# Formulation of Learning Algorithms for Graph Data Classification

A thesis submitted

in partial fulfillment for the award of the degree of

**Doctor of Philosophy**

by

**Asif Salim**

**Department of Mathematics**

**Indian Institute of Space Science and Technology**

**Thiruvananthapuram, India**

**January 2023**

# Certificate

This is to certify that the thesis titled ***Formulation of Learning Algorithms for Graph Data Classification*** submitted by **Asif Salim**, to the Indian Institute of Space Science and Technology, Thiruvananthapuram, in partial fulfillment for the award of the degree of **Doctor of Philosophy** is a bona fide record of the original work carried out by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

| | |
|---|---|
| Dr. S. Sumitra | Dr. C. V. Anil Kumar |
| Associate Professor | Professor & Head |

**Place:** Thiruvananthapuram
**Date:** January 2023

# Declaration

I declare that this thesis titled ***Formulation of Learning Algorithms for Graph Data Classification*** submitted in partial fulfillment for the award of the degree of **Doctor of Philosophy** is a record of the original work carried out by me under the supervision of **Dr. S. Sumitra** , and has not formed the basis for the award of any degree, diploma, associateship, fellowship, or other titles in this or any other Institution or University of higher learning. In keeping with the ethical practice in reporting scientific information, due acknowledgments have been made wherever the findings of others have been cited.

**Place:** Thiruvananthapuram  
**Date:** January 2023

Asif Salim  
(SC16D020)

*This thesis is dedicated to all people who dominate intelligence over their emotions and put rationality above their "blind" devotions.*

# Acknowledgements

Firstly, I would like to express my sincere gratitude to my research advisor Dr. Sumitra S. for the continuous support in my course studies and the research,for the motivation, and the vision. Her guidance helped me in all the time of research works, peer-reviewed publishing, and preparation of this thesis.

Besides my advisor, I would like to thank the rest of my doctoral committee: Prof. Chiranjib Bhattacharyya (Professor, Department of Computer Science and Automation, IISc, Bangalore), Prof. Raju. K. George (Dean - R & D, IIST), Prof. Subrahamanian Moosath. K.S. (Professor, Department of Mathematics, IIST), and Prof. Deepak Mishra (Professor & Head, Department of Avionics, IIST), for their insightful reviews, comments, and encouragement, for the great inspiration to carry out my research from various perspectives.

I thank my fellow lab mates, master students, and research colleagues especially Dr. Shiju S.S for the greater co-operation with my research program, valuable insights, inspiring professionalism, work ethics, and all the fun we had and going to have. I thank my friends at the Indian Institute of Space Science and Technology. I am also grateful to the department office staff for making my life easy for helping with office related works and the friendship.

I would also like to thank my family, my parents, and my fiance for supporting me throughout my research period and for their greater patience.

Last but not the least, I would like to thank every being on this planet living and dead, millions of years of learning and contribution of my predecessors, and the immense wisdom they accumulated that helped me to put a "silly" contribution to that vast sea of knowledge.

Asif Salim

# Abstract

The data in the forms of graphs are crucial in many domains of science and technology like bio-informatics, chemo-informatics, social media analysis, natural language processing, etc. These domains create graphs in a variety of forms. For example, a collection of graphs is created in bio-informatics and chemo-informatics domains where we need to do a conventional classification or regression. The nodes in these graphs may be represented as atoms in a molecule and edges represent the interaction between them. The nodes and edges can be accompanied with a discrete label or some attributes in the form of a vector. In social media and natural language processing, the graph is in the form of a single large network where we need to process on the nodes and edges. This thesis discusses the design of techniques that process different types of graph data by making use of tools in kernel methods and graph signal processing. Although the algorithms are discussed in the context of classification, with appropriate changes in the learning algorithms, they are also applicable for regression problems.

Our first approach is to make use of graph embedding techniques and multiple kernel learning (MKL). The graph embedding is the process of representing the graph in a vector space using properties of the graph while MKL is a framework where the optimal kernel is learned as a linear combination of a set of base kernels. The technique of MKL allows us to incorporate multiple graph embedding into a single learning framework. Hence we designed graph embedding using a multi-view approach, where each view is an embedding of the graph using a graph property. The reproducing kernel used in SVM is represented as a linear combination of the kernels defined on the individual embeddings. The proposed method helps to process a dataset of a collection of graphs with a categorical label information over the nodes and edges.

In the second approach, we made use of the optimal assignment kernel framework to design graph kernels. The bijection associated with the optimal assignment framework is defined between sets that consist of the nodes of the graph kernel arguments. In the proposed kernels, the nodes of the given data are divided into groups named as *neighbourhood sets* on the basis of the labels generated by the Weisfeiler-Lehman (WL) test for graph isomorphism and a matrix representation is defined for them. A kernel is then defined over the domain that consists of the *neighbourhood sets* in terms of the matrix and an aggregate measure of those kernel values is used for defining the kernels. The proposed kernels can be used for analyzing a collection of graphs with categorical labels on the nodes/edges and it can also be extended to the case of attributed graphs in which apart from the labels, the nodes also contain vector information.

The third approach is specifically proposed for attributed graphs. We formulated the design of a reproducing kernel suitable for processing the attributes, in which the similarity between two graphs is defined on the basis of neighborhood information of the graph nodes with the aid of a product graph formulation. We represent the proposed kernel as the weighted sum of two other kernels of which one is an R-convolution kernel that processes the attribute information of the graph and the other is an optimal assignment kernel that processes label information. They are formulated in such a way that the edges processed as part of the kernel computation have the same neighborhood properties and hence the kernel proposed makes a well-defined correspondence between regions processed in the graphs. We found that the kernel value of the argument graphs in each iteration of the WL algorithm can be obtained recursively from the product graph formulated in our method.

The fourth approach is developed to classify the nodes of a single large network in contrast to the previous approaches. The spectral graph convolutional neural networks (SGCN) are utilized for this. In this work, it has been identified that the filters in the state-of-the-art SGCNs are essentially graph kernels in the form of low pass filters. They enforce a smoothness across the graph and use the functions of graph Laplacian as a tool that injects graph structure into the learning algorithm. The existing SGCNs are reviewed in the context of the relationship between graph Laplacian and regularization operators and propose a framework where the state-of-the-art filter designs can be deduced as its special cases. A new set of filters are designed that are associated with a well-defined low pass behavior. We also deduce the connection of support vector kernels and SGCN filters based on our framework.

The efficiency of the first three approaches are evaluated by incorporating the proposed kernels on support vector machines for classification task and the fourth approach on neural networks for semi-supervised node classification task on the real-world data sets and they have shown superior performance in comparison with that of the state-of-the-arts.

# Contents

**List of Figures** **xv**

**List of Tables** **xix**

**List of Algorithms** **xxi**

**Abbreviations** **xxiii**

**Nomenclature** **xxv**

**1 Introduction** **1**

1.1 Organization of the Thesis . . . . . . . . . . . . . . . . . . . . . . . . . . 3

1.2 Major Contributions of the Thesis . . . . . . . . . . . . . . . . . . . . . . 6

**2 Background and Related Works** **9**

2.1 Kernel Theory . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

2.2 1-dimensional Weisfeiler-Lehman color refinement algorithm . . . . . . . . 13

2.3 Spectral Graph Convolutions . . . . . . . . . . . . . . . . . . . . . . . . 14

2.4 Related works . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14

**3 Design of multi-view graph embedding using multiple kernel learning** **21**

3.1 Notations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22

3.2 MKL for multi-view . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23

3.3 Construction of views based on graph embeddings . . . . . . . . . . . . . 24

3.4 Representation Capability of the Views . . . . . . . . . . . . . . . . . . . 31

3.5 Experiment . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34

3.6 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38

**4 Graph Kernels Based on Optimal Node Assignment 41**
4.1 Design of optimal node assignment graph kernels . . . . . . . . . . . . . . 42
4.2 Kernel computation using heirarchy . . . . . . . . . . . . . . . . . . . . . . 47
4.3 Experiment . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 50
4.4 Conclusions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 53

**5 Neighborhood Preserving Kernels for Attributed Graphs 55**
5.1 Neighborhood Preserving Kernel . . . . . . . . . . . . . . . . . . . . . . . 55
5.2 Neighborhood preserving shortest path kernel . . . . . . . . . . . . . . . . 67
5.3 Experiments . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 69
5.4 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 76

**6 Spectral Graph Convolutional Neural Networks in the Context of Regularization Theory 77**
6.1 Regularized graph convolution filters . . . . . . . . . . . . . . . . . . . . . 78
6.2 Experiments . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 86
6.3 Discussion on optimizing network architectures of SGCN . . . . . . . . . . 93
6.4 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 97

**7 Applications 99**
7.1 Brain connectivity and social media data analysis . . . . . . . . . . . . . . 99
7.2 Location-wise spread of Covid-19 . . . . . . . . . . . . . . . . . . . . . . . 101

**8 Conclusions and Future Works 103**
8.1 Future Works . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 104

**Bibliography 107**

**List of Publications 121**

**Appendices 123**

**A From Chapter 4 123**
A.1 Proof of optimal assignment of $\tilde{K}_{ONA}(G, G')$ . . . . . . . . . . . . . . . 123
A.2 Computing $\tilde{K}_{ONA}$ using hierarchy . . . . . . . . . . . . . . . . . . . . . . 124

**B From Chapter 6 127**
B.1 Regularization in graphs, support vector kernels and spectral GCNN filters . 127

B.2 Ablation studies on architectures . . . . . . . . . . . . . . . . . . . . . . . . 128
B.3 Effects of hyper-parameter tuning in Citeseer and Pubmed datasets . . . . . 130

# List of Figures

2.1 Example for hierarchy . . . . . . . . . . . . . . . . . . . . . . . . . . 13

3.1 Multi-view graph embedding . . . . . . . . . . . . . . . . . . . . . . . 23

3.2 View-I formulation: Feature representation of a sample graph (note that node labels are given inside circles and the associated number indicates node identification). The table shows the node label associated with each edges. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25

3.3 View-II formulation: Feature representation of a sample graph. The right table shows the length associated with each shortest paths and left table shows the derived feature representation. . . . . . . . . . . . . . . . . . . 26

3.4 View-III formulation. (a) A sample graph (note that node labels are given inside circles). (b) Labeled shortest path representations, $L^G_\Pi$. (c) Corresponding set $P$ of the graph and its vector representation. . . . . . . . . . . 27

3.5 View IV Formulation. (a) A sample graph. (b) Corresponding set $T$ and vector representation of the graph. (c) Feature construction corresponding to subtrees rooted at node 1, (d) Feature representations corresponding to subtrees rooted at node 2. (e) Feature representations corresponding to subtrees rooted at node 3. (f) Feature representations corresponding to subtrees rooted at node 4. (g) Feature representations corresponding to subtrees rooted at node 5. . . . . . . . . . . . . . . . . . . . . . . . . 30

4.1 **(a)** Two sample graphs. **(b)** The string representation of the nodes and corresponding hashing to a color. **(c)** The graphs with WL labels marked on nodes. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 42

4.2 The illustration of the kernel calculation corresponding to a WL label $l_5$. **(a)** Two graphs $G$ and $G'$ with WL labels marked as different colors on the nodes. **(b)** Constructing the sets $g_{l_i}$. **(c)** Augmenting $g'_{l_5}$ with dummy nodes (node marked as black). **(d)** Constructing the vectors $\mathcal{V}_{v_j}$ and $\mathcal{V}'_{v_j}$ - vectors are obtained from the string representation of nodes 3,4, and 5 in $G$ and node 4 in $G'$ as shown in the Figure 4.1 (b). The dummy node vector representation is taken as zero vector. **(e)** Constructing $M_{g_{l_5}}$ and $M'_{g_{l_5}}$ corresponding to $G$ and $G'$. **(f)** Applying the matrices into kernel definitions $k_{ns_1}$ and $k_{ns_2}$. . . . . . . . . . . . . . . . . . . . . . . . . . 45

5.1 (a) Two sample graphs $G_1$ and $G_2$, (b) WL color refinement, (c) direct product graph, (d) separation of neighborhood preserving or structurally similar edges (bold lines) and dissimilar edges (dashed lines). . . . . . . . 56

5.2 Two nodes $u, v$ in the graph $G$ and two nodes $u', v'$ in the graph $G'$ with the same WL refined labels and corresponding edges having similar label. The product graph $G_P$ containing additional pair of nodes and an additional edge. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 59

5.3 Convolution as vectorization for a sample of two *WL refined edge addresses* $w_1$ and $w_2$. (a) and (b) represents node attributes in $G, G'$ with an address $w_1$ and $w_2$ respectively. (c) represents arranging attribute vectors to compute convolution as a vectorization process, $\oplus$ denotes concatenation and $\times$ denotes Hadamard product. . . . . . . . . . . . . . . . . . . . . . . 67

5.4 Runtime comparison of pairwise and global computation of NP kernel for synthetic graphs at graph density 20%, 40%, 60%, and 80% respectively. . 71

5.5 The plot of cardinality of *WL refined edge addresses* and percentage of kernel values getting updated against WL iterations for the datasets. . . . . 74

6.1 Regularization function, $r(\lambda)$. (a) regularized Laplacian ($s = \{0.5, 1, 1.5, 2\}$), (b) diffusion function ($s = \{0.5, 1, 1.5, 2\}$), (c) one-step random walk ($a = \{2, 3, 4, 5\}$), (d) 2-step random walk ($a = \{2, 3, 4, 5\}$), (e) inverse cosine function. . . . . . . . . . . . . . . . . . . . . . . . . . . . 80

6.2 Regularization function, $r(\lambda)$. (a) ChebyNet, (b) GCN, (c) GraphHeat, (d) IGCN for $k = 2$ , (e) IGCN for $k = 3$. All graphs are for ($c = \{0.2, 0.5, 1.0, 1.5\}$) . . . . . . . . . . . . . . . . . . . . . . . . . . . 84

6.3 SGCN two layer architecture. . . . . . . . . . . . . . . . . . . . . . . . 88

6.4 Accuracy variation with hyper-parameters. (a) Diffusion, (b) 1-step RW, (c) 2-step RW, (d) 3-step RW . . . . . . . . . . . . . . . . . . . . . . . 91
6.5 SGCN three layer architecture. . . . . . . . . . . . . . . . . . . . . . . . . 92
6.6 Accuracy plot with standard deviation of GCN, ChebyNet and best performing model for Flickr, GitHub, Europe Deezer and Facebook Page-page datasets. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 92
6.7 SGCN architecture with a collection of filters. C/A stands for concatenation/averaging . . . . . . . . . . . . . . . . . . . . . . . . . . . . 96

7.1 Procedure in constructing edges of the graph. Red nodes indicate training points and blue nodes testing points. . . . . . . . . . . . . . . . . . . . . 102

B.1 Accuracy variation with hyper-parameters in Citeseer dataset. (a) Diffusion, (b) 1-step RW, (c) 2-step RW, (d) 3-step RW . . . . . . . . . . . . . . 129
B.2 Accuracy variation with hyper-parameters in Pubmed dataset. (a) Diffusion, (b) 1-step RW, (c) 2-step RW, (d) 3-step RW . . . . . . . . . . . . . . 130

# List of Tables

3.1 Dataset details: $|D|$: cardinality of the data set $D$, $|P|$: cardinality of positive class, $|N|$: cardinality negative class, $V^{avg} =$: average of $\{|V|, V \in D\}$, $E^{avg} =$ average of $\{|E|, E \in D\}$, $V^{\max} = \max\{|V|, V \in D\}$, $E^{\max} = \max\{|E|, E \in D\}$. . . . . . . . . . . . . . . . . . . . . . . . . . 36
3.2 Accuracy (along with standard deviation) of the multi-view graph embedding along with state-of-the-art embedding techniques and graph kernels. The rank in statistical significance test is given in the bracket. . . . . . . . 37
3.3 Kernel weights of different views in MKL setting . . . . . . . . . . . . . . 38
3.4 Accuracy using designed R-convolution kernels, graph embeddings & single view. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 39

4.1 Accuracy (along with standard deviation) of the multi-view graph embedding along with state-of-the-art embedding techniques and graph kernels. The rank in statistical significance test is given in the bracket. . . . . . . . 51
4.2 Runtime (wall clock time) of the algorithms . . . . . . . . . . . . . . . . . . 52

5.1 Classification accuracy of the proposed kernels with state-of-the-arts. . . . . 69
5.2 Classification accuracy in MNIST dataset . . . . . . . . . . . . . . . . . . . 73
5.3 Runtime of the proposed kernels with state-of-the-arts. . . . . . . . . . . . . 74

6.1 Frequency response function and output of filters of SGCNs . . . . . . . . 78
6.2 Filters, corresponding regularization function $(r(\lambda))$ and its filter definition 79
6.3 Summary of the datasets . . . . . . . . . . . . . . . . . . . . . . . . . . . 87
6.4 Classification accuracy (in percentage $\pm$ standard deviation) along with average time taken for one epoch (in brackets). . . . . . . . . . . . . . . . 89
6.5 Classification accuracy (in percentage $\pm$ standard deviation). . . . . . . . . 93
6.6 Accuracy of the filters. Std dev. is given in brackets. . . . . . . . . . . . . . 95

7.1 Dataset details: $|D|$: cardinality of the data set $D$, $|P|$: cardinality of positive class, $|N|$: cardinality negative class, $V^{avg} =$: average of $\{|V|, V \in D\}$, $E^{avg}$ = average of $\{|E|, E \in D\}$, $V^{\max} = \max\{|V|, V \in D\}$, $E^{\max} = \max\{|E|, E \in D\}$. . . . . . . . . . . . . . . . . . . . . . . . . . . . 100
7.2 Accuracy of the multi-view graph embedding along with state-of-the-art embedding techniques and graph kernels in brain connectivity and social media data. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 100
7.3 Kernel weights of different views in MKL setting . . . . . . . . . . . . . . . 101

B.1 Classification accuracy (in percentage ± standard deviation) for the models. 131

# List of Algorithms

3.1 Computation of view-III embedding . . . . . . . . . . . . . . . . . . . . . . . 28
3.2 Computation of view-IV embedding . . . . . . . . . . . . . . . . . . . . . . . 31

4.1 Calculating the histogram vector $G_V$ of graph $G$ for $h$ iterations of WL color refinement . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 48

5.1 Computation of histogram vector $G_V$ . . . . . . . . . . . . . . . . . . . . . . 61
5.2 Pairwise computation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 65
5.3 Global computation for one iteration of WL color refinement . . . . . . . . . 66

# Abbreviations

| | |
|---|---|
| MKL | Multiple Kernel Learning |
| WL | Weisfeiler-Lehman |
| SGCN | Spectral Graph Convolutional Networks |
| SimpleMKL | Simple Multiple Kernel Learning |
| SVM | Support Vector Machine |
| OA | Optimal Assignment |
| NP | Neighborhood Preserving |
| G2V | Graph2vec |
| S2V | Subgraph2vec |
| N2V | Node2vec |
| GE-FSG | Graph Embedding with Frequent Sub-Graphs |
| DGK | Deep graph kernels |
| RW | Random walk |
| SP | Shortest path |
| GL | Graphlet |
| WL-S | WL subtree |
| WL-E | WL edge |
| WL-SP | WL shortest path |
| WL-OA | WL optimal assignment |
| TK+MKL | Treelet kernel+MKL |
| ONA | Optimal Node Assignment |
| NPE | Neighborhood Preserving Edge |
| NPO | Neighborhood Preserving Optimal edge assignment |
| NPS | Neighborhood Preserving Shortest path |
| CNN | Convolutional Neural Network |

# Nomenclature

| | |
|---|---|
| $f$ | Function to be learned |
| $x_i$ | Data point indexed by $i$ |
| $\mathcal{X}$ | Space where $x_i$ lies |
| $k$ | Kernel function |
| $y_i$ | Label of data point $x_i$ |
| $N$ | Number of data points |
| $F$ | Reproducing kernel Hilbert space |
| $T$ | Hierarchy tree |
| $V$ | Set of nodes |
| $E$ | Set of edges |
| $G$ | A graph represented as tuple $(V, E)$ |
| $l$ | Function that assigns a label to nodes/edges |
| $\mathcal{G}$ | Set of all graphs and the class labels |
| $\Sigma$ | Alphabet of node/edge labels |
| $\Pi$ | Shortest path |
| $L_\Pi$ | Labelled shortest path |
| $\leq$ | Total order |
| $\Sigma_C$ | Alphabet of WL labels |
| $l_C$ | Function that assigns WL label to nodes |
| $l_i$ | Member in $\Sigma_{WL}$ |
| $d_{l_i}$ | Cardinality difference of $g_{l_i}$ and $g'_{l_i}$ |
| $g_{l_i}$ | Set of nodes with the WL label $l_i$ and $d_{l_i}$ dummy nodes |
| $\mathcal{D}$ | Domain set consisting all $g_{l_i}, \ \forall\ l_i \in \Sigma_C$ for all graphs |
| $\mathcal{V}_{v_j}/\mathcal{V}_{l_i}$ | Vector representation of member node: $j$ in $g_{l_i}$ or equivalently that of $l_i$ |
| $M'_{g_{l_i}}$ | Matrix representation of $g_{l_i}$ whose rows are filled with $\mathcal{V}_{v_j}^T$ |
| $\sigma$ | Function that assigns an integer to the rows of $M_{g_{l_i}}$ |
| $\tilde{V}$ | Set of union of all $g_{l_i}$s in a graph |

| | |
|---|---|
| $B_{l_i}$ | Bijective function from $g_{l_i}$ of $G$ to $g'_{l_i}$ of $G'$ |
| $B$ | Bijective function from $\tilde{V}$ of $G$ to $\tilde{V'}$ of $G'$ |
| $h$ | Number of WL iterations |
| $V_T$ | Set of nodes in $T$ except the nodes in the zero level |
| $G_V$ | Vector that counts frequency of occurrence of elements in $V_T$ for a graph $G$ |
| $N_T$ | Vector that stores norm of $\mathcal{V}_{v_j}$ corresponding to the members in $V_T$ |
| $W$ | Adjacency matrix of $G$ |
| $D$ | Degree matrix of $G$ |
| $L$ | graph Laplacian |
| $\tilde{L}$ | normalized graph Laplacian |
| $\lambda, \tilde{\lambda}$ | Eigenvalue of $L$ and $\tilde{L}$ |
| $\mathcal{F}$ | Filter of a SGCN |
| $g_\theta$ | Frequency response function of $\mathcal{F}$ |
| $r(.)$ | regularization function |

# Chapter 1

# Introduction

The graphs are considered to be universal data structures. It has a philosophical importance in the sense that any real world phenomena can be represented in the form of graphs [1]. Many real world applications can be considered as a set of agents (nodes) interacting together. The interaction can be modelled as the edges of the graph. Nowadays the usage of graphical tools in representing and analyzing real world applications have been increased. For example, fields like bioinformatics, chemoinformatics social networks, and medical imaging generate large amount of structured data in the form of graphs. Not only that, in domains like Natural language processing (NLP), recommender systems and fraud management, the data is represented using graph structure for the analysis. Hence development of efficient algorithms for knowledge extraction from graph based data is very much essential.

The algorithms that process the graph data mainly falls under the domains of kernel methods and graph neural networks. A wide variety of graph kernels were developed for learning tasks such as classification and regression. Most of them were developed for graphs that contain discrete labels over nodes and edges. They are formulated mainly using the principles of R-convolution [2] and focused on a particular graph property for the representation learning. Hence our works focused on (i) developing graph kernels that can make use of multiple graph properties, (ii) processing vector information together with discrete labels and (iii) utilizing other frameworks apart from the conventional R-convolution kernel designs.

The area of graph neural networks has attained significant developments recently. The advancements happened with respect to two classes of algorithms namely spectral and spatial approaches. The spectral graph convolutional networks (SGCNs) use functions of graph Laplacian to define a graph filtering operation. This filtering is defined over the information in graph nodes while spectral approach define a message passing mechanism in graph nodes through information aggregation and sharing. There are a collection of networks that can be categorized into SGCN family. All these networks, in a fundamental level, differs in the function of graph Laplacian used for the filtering. Despite having this common feature, a general platform to combine these networks are not available. We analysed these networks and proposed a common framework connecting graph Laplacian based filtering to its regularization properties. We derived the connection between the SGCN filters and proposed novel ones.

The main challenge related to developing algorithms is to deal with graphs of different types. We developed methods to analyze three types of graph data.

1. Type I: Collection of graphs each having a separate sets of nodes and edges. The nodes and edges have a categorical label associated with them.

   Example: a dataset of chemical molecules to check for carcinogenecity or toxicity. The atomic symbol can be the label on nodes and type of chemical bond as label on edges.

2. Type II: This category is of type-I but contains attribute information in addition. The attribute information are the ones that reside in nodes/edges as a vector information apart from their labels.

   Example: If we consider atomic properties or spatial coordinates of atoms (nodes) they can be categorized as attributed information.

3. Type III: In contrast to type-I and II, this is a single large network consisting of thousands of nodes and edges. The nodes most often contain a vector information and for some graphs the edges also carry such information.

   Example: Social media networks - the persons/pages can be considered as nodes and an edge can be formed between them if they interact. The data can then be analyzed to design an advertisement campaign or to detect malicious activities.

Each type require specialized treatments in designing algorithms for downstream learning tasks. We have leveraged kernel methods paradigms for developing algorithms for type-I and type-II graphs. While for type-III graphs, spectral graph convolutional networks in the context of regularization theory have been made use of. In addition to this, we have also established how the filters are related to graph kernels. The theoretical foundations of these methods are discussed in Chapter 2.

For type-I and type-II graphs, the efficiency of the designed kernels is evaluated for the classification task using the support vector machines (SVM) algorithm, and that of filters developed for type-III graphs is evaluated for the semi-supervised node classification task using a neural network. However, the designed kernels can also be used for regression tasks. For this purpose, a kernel regression algorithm such as support vector regression may be used for type-I and type-II graphs. For type-III graphs, the developed filters can be used for regression with an appropriate neural network architecture and loss function.

## 1.1 Organization of the Thesis

The thesis is organized into the following chapters whose highlights are described below.

### 1.1.1 Theoretical Foundations and Related Works

This chapter describes the foundations of the kernel method theory, state-of-the-art graph kernels, basics of spectral graph neural network and related state-of-the-arts.

### 1.1.2 Design of multi-view graph embedding using multiple kernel learning

The graph embedding is the process of representing the graph in a vector space using properties of the graphs. The existing graph embeddings rely mostly on a single property of graphs for data representation. Hence the effectiveness of embedding depends on the ability of the chosen graphical properties in representing the structural and topological properties of the graph. So a single embedding may be incapable to capture all the characteristics of the data. Hence we designed graph embedding using multi-view approach, where each view is an embedding using a graph property. The method is proposed for learning in type-I graphs.

Multi-view learning deals with the process of learning from data that can be represented by heterogeneous features or views [3], [4]. By considering such data as a single unit could result in problems such as over-fitting as each view has a specific statistical property. This demands for a learning setting where the views shall be treated separately and at the same time it shall also fuse the information from the views in an efficient manner. We made use of multiple kernel learning (MKL) for this purpose. The input space of multi-view learning is then taken as the direct sum of the subspaces in which the graph embeddings lie.

We did analysis on real world data by incorporating the proposed model on support vector machines (SVM). The reproducing kernel used in SVM is represented as the linear combination of the kernels defined on the individual embeddings. The optimization technique used in simple multiple kernel learning (simpleMKL) is used to find the parameters of the optimal kernel.

To analyze the individual representation capability of the embeddings, an R-convolution graph kernel is designed over each of the views. In our experimental analysis, the multi-view graph embedding showed a superior performance in comparison with that of the state-of-the-art graph embeddings as well as graph kernels.

### 1.1.3 Graph Kernels based on Optimal Node Assignment

The R-convolution kernel [2] is a widely adopted framework to design graph kernels where as optimal assignment (OA) framework [5] has not yet gained as much visibility as the former. There are some inherent advantages as well as disadvantages in both the methods. In the case of R-convolution, each graph component has to be compared with each component in the counterpart graph. The same component is part of many computations and hence the graph similarity metric gets over estimated resulting in artificially high value. On the other hand, resorting to OA framework can be a solution to this issue as it compares one component with only one counterpart in the other graph.

The core part of the OA framework is to form a bijection between graph components. The components can be characterized through nodes, edges, shortest paths or any other similar entities. In this work, we formed a well-defined bijection between the nodes to form OA graph kernels. The bijection is enabled through a grouping of the nodes that gathers those ones which have similar neighborhood. The WL test for graph isomorphism is utilized for this purpose. A vector representation is designed for each group that encodes their neighborhood information. The kernel functions are then defined in terms of these vectors. We have proved that the functions are valid OA kernels and hence positive semi-definite. For efficient computations of these kernels, the hierarchy structure associated with the OA framework is utilized. The efficiency of the designed kernels was analyzed using real-world problems by incorporating the kernels in support vector machines. The results were found to be superior in comparison with the other state-of-the-art graph kernels. Although the methods are proposed for type-I graphs, the concepts can be extended to process type-II also.

### 1.1.4 Neighborhood Preserving Kernels for Attributed Graphs

The first two approaches have been developed for type-I graphs. In this chapter, we proposed a kernel that can be used for type-II graphs. It is formulated as a linear combination of two kernels. The attribute information in the graphs are processed by an R-convolution kernel while the label information by an OA kernel.

The main constituent of this is a product graph from which the structurally similar region of the argument graphs are found out. The kernel value is found out by processing only through these similar regions and hence the name neighborhood preserving (NP). An algorithm is proposed in which the kernel value is computed from the product graph by iterating through the edges that are NP. The NP property helps to establish a structural correlation between the edges of the argument graphs. Using this correlation, an R-convolution kernel is defined to process attribute information and an OA kernel to process label information. It has been proved that the product graph in each iteration of the WL color refinement algorithm is a subgraph of that in the previous iteration and this property helps in efficient computation of the kernel values in certain types of graphs.

The concepts formulated are also extended to the case of shortest paths. We have also identified the state-of-the-art graph kernels that can be mapped to the proposed framework. The efficiency of the developed kernels were tested in real world datasets and their performance was found to be superior. The concepts were also applied to image processing and the results are found to be in par with the deep neural networks.

### 1.1.5 Spectral Graph Convolutional Neural Networks in the Context of Regularization Theory

The methods developed for type-III graphs are described in this chapter. They are formulated using the spectral graph neural network (SGCN) approach. The motivation behind this work is the relation between filters of SGCNs and graph kernels which we explain in detail. In this context, the graph kernels are those ones defined between the nodes. It has been found that the filters used in SGCNs are essentially low pass filters that enforce a smoothness across the data in graph nodes.

A framework has been developed for designing SGCN filters in the context of regularization theory. The properties of regularization operators in terms of graph Laplacian are studied and methods are developed to use them as SGCN filters. The condition where the filters become valid support vector kernels on the graph is also discussed. Based on the framework, we developed novel filters and found that the filters used in state-of-the-art SGCNs are its special cases. The filtering property of the state-of-the-art SGCN filters are analyzed with the developed framework and useful observations are deduced. The proposed filters are applied in a semi-supervised node classification task and their performance was found to be superior.

The chapter also discusses about certain points that can help in improving the conventional architecture of SGCN in the context of our observations. We have identified certain latest works that are done in this direction and discuss about further possible improvisations.

### 1.1.6 Practical Applications

In this chapter, the practical applications of the methods have been discussed. The proposed graph kernel methods are applied in brain connectivity and social media data analysis. The proposed filters of SGCN are applied in a proof-of-concept way in the prediction of location wise prediction of Covid-19 cases.

### 1.1.7 Conclusions and Future Works

The conclusion and the future works are discussed in this chapter.

## 1.2 Major Contributions of the Thesis

The major contributions of the thesis are listed below.

1. The existing graph embedding as well as graph kernels are designed mostly on the basis of single graph property and hence cannot capture all the characteristics of the data. Hence in the first approach we proposed a multi-view learning strategy in which the graph data is represented as the direct sum of the vectors corresponding to the views. For applying kernel algorithms on the data, a suitable kernel is selected for each view and the optimal kernel associated with the data is represented as the linear combination of such kernels. The Simple MKL setting assigns a weight to each of the views and they add to one. Hence it is possible to identify the contribution of each of the views in representing the graph via the optimal kernel. This helps to bring valuable insights in identifying the importance of individual graph representations.

2. The graph kernels were designed using the relatively less explored OA kernel framework for graph kernel designs. The bijection required for graph comparison is char-

acterized among the graph nodes. For efficient graph representations, the nodes are assigned to well-defined groups on the basis of their neighborhood information. The graph kernels are then defined in terms of these groups. The validity of the kernels were mathematically proved in the context of OA framework. The algorithms for the efficient computations using the hierarchy structure defined in OA framework were also developed.

3. We designed a product graph from which the structurally similar regions of the argument graphs can be found out. Such regions in the graph are called neighborhood preserving (NP). By processing through NP edges it is possible to define kernels that process the attribute information of the graphs along with their labels. The state-of-the-art graph kernels were identified that fit in this framework. It had also been proved that kernel value obtained in each iteration of the Weisfeiler-Lehman (WL) color refinement algorithm using argument graphs can be obtained in a recursive manner from the product graph formulated in our method. Using this property, we developed algorithms to compute the proposed kernels. The concepts were extended to shortest paths. This method can be applied to image processing use cases. If an image consists of super pixels whose neighborhood structures are visually evident, we can convert the pixels as nodes and connection between structures as edges. An example is the images of digits.

4. The family of Spectral Graph Convolutional Neural Networks (SGCN) in the context of regularization theory in graphs has been explored. We studied regularization operators in graphs characterized by its Laplacian, its relation to SGCN filters and the conditions where the filters become valid support vector kernels. In the context of this work, it can be proved that the SGCN filters are essentially low pass filters. Their filtering behaviour had also been thoroughly examined. A general platform was developed for designing filters of desired filtering behaviours. A set of novel filters were proposed using this platform. The state-of-the-art SGCN filters can be deduced as a special case of these filters. Based on the formulated theories, the improvisations that can be done on the conventional SGCN architecture were discussed.

# Chapter 2

# Background and Related Works

The theoretical background on which the algorithms are formulated and the related works are discussed in this chapter. The methods proposed for type-I and type-II graphs are based on kernel methods and the related theory is discussed in Section 2.1. There are a collection of graph embedding and graph kernels proposed in the literature. These works which are related to the proposed works in this thesis are discussed in Sections 2.4.1 and 2.4.2. The Section 2.4.1 discusses about graph embedding methods and 2.4.2 discusses about graph kernels. These discussions are helpful to understand the approaches and methodologies that are applied to graph classification. It helps in identifying the novelties of the proposed approaches and its benefits. In the proposed works, we have used 1-dimensional Weisfeiler-Lehman color refinement as a major tool for label aggregation in the nodes and it is explained in Section 2.2.

The theoretical foundation of spectral graph theory used for developing methods for type-III graphs is discussed in Section 2.3. A discussion on state-of-the-arts SGCNs is given in Section 2.4.3. These sections describe how spectral graph theory is incorporated into the neural network paradigm to enable learning on graphs. Our proposed work on SGCNs is a generalization of these networks to a common framework in the context of regularization theory in graphs. The works that discuss the regularization in graphs are discussed in Section 2.4.4 and works that analyze spectral properties of SGCNs are discussed in Section 2.4.5.

## 2.1 Kernel Theory

Let $\mathcal{D} = \{(x_i, y_i)|i = 1, ..., N\}$ be the training data points, where $x_i \in \mathcal{X} \subseteq \mathbb{R}^n$ and $y_i \in \mathbb{R}$. Let $f$ be the function that generates the data and let it belongs to a Reproducing Kernel Hilbert Space ($F$). By definition, RKHS is a space of functions in which every point evaluations are bounded linear functionals. Therefore, if we define $L_{x_i}(f) : F \to \mathbb{R}$, such that $L_{x_i}(f) = f(x_i), x_i \in \mathcal{X}, f \in F$, then by definition of RKHS, $L_{x_i}, i = 1, 2, ...$ are bounded linear functional and hence by Riesz representation theorem, $\exists k_{x_i} \in F$ such that

$$L_{x_i}(f) = f(x_i) = \langle f, k_{x_i} \rangle, f \in F$$

where $k_{x_i}$ is unique and depends only on $L_{x_i}$, that is, $x_i$. Hence we can define a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that

$$k(x_i, x_j) = \langle k_{x_i}, k_{x_j} \rangle, x_i, x_j \in \mathcal{X}$$

where $k$ is called the reproducing kernel of $F$. Corresponding to every RKHS there exists a unique reproducing kernel and vice versa. $F$ is the closure of the span of $\{k_{x_i}, x_i \in \mathcal{X}\}$ and hence every $f \in F$ can be written as $f = \sum_i \alpha_i k_{x_i}, \alpha_i \in \mathbb{R}$. The inner product $\langle ., . \rangle_F$ induces a norm on $f \in F : \|f\|_F^2 = \langle f, f \rangle_F = \sum_i \sum_j \alpha_i \alpha_j k(x_i, x_j)$.

The kernel learning methods use regularized risk functional defined as

$$R(f) = \sum_{i=1}^{N} l(y_i, f(x_i)) + \frac{\lambda}{2} \|f\|^2 \tag{2.1}$$

where $\lambda > 0$ is the regularization parameter and $l(y_i, f(x_i))$ is empirical loss term.

The solution to (2.1) has the general form

$$\tilde{f}(x) = f(x) + b$$

where $\alpha_i, b \in \mathbb{R},\ x_i, x \in \mathcal{X}$. In this case, by semi-parametric representer theorem $\tilde{f}$ can be represented as

$$\tilde{f} = \sum_{i=1}^{N} \alpha_i k_{x_i} + \sum_{i=1}^{M} \beta_i \psi_i$$

where $\beta_i \in \mathbb{R}$ and $\{\psi_i\}_{i=1}^{M}$ are a set of real valued functions on $\mathcal{X}$.

### 2.1.1 Multiple Kernel Learning

In multiple kernel learning paradigm, there are many ways of learning the kernel $k$. One of the techniques is to represent it as a linear combination of base kernels under consideration. That is,

$$k(x, z) = \sum_{l=1}^{p} w_l k_l(x, z) \tag{2.2}$$

where kernel weights $w_l \geq 0, l = 1, 2 \ldots p$, $p$ is the number of base kernels and $x, z \in \mathcal{X}$.

The SimpleMKL [6] is one of the prominent works in multiple kernel learning algorithms. In this technique, in order to make the kernel weights sparse, an additional constraint, $\sum_l w_l = 1$, has been incorporated. The dual function of the SVM classification in terms of SimpleMKL can be formulated as,

$$\begin{aligned} J(\alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j & \sum_{l=1}^{p} w_l k_l(x_i, x_j) \\ subj.\ to & \\ & \sum_i \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C \\ & \sum_l w_l = 1 \\ & w_l \geq 0 \end{aligned} \tag{2.3}$$

The positive semi-definiteness of $k$ is guaranteed as the coefficients $w_l, l = 1, 2, \ldots p$ are non-negative [6]. Each $w_l$ value can be considered as the weight corresponding to each kernel. As higher the value of $w_l$, the higher is the influence of the corresponding kernel on determining the optimal kernel.

### 2.1.2 R-convolution kernels

The R-convolution kernel formulated by Haussler [2] is a generalized framework for processing structured data. This framework involves the decomposition of data into its constituent parts and a kernel is defined in terms of such decomposition. The shortest path kernel [7], subgraph matching kernel [8] etc. are examples of R-convolution kernels.

Let $\mathcal{X}$ be a separable metric space consisting of discrete structures like string, trees or graphs. Let $x \in \mathcal{X}$. Assume that $x$ can be decomposed to $D$ number of components or parts. Let $\tilde{x} = \{x_1, \ldots, x_D\}$ be this decomposition, where $x_i \in X_i$ and $X_i, 1 \leq i \leq D$, be a non empty, separable metric space. Define a relation $R$ as

$$R = \{(\tilde{x}, x) \in X_1 \times \cdots \times X_D \times \mathcal{X} | \tilde{x} \text{ are parts of } x\}.$$

That is, $R$ is true iff $\tilde{x}$ is a valid decomposition of $x$. Now it is possible to define the inverse relation, $R^{-1}(x) = \{\tilde{x} : R(\tilde{x}, x)\}$. Note that $R$ is finite if $R^{-1}$ is finite $\forall x \in \mathcal{X}$. Then the

R-convolution kernel $K_{R_{conv}} : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is defined as

$$K_{R_{conv}}(x, y) = \sum_{\tilde{x} \in R^{-1}(x), \tilde{y} \in R^{-1}(y)} \prod_{i=1}^{D} k_i(x_i, y_i)$$

where $(x_i, y_i)$ is the $i^{th}$ component of $(\tilde{x}, \tilde{y})$ and $k_i(.,.) : X_i \times X_i \rightarrow \mathbb{R}, 1 \leq i \leq D$ is the kernel corresponding to $i^{th}$ component.

### 2.1.3 Valid optimal assignment kernel framework

The optimal assignment kernel is defined as follows: Let $\mathcal{X}^n, n \in \mathbb{N}$ denote the set of all $n$-element subsets of $\mathcal{X}$ and $B(X, Y)$ the set of all bijections between $X, Y$, where $X, Y \in \mathcal{X}^n$. The optimal assignment kernel $K_B^k$: $\mathcal{X}^n \times \mathcal{X}^n \rightarrow \mathbb{R}$ is defined as

$$K_B^k(X, Y) = \max_{\beta \in B(X,Y)} W(\beta) \quad where \quad W(\beta) = \sum_{(x,\beta(x))} k(x, \beta(x)), \tag{2.4}$$

where $k$ is a strong kernel defined on $\mathcal{X} \times \mathcal{X}$ [5]. The above concept can be extended to the case where the underlying domain consists of sets with unequal cardinality by adding dummy objects $d$ to the smaller set, where $k(d, x) = 0, \ \forall x \in \mathcal{X}$. The concept of the strong kernel is discussed below.

#### 2.1.3.1 Strong kernels and hierarchies

The strong kernel can be explained using two different concepts:

1. In terms of an inequality constraint on the kernel values.

A function $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}^+$ is called strong kernel if

$$k(x, y) \geq \ min \ \{k(x, z), k(z, y)\} \quad \forall x, y, z \in \mathcal{X}.$$

That is, once we consider $x$ and $y$, there is no other element $z$ in $\mathcal{X}$ where both $x$ and $y$ are more similar to $z$ than themselves.

2. In terms of hierarchy defined on the domain of the kernel.

The hierarchy can be constructed in the form of a rooted tree, $T$, as follows. It is assumed that the leaves of $T$ correspond to elements in $\mathcal{X}$. It has to be noted that the tree forms a nested structure, that is, each inner node in $T$ apart from the leaves corresponds to a subset of elements in $\mathcal{X}$. For example, consider a hierarchy in Figure 2.1, the node $v$ is an inner vertex that corresponds to nodes $a$, $b$ in $\mathcal{X}$, and node $r$ corresponds to node $c$ in $\mathcal{X}$.

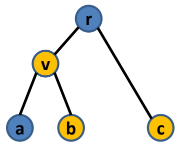


**Figure 2.1:** Example for hierarchy

A weight is defined to each of the nodes in $T$. Let $w : V(T) \to \mathbb{R}^+$ be a weight function such that $w(v) \geq w(p(v))$ for all $v$ in $T$ where $p(v)$ denotes parent node of the node $v$ and $V(T)$ is the set of nodes in $T$. The tuple $(T, w)$ is referred as hierarchy. It has to be noted that the base kernels which are strong in $\mathcal{X} \times \mathcal{X}$ mentioned earlier can be derived from the hierarchy structure [5].

With the above concepts, Kriege et.al [5] proved that *a kernel $k$ on $\mathcal{X} \times \mathcal{X}$ is strong if and only if it is induced by a hierarchy on $\mathcal{X}$*.

It has to be noted that the hierarchy corresponding to a base kernel $k$ is not unique. Considering this, to prove that a kernel is an optimal assignment on a structured data, it is enough to prove that the underlying base kernel is strong and there exists a hierarchy that induces the corresponding base kernel from which (2.4) can be calculated.

Examples for optimal assignment kernels are kernels defined on vertices and edges, WL-optimal assignment kernel by Kriege et.al [5] and optimal assignment kernels on molecular graphs by Frohlich et.al [9].

## 2.2 1-dimensional Weisfeiler-Lehman color refinement algorithm

1-dimensional Weisfeiler-Lehman color refinement [10] algorithm is a tool that has been recurrently used in our graph kernel designs. A brief discussion about the technique is given below.

The algorithm creates a representation for each node in the graph in the form of a string. The node label is the first element of the string and the remaining elements are node labels of the neighbouring nodes, which are arranged in lexicographic order. A new label is then assigned to each such string representation using a hashing function. The above process is repeated until the labels become consistent. An example of the processes involved in the algorithm is given in Figure 4.1.

## 2.3 Spectral Graph Convolutions

The work we proposed for type-III graphs are based on SGCNs. In this section, we discuss the basic foundations of spectral graph convolution and graph filtering.

Spectral convolutions on graphs can be defined as the multiplication of a signal on nodes with a graph filter. We define $f = (f_1, \ldots, f_n) \in \mathbb{R}^n$ as a signal on $n$ nodes on a graph $G$. The graph filter, $\mathcal{F}$, can be regarded as a function that takes $f$ as input and outputs another function $y$. The convolution operation can be written as,

$$y = \mathcal{F}f = U g_\theta(\Lambda) U^T f,$$

where $g_\theta(\Lambda) \in \mathbb{R}^{n\times n} = \text{diag}(g_\theta(\lambda_1), \ldots, g_\theta(\lambda_n))$ is a diagonal matrix. The function $g_\theta() : \mathbb{R} \to \mathbb{R}$ (with parameters $\{\theta\}$) is defined as *frequency response* function of the filter $\mathcal{F}$.

## 2.4 Related works

In this section, we discuss the existing graph embeddings, graph kernels, and spectral graph convolutional networks.

### 2.4.1 Graph embeddings

The hierarchical graph embedding by Mousavi et.al [11] builds a hierarchical representation of graphs by clustering the nodes at each layer to form super nodes for the next layer. The edges corresponding to such newly formed nodes are defined with respect to the edges of the previous layer. At each level, an embedding is done in terms of node and edge attributes and then such embeddings are stacked together to form the final embedding. Riesen et.al [12] made the vector representation of the graph with respect to a set of prototype graphs. A measure based on the concept of graph edit distance computes how the graph under consideration differs from prototype graphs and that information is used for the embedding. Gibert et.al [13] assumes that a multidimensional attribute is present with each node. They proposed a method to map each of such attribute to a prototype vector. The embedding is then built based on the statistical properties of those mappings. In the work of Luqman et.al [14], the features of a graph are constructed from its topological, structural, and attribute information using a predefined set of rules. These features are encoded as fuzzy histograms in a vector space to define the embedding. The frequency of predefined sub graphs is used for vector representation by Sidere et.al (2008) [15] whereas Wilson et.al [16] encode the spectral features of Laplacian matrix into symmetric polynomials to achieve permutation independence of nodes and then the coefficients of those polynomials are used to formulate the graph embedding.

A recent development in graph embedding involves the incorporation of the concepts of neural word and doc2vec embedding methods developed by Mikolov et.al [17], [18] and Le et.al [19] respectively. The neural word embedding or word2vec is a systematic embedding of words to a vector space by taking the context of the word into consideration. This helps to capture the semantic relations between the words. On the other hand, the doc2vec by Le et.al [19] is formulated on the basis of the embedding of documents. The word2vec and doc2vec concepts are adopted to the case of graphs, for which the embeddings are built upon nodes, subgraphs, graph as a whole etc. Such designs can be seen in the works like graph2vec by Narayanan et.al [20], subgraph2vec by Narayanan et.al [21] and node2vec by Grover et.al [22]. This concept is further improvised by Nguyen et.al [23] by taking into consideration the occurrence of frequent subgraphs. Yanardag et.al [24] introduced these concepts into Weisfeiler-Lehman kernel [25] and called it deep graph kernel.

### 2.4.2 Graph kernels

The graph kernels can be classified into two: (a) kernel between nodes within a graph (b) kernel between graphs. In the first category, notable work is the diffusion kernel developed by Kondor et.al [26]. In this, the random walks (of length zero to infinity) happening in graphs are transformed into a diffusion setting and a kernel has been designed in discrete space which is analogous with Gaussian kernel in continuous cases. Another work in this direction is done by Smola et.al [27], called the regularized kernels. In this case, a class of monotonically decreasing functions are utilized to build kernels and corresponding regularization operators [28].

The second category of kernels are formulated by using graph properties such as random walks, shortest paths, tree or subtree structures and subgraphs. In the random walk kernels by Gartner et.al [29], the random walks are done on the graphs under consideration and the number of matching walks are utilized as an information to build kernel. This work was modified by Vishwanathan et.al [30] by taking the pair of graphs into a common platform using direct product graph for finding the similar random walks. Another notable work was done by Kashima et.al [31] called marginalized kernels in which the labeled graphs are used, that is, each node has a label and random walks are done by considering the labels and the count of these labeled paths are taken into consideration for feature representation. This work was modified by Mahe et.al [32] by introducing Morgan indexing [33] in nodes. They also developed a mechanism to avoid tottering (repeated visits to a node). Borgwardt et.al [34] improvised the random walk kernels proposed in [29, 31] to process the attribute information. Zhang et.al [35] used the return probabilities of random walks of fixed length in their kernel design for the attributed graphs.

The graph kernels using shortest paths are usually formulated by constructing a vector for each graph based on the number of occurrence of $i$-lengthed shortest paths. Along with this information, sometimes the label information is also being used. The notable work in this direction was done by Borgwardt et.al [7]. The GraphHopper kernel formulated by Feragen et.al [36] processes the attributed graphs by making use of shortest paths.

The formulation using subtree structures is the most popular approach for graph kernel construction and a lot of work has been done in this direction. The subtree kernel defined by Gartner et.al [37] makes use of subtree patterns and node labels. Mahe et.al [38] modified this work by reducing the complexity of subtree patterns. They also introduced a controlling parameter in the formulation that gives a greater flexibility in kernel definitions. The Weisfeiler-Lehman (WL) kernel defined by Shervashidze et.al [25] makes use of WL test for graph isomorphism and it is formulated by retrieving the subtree patterns with node labels in an iterative fashion. Bai et.al [39] proposed aligned subtree kernel which takes into account the relative positions of the nodes within the subgraphs. Martino et.al [40] designed a kernel from the tree structures by capturing the specific graph visits. Truncated tree based kernel proposed by Ye et.al [41] uses a feature vector for the graphs in which the number of breadth first search (BFS) trees rooted on each node are counted and the structural role of the individual vertices are compared in this way.

There exist graph kernels that are based on the properties of subgraph structures. The cyclic pattern kernels developed by Horvath et al [42] utilized pairs of matching cyclic patterns along with tree patterns in graphs to define kernels. Shervashidze et.al [43] made use of graphlets, that is, graphs as collection of many subgraphs, in a systematic manner such that an exhaustive feature extraction is avoided. A computationally efficient procedure for this kernel is proposed by Aziz et.al [44]. De Grave et.al's [45] formulation is based on the similarity between subgraphs within a graph. The kernel design of Kriege and Mutzel [8] made use of isomorphic subgraphs while the work of Orsini and Frasconi [46] is on the basis of vertex similarity in subgraphs and both of these kernels are suited for attributed graphs also. Bai et.al [47] formulated a subgraph representation on each vertex and kernel is defined on the basis of Jensen-Shannon divergence of the subgraph structures. The ordered decomposition directed acyclic graph (ODD) kernel [40] creates a representation of the graph as a bag of directed acyclic graphs (DAG). Each node pair in each DAG pair is then processed in a similarity function to derive the kernel value.

The other formulations of graph kernels include global graph kernels by Johansson et.al [48] which is built on the basis of geometric properties in graphs by making use of Lovasz number, propogation kernels by Neumann et.al [49] whose framework depends on the pattern of information flow through a graph, kernel fomulated by Guazere et.al [50] that makes use of graph edit distance and the kernels based on information theory by Bai et.al [51] and Xu et.al [52], [53]. Kriege et.al [5] formulated the optimal assignment kernels where optimal bijection is done between graph components to make the kernel definition. There also exist domain specific kernels, for example, the kernel tailored for chemoinformatics domain by Guazere et.al [54], in which the chemical properties such as stereo isomerism and graph theoretic approach are made use of in the design. The Wasserstein WL graph kernels formulated by Togninalli et.al [55] model a distribution over the graph components rather than using the corresponding feature vectors directly into a kernel definition. Multiscale Laplacian kernel proposed by Kondor et.al [56] uses a base kernel to extract information from vertices and this kernel is used recursively on subgraphs in their design. Rieck et.al [57] used persistent homology (PH) concept in topological data analysis in formulating a graph classification algorithm where the PH is modeled on WL subtree features. In addition to this, there are graph kernels proposed for attributed graphs that are based on the discretization of attributes. Propagation kernel [58] by Neumann et.al, Hash graph kernels by Morris et.al [59] etc, are examples.

MKL strategy is applied in several works related with graph learning. Aiolli et.al [60] used MKL to group the similar features involved in a graph kernel design and to define a kernel for each group. Gauzere et.al [54] used MKL to assign a weight for each of the feature they used for kernel construction, whereas Donini et.al [61] applied MKL to subsample the dataset. It has to be noted that our proposed method based on multi-view learning is different from these since we use multiple embeddings and MKL to assign a weight for each embedding to derive the optimal graph representation.

### 2.4.3 Spectral Graph Convolutional Neural Networks

The spectral filter designs for GCNNs start with the work by Bruna et.al [62] where the graph Fourier transform of the signals on nodes is utilized. The filter is learned as a diagonal matrix with learnable weights which can be interpreted as direct interaction with the eigenvalues of graph Laplacian. Henaff et.al [63] improvised this work by incorporating the localization of the filters. In ChebyNet [64], the filter is defined as the linear combination of powers of the Laplacian. The scaling factors of the linear combination are considered as the parameters to be learned from the network. Kipf et.al [65] proposed graph convolutional network (GCN) as the first-order approximation to spectral convolutions defined in ChebyNet. GraphHeat [66] uses a negative exponential processing on the eigenvalues of graph Laplacian to improve the smoothness of the function to be learned by penalizing high-frequency components of the signal. Li et al [67] proposed improved graph convolution networks (ICGN) by proposing higher-order filters used in [65]. Our proposed method analyses these architectures and for the sake of clarity we describe these methods as *traditional* SGCNs. Apart from *traditional* SGCNs, there are a few networks that can be categorized into the spectral class as described below.

The filters of the *traditional* networks are computed in the fourier domain described by the eigenspace of (normalized) graph Laplacian. The graph wavelet network [68] on the other hand defines the convolution over graph wavelet transform rather than graph fourier transform. There are certain works that can be categorized as spectral approaches although they learn additional graph connectivity information apart from that provided by graph Laplacian. Adaptive graph convolutional network [69] models the *residual* connections between the nodes that are not directly connected and given by the graph Laplacian. The dual graph convolutional network [70] proposed positive pointwise mutual information (PPMI) matrix to capture global information along with the Laplacian that captures only local information about the graph structure.

CayleyNets [71] are SGCNs in which the Cayley polynomials are used to compute the filters that give a control over the frequency reponse compared to the traditional approach. LanczosNet [72] uses Lanczos algorithm to construct the low rank approximation of graph Laplacian and constructs learn-able filters.

### 2.4.4 Regularization in graphs

Regularization associated with graphs has emerged along with the development of algorithms related to semi-supervised learning [73]. The data points are converted into a network and the unknown label of a data point can be learned in a transductive setting. A generalized regularization theory is formulated where the classical formulation is augmented with a smoothness functional on a graph in terms of its Laplacian. These concepts are used for semi-supervised learning and in related applications [74], [75], [76], [77]. Smola et.al [27] leverage these concepts to define support vector kernels on graph nodes and they also formulated its connection with regularization operators.

### 2.4.5 Spectral analysis of SGCNs

There are works that analyse the frequency response of the filters of SGCNs. NT et.al [78] have shown that the filters learned by *traditional* SGCNs are low-pass filters. They have experimentally analyzed this by checking the accuracy against the frequency components in citation datasets and found that the addition of high pass components cause decrease in accuracy. They also proposed a simplified architecture for graph learning where the graph filters are made parameter free and had observations that is similar to ours as explained in Section 6.3.1. The low pass filtering property of ChebyNet and GCN networks are analyzed by Wu et.al [79]. They have shown that the renormalization trick [65] applied to GCN shrinks the spectrum of the modified Laplacian from $[0, 2]$ to $[0, 1.5]$ which favors the low pass filtering process. Li et.al [67] and Klicpera et.al [80] has given the frequency response analysis of their proposed filters.

There have been certain works that modify or propose new architecture in the context of low pass behaviors of SGCNs. Li et.al [81] have analyzed the low pass filtering property to observe the need for incorporating high frequency components for certain applications and proposed modifications to the SGCN architectures. Similarly, Balcilar et.al [82] have observed the need for high pass and band pass components for certain applications. Chang et.al [83] have suggested separate learn-able parameters for low pass and high pass components of the features rather than treating them in a single unit. Bo et.al [84] theoretically analyze the influence of low and high pass components for node representation learning and proposed new propagation techniques that incorporate various components suitable for the learning task. Fu et.al [85] have deduced that SGCNs basically denoise the node features while graph attentions [86] denoise edge weights.

Li et.al [87] have observed that graph convolution in *traditional* SGCNs are equivalent to a Laplace smoothing operation. The operation enforces a similar representation for those nodes which are topologically similar. Inspired by the low pass behaviors of graph Laplacian, Wu et.al [79] have simplified the SGCN architecture by removing certain weights and non-linearities between the layers. Zhang et.al [88] analyze the low pas filtering property of GCN and propose higher order graph convolution suitable for graph clustering applications. Adding to this, Gama et.al [89] study the perturbation of graphs, consequent effects in filters, and proposed the conditions under which the filters are stable to small changes in the graph structure.

In the next chapters, we discuss the proposed methods starting with the multi-view MKL approach formulated for type-I graphs.

Chapter 3

# Design of multi-view graph embedding using multiple kernel learning

The graph embedding is the process of representing a graph in a vector space [90]. This is done with the aid of a vector constructed using the graph properties of the data. Hence the effectiveness of embedding depends on the ability of the chosen graph properties in representing the structural and topological properties of the graph. Many methods have been proposed for graph embedding that differs in the graph properties used for its formulation. On the other hand, graph kernel does not undergo any intermediate embedding as it refers to a kernel function whose domain is the input space of graphs [91], [92] [30]. The R-convolution framework proposed by Haussler(1999) is one of the most widely used designs for formulating graph kernels [2].

The multi-view learning deals with the process of learning from data that can be represented by hetrogeneous features or views [3], [4]. By considering such data as a single unit could result in problems such as over-fitting as each view has a specific statistical property. The multi-view approaches use a separate function to optimize each view individually and then these functions are jointly optimized, which in turn helps to improve the learning performance. Examples can be found in [93], [94], [95].

We adopted the technique of multi-view learning in designing the graph embedding where view of a graph related to a graph property involves the generation of a vector from the graph using that property. The data from chemo-informatics, brain connectivity and social media domains are selected for our study. The efficiency of the proposed method was experimentally verified by incorporating the multi-view embedding on support vector machine. The multi-view embedding approach produced superior results in comparison with that of state-of-the-art techniques. The representation capability of the individual embedding has also been analyzed by designing an appropriate graph kernel based on R-convolution kernel framework.

The rest of the chapter is organized as follows. The notations used in the chapter are discussed in Section 3.1. In Section 3.3, the techniques we used for constructing the views are described. The Section 3.4 describes about the R-convolution designs we have formulated to assess the individual embeddings. The experiments results are in Section 4.3 while concluding remarks are given in Section 3.6.

## 3.1 Notations

We represent a graph as $G = (V, E)$ where $V$ and $E$ are the set of nodes and edges respectively. Define a mapping $l : V \to \Sigma$, such that $l(v)$ is the label associated with a node $v$ and $(\Sigma, \leq)$ is a total ordered set that consists of node and edge labels. The shortest path between node $v_i$ and node $v_j$ in a graph is represented as $\Pi(v_i, v_j) = \{v_0, v_1, \ldots, v_n\}$ where $v_0 = v_i$ and $v_n = v_j$ and $(v_l, v_{l+1}) \in E, l = 0, 1, \ldots n-1$. The labeled path is defined as the path where nodes are replaced with their corresponding labels, that is, $L_\Pi(v_0, v_n) = \big[l(v_0), l(v_1), \ldots, l(v_n)\big]$.

Let $\mathcal{G} = \{(G_1, y_1), (G_2, y_2) \ldots (G_N, y_N)\}$ be the given data, where each $G_i$ is a graph and $y_i, i = 1, 2, \ldots N$ is a class label associated with the data. The data for the learning task of classification is considered for our analysis.

The graph properties are used for embedding the graphs into vector spaces. Let $M$ properties of graph are considered and using each property, $G_i, i = 1, 2, \ldots N$ is converted into a vector. Let $\mathcal{X}_m \subseteq \mathbb{R}^n$ be the vector space in which the vectors generated from the graphs using the $m^{th}$ property lie, where $m = 1, 2, ...M$. Then the input space of multi-view learning is

$$\mathcal{X} = \mathcal{X}_1 \oplus \mathcal{X}_2 \oplus \ldots \mathcal{X}_M \tag{3.1}$$

The $M$ components of $x \in \mathcal{X}$ are called its views. Thus each view is a set of attributes (features) of the data based on a graph embedding. We used kernel methods for data analysis and for that, graph kernel on $\mathcal{X} \times \mathcal{X}$ is represented and learned with the help of multiple kernel learning (MKL) framework, whose discussion is given in Section 2.1.1.

The representation capacity of each view has been analysed using graph kernel as well as embedding approach. The graph kernel for individual views is formulated using the concept of R-convolution framework which is explained in Section 2.1.2.

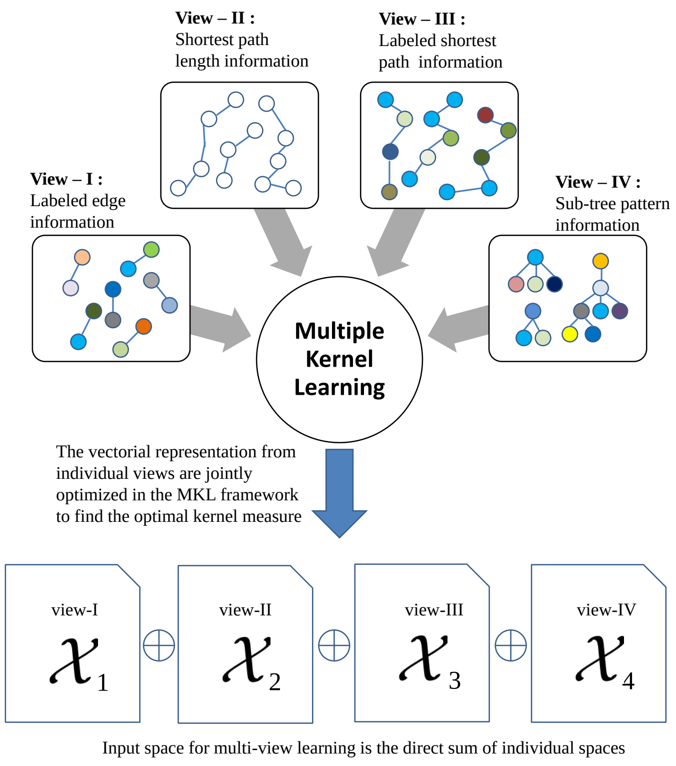

View – II :
Shortest path length information
View – III :
Labeled shortest path information
View – I :
Labeled edge information
View – IV :
Sub-tree pattern information
Multiple Kernel Learning
The vectorial representation from individual views are jointly optimized in the MKL framework to find the optimal kernel measure
view-I
view-II
view-III
view-IV
$\mathcal{X}_1 \oplus \mathcal{X}_2 \oplus \mathcal{X}_3 \oplus \mathcal{X}_4$
Input space for multi-view learning is the direct sum of individual spaces

**Figure 3.1:** Multi-view graph embedding

## 3.2 MKL for multi-view

In (2.2), all the kernels are defined on the same input space while in (3.2), each kernel has its own input space.

The multi-view learning involves as much kernels as the number of views and hence the technique of MKL is adopted for representing the multi-view kernel. We define the multi-view kernel as

$$k_{multi-view}(x_i, x_j) = \sum_{m=1}^{M} d_m k_m(x_i^m, x_j^m), d_m \geq 0 \tag{3.2}$$

where $k_m$ is the kernel defined on $\mathcal{X}_m \times \mathcal{X}_m$, $d_m$ is the associated weight of the kernel $k_m$, $x_i = x_i^1 \oplus x_i^2 \cdots \oplus x_i^M, x_i^m \in \mathcal{X}_m, i = 1, 2, \ldots M$, and $M$ is the total number of views under consideration.

The positive semi-definiteness of $k_{multi-view}$ is guaranteed as the coefficients $d_m$ are non-negative [6]. Each $d_m$ value can be considered as the weight corresponding to that kernel. A higher value of $d_m$ indicates a higher influence of the corresponding kernel on determining the optimal one. Hence this formulation helps to find the important views and brings a further insight into the effectiveness of different feature representations we used. This approach is flexible in the sense that any further feature extraction results can be added as another view in a plug-in mode.

## 3.3 Construction of views based on graph embeddings

The views are constructed corresponding to an embedding strategy by making use of the structural as well as intrinsic properties of the graphs. The description of the formulation of views is given below.

- View-I is formulated on the basis of the information of the edges in the graph.
- View-II is constructed on the basis of shortest path length information.
- View-III considers labels involved with shortest paths.
- View-IV is built using the subtree patterns.

The four views together help to capture the characteristic properties of the data and they form the different embeddings of the graph data. The data point is represented as the direct sum of the vectors corresponding to these views, that is,

$$x = x^{view:I} \oplus x^{view:II} \oplus \ldots x^{view:IV}$$

### 3.3.1 View-I: Embedding based on labeled edge information

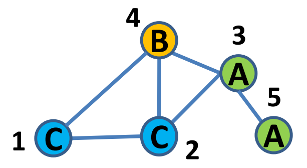
In view I, the information of the labels associated with each edge of the graph is used. As each edge is associated with two labels, for getting their information, a set of strings of length two, that is, $\Sigma^2 = \{xy | x, y \in \Sigma, x \leq y\}$ is constructed. As the strings $xy$ and $yx$ convey the same meaning as far as undirected graphs are concerned, with the aid of the constraint $\leq$ imposed on $\Sigma$, only one among them becomes a member in $\Sigma^2$. The elements of $\Sigma^2$ are arranged on the basis of the total order $\leq$.

The similarity between two graphs increases as the number of similar edges increase. Therefore in this view, the feature vector of each graph is found out by finding the number of times the elements of $\Sigma^2$ appear as edges of the graph. That is,

$$x_{view_I}(G) = (c^1, c^2, \ldots c^{|\Sigma^2|})$$

where $c^i$ is the number of times $i^{th}$ element of $\Sigma^2$ appear as edges in $G$ and $|\Sigma^2|$ is the cardinality of $\Sigma^2$. Therefore the input space for view I is $\mathbb{R}^{|\Sigma^2|}$. An example is shown in Figure 3.2.

| Edge | Labeled representation |
|---|---|
| {1, 2} | {C C} |
| {4, 1} | {B C} |
| {3, 2} | {A C} |
| {4, 2} | {B C} |
| {3, 4} | {A B} |
| {3, 5} | {A A} |



$\Sigma = \{\mathbf{A}, \mathbf{B}, \mathbf{C}\}$

$\Sigma^2 = \{\mathbf{AA}, \mathbf{AB}, \mathbf{AC}, \mathbf{BB}, \mathbf{BC}, \mathbf{CC}\}$

$\mathbf{x}_{\mathbf{view_I}}(\mathbf{G}) = (\mathbf{1}, \mathbf{1}, \mathbf{1}, \mathbf{0}, \mathbf{2}, \mathbf{1})$

**Figure 3.2:** View-I formulation: Feature representation of a sample graph (note that node labels are given inside circles and the associated number indicates node identification). The table shows the node label associated with each edges.

In this view only the immediate neighbours are considered. It is required to have the information of the entire neighbourhood. Hence we constructed two more views that deals with the neighbourhood information.

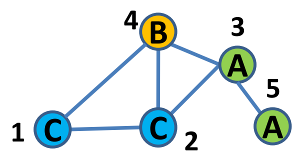
### 3.3.2 View-II:Embedding based on shortest path length information

This view extracts the density of the neighborhood of each vertex by looking into the length of the shortest paths between vertices. That is, it looks into how fast one can reach from one vertex to another. The works like [7] used shortest path in constructing kernels for unlabeled graph processing.

In this view, two graphs become similar when they are similar in terms of the reachability between the nodes. Therefore each graph is represented as a vector

$$x_{view_{II}}(G) = (c^0, c^1, \ldots c^{|p_{max}|})$$

where $c^i, i \geq 0$ is the number of occurrence of shortest paths of length $i$ in $G$, $p_{max}$ is the maximum length of the shortest path we considered and $c^0$ is the number of zero length paths or equivalently number of vertices in the graph. Thus the input space $\mathcal{X}_2$ for this view is $\mathbb{R}^{|p_{max}|}$. An example is shown in the Figure 3.3.



$\Sigma = \{A, B, C\}$

| Shortest paths | Path Length |
|---|---|
| Path 1-2: {1, 2} | 1 |
| Path 1-3: {1, 2, 3} | 2 |
| Path 1-4: {1, 4} | 1 |
| Path 1-5: {1, 4, 3, 5} | 3 |
| Path 2-3: {2, 3} | 1 |
| Path 2-4: {2, 4} | 1 |
| Path 2-5: {2, 3, 5} | 2 |
| Path 3-4: {3, 4} | 1 |
| Path 3-5: {3, 5} | 1 |
| Path 4-5: [4, 3, 5} | 2 |

**Feature representation**

| Attributes following total order | Value |
|---|---|
| Number of nodes | 5 |
| No: of occurrence of Path Length: 1 | 6 |
| No: of occurrence of Path Length: 2 | 3 |
| No: of occurrence of Path Length: 3 | 1 |

$\mathbf{x_{view_{II}}(G) = (c^0, c^1, c^2, c^3) = (5, 6, 3, 1)}$

**Figure 3.3:** View-II formulation: Feature representation of a sample graph. The right table shows the length associated with each shortest paths and left table shows the derived feature representation.

### 3.3.3 View-III: Embedding based on labeled shortest path information

The information of the labels of the nearest neighbors of each node is used to construct this view. For that, the labels appearing in the shortest path between each vertex are considered. The construction of the feature corresponding to view-III is done as given in [96].

A vertex $v_i$ is taken as a nearest neighbour of $v_j$ if the length of the shortest path $v_i$ and $v_j$ is less than or equal to a predefined number $p_{max}$. Let $\mathcal{N}(v_i) = \{v_j \in G : v_j \text{ is a neighbour of } v_i \text{ within a shortest path of finite length}\}$. In this view, we are only interested in which all labels appearing in the shortest path that connects $v_i$ with that of its neighbours and not in their order of appearance in the path. Hence the labeled shortest path is constructed as: $L^G_{\Pi}(v_i, v_j) = [l_1, l_2, \dots l_n], n \leq p_{max}, v_j \in \mathcal{N}(v_i)$. The end points of $L^G_{\Pi}(v_i, v_j)$ are the labels of its arguments and $l_k, k = 2, \dots n-1$ are the labels appearing in the shortest path between $v_i$ and $v_j$ such that

1. $l_1 \leq l_n$
2. $l_s \leq l_t$ , $1 < s < t < n$

As the information related with the shortest path between $v_i$ and $v_j$ is same as that of between $v_j$ and $v_i$, the constraint $l_1 \leq l_n$ has been imposed on $L^G_{\Pi}(v_i, v_j)$.

Consider the class of sets

$$P = \cup_{l=1}^{N} \{L_{\Pi}^{G_l}(v_i, v_j) : v_i \in G_l, v_j \in \mathcal{N}(v_i)\} \tag{3.3}$$

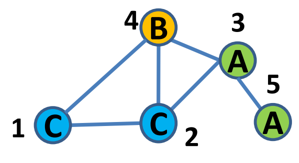
Now

$$x_{view_{III}}(G) = \{c^1, c^2, \ldots .c^{|P|}\}$$

where $c^l = |(v_i, v_j) \in G : L_{\Pi}^{G}(v_i, v_j) \equiv P^l|$, $P^l$ is the $l^{th}$ element of $P$ and $|P|$ is the cardinality of $P$. That is, $c^l$ is the number of vertex pairs in $G$ for which their corresponding labelled shortest path is equivalent to the $l^{th}$ element of $P$. Hence the input space for View-III is $\mathbb{R}^{|P|}$.

An example for the feature extraction process is shown in the Figure 3.4. It should also be noted that the length of the shortest paths so chosen are limited within a tolerance level. The tolerance level was fixed by cross validation.

$$\Sigma = \{A, B, C\}$$



(a)

| Shortest paths | Associated node labels | $\mathbf{L}_{\Pi}^{\mathbf{G}}$ representation |
|---|---|---|
| Path 1-2: {1, 2} | {C C} | {C C} |
| Path 1-3: {1, 2, 3} | {C C A} | {A C C} |
| Path 1-4: {1, 4} | {C B} | {B C} |
| Path 1-5: {1, 4, 3, 5} | {C B A A} | {A A B C} |
| Path 2-3: {2, 3} | {C A} | {A C} |
| Path 2-4: {2, 4} | {C B} | {B C} |
| Path 2-5: {2,3,5} | {C A A} | {A A C} |
| Path 3-4: {3, 4} | {A B} | {A B} |
| Path 3-5: {3, 5} | {A A} | {A A} |
| Path 4-5: [4, 3, 5} | {B A A} | {A A B} |

(b)

$$\mathbf{P} = \{\mathbf{AA}, \mathbf{AB}, \mathbf{AC}, \mathbf{BC}, \mathbf{CC}, \mathbf{AAB}, \mathbf{AAC}, \mathbf{ACC}, \mathbf{AABC}\}$$
$$\mathbf{x}_{\mathbf{view_{III}}}(\mathbf{G}) = (\mathbf{1}, \mathbf{1}, \mathbf{1}, \mathbf{2}, \mathbf{1}, \mathbf{1}, \mathbf{1}, \mathbf{1}, \mathbf{1})$$

(c)

**Figure 3.4:** View-III formulation. (a) A sample graph (note that node labels are given inside circles). (b) Labeled shortest path representations, $L_{\Pi}^{G}$. (c) Corresponding set $P$ of the graph and its vector representation.

### 3.3.3.1 Note on computation of set $P$

Algorithm 3.1 can be used for computation of $P$. It can be seen that the shortest paths between each pair of nodes has to be calculated. In practical computation, set $P$ can be made as a dictionary data type. Every $L_{\Pi}^{G_i}(v_j, v_k)$ representation are stored in $P$ along with an associated *key*. The embedding vectors can be efficiently computed using these keys as shown in the algorithm.

**Algorithm 3.1:** Computation of view-III embedding

```
Input  : The graph dataset $D$
Output: The view-III vector embedding
1  Initialize $P(key, value)$ as empty dictionary and $p = 1$
2  for every graph, $G_i = (V_i, E_i)$ in $\mathcal{G}$ do
3      Initialize $x_{view_{III}}(G_i)$ as zero vector
4      for every node $v_j$ in $V_i$ do
5          for every node $v_k$ in $V_i$ do
6              if $j<k$ then
7                  Find shortest path between $v_j$ and $v_k$
8                  Find $L_{\Pi}^{G_i}(v_j, v_k)$ representation
9                  if $L_{\Pi}^{G_i}(v_j, v_k) \notin P$ then
10                     $P = P \cup \{L_{\Pi}^{G_i}(v_j, v_k)\}$
11                     $key(L_{\Pi}^{G_i}(v_j, v_k)) = p$
12                     $x_{view_{III}}(G_i)(p) = x_{view_{III}}(G_i)(p) + 1$
13                     $p = p + 1$
14                 else
15                     $x_{view_{III}}(G_i)(key(L_{\Pi}^{G_i}(v_j, v_k))) =$
                         $x_{view_{III}}(G_i)(key(L_{\Pi}^{G_i}(v_j, v_k))) + 1$
16                 end
17             end
18         end
19     end
20     Store $x_{view_{III}}(G_i)$
21 end
```

The View-III basically contains information in the form of chains running across the graphs. But these chains could intersect in multiple nodes forming much more complex graph structures. Hence we need to process these complex structures and for this purpose View-IV is introduced.

### 3.3.4 View-IV: Embedding based on subtree patterns

The view-IV is designed as a set of features that could derive the structure of the graph in a global perspective. This is achieved by considering the subtrees originating from each vertex.

The subtree associated with a vertex $v$ of a graph $G$ are hierarchical structures in the form of a tree, whose root node consists of $v$ and the immediate children of each node in the tree are those vertices which are adjacent to it. For subtree generation, the set of nodes that are encountered at each level of the walks of depth $k, (k \geq 0)$ from the root node $v$ is represented as $S_k^G(v)$, where,

$$S_k^G(v) = \begin{cases} \Big\{v\Big\},\ k = 0 \\ \Big\{v_i : (v, v_i) \in E\Big\},\ k = 1 \\ \Big\{v_j : (v_i, v_j) \in E \wedge v_i \in S_{k-1}(v)\Big\},\ k > 1 \end{cases}$$

The View-IV is designed using the adjacency information in the form of labels appearing in each level of the sub-tree. For that a function $\mathcal{L}$ over the domain consisting of such $S_k^G(v)$ is formulated, where

$$\mathcal{L}\big(S_k^G(v)\big) = \begin{cases} l(v), k = 0 \\ \Big\{|\epsilon_1||\epsilon_2|\ldots|\epsilon_{|\Sigma|}|\Big\},\ k > 0 \end{cases}$$

where $\epsilon_i$ is the $i^{th}$ element in $\Sigma$, $|\epsilon_i|$ gives the number of vertices in $S_k(v)$ that has the label $\epsilon_i$ and $|\Sigma|$ is the cardinality of $\Sigma$.

Now corresponding to each $v \in V$ and each depth $d$ under consideration, where $1 \leq d \leq d_{max}$, a feature $F_G(v, d)$ is constructed. Here, $F_G(v, d)$ is a string formed by the concatenation of strings $\mathcal{L}(S_k^G(v)), k = 0, 1, \ldots, d$ and $d_{max}$ is the maximum depth of the subtree considered for the design. The $F_G(v, d)$ has to be found for all nodes at all depths under consideration and hence a total of $|V| \times d_{max}$ number of features is obtained for each graph.

Consider the class of sets

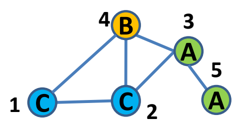
$$T = \cup_{l=1}^{N}\{F_{G_l}(v, d) : v \in V_l, 1 \leq d \leq d_{max}\} \tag{3.4}$$

Note that $T$ contains information about whole subtree pattern upto depth $d$ rooted in vertices of the all graphs.

Now

$$x_{view_{IV}}(G) = \{c^1, c^2, \ldots .c^{|T|}\}$$

where $c^l = |(v, d), v \in G, 1 \leq d \leq d_{max} : F_G(v, d) \equiv T^l|$, $T^l$ is the $l^{th}$ element of $T$ and $|T|$ is the cardinality of $T$. That is, $c^l$ is the number of vertices in $G$ for which its corresponding subtree pattern representation is equivalent to the $l^{th}$ element of $T$. Hence the input space for View-IV is $\mathbb{R}^{|T|}$. An example for the feature extraction scheme is given in the Figure 3.5 where subtree patterns up to depth, $d = 3$ is computed.



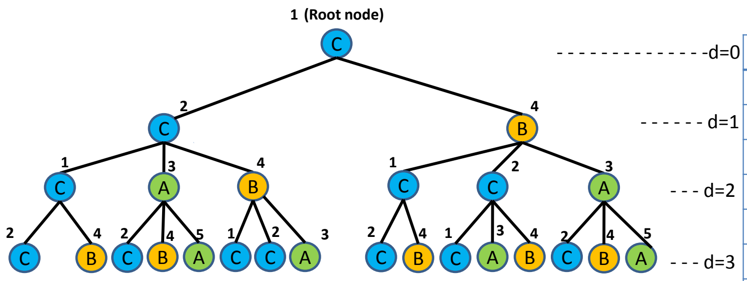
(a)

$$\mathbf{T} = \{\mathbf{C\,0\,1\,1}, \mathbf{C\,1\,1\,1}, \mathbf{A\,1\,1\,1}, \mathbf{B\,1\,1\,1}, \mathbf{A\,1\,0\,0}, \mathbf{C\,0\,1\,1\,2\,1\,3}, \mathbf{C\,1\,1\,1\,2\,2\,4}, \mathbf{C\,1\,1\,1\,3\,1\,3}, \mathbf{B\,1\,0\,2\,2\,3\,3}, \mathbf{A\,1\,0\,0\,1\,1\,1}, \mathbf{C\,0\,1\,1\,2\,1\,3\,4\,5\,7}, \mathbf{C\,1\,1\,1\,2\,2\,4\,7\,5\,9}, \mathbf{C\,1\,1\,1\,3\,1\,3\,5\,6\,8}, \mathbf{B\,1\,0\,2\,2\,3\,3\,7\,4\,10}, \mathbf{A\,1\,0\,0\,1\,1\,1\,3\,1\,3}\}$$

$$\mathbf{x}_{\mathbf{view}_{\mathrm{IV}}}(\mathbf{G}) = (\mathbf{1},\mathbf{1},\mathbf{1},\mathbf{1},\mathbf{1},\mathbf{1},\mathbf{1},\mathbf{1},\mathbf{1},\mathbf{1},\mathbf{1},\mathbf{1},\mathbf{1},\mathbf{1},\mathbf{1})$$

(b)

**Construction details of the vector representation**



| | |
|---|---|
| $\mathbf{S_0^G(v)}$ | {1} |
| $\mathbf{E(S_0^G(v))} = \mathscr{L}(\mathbf{v})$ | {C} |
| $\mathbf{S_1^G(v)}$ | {2,4} |
| $\mathbf{E(S_1^G(v))}$ | {0 1 1} |
| $\mathbf{S_2^G(v)}$ | {1,3,4,1,2,3} |
| $\mathbf{E(S_2^G(v))}$ | {2 1 3} |
| $\mathbf{S_3^G(v)}$ | {2,4,2,4,5,1,2,3,2,4,1,3,4,2,4,5} |
| $\mathbf{E(S_3^G(v))}$ | {4 5 7} |

**Feature representations for subtrees rooted at node: 1**

| Subtree pattern of depth k=1 | | Subtree pattern of depth k=2 | | Subtree pattern of depth k=3 | |
|---|---|---|---|---|---|
| $\mathbf{F_G(v,1)}$ | {C}+{0 1 1} ≡ **{C 0 1 1}** | $\mathbf{F_G(v,2)}$ | {C}+{0 1 1}+{2 1 3} ≡ **{C 0 1 1 2 1 3}** | $\mathbf{F_G(v,3)}$ | {C}+{0 1 1} +{2 1 3}+{4 5 7} ≡ **{C 0 1 1 2 1 3 4 5 7}** |

(c)

**Feature representations for subtrees rooted at node: 2**

| Subtree pattern of depth k=1 | | Subtree pattern of depth k=2 | | Subtree pattern of depth k=3 | |
|---|---|---|---|---|---|
| $\mathbf{F_G(v,1)}$ | {C}+{1 1 1} ≡ **{C 1 1 1}** | $\mathbf{F_G(v,2)}$ | {C}+{1 1 1}+{2 2 4} ≡ **{C 1 1 1 2 2 4}** | $\mathbf{F_G(v,3)}$ | {C}+{1 1 1} +{2 2 4}+{7 5 9} ≡ **{C 1 1 1 2 2 4 7 5 9}** |

(d)

**Feature representations for subtrees rooted at node: 3**

| Subtree pattern of depth k=1 | | Subtree pattern of depth k=2 | | Subtree pattern of depth k=3 | |
|---|---|---|---|---|---|
| $\mathbf{F_G(v,1)}$ | {A}+{1 1 1} ≡ **{A 1 1 1}** | $\mathbf{F_G(v,2)}$ | {A}+{1 1 1}+{3 1 3} ≡ **{C 1 1 1 3 1 3}** | $\mathbf{F_G(v,3)}$ | {A}+{1 1 1} +{3 1 3}+{5 6 8} ≡ **{C 1 1 1 3 1 3 5 6 8}** |

(e)

**Feature representations for subtrees rooted at node: 4**

| Subtree pattern of depth k=1 | | Subtree pattern of depth k=2 | | Subtree pattern of depth k=3 | |
|---|---|---|---|---|---|
| $\mathbf{F_G(v,1)}$ | {B}+{1 0 2} ≡ **{B 1 1 1}** | $\mathbf{F_G(v,2)}$ | {B}+{1 0 2}+{2 3 3} ≡ **{B 1 0 2 2 3 3}** | $\mathbf{F_G(v,3)}$ | {B}+{1 0 2}+{2 3 3} +{7 4 10}≡**{B 1 0 2 2 3 3 7 4 10}** |

(f)

**Feature representations for subtrees rooted at node: 5**

| Subtree pattern of depth k=1 | | Subtree pattern of depth k=2 | | Subtree pattern of depth k=3 | |
|---|---|---|---|---|---|
| $\mathbf{F_G(v,1)}$ | {A}+{1 0 0} ≡ **{A 1 0 0}** | $\mathbf{F_G(v,2)}$ | {A}+{1 0 0}+{1 1 1} ≡ **{A 1 0 0 1 1 1}** | $\mathbf{F_G(v,3)}$ | {A}+{1 0 0}+{1 1 1} +{3 1 3}≡ **{A 1 0 0 1 1 1 3 1 3}** |

(g)

**Figure 3.5:** View IV Formulation. (a) A sample graph. (b) Corresponding set $T$ and vector representation of the graph. (c) Feature construction corresponding to subtrees rooted at node 1, (d) Feature representations corresponding to subtrees rooted at node 2. (e) Feature representations corresponding to subtrees rooted at node 3. (f) Feature representations corresponding to subtrees rooted at node 4. (g) Feature representations corresponding to subtrees rooted at node 5.

#### 3.3.4.1 Note on computation of set $T$

Algorithm 3.2 can be used for computation of $T$. It can be seen that a tree traversal rooted on every nodes are required for the computations till walks of step 3. In practical computation, set $T$ can be made as a dictionary data type. The $F_G(v, d)$ representation can be efficiently calculated at each depth $d$ and can be integrated with dictionary $T$ as *value* along with an associated *key*. The vectors then can be found out in the same way as explained in the case of algorithm 3.1.

**Algorithm 3.2:** Computation of view-IV embedding

```
Input  : The graph dataset $D$
Output: The view-IV vector embedding
1  Initialize $T(key, value)$ as empty dictionary and $p = 1$
2  for every graph, $G_i = (V_i, E_i)$ in $\mathcal{G}$ do
3      Initialize $x_{view_{IV}}(G_i)$ as zero vector
4      for every node $v_j$ in $V_i$ do
5          $\mathcal{L} = l(v)$
6          for depth,k from 1 to 3 do
7              Find $S_k^{G_i}(v_j)$ or nodes reachable through $k$ step walk from $v_j$
8              $F = \mathcal{L} \odot \mathcal{L}\big(S_k^G(v)\big)$, $\odot$ represents string concatenation
9              if $F \notin T$ then
10                 $T = T \cup \{F\}$
11                 $key(F) = p$
12                 $x_{view_{IV}}(G_i)(p) = x_{view_{IV}}(G_i)(p) + 1$
13                 $p = p + 1$
14                 $\mathcal{L} = F$
15             else
16                 $x_{view_{IV}}(G_i)(key(F)) = x_{view_{IV}}(G_i)(key(F)) + 1$
17                 $\mathcal{L} = F$
18             end
19         end
20     end
21     Store $x_{view_{IV}}(G_i)$
22 end
```

## 3.4 Representation Capability of the Views

The views defined in the above section can be regarded as different representations of the graph. The prediction capacity of each view was analysed using kernel algorithm and for that we designed graph kernels using the concept of R-convolution framework[2] whose descriptions are given below.

### 3.4.1 Kernel definition for view: I

In view-I, the similarity between two graphs increases when the number of similar edge increases and hence we designed kernel in the following way.

Let $\Sigma^2_{G_i}$ and $\Sigma^2_{G_j}$ be the class of view-I feature sets corresponding to $G_i$ and $G_j$ respectively. Then

$$K_{view_I}(G_i, G_j) = \sum_{e_u \in \Sigma^2_{G_i}} \sum_{e_v \in \Sigma^2_{G_j}} \delta(e_u, e_v)$$

Because of the $\delta$ function the value of the kernel increases when the number of common edges in $G_i$ and $G_j$ increase.

The proof of the positive definiteness of $K_{view_I}$ can be established once we prove it is an R-convolution kernel.

For each element $e_u \in \Sigma^2_{G_i}$, we define a relation $R(e_u, G_i^\dagger, G_i)$ whose value equals 1 iff $G_i^\dagger$ is the graph obtained from $G_i$ by removing the edge $e_u$. Let $R^{-1}(G_i)$ be the set containing all the decompositions of the graph $G_i$ into $e_u$ and $G^\dagger$.

With this setting, R-convolution kernel $K_{edge}$ can be defined as follows.

$$\begin{aligned} K_{edge}(G_i, G_j) &= \sum_{R^{-1}(G_i)} \sum_{R^{-1}(G_j)} \delta(e_u, e_v) k_{trivial}(G_i^\dagger, G_j^\dagger) \\ &= \sum_{e_u \in \Sigma^2_{G_i}} \sum_{e_v \in \Sigma^2_{G_j}} \delta(e_u, e_v) = K_{view_I}(G_i, G_j) \end{aligned}$$

where $k_{trivial}$ is a trivial kernel whose value is always 1.

It is noteworthy that $K_{edge}(G_i, G_j) = \langle x_{view_I}(G_i), x_{view_I}(G_j) \rangle$. The reason can be stated as: $K_{edge}$ actually counts the common structures generated using the decompositions specified by $R$. The vector representation $x_{view_I}$ of graphs $G_i, G_j$ contains the information of number of decomposed structures specified by the decomposition relation $R$ and taking their inner product is a measure of common structures as in the case of $K_{edge}$.

### 3.4.2 Kernel definition for view: II

The kernel for view-II is defined as follows:

$$\begin{aligned} K_{view_{II}}(G_i, G_j) &= \sum_{u_i, v_i \in V(G_i)} \sum_{u_j, v_j \in V(G_j)} \delta\big(\Pi^{G_i}(u_i, v_i), \Pi^{G_j}(u_j, v_j)\big) \\ &= \langle x_{view_{II}}(G_i), x_{view_{II}}(G_j) \rangle \end{aligned}$$

where $\delta\big(\Pi^{G_i}(u_i, v_i), \Pi^{G_j}(u_j, v_j)\big) = 1$ iff $\Pi^{G_i}(u_i, v_i)$ and $\Pi^{G_j}(u_j, v_j)$ are of same length. Note that this kernel definition is same as that of the shortest path kernel [7].

It is straight forward to define this kernel on the basis of R-convolution kernel as explained in the kernel definition for the View-I. In this case, corresponding to each shortest path a decomposition relation $R(p, G^{\dagger}, G)$ can be made where $G^{\dagger}$ is the graph obtained from $G$ by removing the edges involved in the shortest path $p$.

### 3.4.3 Kernel definition for view: III

The view-III kernel is defined as

$$\begin{aligned} K_{view_{III}}(G_i, G_j) &= \sum_{u_i, v_i \in V(G_i)} \sum_{u_j, v_j \in V(G_j)} \delta(L_{\Pi}^{G_i}(u_i, v_i), L_{\Pi}^{G_j}(u_j, v_j)) \\ &= \langle x_{view_{III}}(G_i), x_{view_{III}}(G_j) \rangle \end{aligned}$$

The above defined kernel is in fact an R convolution kernel. For each summation case, we can define a relation $R(p_L, G^{\dagger}, G)$ such that shortest path representation of the path $p_L$ is $L_{\Pi}(., .)$ and $G^{\dagger}$ is the graph obtained from $G$ by removing edges involved in the path $p_L$.

### 3.4.4 Kernel definition for view: IV

The kernel for view IV is defined as follows:

$$K_{view_{IV}}(G_i, G_j) = \sum_{d=1}^{d_{max}} \sum_{v \in V_i} \sum_{u \in V_j} \delta\big(F_d^{G_i}(v), F_d^{G_j}(u)\big)$$

The View-IV kernel can be explained in R-convolution framework. Let $\mathcal{T}$ be the set containing whole trees with labeled nodes in all graphs. Now consider a graph $G$. For any $t_d \in \mathcal{T}$ we define a relation $R(t^d, G) = 1$ iff $t^d$ is a subtree in $G$ of depth $d, d = 1, \ldots d_{max}$.

With this assumptions, $R^{-1}(G)$ contains the set of all subtrees in $G$. Now we can define a R-convolution kernel called subtree kernel($K_{ST}$) as follows

$$\begin{aligned}
K_{ST}(G_i, G_j) &= \sum_{d=1}^{d_{max}} \sum_{t_i^d \in R^{-1}(G_i)} \sum_{t_j^d \in R^{-1}(G_j)} \delta\big(F_d^{G_i}(v), F_d^{G_j}(u)\big) : \forall v \in V_i \ \wedge \ \forall u \in V_j \\
&= K_{view_{IV}}(G_i, G_j) \\
&= \langle x_{view_{IV}}(G_i), x_{view_{IV}}(G_j)\rangle
\end{aligned}$$

where $F_d^{G_i}(v), F_d^{G_j}(u)$ are the string representations of $t_i^d$ rooted on node $v$ and $t_j^d$ rooted on node $u$ respectively and $\delta\big(F_d^{G_i}(v), F_d^{G_j}(u)\big) = 1$ iff string representations of subtree patterns of depth $d$ rooted on node $v$ in graph $G_i$ and node $u$ in graph $G_j$ is same. Otherwise it is 0.

## 3.5 Experiment

The experiments were done in real world datasets in the domains of chemo-informatics, brain connectivity analysis and social media data analysis.

### 3.5.1 Experimental Setup

The classification algorithm used was SVM (with Libsvm implementation [97]). The performance parameter used was accuracy. The experiments were done with Intel Xeon i5 CPU with 80 GB RAM.

The kernel for each view and other hyperparameters of the model were selected using cross validation techniques, whose procedure was as follows. Using the hold out technique 70% of the data points were assigned for training and the remaining for testing. 10 fold cross-validation was done on training data for selecting the hyperparameters. A model was then built using the entire training data and its performance was tested on the testing data. The above process was repeated 30 times and the results reported were averaged over these 30 iterations to nullify the effects of fold assignments in the hold-out split.

The hyperparameter associated with Gaussian kernel was searched in the interval ($2^{-15}$, $2^{3}$), degree of the polynomial kernels in the set $\{1, 2, 3, 4\}$ and the penalty parameter $C$ of SVM in the interval ($2^{-5}$, $2^{15}$). The $h$ value associated with WL kernel framework was searched in the set {0,1,2,..., 10} for WL-subtree, {0,1,2,3} for WL-edge and {0,1,2} for WL-shortest path kernels.

The models were subjected to paired t-test to check the statistical significance of the results (significance level $\alpha = 0.05$). The ranks to the algorithms were assigned in the following way. Let $P_1$ and $P_2$ are the values of a performance measure $P$ for a given dataset corresponding to algorithms $A_1$ and $A_2$ respectively. Then we say that $A_1$ is better than $A_2$ on the basis of $P$ if $P_1 > P_2$ and the difference between $P_1$ and $P_2$ is statistically significant.

### 3.5.2 Applications in chemoinformatics

The efficiency of the proposed multi-view embedding was analyzed by subjecting them on real world data sets on chemo-informatics domain and compared its performance with state-of-the-art graph embeddings namely: Graph2vec (G2V) [20], Subgraph2vec (S2V) [21], Node2vec (N2V) [22], GE-FSG (Graph Embedding with Frequent Sub-Graphs) [23], Deep graph kernels (DGK) [24] and graph kernels namely: Random walk (RW) kernel [30], Shortest path (SP) kernel [7] (labelled and unlabelled version), Graphlet kernel (GL) [43], Weisfeiler-Lehman (WL) kernels [25] (WL-subtree (WL-S), WL-edge (WL-E) and WL-shortest path (WL-SP), WL-optimal assignment kernel (WL-OA) [5], RetGK kernels (the variants with explicit RKHS mapping (RGK-1) and with approximated mapping (RGK-2)) [35], and Treelet kernel+MKL (TK+MKL) [54]. The results of multi-view approach are abbreviated by MV. Node2vec gives the embedding of individual nodes and hence in our analysis, the vector corresponding to a graph's embedding was taken as the average of the vectors corresponding to its node embeddings. We used deep graph kernel as an embedding for our analysis because it constructs a matrix based on the concepts of word2vec [17] that can be regarded as an embedding matrix, $M$. The feature map of state-of-the-art graph kernels is then utilized to construct an augmented kernel measure of the form $k(x, y) = \phi(x)^T M \phi(y)$ where, $\phi$ is the feature mapping.

#### 3.5.2.1 Datasets

The classification datasets used for analysis were from the chemoinformatics domain. The datasets used were MUTAG, PTC(MR), PTC(FR), PTC(MM), PTC(FM), PROTEINS, ENZYMES, NCI1, and, NCI109. The MUTAG [98] is a dataset of 188 mutagenic aromatic and heteroaromatic nitro compounds that are labeled according to whether or not they have a mutagenic effect on the Gramnegative bacterium Salmonella typhimurium. The PTC-FR, PTC-MR, PTC-FM and PTC-MM [99] describe the carcinogenicity of certain chemical compounds on female rats, male rats, female mice, male mice respectively. The EN-

**Table 3.1:** Dataset details: $|D|$: cardinality of the data set $D$, $|P|$: cardinality of positive class, $|N|$: cardinality negative class, $V^{avg}$ =: average of $\{|V|, V \in D\}$, $E^{avg}$ = average of $\{|E|, E \in D\}$, $V^{\max} = \max\{|V|, V \in D\}$, $E^{\max} = \max\{|E|, E \in D\}$.

| **Dataset** | $\lvert D\rvert$ | $\lvert P\rvert$ | $\lvert N\rvert$ | $V^{avg}$ | $E^{avg}$ | $V^{\max}$ | $E^{\max}$ |
|---|---|---|---|---|---|---|---|
| **MUTAG** | 188 | 125 | 63 | 17.93 | 19.79 | 28 | 33 |
| **PTC-MR** | 344 | 152 | 192 | 25.55 | 25.96 | 109 | 108 |
| **PTC-FR** | 351 | 121 | 230 | 27.08 | 26.47 | 109 | 104 |
| **PTC-FM** | 349 | 143 | 206 | 26.24 | 25.57 | 109 | 104 |
| **PTC-MM** | 336 | 129 | 207 | 26.03 | 25.34 | 109 | 104 |
| **PROTEINS** | 1113 | 663 | 450 | 39.05 | 72.81 | 620 | 1049 |
| **ENZYMES** | 600 | NA | NA | 32.63 | 46.66 | 126 | 81 |
| **NCI1** | 4110 | 2057 | 2053 | 29.86 | 32.30 | 111 | 119 |
| **NCI109** | 4127 | 2079 | 2048 | 29.68 | 32.13 | 111 | 119 |

ZYMES is a data set of 600 protein tertiary structures obtained from the BRENDA enzyme database [100]. The PROTEINS dataset consists of chemical compounds with two classes (enzyme and non-enzyme) [101]. The NCI1 and NCI109 [102] are two datasets of chemical compounds that are screened for activity against non-small cell lung cancer and ovarian cancer cell lines respectively. The ENZYMES is a 6 class classification problem whereas all the other are binary class problems. The further details of the datasets are given in Table 3.1.

#### 3.5.2.2 Results and Discussion

The accuracy results of different models we used in our analysis are tabulated in Table 3.2. The multi-view approach secured rank 1 in all datasets. This shows that relative with other graph embeddings and graph kernels that rely on a single class of graph property, multi view approach succeeded to capture the characteristics of the data in a better way. In the PTC datasets, the multi-view approach has a significantly better performance compared to other models - an improvement of 5%, 4%, 2% and 3% respectively. When it comes to the large datasets - NCI1 and NCI109 - the improvement is approximately 1.3% and 2% respectively compared to the second best performing model.

Among the embedding methods, G2V, S2V and DGK has shown consistent performance. The performance of N2V is low and it can be attributed to the node level averaging. In PCI datasets, G2V has the better performance while in MUTAG it is S2V. Note that the corresponding scores are on par with the best scores from the kernel algorithms. In large datasets - PROTEINS, NCI1, NCI109 - it is DGK that has better performance and GE-FSG that performs lower in small datasets are second best. In the case of ENZYMES, it is G2V that has better performance than others. Comparing the performance with ker-

**Table 3.2:** Accuracy (along with standard deviation) of the multi-view graph embedding along with state-of-the-art embedding techniques and graph kernels. The rank in statistical significance test is given in the bracket.

| **Embed.** | **MUTAG** | **PTC-MR** | **PTC-FR** | **PTC-FM** | **PTC-MM** |
|---|---|---|---|---|---|
| G2V | 84.26 ± 10.25 (5) | 61.23 ± 5.67 (2) | 67.23 ± 4.77 (2) | 61.23 ± 5.05 (3) | 66.58 ± 5.65 (2) |
| S2V | 88.23 ± 1.89 (2) | 60.76 ± 1.49 (3) | 65.44 ± 1.78 (4) | 59.77 ± 2.12 (4) | 63.79 ± 1.83 (4) |
| N2V | 73.42 ± 8.75 (9) | 57.45 ± 8.94 (7) | 61.56 ± 9.44 (6) | 56.82 ± 9.74 (7) | 60.66 ± 8.35 (7) |
| GE-FSG | 83.54 ± 0.03 (5) | 58.94 ± 0.82 (5) | 60.17 ± 0.71 (6) | 56.29 ± 1.13 (6) | 59.83 ± 0.86 (7) |
| DGK | 88.46 ± 2.87 (2) | 60.15 ± 3.08 (4) | 66.36 ± 3.64 (3) | 60.54 ± 4.88 (4) | 65.44 ± 5.14 (3) |
| **Kernel** | **MUTAG** | **PTC-MR** | **PTC-FR** | **PTC-FM** | **PTC-MM** |
| RW | 74.27 ± 3.79 (8) | 58.19 ± 3.63 (5) | 65.56 ± 0.80 (4) | 58.83 ± 2.48 (5) | 62.50 ± 0.76 (4) |
| SP [2] | 87.84 ± 5.66 (3) | 59.65 ± 6.02 (5) | 65.66 ± 4.77 (4) | 63.13 ± 5.12 (2) | 62.88 ± 5.78 (5) |
| GL | 74.16 ± 3.60 (8) | 58.25 ± 3.68 (5) | 66.91 ± 2.44 (3) | 56.70 ± 3.96 (6) | 64.78 ± 3.50 (3) |
| WL-S | 85.04 ± 4.70 (4) | 58.95 ± 3.99 (5) | 67.95 ± 2.74 (1) | 62.41 ± 4.73 (2) | 67.87 ± 3.85 (1) |
| WL-E | 83.70 ± 3.99 (5) | 57.61 ± 4.23 (6) | 67.44 ± 2.05 (2) | 60.06 ± 4.98 (4) | 66.00 ± 2.84 (2) |
| WL-SP | 85.82 ± 5.19 (4) | 58.40 ± 4.78 (5) | 62.86 ± 2.93 (5) | 54.84 ± 4.49 (8) | 60.56 ± 4.21 (6) |
| WL-OA | 87.27 ± 4.46 (3) | 60.59 ± 3.66 (3) | 66.62 ± 2.88 (2) | 59.30 ± 3.91 (4) | 67.15 ± 3.47 (1) |
| RGK-1 | 65.80 ± 3.52 (10) | 58.43 ± 3.63 (5) | 65.22 ± 1.66 (4) | 58.44 ± 2.53 (5) | 60.71 ± 2.55 (6) |
| RGK-2 | 65.51 ± 3.91 (10) | 56.25 ± 2.23 (7) | 66.29 ± 3.08 (3) | 58.77 ± 1.67 (5) | 61.78 ± 0.42 (5) |
| TK+MKL | 81.61 ± 5.61 (7) | 51.12 ± 6.48 (8) | 59.45 ± 4.38 (6) | 56.66 ± 5.22 (6) | 58.33 ± 5.69 (8) |
| MV | **90.24 ± 4.85** (1) | **66.96 ± 4.32** (1) | **72.44 ± 2.78** (1) | **65.32 ± 4.03** (1) | **69.33 ± 3.56** (1) |

| **Embed.** | **PROTEINS** | **ENZYMES** | **NCI1** | **NCI109** | **Avg. Rank** |
|---|---|---|---|---|---|
| G2V | 73.48 ± 1.83 (3) | 36.67 ± 7.05 (6) | 74.05 ± 1.53 (8) | 74.87 ± 1.49 (8) | 4.33 |
| S2V | 74.17 ± 1.82 (2) | 32.55 ± 3.86 (8) | 78.12 ± 1.85 (7) | 78.67 ± 1.64 (7) | 4.55 |
| N2V | 56.82 ± 4.05 (8) | 26.44 ± 9.89 (11) | 55.64 ± 2.06 (13) | 52.37 ± 1.62 (13) | 9.00 |
| GE-FSG | 73.85 ± 0.03 (2) | 28.44 ± 2.68 (9) | 78.61 ± 0.02 (7) | 78.15 ± 0.02 (7) | 6.00 |
| DGK | 74.98 ± 0.82 (2) | 34.62 ± 5.02 (7) | 81.35 ± 0.94 (6) | 80.49 ± 0.92 (6) | 4.11 |
| **Kernel** | **PROTEINS** | **ENZYMES** | **NCI1** | **NCI109** | **Avg. Rank** |
| RW | 59.32 ± 0.38 (7) | 20.40 ± 2.85 (12) | 50.12 ± 0.13 (14) | 50.39 ± 0.06 (12) | 7.88 |
| SP[2] | 73.83 ± 2.93 (3) | 34.77 ± 3.22 (7) | 67.15 ± 1.63 (10) | 70.63 ± 1.80 (9) | 5.33 |
| GL | 71.52 ± 1.76 (4) | 32.06 ± 3.22 (8) | 63.42 ± 1.01 (11) | 63.69 ± 1.42 (10) | 6.44 |
| WL-S | 74.40 ± 2.05 (2) | 49.85 ± 4.15 (4) | 84.61 ± 1.03 (3) | 84.17 ± 1.31 (2) | 2.66 |
| WL-E | 71.05 ± 2.10 (4) | 52.02 ± 4.13 (3) | 82.97 ± 0.99 (4) | 81.25 ± 1.88 (5) | 3.88 |
| WL-SP | 68.12 ± 1.72 (5) | 53.50 ± 2.54 (2) | 82.84 ± 0.96 (4) | 80.62 ± 1.13 (6) | 5.00 |
| WL-OA | 75.18 ± 2.04 (1) | 53.76 ± 3.36 (2) | 85.13 ± 0.83 (2) | 85.22 ± 0.96 (1) | 2.11 |
| RGK-1 | 74.97 ± 1.26 (2) | 55.28 ± 3.89 (1) | 83.14 ± 0.89 (4) | 82.73 ± 0.81 (3) | 4.44 |
| RGK-2 | 74.35 ± 1.46 (2) | 54.85 ± 3.52 (1) | 82.09 ± 1.09 (5) | 81.95 ± 0.78 (4) | 4.66 |
| TK+MKL | 67.91 ± 2.34 (6) | 40.75 ± 4.68 (5) | 72.64 ± 0.98 (9) | 73.02 ± 0.81 (8) | 7.00 |
| MV | **75.31 ± 2.69** (1) | **56.88 ± 3.45** (1) | **86.56 ± 0.84** (1) | **86.13 ± 0.86** (1) | 1.00 |

nels, the kernel algorithms - WL methods, RGK - have better performance than embedding methods.

Among the kernel algorithms, WL kernels show consistent performance across all datasets. The WL-S and WL-OA have better performance among the WL variants. SP kernel has good performance in smaller and PROTEINS datasets but performance is low in others. TK+MKL have good performance in smaller datasets than the larger ones. On the other hand, the performance of RGK kernels is lower in smaller datasets and comparitively

**Table 3.3:** Kernel weights of different views in MKL setting

| Kernel | View-I | View-II | View-III | View-IV |
|---|---|---|---|---|
| MUTAG | 0.38 | 0.41 | 0.11 | 0.10 |
| PTC-MR | 0.25 | 0.35 | 0.06 | 0.34 |
| PTC-FR | 0.31 | 0.28 | 0.15 | 0.26 |
| PTC-FM | 0.23 | 0.31 | 0.08 | 0.38 |
| PTC-MM | 0.28 | 0.23 | 0.17 | 0.32 |
| PROTEINS | 0.00 | 0.03 | 0.11 | 0.86 |
| ENZYMES | 0.00 | 0.07 | 0.16 | 0.77 |
| NCI1 | 0.00 | 0.00 | 0.09 | 0.91 |
| NCI109 | 0.00 | 0.00 | 0.09 | 0.91 |

higher in larger ones. The performance of RW and GL kernels are comparitively lower.

The weights learned by SimpleMKL are given in Table 3.3. This information gives an insight into the prominence of each view in the multi-view setting. It is noteworthy that in large datasets such as PROTEINS, ENZYMES, NCI1 and NCI109, view-IV has the largest weight while for other smaller datasets the weights for view-I and view-II are large compared to others. This points out that global graph properties help to produce better results in larger datasets while in smaller datasets localized features should also be taken into consideration.

The experiments were also done to assess the ability of each view for capturing the information. For that, graph kernels as well as embedding techniques were used. In the case of graph kernels, R-convolution kernels defined in Section 3.4 were used and in this process, for each view, the information corresponding to that was considered for analysing the data. For embedding approach, for each view, the vectors related to that were used for data representation and the optimal kernel was selected using MKL approach. The single view embedding approach was also applied on the data and in this technique, for each data point, a vector was constructed by concatenating the vectors corresponding to each of the views under consideration. The MKL method was used for selecting the optimal kernel. The results are shown in Table: 3.4 in which the best results are given in bold letters. These experiments clearly demonstrated the merit of using multi-view graph embedding approach.

## 3.6 Conclusion

The data in the form of graphs has to be converted into suitable mathematical objects like vectors for designing an embedding technique while for graph kernels an appropriate RKHS mapping has to be found out using the graphical properties. The state-of-the-art

**Table 3.4:** Accuracy using designed R-convolution kernels, graph embeddings & single view.

| Approach | View | MUTAG | PTC-MR | PTC-FR | PTC-FM | PTC-MM |
|---|---|---|---|---|---|---|
| Graph Kernel | View-I | 88.68 $\pm$ 4.82 | 57.81 $\pm$ 5.65 | 60.34 $\pm$ 4.13 | 61.05 $\pm$ 4.59 | 61.22 $\pm$ 4.77 |
| | View-II | 88.68 $\pm$ 3.96 | 58.42 $\pm$ 5.87 | 59.64 $\pm$ 3.21 | 62.26 $\pm$ 4.88 | 60.27 $\pm$ 4.14 |
| | View-III | 89.64 $\pm$ 3.49 | 62.08 $\pm$ 4.00 | 65.27 $\pm$ 3.74 | 62.94 $\pm$ 3.49 | 65.64 $\pm$ 3.18 |
| | View-IV | 88.85 $\pm$ 5.07 | 60.89 $\pm$ 4.19 | 63.67 $\pm$ 3.64 | 60.83 $\pm$ 3.77 | 64.64 $\pm$ 3.02 |
| Graph Embedding | View-I | 88.68 $\pm$ 4.82 | 57.81 $\pm$ 5.65 | 61.77 $\pm$ 3.40 | 61.05 $\pm$ 4.59 | 61.22 $\pm$ 4.77 |
| | View-II | **89.87** $\pm$ **4.15** | 59.27 $\pm$ 5.38 | 59.64 $\pm$ 3.21 | **63.18** $\pm$ **4.71** | 60.27 $\pm$ 4.14 |
| | View-III | 89.64 $\pm$ 3.49 | 62.08 $\pm$ 4.01 | **67.58** $\pm$ **4.53** | 63.00 $\pm$ 3.72 | **66.32** $\pm$ **4.55** |
| | View-IV | 89.84 $\pm$ 4.61 | **62.36** $\pm$ **4.43** | 65.81 $\pm$ 3.45 | 61.17 $\pm$ 4.22 | 65.65 $\pm$ 3.84 |
| Single-view | | 85.93 $\pm$ 4.71 | 61.40 $\pm$ 4.91 | 66.34 $\pm$ 3.67 | 59.40 $\pm$ 4.33 | 64.85 $\pm$ 4.07 |

| Approach | View | PROTEINS | ENZYMES | NCI1 | NCI109 |
|---|---|---|---|---|---|
| Graph Kernel | View-I | 71.45 $\pm$ 2.39 | 35.71 $\pm$ 4.66 | 72.10 $\pm$ 1.42 | 76.11 $\pm$ 1.13 |
| | View-II | 57.56 $\pm$ 2.84 | 40.64 $\pm$ 4.18 | 72.56 $\pm$ 1 74 | 71.73 $\pm$ 1.89 |
| | View-III | 71.52 $\pm$ 2.36 | 47.65 $\pm$ 4.06 | 81.01 $\pm$ 0.91 | 80.32 $\pm$ 1.14 |
| | View-IV | **74.55** $\pm$ **2.83** | 48.77 $\pm$ 4.22 | 81.87 $\pm$ 1.22 | 81.52 $\pm$ 1.27 |
| Graph Embedding | View-I | 72.56 $\pm$ 2.19 | 35.71 $\pm$ 4.66 | 74.35 $\pm$ 1.52 | 78.35 $\pm$ 1.27 |
| | View-II | 58.84 $\pm$ 2.76 | 41.45 $\pm$ 3.95 | 74.24 $\pm$ 1.84 | 74.09 $\pm$ 2.15 |
| | View-III | 72.67 $\pm$ 2.18 | 47.65 $\pm$ 4.06 | 82.24 $\pm$ 1.15 | 82.67 $\pm$ 1.37 |
| | View-IV | **74.55** $\pm$ **2.83** | **49.54** $\pm$ **3.94** | **82.67** $\pm$ **1.34** | **82.78** $\pm$ **1.51** |
| Single-view | | 71.73 $\pm$ 1.80 | 48.33 $\pm$ 3.63 | 82.44 $\pm$ 1.24 | 81.60 $\pm$ 1.09 |

graph embeddings and kernels depend on a single property of graphs in their design. We selected a number of characteristic properties of graphs for their vector space embedding and used multiple kernel learning approach for optimal kernel representation. The optimization technique used to solve the model is same as that in SimpleMKL formulation. The graph embedding designed using multi-view approach gave promising results in our analysis. The approach of multi-view is flexible in the sense that any new feature of a graph can be included by appending it as a view. The MKL approach we used helps to assign appropriate weights to the views and these weights help to create further insight into the relative role of individual views in prediction.

Chapter 4

# Graph Kernels Based on Optimal Node Assignment

The common approaches used for graph kernel designs are R-convolution kernel and graph embedding. In the previous chapter, we used their combination to design multi-view graph kernels. Similar to R-convolution, optimal assignment kernel is a framework for designing kernels on structured data. However, R-convolution domain is widely studied compared to that of the optimal assignment.

The R-convolution kernel processes the structural similarity of its argument graphs by comparing each substructure of a data point with all substructures in the other. This results in visiting the same part multiple times which causes some redundancy in the graph structure learning. The optimal assignment framework is devoid of such problems as it involves the bijection between graph structures. This chapter describes graph kernels we designed using the principles of optimal assignment kernel framework.

In this work, we used optimal assignment (OA) kernel framework [5] and 1-dimensional Weisfeiler-Lehman (WL) color refinement algorithm [10]. A brief description of OA kernel framework is discussed in Section 2.1.3 and that of WL algorithm is discussed in Section 2.2. An example of the processes involved in the WL algorithm is given in Figure 4.1.

The rest of the chapter is organized as follows. Section 4.1 describes the theoretical formulation of the optimal assignment kernels and Section 4.2 discusses the kernel computation procedure using the associated hierarchy. The experimental results and related discussions are in Section 4.3, and concluding remarks are given in Section 4.4.

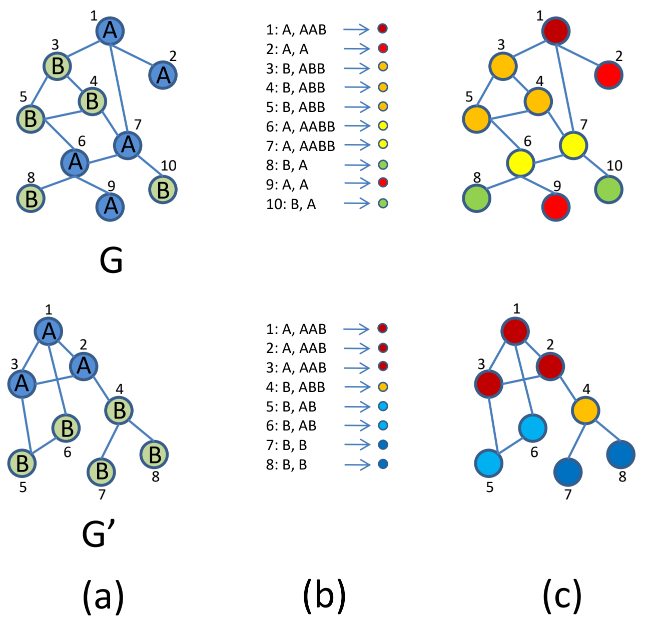


**Figure 4.1:** **(a)** Two sample graphs. **(b)** The string representation of the nodes and corresponding hashing to a color. **(c)** The graphs with WL labels marked on nodes.

## 4.1 Design of optimal node assignment graph kernels

The motivation behind optimal node assignment (ONA) kernels and its description are given in this section.

### 4.1.1 Characteristics of the Problem Under Study

As discussed in Section 2.4.2, most of the kernel designs are formulated based on the enumeration through the graph substructures with the aid of either an R-convolution or some explicitly designed comparison methods. On the other hand, the optimal assignment domain is a less explored area.

The objective of our work is to design a graph kernel by making use of the properties of an assignment problem. The major challenge in this regard is to find out the criterion of assigning a graph substructure to that of one in the counterpart graph. Once this is done, it is important to design the base kernel that processes the substructures and satisfies the *strong* property. Also, the kernel values have to be calculated by making use of the hierarchy structure for faster computations. The kernels designed in this fashion have the optimal comparison operations between the argument graphs as each substructure in one graph is compared with one and only one in the other graph and that is why we have chosen optimal assignment framework for the kernel design.

In our formulation of the optimal assignment kernels, the substructure we used for comparing two graphs are their nodes, as it is one of the convenient ways for comparison. For that, the nodes in a graph are grouped into subsets called *neighbourhood sets*, based on their local neighbourhood structure extracted using WL test and a vector representation is defined for them. A bijection is defined between the corresponding *neighbourhood sets* of the argument graphs after incorporating necessary steps for equalizing the cardinality of the sets. Using this bijection and vector representations, the *neighbourhood sets* are represented as a matrix. A kernel is then defined over the domain that consists of the *neighbourhood sets* in terms of these matrices. An aggregate measure of these kernel values is taken as the final graph kernel value. We designed two OA kernels which differ in the kernel defined over the *neighbourhood sets*. The validity of those kernels has been proved mathematically. We have also described the efficient computation of the kernels using the concept of the hierarchy associated with the OA framework. The efficiency of the designed kernels was analyzed using real-world problems by incorporating the kernels in support vector machines. The results were found to be superior in comparison with the other state-of-the-art graph kernels. The concepts are discussed in detail in the next sections.

### 4.1.2 Neighbourhood sets

For the kernel definitions, a bijection has to be formed between $V$ and $V'$. This is done with the construction of neighbourhood sets as defined below.

#### 4.1.2.1 Formulation of neighbourhood sets

The nodes that have similar neighbourhoods have to be grouped for defining the bijections involved in the optimal assignment kernels we have designed. The labels generated by the 1-dimensional Weisfeiler-Lehman color refinement algorithm [10] on the graphs are used for this purpose.

Let $\Sigma_C$ be the alphabet that consists of labels generated by an iteration of WL algorithm on these graphs. A total order $\leq$ is defined on $\Sigma_C$ such that $a < b$ if $a$ appears before $b$ in $\Sigma_C$. Consider a mapping $l_C : V \to \Sigma_C$, where $l_C(v)$ is the WL refined label of $v$. Corresponding to each element $l_i \in \Sigma_C$, a neighbourhood set $g_{l_i}$ is constructed where, $g_{l_i} = \{v \in V : l_C(v) = l_i\}$. Because of the peculiarity of WL algorithm, the members of $g_{l_i}$ have similar neighbourhood. Similarly $g'_{l_i}$ is constructed for graph $G'$. Consider $\mathcal{D} = \{g^k_{l_i}, k = 1, 2. \dots N, i = 1, 2, \dots |\Sigma_C|\}$ where $|\Sigma_C|$ is the cardinality of $\Sigma_C$. A matrix representation for each element of $\mathcal{D}$ is constructed as given below.

#### 4.1.2.2 Matrix representation for the neighbourhood sets

The cardinality of $g_{l_i}$ and $g'_{l_i}$ may differ. To rectify that, dummy nodes are added to the deficient set where dummy nodes are considered as separate entities that do not affect the graph structure. Let $d_{l_i} = \max(|g_{l_i}|, |g'_{l_i}|) - |g_{l_i}|$. Let $g_{l_i} := g_{l_i} \cup v_{d_{l_i}}$ where $v_{d_{l_i}}$ be a set of $d_{l_i}$ number of dummy nodes with the WL label $l_i$. For each $v_j \in g_{l_i}$, a vector $\mathcal{V}_{v_j}$ of length $|\Sigma|$ is constructed as follows. If $v_j$ is a non-dummy node, its $k^{th}$ component represents how many of its neighbours have the $k^{th}$ element of $\Sigma$ as the node label. If it is a dummy node, all elements of $\mathcal{V}_{v_j}$ are taken to be zero. The vector representation of the non-dummy nodes are identical since their WL label, $l_i$, is same. Hence the vector representation of the WL label $l_i$ is taken to be the vector representation of non-dummy nodes in $g_{l_i}$ and it is notated as $\mathcal{V}_{l_i}$.

Now a matrix $M_{g_{l_i}}$ of dimension $|g_{l_i}| \times |\Sigma|$ and a function $\sigma : g_{l_i} \rightarrow \{1, 2, \ldots |g_{l_i}|\}$ are created where $j^{th}$ row of $M_{g_{l_i}}$ is $\mathcal{V}^T_{v_j}$ and $\sigma(v_j) = j$. Similarly $M'_{g_{l_i}}$ and $\sigma'$ are constructed for graph $G'$. Let $\tilde{V} = \cup_{i=1}^{|\Sigma_C|} g_{l_i}$ and $\tilde{V'} = \cup_{i=1}^{|\Sigma_C|} g'_{l_i}$. Now $\tilde{V}$ and $\tilde{V'}$ are sets of equal cardinality which consist of nodes in $G$ and $G'$ respectively along with the corresponding dummy nodes.

### 4.1.3 Optimal node assignment (ONA) graph kernels

The ONA kernels are defined with the aid of two kernels $k_{ns_1}$ and $k_{ns_2}$ which are formulated using the matrix representation of neighbourhood sets. The definitions of those kernels are as follows.

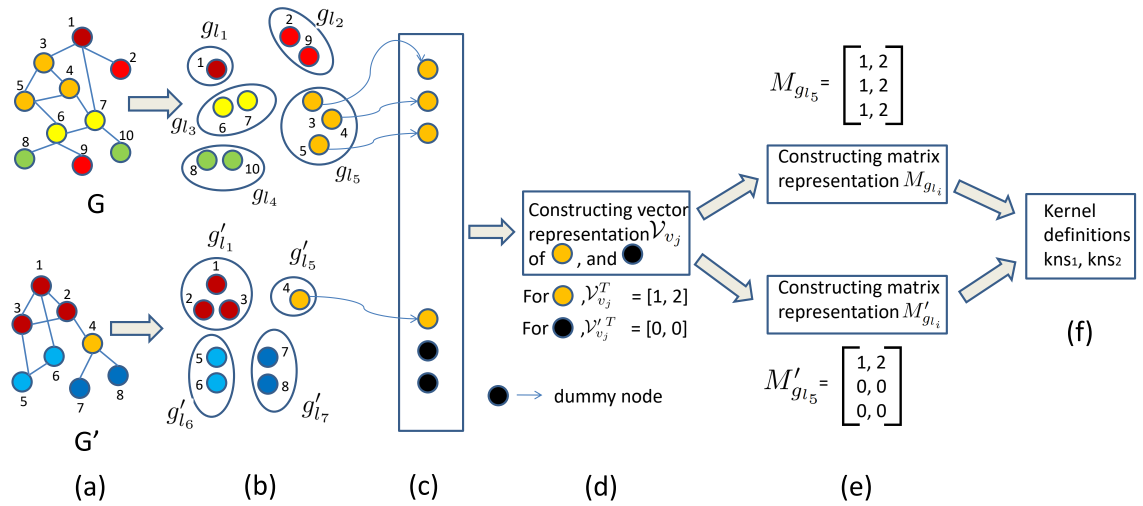
Define kernel

$$k_{ns_1} : \mathcal{D} \times \mathcal{D} \rightarrow \mathbb{R} \text{ such that } k_{ns_1}(g_{l_p}, g'_{l_q}) = \exp\left(-\gamma \left\|(M_{g_{l_p}} - M'_{g_{l_q}})\right\|_{L_{21}}\right) \tag{4.1}$$

where $\gamma > 0$ is a tuning parameter, $l_p, l_q \in \Sigma_C$, and $\|.\|_{L_{21}}$ is the $L_{21}$ matrix norm. Here $L_{21}$ norm is defined for a matrix $A$ as, $\|A\|_{L_{21}} = \sum_i \|A[i]\|$ where $A[i]$ is the $i^{th}$ row of $A$ and $\|.\|$ is the Euclidean norm.

Define kernel

$$k_{ns_2} : \mathcal{D} \times \mathcal{D} \rightarrow \mathbb{R} \text{ such that } k_{ns_2}(g_{l_p}, g'_{l_q}) = \sum_i \left\langle (M_{g_{l_p}}(i, .)), (M'_{g_{l_q}}(i, .)) \right\rangle \tag{4.2}$$



G
G′
$g_{l_1}$
$g_{l_2}$
$g_{l_3}$
$g_{l_4}$
$g_{l_5}$
$g'_{l_1}$
$g'_{l_5}$
$g'_{l_6}$
$g'_{l_7}$
Constructing vector representation $\mathcal{V}_{v_j}$ of ●, and ●
For ●, $\mathcal{V}_{v_j}^T$ = [1, 2]
For ●, $\mathcal{V}'^T_{v_j}$ = [0, 0]
● → dummy node
$M_{g_{l_5}}$ = [1, 2; 1, 2; 1, 2]
Constructing matrix representation $M_{g_{l_i}}$
Constructing matrix representation $M'_{g_{l_i}}$
$M'_{g_{l_5}}$ = [1, 2; 0, 0; 0, 0]
Kernel definitions kns₁, kns₂
(a)
(b)
(c)
(d)
(e)
(f)

**Figure 4.2:** The illustration of the kernel calculation corresponding to a WL label $l_5$. **(a)** Two graphs $G$ and $G'$ with WL labels marked as different colors on the nodes. **(b)** Constructing the sets $g_{l_i}$. **(c)** Augmenting $g'_{l_5}$ with dummy nodes (node marked as black). **(d)** Constructing the vectors $\mathcal{V}_{v_j}$ and $\mathcal{V}'_{v_j}$ - vectors are obtained from the string representation of nodes 3,4, and 5 in $G$ and node 4 in $G'$ as shown in the Figure 4.1 (b). The dummy node vector representation is taken as zero vector. **(e)** Constructing $M_{g_{l_5}}$ and $M'_{g_{l_5}}$ corresponding to $G$ and $G'$. **(f)** Applying the matrices into kernel definitions $k_{ns_1}$ and $k_{ns_2}$.

where $M_{(..)}(i,.)$ denotes $i^{th}$ row of matrix $M$.

The entire process involved in the construction of $k_{ns_1}$ and $k_{ns_2}$ is summarized in Figure 4.2. Using $k_{ns_1}$ and $k_{ns_2}$, ONA graph kernels are formulated whose description is given below.

ONA kernel definition corresponding to $k_{ns_1}$ is defined as,

$$K_{ONA} : \mathcal{G} \times \mathcal{G} \rightarrow \mathbb{R} \text{ such that } K_{ONA}(G, G') = \prod_{l_i \in \Sigma_C} k_{ns_1}(g_{l_i}, g'_{l_i}) \tag{4.3}$$

ONA kernel definition corresponding to $k_{ns_2}$ is defined as,

$$\tilde{K}_{ONA} : \mathcal{G} \times \mathcal{G} \rightarrow \mathbb{R} \text{ such that } \tilde{K}_{ONA}(G, G') = \sum_{l_i \in \Sigma_C} k_{ns_2}(g_{l_i}, g'_{l_i}) \tag{4.4}$$

**Theorem 4.1.** *$K_{ONA}(G, G')$ is an optimal assignment kernel.*

*Proof.* First we prove that corresponding to a WL label $l_i$, $k_{ns_1}(g_{li}, g'_{li})$ is an optimal assignment kernel. Consider,

$$\begin{aligned} k_{ns_1}(g_{l_i}, g'_{l_i}) &= \exp\Big(-\gamma \|(M_{g_{l_i}} - M'_{g_{l_i}})\|_{L21}\Big) \\ &= \prod_{j=1}^{|g_{l_i}|} \exp\Big(-\gamma \|M^j_{g_{l_i}} - M'^j_{g_{l_i}}\|\Big) \end{aligned}$$

where $M^j_{g_{l_i}}$ and $M'^j_{g_{l_i}}$ are the $j^{th}$ row of $M_{g_{l_i}}$ and $M'_{g_{l_i}}$ respectively.

Define the bijective function

$$B_{l_i} : g_{l_i} \to g'_{l_i}, \text{ where } B_{l_i}(v) = v' \text{ if } \sigma(v) = \sigma(v')$$

Using $B_{l_i}$, $k_{ns_1}$ can be written as,

$$\begin{aligned} k_{ns_1}(g_{l_i}, g'_{l_i}) &= \prod_{(v, B_{l_i}(v))} \exp\Big(-\gamma \|\mathcal{V}_v - \mathcal{V}_{B_{l_i}(v)}\|\Big) \\ &= \prod_{(v, B_{l_i}(v))} k_{l_i}(v, B_{l_i}(v)) \end{aligned}$$

where $\mathcal{V}_v$ and $\mathcal{V}_{B_{l_i}(v)}$ are the vectors as defined in Section 4.1.2.2 and for each $l_i \in \Sigma_C$, $k_{l_i} : g_{l_i} \times g'_{l_i} \to \mathbb{R}$ is a valid kernel. $k_{l_i}$ is a strong kernel since its range set $\big\{1, \exp(-\gamma \|\mathcal{V}_v - \mathcal{V}_{B_{l_i}(v)}\|)\big\}$ is of cardinality two [5].

The permutations of the rows of the matrices $M_{g_{l_i}}$ and $M^{'}_{g_{l_i}}$ correspond to different bijections. The value of $k_{ns_1}$ remains the same for all of them due to the formulation characteristics of $k_{l_i}$ and the fact that the matrices $M_{g_{l_i}}$ and $M'_{g_{l_i}}$ have identical non-zero rows by virtue of the WL label $l_i$. Hence $k_{ns_1}(g_{l_i}, g'_{l_i})$ is an optimal assignment kernel as per (2.4).

Now consider

$$B : \tilde{V} \to \tilde{V}' \text{ where } B(v) = B_{l_i}(v) \text{ if } l_C(v) = l_i$$

.

$B$ is a bijection as each $B_{l_i}$s are bijective functions. Using $B$, $K_{ONA}$ can be written as

$$K_{ONA}(G, G') = \prod_{v, B(v)} \exp(-\gamma \|\mathcal{V}_v - \mathcal{V}_{B(v)}\|)$$

Therefore $K_{ONA}(G, G')$ is an optimal assignment kernel. $\square$

The proof to show $\tilde{K}_{ONA}$ is an optimal assignment kernel is given in A.1. As the bijection involved in the kernel formulations is between the sets of nodes of the argument graphs augmented with dummy nodes, we named the kernels as optimal node assignment kernels.

Till now we have considered the case of one WL iteration. It is straight forward to extend the concepts to further iterations of WL color refinement. Let $h$ be the number of iterations. Then the graph kernel corresponding to $K_{ONA}$ for $h$ iterations can be defined as

$$K_{ONA}(G, G') = \prod_{j=1}^{h} \prod_{l_i \in \Sigma_C^j} k_{ns_1}(g_{l_i}, g'_{l_i}) \tag{4.5}$$

Similarly

$$\tilde{K}_{ONA}(G, G') = \sum_{j=1}^{h} \sum_{l_i \in \Sigma_C^j} k_{ns_2}(g_{l_i}, g'_{l_i}) \tag{4.6}$$

where $\Sigma_C^j$ is the WL alphabet obtained in iteration $j$.

The ONA kernels can be extended to the case of attributed graphs which contains additional vector information in the nodes along with its label. In this case, the base kernel $k_{l_i}$ can be multiplied with a function that process these vectors such that the resultant function is a strong kernel.

It has been shown that the optimal assignment kernel can be computed efficiently from the hierarchy corresponding to the strong kernel rather than using the definition equations as such [5]. In the following section, we describe how the hierarchy can be constructed for the proposed ONA kernels and the tree traversal algorithms for their computation.

## 4.2 Kernel computation using heirarchy

The WL iterations are applied on the set $\mathcal{G}$ and the hierarchy $T$ is then constructed as follows. As there are $h$ WL iterations, the number of levels in $T$ is $h + 1$. Elements in $\Sigma$ form the nodes in the zero level of $T$. The nodes in the $0 < i \leq h$ level are formed by the labels in $\Sigma_C^i$. If the first element of the WL string of node $v$ in the level $i$ is the WL label of node $v'$ in the level $(i - 1)$, we define $v$ as the child of $v'$ and an edge is formed between them.

Let $V_T$ be the set of nodes in $T$ except the nodes in the zero level. Let $V_{T_{ij}}$ be the node in $T$ that lies in position $0 < j \leq |\Sigma_C^i|$ in the WL iteration $0 < i \leq h$. A total order, $\leq$, is defined on nodes of $T$ as follows: (i) $V_{T_{ij}} < V_{T_{ik}}$ if $j < k$ and (ii) $V_{T_{ip}} < V_{T_{jq}}$ if $i < j$.

The number of occurrences of WL labels across whole iterations in a graph $G$ can be found out from $T$ as follows. As each leaf node in $T$ corresponds to a label in WL iteration $h$, there exists a unique path that connects them to the root node. By the characteristics of the WL algorithm, each node in a graph $G$ is associated with a leaf node in $T$. Hence by traversing through the paths that connect the leaf nodes to the root node in $T$, it is possible to find the WL labels that have been generated for $G$ in the whole WL iterations. Using this information, a vector $G_V$ of length $|V_T|$ is constructed where $i^{th}$ element of $G_V$ represents the number of times $i^{th}$ element of $V_T$ appears in WL iterations of $G$. The process is summarized in Algorithm 4.1.

**Algorithm 4.1:** Calculating the histogram vector $G_V$ of graph $G$ for $h$ iterations of WL color refinement

```
Input  : The hierarchy tree $T$ for $h$ number of WL iterations and a graph $G(V, E)$.
Output: The vector $G_V$ for graph $G$.
1  Initialize $G_V$ to zero vector of dimensions $|V_T| \times 1$
2  for every node $v$ in $V$ do
3      Find the leaf node in $T$ corresponding to WL label of $v$ in iteration: $h$
4      Find the path $P$ corresponding to the leaf node
5      for every elements $p$ in $P$ do
6          Find the position $(i, j)$ of $p$ in $T$ as per the total order, $\leq$
7          $G_{V_{ij}} = G_{V_{ij}} + 1$
8      end
9  end
10 return $G_V$
```

A vector $N_T$ of the dimension $|V_T|$ is constructed whose $i^{th}$ element is the norm of the vector representation $\mathcal{V}$ (as explained in Section 4.1.2.2 ) of the $i^{th}$ element of $V_T$. Now we define a kernel that is calculated using $N_T$, $G_V$, and $G'_V$ as follows.

$$K_T(G, G') = \exp\big( - \gamma \times \|(G_V - G'_V) \odot N_T\|_1\big) \tag{4.7}$$

**Theorem 4.2.** $K_T(G, G') = K_{ONA}(G, G') \; \forall \; G, G' \in \mathcal{G}$.

*Proof.*

$$\begin{aligned} K_T(G, G') &= \exp\big(-\gamma \times \|(G_V - G'_V) \odot N_T\|_1\big) \\ &= \exp\Big(-\gamma \times \sum_{j=1}^{|V_T|} |G_V(j) - G'_V(j)| \times N_T(j)\Big) \\ &= \exp\Big(-\gamma \times \sum_{j=1}^{|V_T|} |G_V(j) - G'_V(j)| \times \|\mathcal{V}_j\|\Big) \end{aligned}$$

where $\mathcal{V}_j$ is the vector representation of the WL label corresponding to $j^{th}$ node in $T$ as explained in Section 4.1.2.2.

$G_V(j)$, and $G'_V(j)$ are equal to the number of non-dummy nodes in $g_j$, and $g'_j$ respectively or equivalently, the number of non-zero rows in $M_{g_j}$ and $M'_{g_j}$ respectively. Therefore,

$$|G_V(j) - G'_V(j)| \times \|\mathcal{V}_j\| = \|(M_{g_j} - M'_{g_j})\|_{L_{21}}.$$

Hence,

$$\exp\Big(-\gamma \times \sum_{j=1}^{|V_T|} |G_V(j) - G'_V(j)| \times \|\mathcal{V}_j\|\Big) = \exp\Big(-\gamma \times \sum_{j=1}^{|V_T|} \|(M_{g_j} - M'_{g_j})\|_{L_{21}}\Big).$$

Now $V_T = \bigcup_{j=1}^{h} \bigcup_{i=1}^{|\Sigma_C^j|} \{l_i\}$, where $l_i$ is the $i^{th}$ element of $\Sigma_C^j$. Therefore,

$$\begin{aligned} \exp\left(-\gamma \times \sum_{j=1}^{|V_T|} \left\|\left(M_{g_j} - M'_{g_j}\right)\right\|_{L_{21}}\right) &= \exp\left(-\gamma \times \sum_{j=1}^{h} \sum_{l_i \in \Sigma_C^j} \left\|\left(M_{g_{l_i}} - M'_{g_{l_i}}\right)\right\|_{L_{21}}\right) \\ &= \prod_{j=1}^{h} \prod_{l_i \in \Sigma_C^j} \exp\left(-\gamma \left\|\left(M_{g_{l_i}} - M'_{g_{l_i}}\right)\right\|_{L_{21}}\right) \\ &= \prod_{j=1}^{h} \prod_{l_i \in \Sigma_C^j} k_{ns_1}(g_{l_i}, g'_{l_i}) = K_{ONA}(G, G') \end{aligned}$$

□

Similarly, kernel value $\tilde{K}_{ONA}(G, G')$ in (4.6) can be computed as per (4.8),

$$\tilde{K}_T(G, G') = \sum_{k=1}^{|V_T|} \min\big(G_V(k), G'_V(k)\big) \times \big(N_T(k)\big)^2 \tag{4.8}$$

The proof of $\tilde{K}_T(G, G') = \tilde{K}_{ONA}(G, G') \; \forall \; G, G' \in \mathcal{G}$ is shown in A.2.

### 4.2.1 Computational complexity analysis

The kernel evaluation is done in two stages. First stage involves the application of WL algorithm and formulation of the vector $G_V$. This involves operations of $\mathcal{O}(hN|\Sigma||V|)$. In the second stage, to find kernel value using hierarchy, $N^2\mathcal{O}(|V_T|)$ operations are required. Considering the above two steps the kernel computation requires operation of $\mathcal{O}(hN|\Sigma||V| + N^2|V_T|)$.

## 4.3 Experiment

The efficiency of the proposed ONA kernels ($K_{ONA}$ and $\tilde{K}_{ONA}$) was analyzed by subjecting them on real-world data sets and compared its performance with state-of-the-art graph embeddings namely: Graph2vec (G2V) [20], Subgraph2vec (S2V) [21], Node2vec (N2V) [22], GE-FSG (Graph Embedding with Frequent Sub-Graphs) [23], Deep graph kernels (DGK) [24] and graph kernels namely: Random walk (RW) kernel [30], Shortest path (SP) kernel [7] (labelled and unlabelled version), Graphlet kernel (GL) [43], Weisfeiler-Lehman (WL) kernels [25] (WL-subtree (WL-S), WL-edge (WL-E) and WL-shortest path (WL-SP), WL-optimal assignment kernel (WL-OA) [5], RetGK kernels (the variants with explicit RKHS mapping (RGK-1) and with approximated mapping (RGK-2)) [35], and Treelet kernel+MKL (TK+MKL) [54]. The description about the datasets are given in Section 3.5.2.1 and that of the experimental setup in Section 3.5.1.

### 4.3.1 Results and Discussion

The accuracy results are shown in Table 4.1. The best results are marked in bold letters. The $K_{ONA}$ and $\tilde{K}_{ONA}$ have shown excellent performance in all datasets as they secured an average rank of 1 and 2 respectively. In PTC-MR and PTC-FM datasets, the performance of the $K_{ONA}$ kernel is significantly higher from the rest of the models as it has more than 1% increase in accuracy from the second-best performer of those data. Comparing $K_{ONA}$, and

**Table 4.1:** Accuracy (along with standard deviation) of the multi-view graph embedding along with state-of-the-art embedding techniques and graph kernels. The rank in statistical significance test is given in the bracket.

| **Embed.** | **MUTAG** | **PTC-MR** | **PTC-FR** | **PTC-FM** | **PTC-MM** |
|---|---|---|---|---|---|
| G2V | 84.26 ± 10.25 (5) | 61.23 ± 5.67 (2) | 67.23 ± 4.77 (2) | 61.23 ± 5.05 (3) | 66.58 ± 5.65 (2) |
| S2V | 88.23 ± 1.89 (2) | 60.76 ± 1.49 (3) | 65.44 ± 1.78 (4) | 59.77 ± 2.12 (4) | 63.79 ± 1.83 (4) |
| N2V | 73.42 ± 8.75 (9) | 57.45 ± 8.94 (7) | 61.56 ± 9.44 (6) | 56.82 ± 9.74 (7) | 60.66 ± 8.35 (7) |
| GE-FSG | 83.54 ± 0.03 (5) | 58.94 ± 0.82 (5) | 60.17 ± 0.71 (6) | 56.29 ± 1.13 (6) | 59.83 ± 0.86 (7) |
| DGK | 88.46 ± 2.87 (2) | 60.15 ± 3.08 (4) | 66.36 ± 3.64 (3) | 60.54 ± 4.88 (4) | 65.44 ± 5.14 (3) |
| **Kernel** | **MUTAG** | **PTC-MR** | **PTC-FR** | **PTC-FM** | **PTC-MM** |
| RW | 74.27 ± 3.79 (8) | 58.19 ± 3.63 (5) | 65.56 ± 0.80 (4) | 58.83 ± 2.48 (5) | 62.50 ± 0.76 (4) |
| SP | 87.84 ± 5.66 (3) | 59.65 ± 6.02 (5) | 65.66 ± 4.77 (4) | 63.13 ± 5.12 (2) | 62.88 ± 5.78 (5) |
| GL | 74.16 ± 3.60 (8) | 58.25 ± 3.68 (5) | 66.91 ± 2.44 (3) | 56.70 ± 3.96 (6) | 64.78 ± 3.50 (3) |
| WL-S | 85.04 ± 4.70 (4) | 58.95 ± 3.99 (5) | 67.95 ± 2.74 (1) | 62.41 ± 4.73 (2) | 67.87 ± 3.85 (1) |
| WL-E | 83.70 ± 3.99 (5) | 57.61 ± 4.23 (6) | 67.44 ± 2.05 (2) | 60.06 ± 4.98 (4) | 66.00 ± 2.84 (2) |
| WL-SP | 85.82 ± 5.19 (4) | 58.40 ± 4.78 (5) | 62.86 ± 2.93 (5) | 54.84 ± 4.49 (8) | 60.56 ± 4.21 (6) |
| WL-OA | 87.27 ± 4.46 (3) | 60.59 ± 3.66 (3) | 66.62 ± 2.88 (2) | 59.30 ± 3.91 (4) | 67.15 ± 3.47 (1) |
| RGK-1 | 65.80 ± 3.52 (10) | 58.43 ± 3.63 (5) | 65.22 ± 1.66 (4) | 58.44 ± 2.53 (5) | 60.71 ± 2.55 (6) |
| RGK-2 | 65.51 ± 3.91 (10) | 56.25 ± 2.23 (7) | 66.29 ± 3.08 (3) | 58.77 ± 1.67 (5) | 61.78 ± 0.42 (5) |
| TK+MKL | 81.61 ± 5.61 (7) | 51.12 ± 6.48 (8) | 59.45 ± 4.38 (6) | 56.66 ± 5.22 (6) | 58.33 ± 5.69 (8) |
| $K_{ONA}$ | **88.50 ± 3.10** (1) | **62.40 ± 4.10** (1) | **68.63 ± 2.20** (1) | **64.27 ± 3.56** (1) | 67.65 ± 3.86 (1) |
| $\tilde{K}_{ONA}$ | 86.88 ± 3.75 (3) | 60.40 ± 3.77 (3) | 67.85 ± 2.33 (1) | 60.81 ± 1.24 (3) | **68.37 ± 4.03** (1) |

| **Embed.** | **PROTEINS** | **ENZYMES** | **NCI1** | **NCI109** | **Avg. Rank** |
|---|---|---|---|---|---|
| G2V | 73.48 ± 1.83 (3) | 36.67 ± 7.05 (6) | 74.05 ± 1.53 (8) | 74.87 ± 1.49 (8) | 4.33 |
| S2V | 74.17 ± 1.82 (2) | 32.55 ± 3.86 (8) | 78.12 ± 1.85 (7) | 78.67 ± 1.64 (7) | 4.55 |
| N2V | 56.82 ± 4.05 (8) | 26.44 ± 9.89 (11) | 55.64 ± 2.06 (13) | 52.37 ± 1.62 (13) | 9.00 |
| GE-FSG | 73.85 ± 0.03 (2) | 28.44 ± 2.68 (9) | 78.61 ± 0.02 (7) | 78.15 ± 0.02 (7) | 6.00 |
| DGK | 74.98 ± 0.82 (2) | 34.62 ± 5.02 (7) | 81.35 ± 0.94 (6) | 80.49 ± 0.92 (6) | 4.11 |
| **Kernel** | **PROTEINS** | **ENZYMES** | **NCI1** | **NCI109** | **Avg. Rank** |
| RW | 59.32 ± 0.38 (7) | 20.40 ± 2.85 (12) | 50.12 ± 0.13 (14) | 50.39 ± 0.06 (12) | 7.88 |
| SP | 73.83 ± 2.93 (3) | 34.77 ± 3.22 (7) | 67.15 ± 1.63 (10) | 70.63 ± 1.80 (9) | 5.33 |
| GL | 71.52 ± 1.76 (4) | 32.06 ± 3.22 (8) | 63.42 ± 1.01 (11) | 63.69 ± 1.42 (10) | 6.44 |
| WL-S | 74.40 ± 2.05 (2) | 49.85 ± 4.15 (4) | 84.61 ± 1.03 (3) | 84.17 ± 1.31 (2) | 2.66 |
| WL-E | 71.05 ± 2.10 (4) | 52.02 ± 4.13 (3) | 82.97 ± 0.99 (4) | 81.25 ± 1.88 (5) | 3.88 |
| WL-SP | 68.12 ± 1.72 (5) | 53.50 ± 2.54 (2) | 82.84 ± 0.96 (4) | 80.62 ± 1.13 (6) | 5.00 |
| WL-OA | 75.18 ± 2.04 (1) | 53.76 ± 3.36 (2) | 85.13 ± 0.83 (2) | 85.22 ± 0.96 (1) | 2.11 |
| RGK-1 | 74.97 ± 1.26 (2) | 55.28 ± 3.89 (1) | 83.14 ± 0.89 (4) | 82.73 ± 0.81 (3) | 4.44 |
| RGK-2 | 74.35 ± 1.46 (2) | 54.85 ± 3.52 (1) | 82.09 ± 1.09 (5) | 81.95 ± 0.78 (4) | 4.66 |
| TK+MKL | 67.91 ± 2.34 (6) | 40.75 ± 4.68 (5) | 72.64 ± 0.98 (9) | 73.02 ± 0.81 (8) | 7.00 |
| $K_{ONA}$ | **75.91 ± 2.07** (1) | **55.36 ± 3.04** (1) | **85.90 ± 0.78** (1) | **85.34 ± 0.97** (1) | 1.00 |
| $\tilde{K}_{ONA}$ | 74.90 ± 2.00 (2) | 52.39 ± 3.30 (2) | 85.24 ± 0.81 (2) | 85.12 ± 0.97 (1) | 2.00 |

$\tilde{K}_{ONA}$, $K_{ONA}$ performed significantly better in the case of MUTAG, PTC-MR, PTC-FM, PROTEINS, and ENZYMES datasets whereas the performance is similar in other datasets. It has to be noted that the average rank of $\tilde{K}_{ONA}$ kernel and WL-OA kernel are almost the same.

The proposed kernels and WL-OA belong to the category of optimal assignment kernels while the rest of the models can be considered as R-convolution kernels. Comparing the models based on this categorization, the performance of the optimal assignment models were found to be better in the datasets we used.

The RW, SP, and GL kernels process only structural information of the graphs and do not make use of label information compared with the rest of the models. That might have attributed to the lower performance of these kernels. Comparing the performances of WL kernels and RetGK kernels, the WL-subtree (WL-S) variant had the best performance. The performance of the WL-edge (WL-E) variant and RetGK kernels were similar and that of the WL-shortest path (WL-SP) variant was the lowest.

The run time of the algorithms except the methods based on embeddings and MKL is shown in Table 4.2. These algorithms are avoided as they involve an embedding process and then applying MKL algorithms and as our aim is to compare between the kernel algorithms. We can see that the runtime of OA kernels are higher compared to that of R-convolution kernels despite having higher performance.

**Table 4.2:** Runtime (wall clock time) of the algorithms

| **Kernel** | **MUTAG** | **PTC-MR** | **PTC-FR** | **PTC-FM** | **PTC-MM** |
|---|---|---|---|---|---|
| RW | 4.5" | 25" | 23" | 21" | 19" |
| SP | 0.37" | 1.20" | 0.94" | 1.10" | 1.25" |
| GL | 0.67" | 1.45" | 3.12" | 3.06" | 2.68" |
| WL-S | 2.2" | 4.5" | 4.6" | 4.3" | 4.1" |
| WL-E | 1.0" | 1.9" | 1.6" | 1.5" | 1.6" |
| WL-SP | 0.8" | 1.7" | 1.8" | 1.5" | 1.6" |
| WL-OA | 5.6" | 42.7" | 1’ 04" | 57" | 51" |
| RGK-1 | 2" | 6" | 6.2" | 6.2" | 6.4" |
| RGK-2 | 0.15" | 0.20" | 0.20" | 0.20" | 0.20" |
| $K_{ONA}$ | 2.2" | 6.8" | 7.7" | 6.9" | 7" |
| $\tilde{K}_{ONA}$ | 2.1" | 6.3" | 7.2" | 6.6" | 6.7" |
| **Kernel** | **PROTEINS** | **ENZYMES** | **NCI1** | **NCI109** | |
| RW | 9’ 42" | 1’ 08" | $>$ 2h | $>$ 2h | |
| SP | 1’ 53" | 3.25" | 21" | 23" | |
| GL | 26.0" | 12.1" | 21" | 20" | |
| WL-S | 24.6" | 10.5" | 1’ 14" | 1’15" | |
| WL-E | 14.0" | 4.8" | 27" | 28" | |
| WL-SP | 36.3" | 14.4" | 1’ 15" | 1’ 17" | |
| WL-OA | 1h 29" | 7’ 30" | $>$ 2h | >2 h | |
| RGK-1 | 1’ 26" | 23" | 16’ 50" | 16’ 57" | |
| RGK-2 | 2" | 0.60" | 8.0" | 6.5" | |
| $K_{ONA}$ | 2’ 18" | 50" | 1h 12’ | 1h 13’ | |
| $\tilde{K}_{ONA}$ | 1’ 50" | 27" | 54’ | 54’ | |

## 4.4 Conclusions

We designed two graph kernels based on the optimal assignment kernel framework. The bijection as a part of the framework is built between the nodes of the argument graphs and the validity of the proposed kernels is mathematically proved. For making the kernel computation effective, the hierarchy tree associated with the framework has been utilized. By plugging the ONA kernels into the SVM, their efficiency was analyzed using real-world graph datasets and their performance was found to be superior compared to the other state-of-the-art graph kernels.

We also found that the performance of the kernels belonging to the family of optimal assignment framework is better than the R-convolution kernels. This is a promising direction to further explore the kernel designs based on the OA kernel framework. However, the runtime of OA kernels is higher compared with R-convolution instances as the latter can be computed through a matrix multiplication operation. The higher runtime of the OA kernel designs is attributed to the element wise subtraction/minimum computation of the feature vector. But the runtime can be improved through the parallel computing techniques.

# Chapter 5

# Neighborhood Preserving Kernels for Attributed Graphs

Generally, the graph data has labels and attributes associated with each of its node and edge, where the label is usually a discrete character while an attribute is a vector. The main challenge in designing a kernel for attributed graphs is to define a similarity function that could capture all the characteristic properties of data by making use of the label as well as the attribute information along with the overall graph structure. By taking into account all these aspects, the description of the kernel we have designed is presented in this chapter.

Most of the existing graph kernels process only the label information in the nodes and edges ignoring the attributes.The kernel we propose use the node and edge labels as well as their attribute information for finding kernel values. For that, we formulated the reproducing kernel as the weighted sum of two kernels of which one is an R-convolution kernel that processes the attribute information and the other is an optimal assignment kernel that processes the label information. By incorporating the proposed kernels on support vector machines, we experimentally verified its performance using real-world data sets selected from the chemoinformatics domain.

The chapter is organized as follows. We define the concept of neighborhood preserving regions and the neighborhood preserving kernel based on edges in Section 5.1. The extension of these concepts to shortest paths are discussed in Section 5.2. Experiments and related discussion are in Section 5.3 and conclusions in Section 5.4.

## 5.1 Neighborhood Preserving Kernel

In this section, we discuss the design aspects of the neighborhood preserving kernel.

### 5.1.1 Neighborhood preserving regions

The Neighborhood Preserving (NP) kernels process the structurally similar regions of the argument graphs. In our design, the concept of structural similarity is defined using neighborhood preserving property, whose concept is given below.

To formalize the neighborhood preserving regions, 1-dimensional WL color refinement [10] is done on the graphs. A product graph [29] is then constructed to find the structural similar regions, where the product graph is defined as follows :

**Definition 5.1.** [**Direct product graph**]: Let $G = (V, E)$ and $G' = (V', E')$ be two graphs under consideration. The direct product graph, $G_P = G \times G'$, represented as $(V_P, E_P)$ is defined as

$$V_P = \{(u, u') \in V \times V' : l_C(u) = l_C(u')\}$$

$$E_P = \Big\{\big((u, u'), (v, v')\big) : (u, v) \in E \land (u', v') \in E' \land l((u, v)) \;=\; l((u', v'))\Big\}$$

An example is shown in the Figure 5.1(a),(b),(c) of two sample graphs, their WL coloring, and product graph formation. The set of neighborhood preserving or structurally similar edges of the two given graphs based on preserving its structure and node/edge labels information can be deducted from the product graph as per the definition given below.

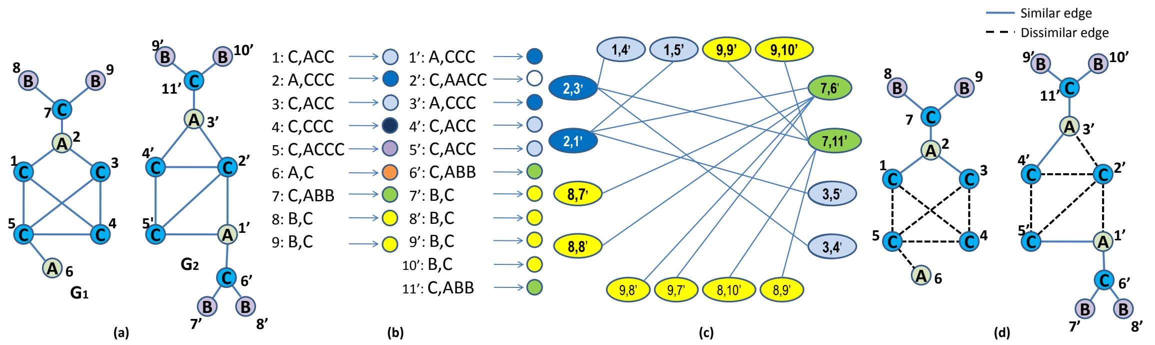


**Figure 5.1:** (a) Two sample graphs $G_1$ and $G_2$, (b) WL color refinement, (c) direct product graph, (d) separation of neighborhood preserving or structurally similar edges (bold lines) and dissimilar edges (dashed lines).

**Definition 5.2.** [**Neighborhood preserving edges**]:

The set of neighborhood preserving edges in graph $G$ with respect to $G'$, is

$$\mathcal{S}_{G:G'} = \{(u, v) \in E : ((u, u'), (v, v')) \in E_P\}$$

An example for the extraction of neighborhood preserving edges of the graphs is shown in Figure 5.1 (d). Note that the WL color refinement algorithm processes the information about the neighborhood of the nodes. Now, corresponding to each $(u, v) \in \mathcal{S}_{G:G'}$ there exists atleast one $(u', v') \in E'$ such that $u, u'$ as well as $v, v'$ has the same label and neighborhood information. Hence the edges are termed as *neighborhood preserving edges.* This helps to process kernel computation with subgraphs having well defined correspondences in terms of the neighborhood of the component nodes and it also gives a visualization of the structural similarity between the graphs.

We designed a kernel that defines the similarity of the argument graphs with the aid of neighborhood preserving edges, whose description is given below.

## 5.1.2 Neighborhood preserving edge kernel

Let $\kappa_V$ and $\kappa_E$ be positive semi-definite kernels defined on $V \times V'$ and $E \times E'$ respectively and $\lambda \in \mathbb{R}^+$ be a weight function. Then the neighborhood preserving edge kernel, $K_{NPE}$, is defined as

$$K_{NPE}(G, G') = \lambda(G, G') \sum_{(u,v) \in E} \sum_{(u',v') \in E'} \kappa_V(u, u').\kappa_E\big((u, v), (u', v')\big).\kappa_V(v, v') \quad (5.1)$$

where $\kappa_V$ is defined in terms of the attribute information of argument nodes and $\kappa_E$ is defined as

$$\begin{aligned}\kappa_E\big((u, v), (u', v')\big) = \delta\big(l(u, v), l(u', v')\big).\\ \delta(l_C(u), l_C(u')).k(\big((u, v), (u', v')\big)).\delta(l_C(v), l_C(v'))\end{aligned}$$

where $\lambda$ is a normalization constant, $\delta$ is the Kronecker delta function and $k$ is a valid kernel defined in terms of attributes on edges. If no attributes are present on edges, the value of $k$ can be taken as 1. Note that delta functions ensure that the edges processed for the kernel computation are from the neighborhood preserving edges only and hence the name for the kernel. The function of weight $\lambda$ is to give a normalizing effect to avoid artificially high similarity value if the argument graph sizes vary unevenly and if some WL refined labels in a graph is very high compared to its counterpart resulting in a large number of edge combinations of a particular type.

**Theorem 5.1.** *The neighborhood preserving edge kernel is a valid kernel.*

*Proof.* We prove $K_{NPE}$ as an R-convolution kernel [2]. Let $\Sigma_3$ be the set of strings of length three formed by members of $\Sigma_C$ and $\Sigma$, that is, $\Sigma_3 = \{xyz | x, z \in \Sigma_C, x \leq z, y \in \Sigma\}$ which satisfy a total order, $\leq$. Consider two graphs $G$ and $G'$ and $\Lambda \subseteq \Sigma_3$ where

$$\Lambda = \bigcup_{(u,v)\in E} l_C(u) \odot l(u,v) \odot l_C(v) \bigcap \bigcup_{(u',v')\in E'} l_C(u') \odot l(u',v') \odot l_C(v')$$

where $\odot$ is a concatenation operator, $l_C(u) \odot l(u,v) \odot l_C(v),\ l_C(u') \odot l(u',v') \odot l_C(v') \in \Sigma_3$. We call elements of $\Lambda$ as *WL refined edge address*.

Now we define a relation $R$ corresponding to each member $\epsilon \in \Lambda$. Let $R(e, G^\dagger, G; \epsilon)$ be a relation where $R(e, G^\dagger, G; \epsilon) = 1$ iff $G^\dagger$ is the graph obtained from $G$ if an edge $e$ corresponding to a member $\epsilon$ in $\Lambda$ is removed. Now, $R^{-1}(G; \epsilon) = \{(e, G^\dagger) : R(e, G^\dagger, G; \epsilon) = 1\}$ is the decomposition of a graph into an edge with *WL refined edge address* $\epsilon$ and rest of the graph. Now $K_{NPE}$ is defined as

$$K_{NPE}(G, G') = \sum_{\epsilon\in\Lambda} \frac{1}{|E_\epsilon \times E'_\epsilon|} \sum_{\substack{(e,G^\dagger)\in R^{-1}(G;\epsilon) \\ (e',G^{\dagger'})\in R^{-1}(G';\epsilon)}} k_{edge}(e, e') \times k_{triv}(G^\dagger, G^{\dagger'}) \tag{5.2}$$

where $E_\epsilon, E'_\epsilon$ are set of edges in $G, G'$ corresponding to *WL refined edge address* $\epsilon$, $E_\epsilon \times E'_\epsilon$ being their cartesian product, $k_{triv}$ is a trivial kernel whose value is always 1 and

$$k_{edge}(e, e') = \kappa_V(u, u') \times k(e, e') \times \kappa_V(v, v') \tag{5.3}$$

where $e = (u, v)$ and $e' = (u', v')$. Now $k_{edge}$ is a valid kernel [103]. Also, $|E_\epsilon \times E'_\epsilon|$ is a well-defined function on $G \times G'$ and hence $K_{NPE}$ is a R convolution kernel. □

The effect of the cardinality of $\Lambda$ has to be considered and hence included the weight $\frac{1}{|E_\epsilon \times E'_\epsilon|}$ in (5.2). The $\lambda$ in (5.1) can be considered as the counterpart of this term.

By constructing a product graph, the information used in $K_{NPE}$ kernel can be extracted as explained below.

**Theorem 5.2.** *Each edge in the direct product graph corresponds to a product calculation in convolution operation defined in* (5.2) *except for few edges of the form* $\big((u, u'), (v, v')\big) \in E_P$ *where* $l_C(u) = l_C(v) = l_C(u') = l_C(v')$ *and* $l(u, v) = l(u', v')$.

*Proof.* Consider a particular edge $(u, v) \in E$ in a graph $G$ which is neighborhood preserving. Note that the nodes in the direct product graph contain all combination of nodes from graphs $G$ and $G'$ whose label and neighborhood information is identical. Hence the

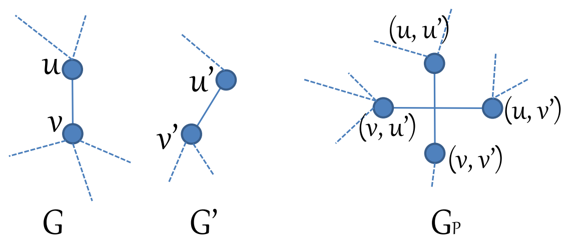


**Figure 5.2:** Two nodes $u, v$ in the graph $G$ and two nodes $u', v'$ in the graph $G'$ with the same WL refined labels and corresponding edges having similar label. The product graph $G_P$ containing additional pair of nodes and an additional edge.

edge $(u, v)$ is a part of a set of edges of the form $\big((u, u'), (v, v')\big)$ in direct product graph where $u' \in \{x \in V' : l_C(x) = l_C(u)\}$, $v' \in \{y \in V' : l_C(y) = l_C(v)\}$, $(u', v') \in E'$ and $l(u, v) = l(u', v')$. Hence these set of edges correspond to a convolution of edge $(u, v) \in E$ with its counterpart edges in $E'$. Similarly, we can find out other convolutions corresponding to every edge in $G$.

But a duplication arises in a special case when $l_C(u) = l_C(u') = l_C(v) = l_C(v')$ and $l(u, v) = l(u', v')$. Consider such a case as shown in the Figure 5.2. We can see that in the direct product graph, $G_P$, 4 nodes are created with two edges. But as per convolution operation the edge pair $(u, v)$ and $(u', v')$ shall be counted only once. Hence avoiding this duplication, $K_{NPE}$ can be calculated from $G_P$.

$\square$

Note that the ordering of the nodes in the case of the duplication can be resolved through a lexicographic ordering among the graph nodes.

The NPE kernel can be found out for multiple iterations of WL color refinement. In this case, the final kernel is taken as the sum of kernels in individual iteration.

The neighborhood preserving edge kernel processes node and edge attributes with the labels on them acting as a guidance for the R-convolution process. For processing the node and edge labels alone, another kernel named neighborhood preserving optimal edge assignment kernel is formulated, whose description is given in the next section. The objective is to analyze the importance of the label information.

## 5.1.3 Neighborhood preserving optimal edge assignment kernel

The neighborhood preserving optimal edge assignment kernel $K_{NPO}$ processes only node labels and defined as

$$K_{NPO}(G, G') = \sum_{\epsilon \in \Lambda} \min(|E_\epsilon|, |E'_\epsilon|) \tag{5.4}$$

The minimum value is taken as it represents a normalized score of structural similarity. Note that the edges processed for $K_{NPO}$ computation is only neighborhood preserving ones. Note that $K_{NPO}$ can also be defined for multiple WL color refinement steps where the kernel value in the individual iteration are summed up.

With the aid of an optimal assignment kernel $K_B(E, E')$, we prove $K_{NPO}(G, G')$ as a valid kernel. We define $K_B(E, E')$ as,

$$K_B(E, E') = \sum_{i=1}^{h} \max_{\beta^i \in B(E,E')} \sum_{\left((u,v), \beta^i(u,v)\right)} k_b^i\big((u,v), \beta^i(u,v)\big) \tag{5.5}$$

where $B(E, E')$ is the set of all bijections between the members of the set $E$ and $E'$ corresponding to the base kernel $k_b^i$ at iteration:$i$ of WL color refinement, $(u, v) \in E$, $\beta^i(u, v) \in E'$ and, $\beta^i$ is assumed to be that bijection in $B$ that maximizes the kernel value in iteration:$i$. Here we assume that the sets among $E$ and $E'$ that contain the least cardinality have enough dummy members to make up the difference between the cardinalities and WL iteration is done $h$ times.

The base kernel $k_b^i$ is defined as,

$$k_b^i\big((u,v), (u',v')\big) = \begin{cases} 1, \text{ if } l_C^i(u) = l_C^i(u') \land l_C^i(v) = l_C^i(v') \\ \quad \land\, l\big((u,v)\big) = l\big((u',v')\big) \\ 0, \text{ otherwise} \end{cases} \tag{5.6}$$

where $l_C^i(.)$ indicates the WL color of the concerned node at iteration:$i$ of the WL color refinement. The base kernel defined above is strong as the cardinality of its range set is two. [5].

The hierarchy tree corresponding to base kernel $k_b$ can be constructed in the following way. Consider the application of WL color refinement algorithm on the given graphs. Corresponding to each *WL refined edge address* obtained in the first iteration, a root node is created. The *WL refined edge addresses* obtained for iteration $i > 1$ form the nodes for the level $i$ of the hierarchy tree. As the nodes in the higher level are related to the nodes in the lower level, a parent-child relationship between two subsequent levels can be established. Thus a tree structure $T$ can be built by adding edge between parents and children. Note that $T$ is constructed for the entire graphs in the training data considering the whole WL iterations.

**Lemma 5.1.** *$K_B(E, E')$ can be calculated from the hierarchy.*

*Proof.* With the help of the hierarchy $T$, $K_B$ can be calculated as a histogram intersection kernel [5] as follows. Corresponding to each graph $G$, a histogram vector $G_V$ of length $V_T$ is created where $V_T$ is the set of nodes of the tree. The $i^{th}$ element of $G_V$ is the number of times it produces the $i^{th}$ element of $V_T$ during the WL color refinement algorithm procedure. The histogram vector calculation is explained in Algorithm 5.1.

**Algorithm 5.1:** Computation of histogram vector $G_V$

```
Input  : The hierarchy $T$ for $h$ iterations of WL color refinement algorithm and a
         graph $G = (V, E)$.
Output: Histogram vector $G_V$
1  Assign a unique location starting from 1 to $|V_T|$ to all the nodes in $T$
2  Initialize $G_V$ as $|V_T| \times 1$ vector
3  for every edge $e$ in $E$ do
4      Find the leaf node $n$ in $T$ corresponding to WL refined edge addresses of edge
         $e$ at WL iteration: $h$
5      $i = h$
6      while $i \geq 1$ do
7          $G_V(\text{location}(n)) = G_V(\text{location}(n)) + 1$
8          $n$ = parent($n$)
9          $i = i - 1$
10     end
11 end
```

Note that the above algorithm corresponds to a tree traversal for each edge in a graph starting from the leaf nodes. Since the hierarchy contains all possible occurrences of *WL refined edge addresses* in $h$ iterations of WL color refinement, each visit of a node in $T$ at level $i$ with respect to the edge under traversal corresponds to an occurrence of that particular *WL refined edge address* characterizing the node in $T$ at iteration:$i$. Hence the tree traversals as per the algorithm give the frequency of occurrences of *WL refined edge addresses* in the entire $h$ number of WL iterations, that is, $G_V$. Let $G_V$ and $G'_V$ be the histogram vectors of graphs $G$ and $G'$. Then $K_B(E, E')$ in (5.5) can be calculated as histogram intersection kernel of $G_V$ and $G'_V$ [5], that is,

$$K_B(E, E') = \sum_{i=1}^{|G_V|} \text{minimum}\ (G_V(i), G'_V(i)) \tag{5.7}$$

It is straightforward now to establish that $K_B(E, E')$ for $h$ iterations of WL color refinement steps is the summation of $K_{NPO}$ kernels in individual refinement steps. That is,

$$K_B(E, E') = \sum_{i=1}^{h} K^i_{NPO}(G, G')$$

where $K^i_{NPO}$ is the NPO kernel at iteration $i$. $\square$

Based on the above discussion, it is clear that neighborhood preserving optimal edge assignment kernel is a valid optimal assignment kernel as the base kernel is strong and it is induced by the hierarchy $T$.

It has to be noted that $K_{NPO}$ in actual experiments is calculated as explained in Section 5.1.8.

### 5.1.4 Neighborhood preserving kernel definition

The neighborhood preserving (NP) kernel for $h$ iterations of WL color refinement algorithm is defined as

$$K(G, G') = \alpha \sum_{i=1}^{h} K^i_{NPE}(G, G') + (1 - \alpha) \sum_{i=1}^{h} K^i_{NPO}(G, G') \tag{5.8}$$

where $\alpha \in (0, 1)$ is a tuning parameter, $K^i_{NPE}$ is the NPE kernel and $K^i_{NPE}$ is the NPO kernel defined for $i^{th}$ iteration. The purpose of $\alpha$ is to have a trade-off between the two components of NP kernel.

### 5.1.5 Other neighborhood preserving kernels

It can be seen that to ensure a graph kernel to process only on the neighborhood preserving regions, it is enough to take features from the edges that have common *WL refined edge address* in the graphs. The application of WL refinement iteration on WL-edge kernel [25] results in the process of the neighborhood preserving regions. However, this kernel does not process attribute information. Hence the neighborhood preserving edge (NPE) kernel can be considered as a generalization of WL-edge kernel.

In the case of the WL-subtree kernel and WL-shortest path kernel [25], the regions processed may include non-neighborhood preserving edges as well. However, the WL-shortest path kernel can be made to a neighborhood preserving kernel by imposing an additional constraint such that the shortest paths considered for feature extraction shall contain only neighborhood preserving edges.

It is evident that the walks in the formulated product graph corresponds to neighborhood preserving walks in the argument graphs. Hence random walk kernel [34] can be made neighborhood preserving. It is possible to distinguish the shortest paths from the neighborhood preserving walks. Hence it is possible to make GraphHopper kernel neighborhood preserving and WL-shortest path kernel [25] can be generalized to attributed graphs in the same way as the proposed NPE kernel generalizes the WL edge kernel.

If we insert $d$-edges in the direct product graph definition, as explained in [8], the cliques correspond to neighborhood preserving subgraph isomorphism and hence subgraph matching kernel [8] can be made neighborhood preserving.

### 5.1.6 Recursive computation of NPE kernel from product graph

We prove with the following theorem that the neighborhood preserving edge kernel can be computed at each WL iteration from the product graph defined for the previous iteration which is obtained in a recursive fashion, i.e, the initial product graph formulation alone is enough to find out the product graphs in proceeding WL iterations without explicitly finding them.

**Theorem 5.3.** *Product graph at any iteration >1 of WL color refinement algorithm is the subgraph of the product graph defined for the previous iteration.*

*Proof.* Product graph ($G_{P_i}$) in iteration : $i > 1$ can be obtained from product graph ($G_{P_{i-1}}$) in iteration: $(i-1)$ by deleting certain edges. Suppose $G_{P_i}$ is the product graph obtained by deleting the whole edges of the form $((u, v), (u', v'))$ in $G_{P_{i-1}}$ where $l_{C_i}(u) \neq l_{C_i}(u')$ and $l_{C_i}(v) \neq l_{C_i}(v')$ where $l_{C_i}$ is the WL alphabet in iteration : $i$. We argue that the edges that can occur in $G_{P_i}$ is already embedded in $G_{P_{i-1}}$ except for the deleted edges based on the above rule.

Suppose there exists an edge $((y, y'), (z, z'))$ in $G_{P_i}$ that does not exist in $G_{P_{i-1}}$. That is label and neighborhood of $y, y'$ and $z, z'$ with respect to WL labels $l_{C_{i-1}}$ at the iteration $h = i - 1$ are identical. If their neighborhood are identical at the iteration $h = i - 1$, they would have formed nodes in $G_{P_{i-1}}$ and so the edge between them. Hence the edge $((y, y'), (z, z'))$ in $G_{P_i}$ is already embedded in $G_{P_{i-1}}$. Hence $G_{P_i}$ formed with the above edge deletion rule is the subgraph of $G_{P_{i-1}}$.

$\square$

### 5.1.7 Computational complexity analysis

With efficient sorting techniques, WL color refining for one iteration can be done in $\mathcal{O}(m)$ operations, where $m$ is the number of edges. NPE kernel defined in 5.1 requires $\mathcal{O}(m^2)$ operations. Considering $d$ be the dimension of attributes, the base kernel computation is of $\mathcal{O}(d)$. Hence NPE kernel is computed in $\mathcal{O}\big(h \times N^2(m^2 \times d)\big)$ where $N$ is the dataset size and $h$ is the number of WL iterations.

The computational complexity of $K_{NPO}$ is $\mathcal{O}(N^2 \times |\Sigma_3|^3)$, where $|\Sigma_3|^3$ is the maximum possible size of $\Lambda$ bounded by $m^2$ for a pair of graphs. This makes the overall computational complexity of NP kernel to be $\mathcal{O}\big(h \times N^2(m^2 \times d)\big)$.

### 5.1.8 Algorithms for computing NP kernel

We can compute $K_{NPE}$ and hence $K(G, G')$ in two ways.

1. In a pairwise manner with product graph.

Although kernel computation with product graph using recursion property described in Theorem 5.3 can have $n^2$ nodes in the worst case where $n$ is the number of nodes, the number of edges are bounded by $c \times m^2$ where $c$ is a factor that accounts for the edges that result in duplication as explained in Theorem 5.2. The computation steps are detailed in Algorithm 5.2.

2. In a global manner.

We can create a list of *WL refined edge addresses* derived from $\Lambda \subseteq \Sigma_3$ that occurs across the training data and a pre-computed feature information data can be formed for each graph listing the edges corresponding to each *WL refined edge address*. Then the kernel can be computed by iterating through this pre-computed feature information data. This method by default provides the provision to calculate $K_{NPO}$ because it only requires to process the cardinality of individual refined addresses in the feature information data. The computation steps are detailed in Algorithm 5.3.

In the Algorithm 5.3, the convolution operation is done by vectorization rather than the brute force approach of running for loops as required in computing 5.2. As part of the vectorization, the vectors to be processed are stacked one after the other corresponding to

**Algorithm 5.2:** Pairwise computation

```
Input  : The graph dataset $\mathcal{G}$, $h$
Output: The neighborhood preserving edge kernel matrix $K_{NPE}$ and optimal edge
        assignment kernel matrix $K_{NPO}$
1  Do WL color refinement on the graphs $h$ times and store in memory
2  for every graph, $G_i = (V_i, E_i)$ in $|\mathcal{G}|$ do
3      Initialize $\Lambda_i(key, value)$ as empty dictionary
4      for every edge $e = (v_p, v_q)$ in $E_i$ do
5          Find WL refined edge address, $\epsilon$
6          if $\epsilon \notin \Lambda_i$ then
7              Add $\epsilon$ as a key in $\Lambda_i$
8              Initialize the value of the key: $\epsilon$, as an empty list
9              Append edge $e$ to the list
10         else
11             Append edge $e$ to the list corresponding to the key:$\epsilon$
12         end
13     end
14     Save $\Lambda_i$
15 end
16 for every graph, $G_i = (V_i, E_i)$ in $|\mathcal{G}|$ do
17     for every graph, $G_j = (V_j, E_j)$ in $|\mathcal{G}|$ do
18         if $i <= j$ then
19             Find the NP edges $E_P = \{(u, u'), (v, v')\}$from $G_P$ for $h = 1$ using $\Lambda_i$
                and $\Lambda_j$
20             if NP edges exist then
21                 Find $K_{NPE}(i, j)$ and $K_{NPO}(i, j)$ for $h = 1$ using $E_P$.
22                 for $k$ = 2 to $h$ do
23                     Delete the edges in $E_P$ that does not satisfy NP property at WL
                        iteration: $k$
24                     Find $K^k_{NPE}(i, j)$ and $K^k_{NPO}(i, j)$ at iteration: $k$
25                     $K_{NPE}(i, j) = K_{NPE}(i, j) + K^k_{NPE}(i, j)$
26                     $K_{NPO}(i, j) = K_{NPO}(i, j) + K^k_{NPO}(i, j)$
27                 end
28             else
29                 $K_{NPE}(i, j)$ = 0
30                 $K_{NPO}(i, j)$ = 0
31             end
32         end
33     end
34 end
```

**Algorithm 5.3:** Global computation for one iteration of WL color refinement

```
Input  : The graph dataset $\mathcal{G}$ and $\Sigma_C$
Output: Kernel matrices $K_{NPE}$ and $K_{NPO}$
1  for every graph, $G_i = (V_i, E_i)$ in $|\mathcal{G}|$ do
2      Initialize $\Lambda_i(key, value)$ as empty dictionary
3      for every edge $e = (v_p, v_q)$ in $E_i$ do
4          Find WL refined edge address, $\epsilon$
5          if $\epsilon \notin \Lambda_i$ then
6              Add $\epsilon$ as a key in $\Lambda_i$
7              Initialize the value of the key: $\epsilon$, as an empty list
8              Append edge $e$ to the list
9          else
10             Append edge $e$ to the list corresponding to the key:$\epsilon$
11         end
12     end
13     Save $\Lambda_i$
14 end
15 Initialize $K_{NPE}$ and $K_{NPO}$ as kernel matrices
16 for every graph, $G_i = (V_i, E_i)$ in $|\mathcal{G}|$ do
17     for every graph, $G_j = (V_j, E_j)$ in $|\mathcal{G}|$ do
18         if $i <= j$ then
19             Find the common keys in $\Lambda_i$ and $\Lambda_j$
20             if common keys exist then
21                 Initialize the empty array $f^{E_i}_{end_1}$ and $f^{E_i}_{end_2}$ for storing the vectors
                     corresponding the pair of nodes of edges in $E_i$ for $G_i$
22                 Initialize empty matrix $a^{E_i}$ for storing vector corresponding to the
                     edges (if any) for $G_i$
23                 Initialize the empty matrices $f^{E_j}_{end_1}$, $f^{E_j}_{end_2}$, and $a^{E_j}$ for $G_j$
24                 for every common key $\epsilon$ do
25                     Find the lists of edges corresponding to $\epsilon$ in $\Lambda_i$ and $\Lambda_j$
26                     Using the lists, populate $f^{E_i}_{end_1}$, $f^{E_i}_{end_2}$, $a^{E_i}$, $f^{E_j}_{end_1}$, $f^{E_j}_{end_2}$, and $a^{E_j}$
                         with the corresponding attribute vectors
27                 end
28                 Find $K_{NPE}(i, j)$ and $K_{NPO}(i, j)$ by processing $f^{E_i}_{end_1}$, $f^{E_i}_{end_2}$, $a^{E_i}$,
                     $f^{E_j}_{end_1}$, $f^{E_j}_{end_2}$, and $a^{E_j}$
29             else
30                 $K_{NPE}(i, j) = 0$, $K_{NPO}(i, j) = 0$
31             end
32             $K_{NPE}(j, i) = K_{NPE}(i, j)$, $K_{NPO}(j, i) = K_{NPO}(i, j)$
33         end
34     end
35 end
```

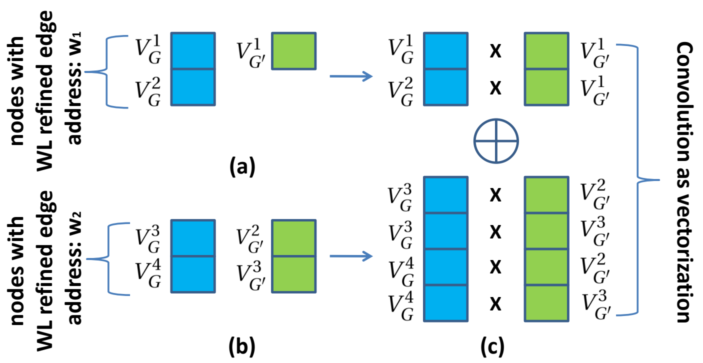

nodes with WL refined edge address: $w_1$
nodes with WL refined edge address: $w_2$
$V_G^1$
$V_G^2$
$V_{G'}^1$
(a)
$V_G^3$
$V_G^4$
$V_{G'}^2$
$V_{G'}^3$
(b)
(c)
Convolution as vectorization

**Figure 5.3:** Convolution as vectorization for a sample of two *WL refined edge addresses* $w_1$ and $w_2$. (a) and (b) represents node attributes in $G, G'$ with an address $w_1$ and $w_2$ respectively. (c) represents arranging attribute vectors to compute convolution as a vectorization process, $\oplus$ denotes concatenation and $\times$ denotes Hadamard product.

the common *WL refined edge addresses* or common keys (lines 25 to 28 of algorithm 5.3). Then the vectors are duplicated in an appropriate form necessary to perform the convolution in the form of matrix Hadamard product and then row wise summation later. An example is given in Figure 5.3.

## 5.2 Neighborhood preserving shortest path kernel

It is straightforward to extend the concepts of neighborhood preserving edge kernel defined in Section 5.1.2 to the shortest paths. One disadvantage with NPE kernel is that the information that gets processed is in terms of edges only, even though the neighborhood preserving regions may be connected. Hence if we process larger subgraphs, the kernel measure is much more accurate. The shortest paths are a good candidate for analyzing these larger connected regions.

Let $\kappa_V$ be a positive semi-definite (psd) kernel defined on $V \times V'$. We assume $P, P'$ as the sets that contain the shortest paths in graphs $G, G'$ respectively. Then the neighborhood preserving shortest path kernel, $(K_{NPS})$, is defined as,

$$K_{NPS}(G,G') = \sum_{\Pi(u_1,u_n)\in P} \; \sum_{\Pi(u'_1,u'_n)\in P'} \kappa_V(u_1,u'_1)\times k_\delta\big(\Pi(u_1,u_n),\Pi(u'_1,u'_n)\big)\times\kappa_V(u_n,u'_n) \tag{5.9}$$

$k_\delta$ can be defined as a psd kernel as follows.

$$k_\delta\big(\Pi(u_1,u_n),\Pi(u'_1,u'_n)\big) = \begin{cases} 1, \text{ if } |\Pi(u_1,u_n)| = |\Pi(u'_1,u'_n)| \wedge \\ \quad \sum_{i=1}^{n} \delta\big(l_C(u_i), l_C(u'_i)\big) = n \wedge \\ \quad \sum_{i=1}^{n-1} \delta\big(l(u_i,u_{i+1}), l(u'_i,u'_{i+1})\big) = n-1 \\ 0, \text{ otherwise} \end{cases} \tag{5.10}$$

In the above definition, edges involved in the shortest paths are compared for neighborhood preserving property. Note that for kernel computation, only attributes in source and sink nodes are considered. We can modify this to have kernel value computation of attributes of the nodes in between as well.

The above kernel can be proved as a psd kernel in the same way as the NPE kernel. We can formulate the R-convolution relation as a decomposition of the shortest path and the rest of the graph. For each shortest path, we can assign an address like we assign an edge with a *WL refined edge address*. The address can be a string where WL labels of the nodes and edge labels involved in the concerned shortest path are concatenated. With this setting, as in the case of the NPE kernel in Section 5.1.2 we can define an R-convolution kernel corresponding to these addresses. Note that if these addresses take the form of a palindrome, the matching between sink and source nodes can happen in either way. In such a case, matching can be resolved based on the lexicographic ordering of the nodes.

### 5.2.1 Computational complexity analysis

The complexity associated with finding the shortest paths for a single graph is of $\mathcal{O}(V^2)$. For kernel computation, the shortest paths have to be compared against each other which is of $\mathcal{O}(V^2)$ and along with each comparison, the base kernel has to be computed twice for source and sink nodes. If the attributes are of dimension $d$, this makes the kernel computation $\mathcal{O}(V^2 \times d)$. So the overall computational complexity for a set of $N$ graphs as $\mathcal{O}(NV^2 + hN^2V^2d)$.

To improve the computational complexity, it is possible to relax the conditions for neighborhood preserving property of the shortest paths. A simple way is to relax the condition in (5.10). For example, the number of neighborhood preserving edges in the paths can be limited to a predefined threshold. This prevents the requirement to check labels of each and every nodes/edges in the paths. Now it is possible to use computationally efficient shortest path kernel computation algorithm such as the one described in [36]. This will help to reduce the complexity approximately to $\mathcal{O}(N^2Vd)$.

## 5.3 Experiments

The efficiency of the proposed neighborhood preserving kernels was analyzed by subjecting them to real-world data sets and compared its performance with state-of-the-art graph kernels namely: Shortest path(SP) kernel [7], GraphHopper(GH) kernel [36], RetGK kernels [35], Graph invariant kernel (GIK) [46], Propagation kernels [58] and Hash graph kernels [59].

The components of the proposed NP kernel is analyzed separately, i.e, when $\alpha = 1$ in 5.8, it corresponds to NPE kernel component alone that process the attributes and when $\alpha = 0$ in 5.8, it corresponds to the NPO kernel component alone that processes the labels. This helps in evaluating the role of attributes and labels separately and also the effectiveness of NP kernel where both information is utilized.

**Table 5.1:** Classification accuracy of the proposed kernels with state-of-the-arts.

| Kernel | PROTEINS | ENZYMES | BZR | COX2 | DHFR | SYN.new |
|---|---|---|---|---|---|---|
| SP | $73.42 \pm 1.11$ | $\mathbf{66.58 \pm 4.06}$ | $85.76 \pm 2.35$ | $79.88 \pm 1.73$ | $79.52 \pm 2.40$ | $86.72 \pm 3.68$ |
| GH | $73.19 \pm 1.79$ | $66.33 \pm 2.78$ | $82.90 \pm 2.73$ | $79.66 \pm 1.17$ | $76.65 \pm 3.21$ | $88.81 \pm 3.41$ |
| RetGK-I | $\mathbf{75.94 \pm 1.79}$ | $65.67 \pm 3.07$ | $85.58 \pm 2.20$ | $78.19 \pm 0.47$ | $80.83 \pm 2.10$ | $97.08 \pm 1.54$ |
| RetGk-II | $74.69 \pm 1.60$ | $62.34 \pm 2.94$ | $85.76 \pm 1.61$ | $78.25 \pm 0.35$ | $81.66 \pm 2.41$ | $96.94 \pm 1.47$ |
| GIK | $72.36 \pm 2.17$ | $52.32 \pm 3.89$ | $86.31 \pm 2.25$ | $79.68 \pm 2.41$ | $81.25 \pm 2.56$ | $91.73 \pm 2.22$ |
| Prop-diff | $72.60 \pm 2.32$ | $38.19 \pm 2.27$ | $78.46 \pm 0.59$ | $77.94 \pm 0.47$ | $72.58 \pm 0.78$ | $48.59 \pm 1.17$ |
| Prop-WL | $74.23 \pm 1.80$ | $44.85 \pm 1.63$ | $78.92 \pm 0.41$ | $78.25 \pm 0.35$ | $73.41 \pm 0.53$ | $46.38 \pm 1.39$ |
| HGK-WL | $74.69 \pm 1.98$ | $64.30 \pm 3.37$ | $81.48 \pm 1.84$ | $78.50 \pm 0.57$ | $75.47 \pm 2.40$ | $81.00 \pm 3.69$ |
| HGK-SP | $75.57 \pm 1.89$ | $62.60 \pm 3.00$ | $82.38 \pm 1.79$ | $78.55 \pm 0.65$ | $76.61 \pm 2.72$ | $96.38 \pm 1.92$ |
| $\text{NPE}_{(\alpha=1)}$ | $71.08 \pm 2.80$ | $64.93 \pm 2.97$ | $87.13 \pm 1.98$ | $82.36 \pm 2.02$ | $82.70 \pm 2.35$ | $99.51 \pm 0.67$ |
| $\text{NPO}_{(\alpha=0)}$ | $72.74 \pm 2.02$ | $45.67 \pm 3.24$ | $88.81 \pm 1.70$ | $80.71 \pm 2.92$ | $81.07 \pm 2.89$ | $97.81 \pm 1.53$ |
| NP | $73.66 \pm 2.55$ | $58.10 \pm 3.16$ | $\mathbf{89.12 \pm 2.03}$ | $\mathbf{81.16 \pm 2.13}$ | $\mathbf{83.98 \pm 2.35}$ | $\mathbf{99.74 \pm 0.64}$ |
| NPS | $73.87 \pm 2.47$ | $58.94 \pm 3.58$ | $\mathbf{89.24 \pm 2.18}$ | $\mathbf{81.37 \pm 2.26}$ | $\mathbf{84.16 \pm 2.43}$ | $\mathbf{99.85 \pm 0.71}$ |

### 5.3.1 Datasets

The classification datasets used for analysis were ENZYMES, PROTEINS, BZR, COX2, DHFR, and SYNTHETICnew. The description of ENZYMES and PROTEINS dataset are given in Section 3.5.2.1. In contrast to two previous chapters, their attribute information are also considered for the experiments. BZR, COX2 and DHFR [104] are taken from the

Graph data repository [105]. SYNTHETICnew is a synthetic dataset introduced in [36]. The datasets are all of binary class except for ENZYMES which has six classes.

### 5.3.2 Experimental setup

The validation process was carried out in the following way. Using hold out technique, 70% of the data points were assigned for training and the remaining for testing. The 10 fold cross-validation was done on training data for selecting the hyperparameters. A model was then built using the entire training data and its performance was tested on the testing data. The above process was repeated 30 times and the results reported were averaged over these 30 iterations to nullify the effects of fold assignments.

The classification algorithm used was SVM (with Libsvm implementation [97]). The penalty parameter $C$ of SVM was searched in the interval $[2^{-7}, 2^{15}]$. The performance parameter used was accuracy. The number of iterations of WL color refinement algorithm was from $\{1, 2, 3\}$ which is fixed through cross-validation. The experiments were done in a machine with Intel Xeon i5 2.4 GHz CPU with 80 GB RAM.

We wrote the code for SP and GIK kernel while GH, RetGK kernels and Hash graph kernel were done with the codes published by authors. Propagation kernel implementation and (its hyper-parameter selection) was done by the codes published by authors of GH kernel. The linear and Gaussian kernel $\left(k(x, y) = e^{-\beta\|x-y\|^2}\right)$ where $\beta = 1/d$, ($d$ the dimension of attribute information) were used as the base kernels within the state-of-the-arts. The coding of the proposed kernels and Hash graph kernel is done in Python while others in Matlab.

For GIK kernels, WL coloring was taken as vertex invariant. Two variants of the Propagation kernel were implemented. In the 'Prop-diff' variant, the propagation scheme used is diffusion [58]. In the 'Prop-WL' variant, labels of the nodes are first hashed and a WL propagation scheme is used. Total variance distance was used as the hashing function in both the propagation schemes. The bin width of the hash function was set to $10^{-5}$ and the number of propagation steps for both variants was fixed through cross-validation from the set $\{1, 2, 3, 4, 5\}$. RetGK kernels used also have two variants. RetGK-I is the one with an explicit feature map in RKHS and RetGK-II is the one with approximated mapping. For both approaches, 50 steps of random walks are assumed. For Hash graph kernels, WL subtree (HGK-WL) and shortest path (HGK-SP) kernels [25] were employed as the base kernels, and the hashing function used is 2-stable Locality sensitive hashing (LSH) with bin width 1, label and hashed attribute information were propagated separately as suggested by the authors. Number of WL refinement steps were fixed through cross-validation from the set $\{1, 2, 3, 4, 5\}$.

For the SYNTHETICnew dataset, the node labels were given identical label discarding the original continuous type values and attributes are used as such. For the proposed kernels and GH kernel, the result reported is the best among the case between Linear and Gaussian kernels where they are employed as base kernels.

### 5.3.3 Runtime experiments with pairwise and global computation of NP kernel

For the algorithms explained in Section 5.1.8, an experiment is done to evaluate the pairwise and global computation schemes in calculating the NP kernel [1]. For this experiment, 4 artificial datasets of 100 graphs with 300 nodes and graph density 20%, 40%, 60% and 80% respectively were created. Each dataset has experimented 3 times with the nodes being selecting a random label out of an alphabet $\Sigma$ of size 1, 2, and 3 respectively and edge labels are assumed to be identical. The node attributes are assumed to be in a dimension of 30. The experiment is done for 3 steps of WL color refinement where calculation for pairwise computation is done as per Theorem 5.2 and Theorem 5.3. The runtime (in seconds) for both approaches are plotted against the size of WL alphabet $\Sigma_C$ and size of $\Lambda$ for the label size, $|\Sigma|$ =1,2, and 3 respectively in Figure 5.4. The variation in graph density is studied since it is the number of edges that affects the computation time as explained in Section 6.1.2.1.

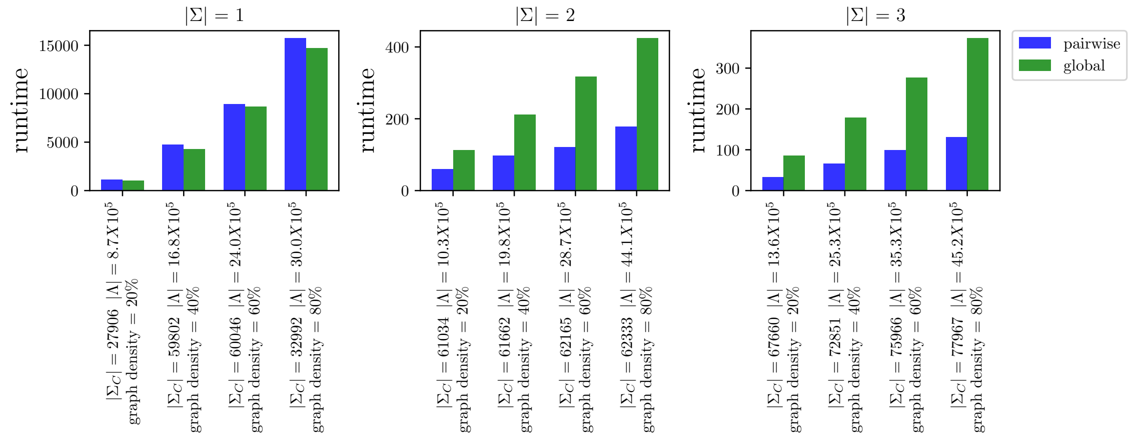


**Figure 5.4:** Runtime comparison of pairwise and global computation of NP kernel for synthetic graphs at graph density 20%, 40%, 60%, and 80% respectively.

[1]Implementations in : https://github.com/asif-salim/NP-graph-kernels

It can be seen from Figure 5.4 that when $|\Sigma|$ is small, global computation takes less amount of time. In this case, more nodes are sharing common WL labels and this results in smaller $|\Lambda|$. But the product graph involves lots of nodes and hence larger computation time for pairwise computation scheme. But as $|\Sigma|$ becomes larger, pairwise computation time is much better compared to the global. In this case, the nodes sharing common WL labels are relatively less and it will result in a larger $|\Lambda|$. In comparison with the smaller number of nodes in the product graph, global computation of these larger $\Lambda$ feature information takes more time than pairwise computation. These effects are more evident as graph density increases. Hence pairwise computation is suitable for dense graphs with a large $|\Lambda|$ or $|\Lambda_C|$.

### 5.3.4 Results and discussion

The accuracy with standard deviation obtained are tabulated in Table 5.1 and runtime (wall-clock time) in Table 5.3. We did experiments with the proposed kernels with NPE kernel alone ($\alpha$ = 1) and NPO kernel alone ($\alpha$ = 0) as well as with the formal NP kernel definition, NPS kernel and compared the results with that of state-of-the-arts. The best results are given in bold letters.

The NP and NPS kernels have a significant improvement over state-of-the-arts in the case of BZR, COX2, DHFR and SYNTHETICnew datasets and their performance is reasonably good in the case of PROTEINS and ENZYMES datasets. Note that the NPE kernel which processes attribute information and the NPO kernel which processes only labels where $\alpha$ tuning is not required performs better than state-of-the-art in BZR, COX2, and SYNTHETICnew datasets. In the case of DFHR dataset, NPE kernel outperforms the state of the arts whereas the performance of NPO kernel is in par with them. For datasets except for ENZYMES and COX2, NP kernel augmented with NPO kernel performs significantly better than the NPE kernel. This validates our argument about the need for processing the label and attribute information independently. This also gives evidence to the fact that in the case of graph data analysis, the attributes cannot be neglected if they are present. This is important since most of the kernels developed in this regard could only process labels. In comparison with the performance of the proposed kernels with the discretization algorithms (Propagation and HGK kernels), the proposed designs perform better although the runtime of discretization-based approaches are low.

The runtime of NP and NPS kernels is reported for the global computation approach with the linear kernel being the base kernel. An exception is for SYNTHETICnew whose

**Table 5.2:** Classification accuracy in MNIST dataset

| | CNN | NPE |
|---|---|---|
| Accuracy | 95.57 | 94.46 |

NP kernel is calculated with the pairwise approach. The runtime performance is better than most of the state-of-the-arts. Considering the runtime, propagation kernels are the fastest. But they use discretized attribute information and hence their performance is lower than the state-of-the-arts. Although the runtime of RetGK kernels is better, NP kernel processes the label and the attribute information independently which makes the performance of those better. Since the NP and NPS kernel establish a well-defined correspondence between subgraphs, their performance is better than GraphHopper kernel with better runtime. In comparison with Shortest path kernel and Graph invariant kernels, the runtime of NP kernel is better. The performance of GIK kernel is close to that of NP kernel while performance of NP kernels compared to Shortest path kernel is better in the datasets, an exception being ENZYMES.

Note that the runtime of NPS kernel can be improvised with the efficient computation approaches like the strategies introduced in [36]. The difference in the runtime of NPE and NP kernels are negligible. The reason is that when we calculate the NPE kernel, the steps involved by default provides a way to calculate NPO kernel as well with the help of dictionaries $\Lambda$'s defined in Algorithm 5.3. To be specific, NPO kernel can be calculated by taking the minimum of the cardinality of the lists of edges corresponding to common keys in $\Lambda$. But once we are only concerned with the computation of NPO kernel, the runtime is significantly reduced since the processing of attributes are not required.

### 5.3.5 A heuristic to choose value of $h$

If we consider a pair of graphs, it is highly likely that some nodes may get a different color in WL iterations from any other nodes in both graphs and hence they do not contribute any neighborhood preserving (NP) edges for the kernel value. As the WL iteration proceeds, the number of NP edges are going to decrease because as more hops are taken in the graphs with respect to nodes, the likelihood of similarity decreases. Hence at a particular iteration the NP edges vanish for all pair of distinct graphs. Any further iterations are not going to contribute to the kernel value but they create computation overheads.

One heuristic to carefully choose the value of $h$ is to track the cardinality of the unique *WL refined edge addresses* that occur across every graph and the pair of graph whose kernel

**Table 5.3:** Runtime of the proposed kernels with state-of-the-arts. .

| Kernel | PROTEINS | ENZYMES | BZR | COX2 | DHFR | SYNnew* |
|---|---|---|---|---|---|---|
| SP | >5 day | >3 day | >2 day | >2 day | >3 day | >2 day |
| GH | 13’ 1" | 3’ 4" | 58" | 1’ 22" | 3’ 7" | 4’ 2" |
| RetGK-I | 2’ 57" | 37" | 17" | 24" | 1’ 15" | 30" |
| RetGk-II | 2.5" | 1" | 0.7" | 0.9" | 1.4" | 13.3" |
| GIK | 22’ 34" | 8’ 43" | 7’ 16" | 12’ 49" | 30’ 39" | 35’ 11" |
| Prop-diff | 7" | 5" | 3.5" | 3.7" | 7.5" | 4" |
| Prop-WL | 12" | 5.3" | 4" | 5.2" | 9" | 8" |
| HGK-WL | 3’ 42" | 1’ 25" | 1’ 2" | 1’ 9" | 1’ 55" | 2’ 11" |
| HGK-SP | 3’ 8" | 1’ 6" | 48" | 52" | 1’ 28" | 1’ 38" |
| NPE$_{(\alpha=1)}$ | 1’ 40" | 34" | 48" | 1’ 20" | 3’ 40" | 1’ 46" |
| NPO$_{(\alpha=0)}$ | 18.3" | 6" | 2.7" | 3.6" | 9" | 19" |
| NP | 1’ 49" | 35" | 52" | 1’ 26" | 3’ 58" | 1’ 58" |
| NPS | 27’ 42" | 2’ 19" | 3’ 55" | 7’ 49" | 16’ 10" | 55’ 58" |

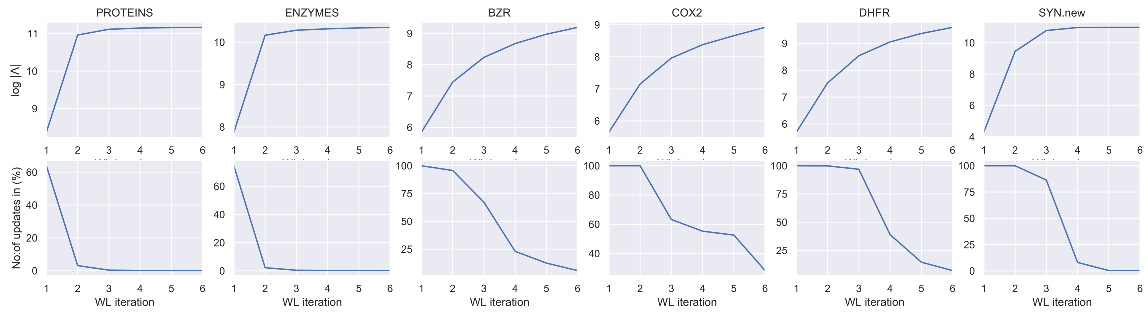


**Figure 5.5:** The plot of cardinality of *WL refined edge addresses* and percentage of kernel values getting updated against WL iterations for the datasets.

values are getting updated. The cardinality of *WL refined edge addresses* increases as the WL iteration proceeds and it can become saturated after a finite number of iterations. On the other hand, the graph similarity decreases as more neighborhood hops are taken as the WL iterations proceed and the pair of graphs whose kernel values getting updated decreases.

The Figure 5.5 gives the cardinality of *WL refined edge addresses* and the percentage of kernel values getting updated (in logarithmic scale) against WL iterations. It is observed that at a particular iteration, the number of kernel values getting updated steeply decreases whereas the cardinality does not have significant increase. This point can be used as a heuristic to choose the value of $h$ in the context of the above discussions.

### 5.3.6 Scaling for large graphs

Scaling requirement for the graph kernels in the case of attributed graphs can happen in two scenarios - (1) when the number of nodes is large and (2) the attribute dimension is very high. As far as the proposed kernel is concerned, the computation load happens around finding the *WL refined edge address*, correlating the edges to these addresses and finding the kernel values via convolution by processing the attributes over nodes/edges.

Now considering the scaling requirement when the number of nodes are larger, finding *WL refined edge address* and correlating the edges to them can be done via parallel processing. The graph structure that is stored in the form of an adjacency list can facilitate this. The nodes can be divided into smaller groups and each group can be fed into separate cores for the processing. The individual results from the cores can then later be consolidated. For both the scaling scenarios, convolution in the form of vectorization can be done as discussed in Section 5.1.8. But in the scaling scenario of attributes of larger dimension, the hardware accelerators like GPU can be utilized.

### 5.3.7 Applications in image processing

We applied the proposed kernel design in image processing tasks in which neighborhood structures are implicitly defined. An example case is the MNIST digit dataset [106]. In this case, apart from the natural image features the filters of a convolutional neural network (CNN) learn, it is the underlying neighborhood structure that helps in the classification tasks. In this context, we have applied the NPE kernel to MNIST images after converting them to graphs and compared the results with CNN. The experiment is done as a sanity check towards the efficiency of neighborhood matching procedure in the proposed approach in the fields apart from the chemo-informatics domain.

For converting the image to a graph, the method described by Defferrard et.al [64] is adopted. The MNIST image data is in the form of a 2D grid of size $28 \times 28$. Following [64], a 8-nearest neighborhood graph of the 2D grid is constructed in the form of a grid with 976 nodes ($28^2 = 784$ and 192 fake nodes), with 3198 edges, the attributes over the nodes are pixel values scaled in the range [0,1] and, the node/edge labels are taken as uniform. This graph conversion process is done using the Spektral library [107]. The NPE kernel is applied only for one iteration of WL color refinement.

For the experiments, 10000 images are chosen at random out of the total of 70000. The architecture followed in [64] is used for CNN. The results are given in Table 5.2. From the experiments we can see that the performance of the NPE kernel is on par with that of

CNN. In this context, in the image processing tasks in which the neighborhood structure is relevant, NPE kernel can be useful for the learning tasks. The challenge associated with such tasks is to define a suitable graph. The nearest neighbor approach over the pixels as described above is an obvious choice. If the domain knowledge permits in selecting the region of interest as nodes and the interaction between them as edges, such a method can be more effective than the nearest neighbor approach. The method can also be applied in volumetric data analysis where an underlying graph structure can be defined.

## 5.4 Conclusion

We designed kernels based on the neighborhood preserving property where the attribute and label information of nodes/edges are utilized. It helps to define a well-defined correspondence between subgraphs computed during kernel computations. The kernels can be recursively computed from the product graph that helps in an efficient computation procedure for dense graphs with large alphabets occurring in the WL color refinement algorithm. The proposed kernel which independently processes the attribute and label information is found to be very effective in the graph classification tasks. The method can also be extended to image analysis where an underlying graph structure can be formulated.

# Chapter 6

# Spectral Graph Convolutional Neural Networks in the Context of Regularization Theory

The success of CNNs as a powerful feature extractor for data in the Euclidean domain motivated researchers to extend the concepts to non-euclidean domains such as manifolds and graphs [108]. Among this, spectral graph convolutional networks based on the principles of spectral graph theory [109] and signal processing on graphs [110] have emerged as a powerful tool.

The relation between regularization theory and neural networks were formulated by Girosi et.al [111]. They have found that a class of smoothness functionals when used as stabilizers for the ill-posed problems [112] leads to regularization scheme that are equivalent to a single layer neural network called regularization networks. In the case of graphs, Smola et.al [27], [28] have studied the regularization property of graph Laplacian and how the regularization operators and support vector kernels are related. They have found that a smoothness functional on graphs can be obtained in terms of graph Laplacian and regularization properties on graphs can be achieved by processing on its eigenfunctions. Our work is mainly motivated by Smola et.al [27] as we observe that the support vector kernels proposed by them using the function of graph Laplacian can act as a convolution filter for SGCNs.

In this work, we propose a framework for filter designs in SGCNs from which its regularization behavior can be analyzed. The regularization behavior is attributed to smoothness the filters impart in the learning. It has to be noted that the proposed theoretical analysis is equally applicable for filter designs with high pass or band-pass behaviors. We also noted some insights in the direction of optimizing the network and the possibilities of new

**Table 6.1:** Frequency response function and output of filters of SGCNs

| Network | Freq. response ($g_\theta(\lambda)$) | Output, $y$ |
|---|---|---|
| ChebyNet [64] | $\sum_{k=0}^{K-1}\theta_k\lambda^k$ | $(\theta_0 I + \sum_{k=1}^{K-1}\theta_k\tilde{L}^k)f$ |
| GCN [65] | $\big(\theta(1-\lambda)\big)$ | $\theta(I-\tilde{L})f$ |
| GraphHeat [66] | $\theta_0+\theta_1\text{exp}\,(-s\lambda))$ | $(\theta_0 I+\theta_1 e^{-s\tilde{L}})f$ |
| IGCN [67] | $\big(\theta(1-\lambda)\big)^K$ | $\theta(I-\tilde{L})^K f$ |

architectures for graph learning.

The rest of the chapter is organized as follows. Section 6.1 discusses the framework for designing regularized graph convolution filters. Section 6.2 discusses the experiments and results. A discussion on improvising the SGCN architecture is discussed in Section 6.3 and conclusions are made in Section 6.4.

## 6.1 Regularized graph convolution filters

The adjacency matrix of the graph $G=(V,E)$ is defined as $W$ where $W_{ij}=w_{ij}$ denotes the weight associated with the edge $[i,j]$ and otherwise $0$. The degree matrix, $D$, is defined as the diagonal matrix where $D_{ii}=\sum_j w_{ij}$. The Laplacian of $G$ is defined as $L:=D-W$ and the normalized Laplacian is defined as $\tilde{L}:=D^{-\frac{1}{2}}LD^{-\frac{1}{2}}=I-D^{-\frac{1}{2}}WD^{-\frac{1}{2}}$. As $\tilde{L}$ is a real symmetric positive semi definite matrix, it has a complete set of orthonormal eigenvectors $\{u_l\}_{l=1}^n \in \mathbb{R}^n$, known as the graph *Fourier modes* and the associated ordered real non negative eigenvalues $\{\lambda_l\}_{l=1}^n$, identified as the *frequencies* of the graph. Let the eigen decomposition of $\tilde{L}$ be $U\Lambda U^T$ where $U$ is the matrix of eigenvectors $\{u_l\}_{l=1}^n$ and $\Lambda$ is the diagonal matrix of eigenvalues. Graph Fourier Transform (GFT) of a signal $f:V\to\mathbb{R}$ is defined as $\hat{f}=U^T f$ and inverse GFT is defined as $f=U\hat{f}$ [110].

The frequency response function and corresponding output of SGCNs discussed in Section 2.4.3 are summarized in the Table 6.1. In our work, the objective is to propose a framework to design regularized graph convolution filters and for this, regularization theory over graphs via graph Laplacian is used as discussed in the following section. We also found that the spectral filters in Table 6.1 can be deduced as special cases of the proposed *frequency response* functions in our framework.

It has to be noted that our framework is proposed for the *traditional* SGCN filters. However, it is straight forward to extend this theory to the networks such as graph wavelet network [68]. The only difference is that instead of taking the spectrum of $\tilde{L}$, the wavelet spectrum may be considered. Hence the theory developed can be extended to any networks

**Table 6.2:** Filters, corresponding regularization function $(r(\lambda))$ and its filter definition

| Filter | Regularization function $(r(\lambda))$ | Filter definition |
|---|---|---|
| Reg. Laplacian | $1 + s\lambda,\ s > 0$ | $(I + s\tilde{L})^{-1}$ |
| Diffusion | $exp\ \big(s\lambda\big),\ s > 0$ | $exp\ (-s\tilde{L})$ |
| $p$-step random walk | $(aI - \lambda)^{-p}\ a \geq 2,\ p \in \mathbb{N}$ | $(aI - \tilde{L})^{p}$ |
| Cosine | $\big(cos\ \frac{\lambda\pi}{4}\big)^{-1}$ | $cos\ (\frac{\tilde{L}\pi}{4})$ |

that operate on wavelet transform.

Consider a signal on the nodes of the graph which is generated from the function $f : V \to \mathbb{R}$. Without loss of generality, we assume only the case of 1-dimensional signal in this section. It is being identified that the eigenvectors of $L$ corresponding to lower frequencies or smaller eigenvalues are smoother on graphs [113]. The smoothness corresponding to the $k-$th eigenvector is, $\sum_{i\sim j} w_{ij}[u_k(i) - u_k(j)]^2 = u_k^T L u_k = \lambda_k$ where $i \sim j$ implies that nodes $i$ and $j$ are connected by an edge. It can be inferred that a smoothly varying graph signal corresponds to eigenvectors with smaller eigenvalues. This is under the assumption that the neighborhood of topologically identical nodes would be similar. In real-world applications, the signals over the graph could be noisy. In this context, the high-frequency content of the signal should be filtered out as it contains noise and low-frequency contents (eigenvectors corresponding to lower eigenvalues) should be maintained as it contains robust information. In other words, smoothness corresponds to spatial localization in the graphs which is important to infer local variability of the node neighborhoods and it attributes to lower eigenvalues. This is where the regularization behavior of the *frequency response* function of a SGCN becomes important. Now we can define the smoothness functional on graph $G$ as,

$$S_G(f) = \sum_{i\sim j} w_{ij}(f_i - f_j)^2 = f^T L f. \tag{6.1}$$

The smoothness property associated with $L$ or $\tilde{L}$ also indicates its potential application to design regularized filters for SGCNs. Since the spectrum of $\tilde{L}$ is limited in $[0, 2]$, normalized Laplacian is used in this work. In the following section, we discuss how graph Laplacian can be used for the regularization in graphs and propose our framework to design regularized graph convolution filters. It has to be noted that the regularized filters will be later defined as the ones that amplify low pass filtering and/or attenuates the high frequency components. Although there is an assumption that the low frequency components as robust and high frequency components as noise, there are instances where the latter can be useful for the learning tasks.

### 6.1.1 Graph Laplacian and regularization

Regularization functionals on $\mathbb{R}^n$ can be written as,

$$\langle f, Pf \rangle = \int |\bar{f}(\omega)|^2 r(\|\omega\|^2) d\omega = \langle f, r(\Delta) f \rangle \tag{6.2}$$

where $f \in L_2(\mathbb{R}^n)$, $P$ is a regularization operator, $\bar{f}(\omega)$ denotes Fourier transform of $f$, $r(\|\omega\|^2)$ is a frequency penalizing function and $r(\Delta)$ is a function that acts on the spectrum of the continuous Laplace operator $\Delta$. The equation in (6.2) can be mapped into the case of graphs by making an analogy between the continuous Laplace operator and its discrete counterpart which is the graph Laplacian [27]. Analogous to (6.2), Smola et.al [28] used a function of Laplacian, $r(\tilde{L})$, in the place of $P$ under Laplacian's capability to impart a smoothness functional on graphs. Hence regularization functionals on graphs can be written as $\langle f, Pf \rangle = \langle f, r(\tilde{L}) f \rangle$, where $r(\tilde{L}) = \sum_{i=1}^{n} r(\lambda_i) u_i u_i^T$.

The choice of $r(\lambda)$ should be in such a way that it favors the *low pass filtering* of the graph convolution filter, that is, the function $r(\lambda)$ should be high for a higher value of $\lambda$ to impose more penalization on high frequency (high eigenvalue) content of the graph signal. Similarly, the penalization of low frequency should be less. Hence we name $r(\lambda)$ as *regularization function*. The examples for choices of $r(\lambda)$ are listed in the second column of Table 6.2 and they are plotted in Figure 6.1.

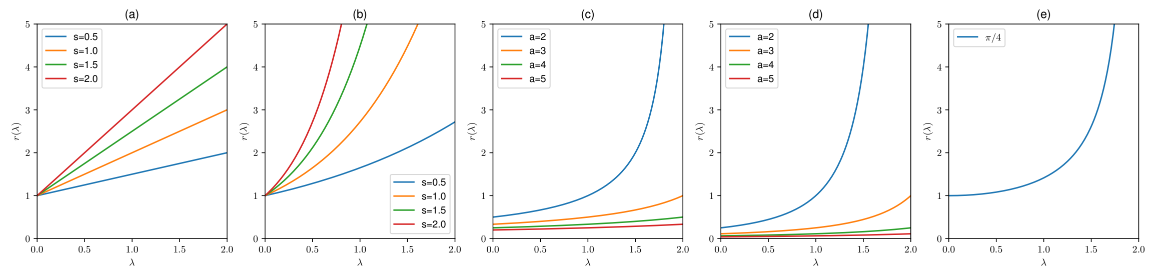


**Figure 6.1:** Regularization function, $r(\lambda)$. (a) regularized Laplacian ($s = \{0.5, 1, 1.5, 2\}$), (b) diffusion function ($s = \{0.5, 1, 1.5, 2\}$), (c) one-step random walk ($a = \{2, 3, 4, 5\}$), (d) 2-step random walk ($a = \{2, 3, 4, 5\}$), (e) inverse cosine function.

*Remark* 6.1. There exists an inverse relationship between the *regularization function* and *frequency response* function. To impose high penalization on higher frequencies, the *regularization function* is supposed to be a monotonically increasing function of the eigenvalues. At the same time, for the low pass filtering characteristics, to make high filter gain on a lower frequency and vice versa, the *frequency response* function is supposed to be a monotonically decreasing function of the eigenvalues.

*Remark* 6.2. Smola et.al [28] has shown that $P^{-1}$ (pseudo-inverse if not invertible) is a positive semidefinite (p.s.d) support vector kernel in a reproducing kernel Hilbert space (RKHS) $\mathcal{H}$ where $P \in \mathbb{R}^{n \times n}$ is a positive semidefinite regularization matrix and $\mathcal{H}$ is the image of $\mathbb{R}^n$ under $P$.

Remark 6.1 and 6.2 points out in the direction of using the inverse of a *regularization function*, as a *frequency response* function for SGCN filters. The corresponding filters take the form, $\big(r(\tilde{L})\big)^{-1}$. In this context, regularized filters corresponding to their *regularization functions* can be obtained as,

$$\mathcal{F} = \big(r(\tilde{L})\big)^{-1} = \sum_{i=1}^{n} \big(r(\lambda)\big)^{-1} u_i u_i^T \tag{6.3}$$

where $\{(u_i, \lambda_i)\}$ is the eigensystem of $\tilde{L}$ and $\big(r(\lambda)\big)^{-1}$ is the reciprocal function of *regularization function*. In the context of Remark 6.2, we can see that filters of SGCNs defined as per (6.3) are also support vector kernels on graphs provided their parameterization (if any) maintains positive semidefiniteness. The detailed discussion is provided in Appendix B.1.

Regularized graph filters which are defined as follows, can be designed by making use of (6.3).

**Definition 6.1** (Regularized graph convolution filter)**.** : The graph filter whose *frequency response* function $g_\theta(\lambda)$ behaves like a low-pass filter, i.e, $g_\theta(\lambda)$ should be a monotonically decreasing function in $\lambda$ or equivalently the associated regularization function $r(\lambda) = \big(g_\theta(\lambda)\big)^{-1}$ should be a monotonically increasing function in $\lambda$.

The design strategy for regularized graph convolution filters is summarized in Theorem 6.1.

**Theorem 6.1.** *A monotonically increasing function in the interval* $[0, \lambda_{max}]$ *is a valid regularization function to design regularized graph convolution filters using* (6.3) *where* $\lambda_{max}$ *is the largest eigenvalue of* $\tilde{L}$.

*Proof.* Note that Laplacian $\tilde{L}$ can be decomposed as $U\Lambda U^T$. Equivalently, this decomposition can be considered as a sum of the matrix of projections onto a one-dimensional subspace spanned by the eigenvectors (Fourier basis), i.e, $\tilde{L} = \sum_{i=1}^{n} \lambda_i P_{\lambda_i}$, where the linear map $P_{\lambda_i}(x) = u_i u_i^T x$ is the orthogonal projection onto the subspace spanned by the Fourier basis vector $u_i$ corresponding to the eigenvalue $\lambda_i$. Consider a regularization function $r(\lambda)$ that is monotonically increasing. Note that frequency response function $g_\theta(\lambda) = 1/r(\lambda)$ is monotonically decreasing. The filtering operation of a signal $f$ by the filter $\mathcal{F}$ can be written in terms of the mappings $P_{\lambda_i}, 1 \leq i \leq n$, i.e, $y = \mathcal{F}f = \sum_{i=1}^{n} g(\lambda_i)P_{\lambda_i}(f) = \sum_{i=1}^{n} g_\theta(\lambda_i)u_i u_i^T f$.

The values $g_\theta(\lambda_i)$ can be considered as weights that measure the importance of the corresponding eigenspace in the amplification or attenuation of the signal $f$. As $g_\theta(\lambda)$ is monotonically decreasing the weight of eigenspace corresponding to eigenvectors of lower frequencies (lower eigenvalues) of $\tilde{L}$ are higher and vice versa. The filter gain of the lower frequency components of $f$ is higher compared to the higher frequencies or $\mathcal{F}$ is a low pass filter. The validity of $r(\lambda)$ is established by the low pass filtering and hence the proof. □

Note that the proof is under the assumption that the parameters $\theta$ involved in the SGCN learning are in such a way that it ensures the monotonically decreasing property of $g_\theta(\lambda)$. The theorem also holds for other definitions of normalized or unnormalized Laplacian and any spectrum in $[0, \infty)$, provided the monotonicity property is maintained. To design regularized filters in the context of Theorem 6.1, it is enough to pick a valid *regularization function* and to plug it into (6.3) to define the filter.

Note that although Theorem 6.1 specifies the low pass filters, we can design filters with custom frequency responses like high pass or band pass filtering characteristics using the same settings. In this case, we need to choose an appropriate function $r(\lambda)$ with desired filtering behavior and plug it in (6.3). Hence the framework has its utility beyond low pass filters.

#### 6.1.1.1 Factors affecting the choices of the regularization function

We can have custom designs for the regularization function. However, to define a closed-form expression for the filter $\mathcal{F}$, the *regularization function* should be able to be expressed in a closed-form. Hence the choice of $r(\lambda)$ should be limited to the functions with power series expansion to get the closed-form expressions for easier computations.

The powers of the graph Laplacian involved in the expression can also affect graph learning since $(L^K)_{ij} = 0$ if the shortest path distance between nodes $i$ and $j$ is greater than $K$ [114]. This can be a factor in the choice of filters. For example, the regularization function form of a one-step random walk (where $a$ = 2) and inverse cosine is approximately the same. But the computation of cosine filter involves higher-order even powers of Laplacian whose non-zero elements are determined by graph structure and it also lacks the information from the neighborhood of odd number hops. Similarly, it is possible to pre-compute the values of hyper-parameters to design the form of the regularization function by knowing the precise spectrum of the Laplacians. In the next section, we discuss a set of filters corresponding to the regularization functions given in Table 6.2.

### 6.1.2 Regularized filters for SGCNs

We take equations in the second column of the Table 6.2 and plug into (6.3) to define the regularized filters. The results are summarized in the third column of Table 6.2. Note that variants of some filters are already familiar in the literature as explained below.

- *Case 1:* In a $p$-step random walk filter, if we put $a = 1, p = 1$ and add self loops, we get the filter corresponding to GCN [65].

- *Case 2:* The filter used in IGCN [67] uses higher powers of the GCN filter. Hence it corresponds to a $p$-step random walk filter with a value of $p \geq 2$, $a = 1$ and self loops.

- *Case 3:* As per [67], the graph filter of the label propagation (LP) method for semi-supervised learning takes the form of the regularized Laplacian filter.

- *Case 4:* GraphHeat filter is similar to a *diffusion* filter together with an identity matrix.

#### 6.1.2.1 Computational complexity

The learning complexity of *regularized Laplacian* is $\mathcal{O}(n^3)$ as it involves matrix inversion. For other filters, it is $\mathcal{O}(K|E|)$, where $K$ is the maximum power of the Laplacian involved in the approximation of the filter equation.

### 6.1.3 Analysis of regularization behavior of SGCNs

In this section, we analyze the regularization behaviors of state-of-the-art SGCNs. The idea is to identify the *regularization function* corresponding to the state-of-the-art networks from their filter definition. This helps to analyze the regularization capability of their filters.

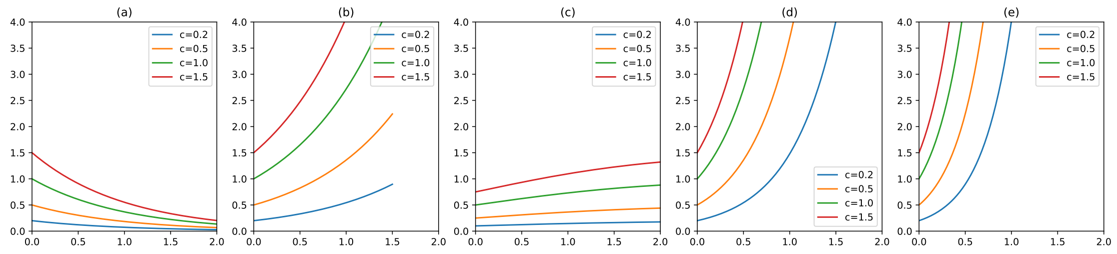


**Figure 6.2:** Regularization function, $r(\lambda)$. (a) ChebyNet, (b) GCN, (c) GraphHeat, (d) IGCN for $k = 2$ , (e) IGCN for $k = 3$. All graphs are for ($c = \{0.2, 0.5, 1.0, 1.5\}$)

#### 6.1.3.1 Chebynet

ChebyNet filtering [64] is defined as, $y = U(\sum_{i=0}^{K-1} \theta_k \Lambda^k)U^T f \approx \exp(\tilde{L})f$, where we assume parameters $\theta_k$ are the coefficients of the expansion of matrix exponential $\exp(\tilde{L})$. In this case, the regularization function, $r(\lambda) = \big(g_\theta(\lambda)\big)^{-1} = (\sum_{i=0}^{K-1} \theta_k \lambda^k)^{-1} \approx c.\exp(-\lambda)$ where $c$ is a constant determined by $\{\theta\}$, the parameters learned by the network.

Hence the regularization happening in ChebyNet is the exact opposite of the expected behavior since small eigenvalues are attenuated more and large ones are attenuated less as shown in Figure 6.2(a). In other words, the corresponding filter is a high pass filter. To make the ChebyNet a low pass filter, the filtering shall be done as per the negative exponential or *diffusion regularization* function and for this, we shall change the filtering operation as, $y = U(\sum_{i=0}^{K-1}(-1)^k \theta_k \Lambda^k)U^T f \approx \exp(-\tilde{L})f$ with the constraint $\theta_k > 0$ to keep up with the desired regularization property. We can also bring an additional hyper-parameter $s > 0$ into the power of exponential function. Note that in this case, it corresponds to the *diffusion* filter.

#### 6.1.3.2 GCN

GCN filtering [65] operation can be written as, $y = \theta(I - \tilde{L})f \approx \exp(-\lambda)$, where we assume parameter $\theta$ is 1 in the approximation of an exponential function. The regularization function, $r(\lambda) = c.(1-\lambda)^{-1} \approx c.\exp(\lambda)$ where $c$ is a constant determined by the parameter $\theta$. Hence the regularization happening in GCN is like that of a low-pass filter shown in Figure 6.2(b). Note that the filter of GCN corresponds to the first-order approximation of the *diffusion* filter. So in effect, it is the diffusion process that harnesses the representation capability of GCN by changing the sign of parameters [65] ($\theta_0 = -\theta_1 = \theta$) compared to ChebyNet.

*Spectral analysis of renormalization trick :* The trick refers to the process of adding self-loops [65] to the graphs for stable training of the network. Wu et.al [79] have shown that adding self-loops help to shrink the Laplacian spectrum from [0, 2] to [0, 1.5] which boosts the low pass filtering behavior. So the *regularization function* remains in the same form as mentioned above, but the range of eigenvalues being in the interval [0, 1.5].

#### 6.1.3.3 GraphHeat

GraphHeat filtering [66] operation can be written as $y = (\theta_0 I + \theta_1 e^{-s\tilde{L}})f$. The *regularization function*, $r(\lambda) = c.(1 + \exp{(-s\lambda)})^{-1}$ where we assume $c$ is a factor determined by $\theta_0$ and $\theta_1$. The regularization function is shown in Figure 6.2(c) and it behaves as that of a low pass filter.

#### 6.1.3.4 IGCN

Improved graph convolutional network (IGCN) filtering [67] operation can be written as, $y = \theta(I - \tilde{L})^k f \approx \exp(-k\lambda)$. The regularization function, $r(\lambda) = c.(1-\lambda)^{-k} \approx c.\exp(k\lambda)$ where $c$ is a constant determined by the parameter $\theta$. Hence the regularization happening in IGCN is like that of a low-pass filter as shown in Figure 6.2 (d).

#### 6.1.3.5 Generalization via rational filters

A rational filter design [115] takes the form,

$$g(\lambda) = \frac{\sum_{i=0}^{q} b_i \lambda^i}{1 + \sum_{i=1}^{p} a_i \lambda^i}.$$

The filter of this form is called the auto-regressive moving average (ARMA) filter of order $(p, q)$ denoted by ARMA$(p, q)$. It is possible to approximate desired frequency response using ARMA filters although a matrix inversion process is involved. If $a_i$s are set to zero, $g(\lambda)$ reduces to a polynomial filter. The filters of state-of-the-art SGCNs belong to this category. It has to be noted that the theoretical analysis provided in Section 6.1.1 is applicable for ARMA filters also. For example, Bianchi et.al [116] have proposed a convolutional ARMA filter of order $K$ whose frequency response, $g(\lambda) = \sum_{k=1}^{K} \frac{b_k}{1-a_k\lambda}$. The regularization behavior or the filtering property, in this case, is dependent on the values of the parameters {a,b} and the corresponding regularization function is provided by $(g(\lambda))^{-1}$.

### 6.1.4 Limitations

The analysis in the proposed framework is limited to polynomial filters of SGCNs in which the *regularization* function can be deduced. Hence it cannot be applied to spatial networks such as GraphSage [117], GAT [86], etc unless we can recover their corresponding *frequency response* function.

The regularization behavior of SGCNs shown in Figure 6.2 is based on the approximation of learning parameters collapsed into a single hypothetical parameter $c$. However, in practice, the exact form of $r(\lambda)$ of SGCNs may deviate depending on the actual values of parameters, $\theta$. Hence the default filtering behaviors, as explained in Section 6.1.3 of the models, are also subjected to change and is dependent upon the features and downstream learning task.

## 6.2 Experiments

The variants of proposed filters as in Table 6.2 are compared with state-of-the-art SGCNs namely ChebyNet [64], GCN [65], GraphHeat [66], and IGCN (RNM variant of the filter) [67]. The comparison is also made with graph regularization based algorithms for semi-supervised learning namely - manifold regularization (ManiReg) [73], semi-supervised embedding (SemiEmb) [77], and label propagation (LP) [76]. Other baselines used are Planetoid [118], DeepWalk [119], and iterative classification algorithm (ICA) [120]. The computational complexity of SGCN baselines is $\mathcal{O}(K|E|)$, where $K$ is the maximum power of the Laplacian as explained in Section 6.1.2.1. For other baselines, the approximate complexity is as follows, for ManiReg, SemiEmb, and Planetoid- $\mathcal{O}(n^2)$, for LP, DeepWalk, and ICA- $\mathcal{O}(nd)$ where $n$ is the number of nodes and $d$ is the dimension of the node feature space.

The citation network datasets [118] - Cora, Citeseer, and Pubmed is used for the study. In these graphs, nodes represent documents and edges represent citations. The datasets also contain 'bag-of-words' feature vectors for each document and further details are given in Table 6.3 where the label rate denotes the ratio of labeled nodes that are used for training to the total number of nodes.

*Diffusion filter* which is a matrix exponential is approximated for first $K+1$ terms, i.e,

$$g_\theta(\Lambda) = \theta \sum_{k=0}^{K} (-1)^k \frac{1}{k!} \Lambda^k \tag{6.4}$$

**Table 6.3:** Summary of the datasets

| Dataset | Nodes | Edges | Classes | Features | Label rate |
|---|---|---|---|---|---|
| Cora | 2708 | 5429 | 7 | 1433 | 0.052 |
| Citeseer | 3327 | 4732 | 6 | 3703 | 0.036 |
| Pubmed | 19717 | 44338 | 3 | 500 | 0.003 |
| Flickr | 89250 | 899756 | 7 | 500 | 0.010 |
| GitHub | 37300 | 578006 | 2 | 128 | 0.010 |
| Deezer Eur. | 28281 | 185504 | 2 | 128 | 0.010 |
| Facebook-PP | 22470 | 342004 | 4 | 128 | 0.010 |

where there is only a single parameter $\theta$ is learned. For *diffusion* filter, ChebyNet and GraphHeat the value of $K$ used for the approximation of matrix exponential is tuned from $\{1, 2, 3, 4\}$. Similarly, *cosine filter* which involves cosine of a matrix is taken as,

$$g_\theta(\Lambda) = \theta \sum_{k=0}^{K} (-1)^k \frac{1}{2k!} \Lambda^{2k}. \tag{6.5}$$

The value of $K$ is tuned from $\{1, 2, 3\}$. For *diffusion* and GraphHeat filter, the value of $s$ is tuned in the range [0.5, 1.5] and for *$p$-step randomwalk* filters, the value of $a$ is tuned in the range [2,24]. For GCN and IGCN, the author's code was reproduced for experiments.

### 6.2.1 Experimental setup

In the experiments, network architecture proposed by Kipf et.al [65] is used. An ablation study with networks of one layer, two layers, and three layers of graph convolution (GC) are used to evaluate all the filters under study and along with this, networks with a GC layer followed by one and two layers of dense layers are also studied. In the ablation studies, it has been found that the network with two layers of GC has outperformed other architectures. The architecture takes the form,

$$Z = \text{softmax}(\mathcal{F}(\tilde{L})\ \text{ReLU}(\mathcal{F}(\tilde{L}) X \Theta^{(1)}) \Theta^{(2)}) \tag{6.6}$$

where $\mathcal{F}(\tilde{L}) \in \mathbb{R}^{n\times n}$ is the filter, $X \in \mathbb{R}^{n\times d}$ is the input feature matrix, $\Theta^{(1)} \in \mathbb{R}^{d\times c_1}$ is the filter parameters of first layer ($c_1$ is the number of filters) and $\Theta^{(2)} \in \mathbb{R}^{c_1\times c_2}$ is the filter parameters of second layer ($c_2$ is the number of filters). The architecture is illustrated in Figure 6.3. Note that the value of $c_2$ equals the total number of classes in the data. Here $Z \in \mathbb{R}^{n\times c_2}$ in which the softmax function is applied row wise. The loss function optimized is the cross-entropy error over the labeled examples [65] defined as follows,

$$\mathcal{L} = -\sum_{i \in \mathcal{Y}} \sum_{j=1}^{c_2} y_{ij} \ln(Z_{ij}) \tag{6.7}$$

where $\mathcal{Y}$ is the set of nodes whose labels are known and $y_{ij}$ is defined as 1 if label of node $i$ is $j$ and 0 otherwise.

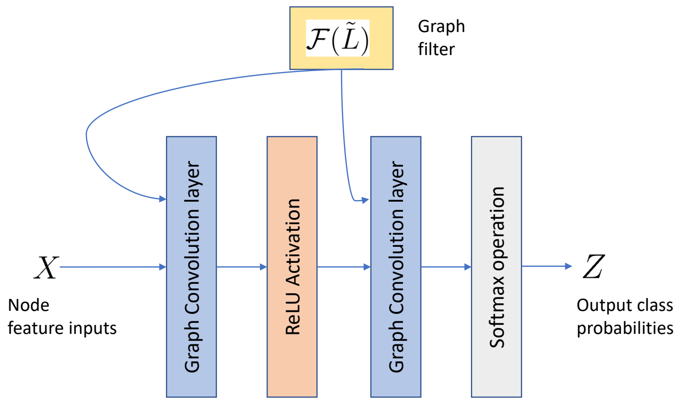


**Figure 6.3:** SGCN two layer architecture.

For training, all the feature vectors and 20 labels per class are used. The same dataset split as used by Yang et.al [118] is followed in the experiments. All the SGCN models corresponding to different filters are trained 10 times each according to a unique random seed selected at random. All models are trained for a maximum of 200 epochs using the ADAM optimizer [121] with the learning rate fixed as 0.01. Early stopping is done in the training if the validation loss does not decrease for 10 consecutive epochs. Network weight initialization and normalization of input feature vectors of the nodes are done as per [122]. Implementation is done using Tensorflow [123]. The hardware used for the experiments is Intel Xeon E5-2630 v3 2.4 GHz CPU, 80 GB RAM, and Nvidia GeForce GTX 1080-Ti GPU.

The higher powers of graph Laplacian are computed with the Chebyshev polynomial approximations [114] in calculating ChebyNet and $p$-step random walk filters considering its computational advantage. Chebyshev polynomial of order $k$ is computed by the recurrence relation $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$, where $T_0$ and $T_1$ is defined as 1 and $x$ respectively. The polynomials form an orthogonal basis for $L^2([-1,1], \frac{dy}{\sqrt{1-y^2}})$, i.e, the space of square integrable-functions with respect to the measure $dy/\sqrt{1-y^2}$. Hence when

**Table 6.4:** Classification accuracy (in percentage $\pm$ standard deviation) along with average time taken for one epoch (in brackets).

| Methods | Cora | Citeseer | Pubmed |
|---|---|---|---|
| ManiReg | 59.5 | 60.1 | 70.7 |
| SemiEmb | 59.0 | 59.6 | 71.1 |
| LP | 68.0 | 45.3 | 63.0 |
| DeepWalk | 67.2 | 43.2 | 65.3 |
| ICA | 75.1 | 69.1 | 73.9 |
| PLanetoid | 75.7 | 64.7 | 77.2 |
| MLP | 56.2 | 57.1 | 70.7 |
| GCN | 81.78 $\pm$ 0.64 (1.02) | 70.73 $\pm$ 0.53 (1.03) | 78.48 $\pm$ 0.58 (1.21) |
| IGCN | 80.49 $\pm$ 1.58 (1.02) | 68.86 $\pm$ 1.01 (1.06) | 77.87 $\pm$ 1.55 (1.25) |
| ChebyNet | 82.16 $\pm$ 0.74 (1.03) | 70.46 $\pm$ 0.70 (1.04) | 78.24 $\pm$ 0.43 (1.21) |
| GraphHeat | 81.38 $\pm$ 0.69 (1.04) | 69.90 $\pm$ 0.50 (1.05) | 75.64 $\pm$ 0.64 (1.34) |
| Diffusion | **83.12** $\pm$ **0.37** (1.11) | 71.17 $\pm$ 0.43 (1.06) | **79.20** $\pm$ **0.36** (1.80) |
| 1-step RW | 82.36 $\pm$ 0.34 (1.02) | 71.05 $\pm$ 0.34 (1.03) | 78.74 $\pm$ 0.27 (1.21) |
| 2-step RW | 82.51 $\pm$ 0.22 (1.03) | 71.18 $\pm$ 0.59 (1.05) | 78.64 $\pm$ 0.20 (1.29) |
| 3-step RW | 82.56 $\pm$ 0.24 (1.05) | **71.21** $\pm$ **0.63** (1.04) | 78.28 $\pm$ 0.36 (1.81) |
| Cosine | 75.53 $\pm$ 0.52 (1.03) | 67.29 $\pm$ 0.64 (1.03) | 75.52 $\pm$ 0.53 (1.29) |

it comes to computing the powers of $\tilde{L}$, its spectrum has to be rescaled in the interval [-1, 1] as follows $L_s = \frac{2}{\lambda_{max}}\tilde{L} - I_n$, where $L_s$ is the rescaled Laplacian, $\lambda_{max}$ is the maximum eigenvalue of $\tilde{L}$.

Except for the spectral SGCNs, the results related to other models are taken from [65] and [118]. The comparison is meaningful since all the results are taken from the same dataset split as in [118] and we also follow the same split for SGCN models. The hyper-parameters used for the models are: dropout rate = 0.5, L2 regularization factor for the first layer weights = $5 \times 10^{-4}$, and the number of filters used in each layer is tuned from 16, 32, 64, and 128. These hyper-parameters are optimized on an additional validation set of 500 labeled examples as followed in [65]. Accuracy is used as the performance measure where models are evaluated on a test set of 1000 labeled examples. Since the focus of the study is on the comparison of spectral filters, mean accuracy along with standard deviation is reported for SGCN variants. For algorithms other than SGCN, accuracy on a single dataset split reported by Kipf et.al [65] is given.

### 6.2.2 Results and Discussion

The results are tabulated in Table 6.4. The best results are bolded. Compared with graph regularization and label propagation methods, SGCN methods have better performance. The *diffusion* filter has the highest accuracy in Cora and Pubmed. In Citeseer, it is the

*3-step RW* but the difference with *diffusion* filter is negligible. Among methods other than SGCNs, ICA and Planetoid have better performance.

*1-step RW* filter is an improvised version of GCN and IGCN in terms of regularization capability. Analyzing their comparison, *1-step RW* filter performs better in all datasets. The three variants of RW filters have higher performance in Cora and Citeseer datasets compared with GCN and IGCN, the reason being attributed to the tuning of parameter $a$ and in the case of Pubmed, RW filters and GCN have better performance than IGCN. From Figure 6.1 (c),(d) and Figure 6.2 (b),(d),(e), we can observe that RW filters, GCN and IGCN have better regularization properties and hence their performance is stable across all datasets. This observation suggests that selection of hyper-parameters $a$, $p$ and $k$ can be done prior to training by designing the regularization function and this method can be applied for any filters with hyper-parameters.

ChebyNet, GraphHeat, and *diffusion* filters calculate the first few powers of the Laplacian in their learning settings. It has been observed that the performance of the *diffusion* filter is better than both despite using one parameter to learn. It is noteworthy that although ChebyNet is having an opposite regularization behavior compared with *diffusion* filters and GraphHeat, there is no significant difference in their performance. This can be explained as follows. The parameters $\Theta$s associated with the network are optimized with respect to the loss function. The final parameters learned after the network learning can affect the form of the *regularization function* since the loss function does not explicitly optimize the desired filtering behavior. This suggests the need for a regularized loss function as discussed in Section 6.3.

The performance of the *cosine* filter is lower compared with other filters. The reason is due to the approximation of the *cosine filter* that requires higher even powers of the Laplacian whose elements can be mostly zeros based on the graph structure as discussed in Section 6.1.1.1. It also requires skipping odd hops in the graph resulting in some information loss while learning. The experiments with *regularized Laplacian* filter are omitted since it involves costly matrix inversion.

The average time required for one training epoch (in seconds) is shown in the brackets along with the accuracy. It can be noted that the time taken increases for those models which use the higher-order powers of the Laplacian and a higher number of filters.

#### 6.2.2.1 Effects of hyper-parameter tuning

The variation in accuracy against hyper-parameters of the proposed filters applied to the Cora dataset is given in Figure 6.4. For *diffusion* filter ($K = 3$), accuracy increases as $s$

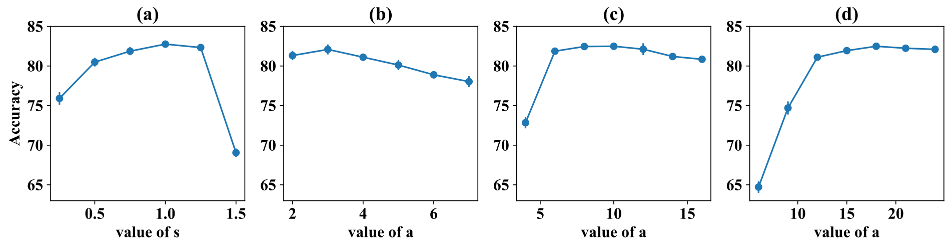


**Figure 6.4:** Accuracy variation with hyper-parameters. (a) Diffusion, (b) 1-step RW, (c) 2-step RW, (d) 3-step RW

is increased but there is a drop in accuracy after a peak value of $s$. Similar is the case of *1-step RW*. In the case of *2-step* and *3-step RW* filters, accuracy increases as the value of $a$ increases but after a threshold point, accuracy variation is minimal. Similar is the trend observed for other datasets. For these experiments, the network with two layers of GC having 32 filters is used.

### 6.2.3 Experiments with large datasets

The performance of the proposed filters is analyzed with large datasets to check their efficiency. They are compared with GCN and ChebyNet. The datasets used are Flickr [124], GitHub [125], Deezer Europe [126] and Facebook Page-page [127]. The nodes in Flickr dataset correspond to images uploaded. The two images with similar properties are connected by an edge. These properties are identical geographic location, similar gallery, comments by the same user, etc. The node features correspond to 500 dimensional bag-of-word representation of the corresponding image. The task is to predict the class of the nodes from 7 choices. In GitHub dataset, the nodes correspond to the developers and edges are formed if one of the user follows the other. The features are based on location, biography, etc. The learning task is to classify the nodes into either machine learning or web developer.

The Deezer Europe dataset is a social network data of Deezer users in Europe. The users are represented as nodes and links are formed based on follower relationships. The classification task is to predict the user's gender. Facebook Page-Page is a dataset where nodes correspond to Facebook pages. Edges are formed based on mutual likes between the pages. The node features are extracted from the site descriptions and the task is to classify them into one of the 4 classes. The datasets are downloaded using PyTorch-Geometric [128] and their details are provided in Table 6.3.

#### 6.2.3.1 Experimental Setup

The label rate for the experiments is 0.01 and 1 % of the nodes are taken for the validation set. The nodes are selected at random for training and validation. The testing is done with the remaining nodes. A three-layer network is used for the experiments, that is, output $Z = \text{softmax}\left(\mathcal{F}(\tilde{L})\left(\text{ReLU}(\mathcal{F}(\tilde{L})X_2\Theta^{(2)})\right)\Theta^{(3)}\right)$, where $X_2 = \text{ReLU}(\mathcal{F}(\tilde{L})X\Theta^{(1)})$ and batch normalization is applied in the first two layers. The architecture is illustrated in Figure 6.5. However, two-layer network as in (6.6) is used for Deezer Europe dataset. The hyper-parameter tuning is as per the details given in Section 6.2.1. The proposed filters are compared with GCN and ChebyNet. The other experimental setup remains the same as explained in Section 6.2.1 except the hardware - 2 x Intel Xeon Silver 4114, 128 GB RAM and Tesla V-100/16 GB GPU.

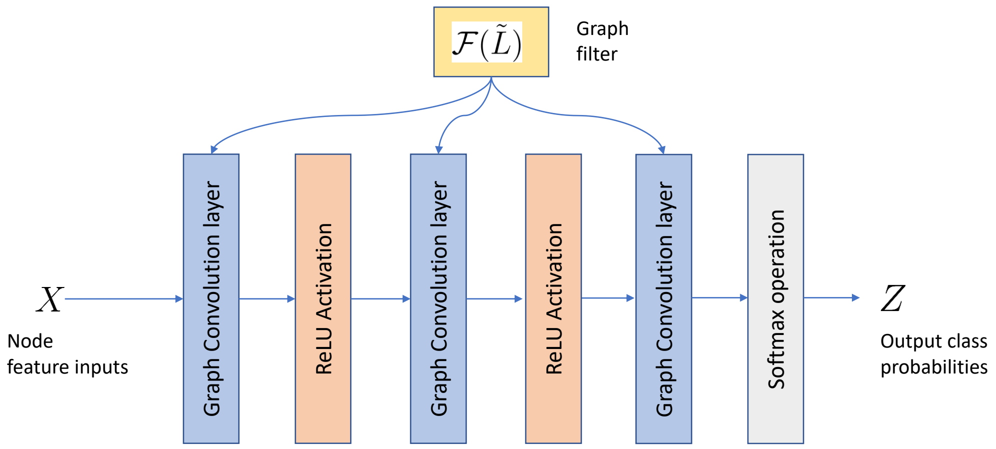


**Figure 6.5:** SGCN three layer architecture.

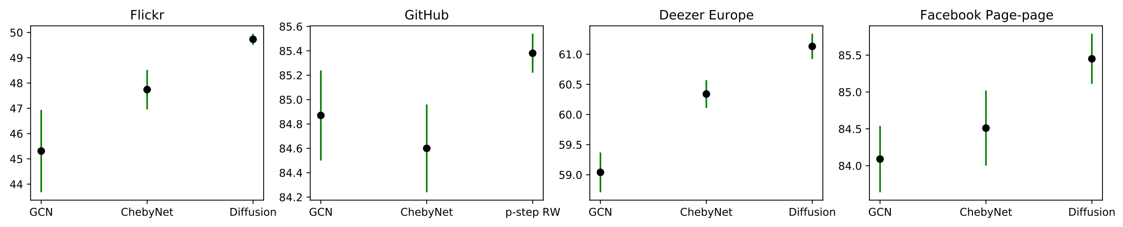


**Figure 6.6:** Accuracy plot with standard deviation of GCN, ChebyNet and best performing model for Flickr, GitHub, Europe Deezer and Facebook Page-page datasets.

**Table 6.5:** Classification accuracy (in percentage $\pm$ standard deviation).

| Methods | Flickr | GitHub | Deezer E. | Facebook-PP |
|---|---|---|---|---|
| GCN | $45.31 \pm 1.63$ | $84.87 \pm 0.37$ | $59.04 \pm 0.33$ | $84.09 \pm 0.45$ |
| ChebyNet | $47.74 \pm 0.78$ | $84.60 \pm 0.36$ | $60.34 \pm 0.23$ | $84.51 \pm 0.51$ |
| Diffusion | $\mathbf{49.73 \pm 0.22}$ | $85.33 \pm 0.56$ | $\mathbf{61.13 \pm 0.21}$ | $\mathbf{85.45 \pm 0.34}$ |
| p-step RW | $48.92 \pm 0.24$ | $\mathbf{85.38 \pm 0.16}$ | $60.88 \pm 0.36$ | $85.28 \pm 0.46$ |
| Cosine | $42.67 \pm 0.35$ | $70.88 \pm 0.61$ | $56.96 \pm 0.40$ | $35.37 \pm 0.78$ |

#### 6.2.3.2 Results and Discussion

The results are provided in Table 6.5 with the best results bolded and they are reported for 10 runs at different random seeds. From the results, it can be seen that the performance of *diffusion* and *p-step RW* are better than GCN and ChebyNet. The standard deviation of the best performing models is lower as shown in the Figure 6.6 and it shows their robustness. The performance of the *Cosine* filter is lower compared to other methods. As discussed in Section 6.2.2, this may be due to the skipping of odd hops in the graph that results in some information loss during message passing.

## 6.3 Discussion on optimizing network architectures of SGCN

We discuss certain points on optimizing the SGCN architectures in the context of our proposed framework. This is mainly motivated by the following observations.

- The *traditional* SGCNs work only on the basis of low pass filtering. But it has been shown that for some applications high frequency components are also useful as discussed in Section 2.4.5.

- The loss function optimization for network parameters $\Theta$s associated with the downstream tasks in SGCNs may not guarantee the expected regularization behavior of the *frequency response* function. Although the low pass filtering is implicitly embedded in the procedure, it is not reflected in the loss function. This case is very evident in the case of ChebyNet as it is showing decent performance in the learning task despite having a high pass behavior.

As an investigation to analyze the existing architecture in the above contexts, we have experimented as described below.

### 6.3.1 Decoupling filtering from network learning

To understand the practical impact of the framework we proposed, an experiment is done that decouples the low pass filtering from the network learning inspired by [79]. First, the filtering is done separately using $\mathcal{F}$ (with no learning parameters) and the resulting filtered features are given into a two-layer MLP (chosen after ablation studies among 1 & 3 layer models). This helps to identify the impact of the choice of $r(\lambda)$ formulated in our work independent of the filter parameters in the routine learning. The results are given in Table 6.6.

#### 6.3.1.1 Observations

From the results, we can see that a simple low pass filtering is enough to give the result within a 2-3% margin of the results given by classical SGCNs. The performance of ChebyNet in this experiment is lower compared to other filters. In Section 6.1.3, we found that, unlike other networks, ChebyNet is a high pass filter. Its performance degradation is evident since the filter is made parameter-free and its high pass behavior is exposed. This has to be read in conjunction with the observation that it is the low-frequency component that contributes more to the efficient learning in the case of Cora, Citeseer, and Pubmed datasets [78].

All other filters except the ChebyNet satisfy Theory 6.1, and hence their performance is better under the low pass filtering property. But the overall results are lower compared to the conventional GCN architecture [65] followed in the experiment described in the Section 6.2.1. This points out to the possibility that the network optimization is not explicitly guaranteeing the desired filtering properties. The case of the ChebyNet is an example as its performance is good in conventional GCN architecture despite having contradictions with Theory 6.1 whereas its performance in the new experiment is lower.

From the experiment, we can make further research directions that help in optimizing the current SGCN architectures proposed in [65]. They are briefly discussed in the following sections.

### 6.3.2 A new regularization term in the loss function

We have seen that the default nature of *traditional* SGCNs is to employ a low pass filtering of the features. But at the same time a high pass filter with a good localization property like ChebyNet can give good performance. This can be explained as follows.

**Table 6.6:** Accuracy of the filters. Std dev. is given in brackets.

| Methods | $g(\lambda)$ | Cora | Citeseer | Pubmed |
|---|---|---|---|---|
| MLP | – | 56.50 (1.21) | 53.57 (2.71) | 71.87 (0.21) |
| GCN | $(1-\lambda)$ | 77.99 (0.75) | 68.28 (0.51) | 76.05 (0.33) |
| IGCN | $(1-\lambda)^K$ | 81.44 (0.41) | 70.64 (0.67) | 78.46 (0.70) |
| ChebyNet | $\sum_{k=0}^{K-1}\lambda^k$ | 27.18 (1.46) | 28.63 (1.08) | 59.99 (0.58) |
| GraphHeat | 1+exp$(-s\lambda)$ | 73.57 (0.72) | 66.31 (0.59) | 73.50 (0.85) |
| Diffusion | exp$(-s\lambda)$ | 78.47 (0.32) | 68.47 (0.61) | 76.72 (0.59) |
| 1-step RW | $(a-\lambda)$ | 77.31 (0.52) | 67.95 (0.80) | 76.34 (0.47) |
| 2-step RW | $(a-\lambda)^2$ | 78.08 (0.62) | 69.41 (0.44) | 76.90 (0.23) |
| 3-step RW | $(a-\lambda)^3$ | 78.98 (0.28) | 69.23 (0.71) | 71.49 (0.81) |
| Cosine | cos$(\lambda\pi/4)$ | 70.38 (0.82) | 64.67 (0.62) | 72.01 (0.74) |

As per (6.7), the parameter optimization is with respect to the classification loss only. But the weights learned by minimizing this loss is not reflecting any *explicit* information about the filtering characteristics required for the task. Hence we can think about adding a regularizer in the loss function that finds a trade-off between the downstream learning task and the required filtering.

In this context, it is useful to recollect that the GCN architecture followed in this study was proposed as an alternative to minimize the explicit graph-based regularization. It is characterized by the loss function, $\mathcal{L} = \mathcal{L}_0 + \lambda\ f^T\tilde{L}f$, of a semi-supervised learning problem [76] where $\mathcal{L}_0$ is the loss function specific to the corresponding learning task, $\lambda$ is a hyper-parameter and $f$ is the function to be learned. The GCN architecture is proposed as a means to optimize $L_0$ with the assumption that the graph-based regularization characterized by the second term is not required. The reason is attributed to the property of $f(.)$ learned by the network being conditioned on the adjacency matrix of the graph and hence it could learn the representation of both labeled and unlabeled nodes together [65]. But the low pass filtering property of the filters used in this architecture was not taken into consideration until recently. Hence there is room for further research in proposing a regularizer term into the current loss function optimization given by (6.7).

Some recent works discuss adding a new regularizer term although it is not strictly attributed to the above discussion. Yang et.al [129] further propagate the output $Z$ given in (6.6) by the SGCN by multiplying $Z$ with $\tilde{L}$. Then the regularizer term is introduced as a function that reduces the difference between $Z$ and $\tilde{L}Z$. The motivation is to use the prediction of the neighborhood nodes as soft supervision for the model output on the node of interest. To alleviate the over-smoothness issue that can arise from the conventional filtering in SGCNs, Chen et.al [130] proposed to use a regularizer term in the loss function.

The regularizer is inspired by a metric that calculates mean average distance of the node embeddings and the optimization is done to improve the information-to-noise ratio. A different approach was proposed by Bianchi et.al [116] in which the desired *frequency response* is learned by using rational filters as explained in Section 6.1.3.5.

Apart from adding a regularizer, another alternative is to use a collection of filters that give the individual amplification to a wider portion in the eigenspectrum. This will be discussed in the following section.

## 6.3.3 Architectures that use a collection of filters

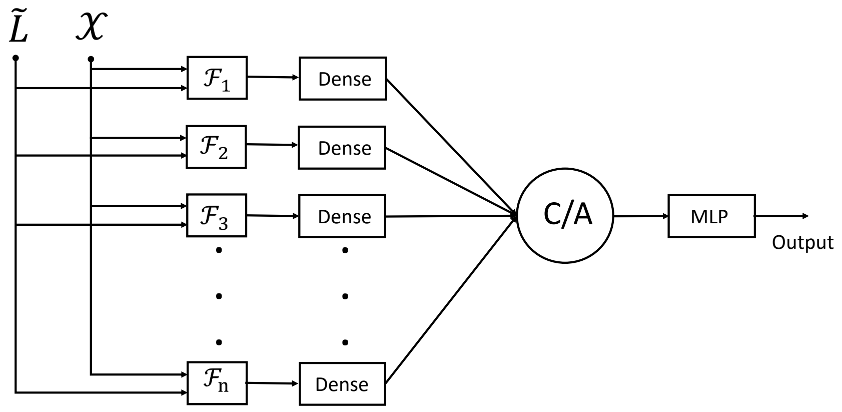


**Figure 6.7:** SGCN architecture with a collection of filters. C/A stands for concatenation/averaging

It has been noted that the frequency components useful for a learning task for an arbitrary dataset cannot be known in advance. The computationally feasible SGCN filters can only be made as either low pass or high pass filters. Hence an obvious choice is to deploy a collection of filters that can cover a wide spectrum of frequency amplifications by designing custom *frequency response* functions and combining the resultant features through some aggregation function like averaging or concatenation. Such an architecture is shown in Figure 6.7. Associated with this, there are multiple ways to configure such a network. For example, one can use learn-able parameters in the filters $\mathcal{F}_i$s or can make it simply as a feature pre-processor as explained in the decoupling experiment in Section 6.3.1. The dense layers in the architecture can be optional, that is, they can be removed in the case of using learn-able filters. The concatenated or averaged feature representation can then be used as an input to a perceptron or a dense layer for downstream learning tasks.

A few works were done in this direction although most of them are proposed as a new architecture rather than being presented as a solution to include all types of frequency components. The network proposed by Frasca et.al [131] uses a collection of learn-able low pass filters and the features smoothed by them are concatenated and given to a dense layer for prediction. The architecture proposed by Wang et.al [132] uses a collection of *traditional* SGCN filters as well as specific feature extractors for node embeddings and topological structures. The features corresponding to each node are taken as a linear combination of the output of filters/extractors learned by an attention mechanism. The architecture proposed by Gao et.al [133] projects the input features to a collection of subspaces where a filter is used to learn about the relevant frequency characteristics of the projection. They specifically deploy the filters to learn from the wide frequency spectrum. Wang et.al [134] proposed a scoring mechanism to assess the effectiveness of a filter in downstream tasks and used the score to make a linear combination of the filtered features given by the filters.

#### 6.3.3.1 Challenges

While the type of architectures depicted in Figure 6.7 has a theoretical justification in the context of our proposed framework, it comes with certain challenges when it comes to the implementation. Using a large collection of filters may result in over parameterization of the networks that could result in large variance and over-fitting in small graphs. It is also difficult to identify the utility of a particular filter beforehand and the optimal filter combination varies according to the applications and datasets.

## 6.4 Conclusion

We formulated a framework to design regularized filters for spectral graph convolution networks based on regularization in graphs modeled by graph Laplacian. A new set of filters are proposed and identified the state-of-the-art filter designs as their special cases. The regularization behavior of the state-of-the-arts are also analyzed. The new filter designs proposed in the context of the framework have shown superior performance in semi-supervised classification tasks compared to conventional methods and state-of-the-art SGCNs. Considering the practical impacts of the framework, we proposed two directions that can further optimize the SGCN architectures. The first one is to add a regularizer to the current optimization problem. The second one is to use a collection of filters that could give the filtering outputs across the wide spectrum of frequencies rather than relying only on low pass features. We have also identified certain related works in both directions.

# Chapter 7

# Applications

In this section, we discuss certain applications of the proposed approaches as a proof-of-concept. For analyzing the proposed kernels, multi-view and optimal node assignment, they are applied in the areas of brain connectivity, social media data analysis and prediction of spread of Covid-19 pandemic.

## 7.1 Brain connectivity and social media data analysis

The proposed methods - mutli-view MKL kernel and optimal assignment kernel - is applied in brain net datasets OHSU, KKI and PEKING and social media dataset IMDB-BINARY and REDDIT-BINARY along with certain state-of-the-art graph embeddings and kernels namely graph2vec [20], GE-FSG [23], WL-subtree kernel [25] and Treelet kernel+MKL [54].

### 7.1.1 Datasets

The OHSU, KKI and PEKING data is constructed from the whole brain fMRI atlas [135]. The graphical data is basically a mapping where brain is considered as a network (or a graph) where each node corresponds to a region of interest (ROI) and the edges indicate correlations between two ROIs. The label information of the datasets are with respect to attention-deficit/hyperactivity disorder (ADHD) classification, hyperactive-impulsive (HI) classification, and gender classification (GD) respectively.

The social media dataset used is IMDB-BINARY [24]. It is a movie collaboration dataset. The nodes in the graph represent actors/actresses and there is an edge between them if they appear in the same movie. The objective is to classify the genre of the corresponding film as *Action* or *Romance*. REDDIT-BINARY [24] is a balanced dataset where each graph corresponds to an online discussion thread where nodes correspond to users, and there is an edge between two nodes if at least one of them responded to another's comment. The classification task is to identify whether the given graph belongs to *question/answer-based community* or a *discussion-based community*. The dataset details are given in Table 7.1.

**Table 7.1:** Dataset details: $|D|$: cardinality of the data set $D$, $|P|$: cardinality of positive class, $|N|$: cardinality negative class, $V^{avg}$ =: average of $\{|V|, V \in D\}$, $E^{avg}$ = average of $\{|E|, E \in D\}$, $V^{\max} = \max\{|V|, V \in D\}$, $E^{\max} = \max\{|E|, E \in D\}$.

| **Dataset** | $\|D\|$ | $\|P\|$ | $\|N\|$ | $V^{avg}$ | $E^{avg}$ | $V^{\max}$ | $E^{\max}$ |
|---|---|---|---|---|---|---|---|
| OHSU | 79 | 44 | 35 | 82.0 | 199.6 | 171 | 823 |
| KKI | 83 | 46 | 37 | 26.9 | 48.4 | 90 | 237 |
| PEKING | 85 | 36 | 49 | 39.3 | 77.3 | 134 | 535 |
| IMDB.BIN | 1000 | 500 | 500 | 19.7 | 96.5 | 136 | 1249 |
| REDD.BIN | 2000 | 1000 | 1000 | 429.6 | 497.7 | 3782 | 4071 |

### 7.1.2 Results and Discussion

The results obtained are given in the Table 7.2 and the weights learned by SimpleMKL are given in Table 7.3. The result reported for OA kernel is the best among $K_{ONA}$ and $\tilde{K}_{ONA}$. It has to be noted that the brain connectivity datasets, OHSU, KKI, and PEKING have similar weight settings with View-II weights being prominent compared to others. In the case of the social media dataset which is comparatively larger than the other domain, view-IV weight is prominent followed by view-III. This is similar to the observation of weight pattern for larger datasets in chemoinformatics domain namely NCI1 and NCI109.

**Table 7.2:** Accuracy of the multi-view graph embedding along with state-of-the-art embedding techniques and graph kernels in brain connectivity and social media data.

| **Methods** | **OHSU** | **KKI** | **PEKING** | **IMDB-BIN.** | **REDD-BIN.** |
|---|---|---|---|---|---|
| Graph2vec | $54.81 \pm 10.47$ | $52.61 \pm 10.57$ | $56.81 \pm 11.47$ | $68.81 \pm 10.57$ | $55.82 \pm 8.73$ |
| GE-FSG | $53.17 \pm 0.04$ | $51.18 \pm 0.02$ | $55.64 \pm 0.03$ | $70.48 \pm 0.01$ | $59.38 \pm 0.01$ |
| WL subtree | $57.06 \pm 9.82$ | $53.90 \pm 8.45$ | $58.12 \pm 7.27$ | $73.79 \pm 2.15$ | $\mathbf{62.91 \pm 2.10}$ |
| TK+MKL | $52.47 \pm 8.84$ | $50.45 \pm 8.49$ | $53.28 \pm 9.42$ | $65.76 \pm 2.59$ | $54.27 \pm 3.84$ |
| Multi-view | $\mathbf{58.89 \pm 8.39}$ | $\mathbf{55.13 \pm 8.78}$ | $56.83 \pm 8.30$ | $\mathbf{75.86 \pm 1.85}$ | $60.\ 57 \pm 1.94$ |
| OA kernel | $57.66 \pm 7.19$ | $54.27 \pm 8.83$ | $\mathbf{59.22 \pm 7.65}$ | $75.43 \pm 2.12$ | $61.\ 58 \pm 2.05$ |

For OHSU, and KKI datasets, the best result is obtained for multi-view method while in PEKING it is the OA method. In the case of social media datasets, multi-view has better result for IMDB-BINARY while in the case of REDDIT-BINARY it is WL subtree kernel.

**Table 7.3:** Kernel weights of different views in MKL setting

| **Kernel** | **View-I** | **View-II** | **View-III** | **View-IV** |
|---|---|---|---|---|
| OHSU | 0.00 | 0.50 | 0.25 | 0.25 |
| KKI | 0.01 | 0.51 | 0.26 | 0.22 |
| PEKING | 0.03 | 0.49 | 0.27 | 0.21 |
| IMDB-BIN. | 0.00 | 0.00 | 0.19 | 0.81 |
| REDD-BIN. | 0.00 | 0.00 | 0.15 | 0.85 |

## 7.2 Location-wise spread of Covid-19

The proposed SGCN filters are used for predicting the number of Covid-19 outbreaks geographically, that is, if we are supplied with the data of number of Covid-19 outbreaks in specific areas, the number of outbreaks in the surrounding areas is predicted.

### 7.2.1 Dataset

We attempted to predict the spread of Covid-19 in the counties of New-Jersey, USA for the study. The data is taken from the internet. The county wise count of number of Covid-19 patients from 30 March 2020 to 04 April 2020 is taken for the modeling. The spread of 8 counties on 4th April is predicted using the past data of all counties till 03rd April 2020 and using the data of remaining 13 counties on 4th April. The objective of the case study is to predict the spread of Covid-19 across a region given the information of the surrounding regions.

#### 7.2.1.1 Data construction procedures

Nodes of the graph represent the counties. A link between two counties is made if the distance between the head quarters of the counties is less than 75 miles. The nodes colored blue are used for prediction of the model and the nodes colored red are used to train the machine learning model. An example is shown in the Figure 7.1.

For each node, a set of information is assumed for building the model. The nodes contain number of patients from $30^{th}$ March to $03^{rd}$ April and the population density of the county. Each county was classified into one among four groups –

1. If number of patients less than 100 - class A.

2. If number of patients between 100 and 1000 - class B.

3. If number of patients between 1000 and 2600 - class C.

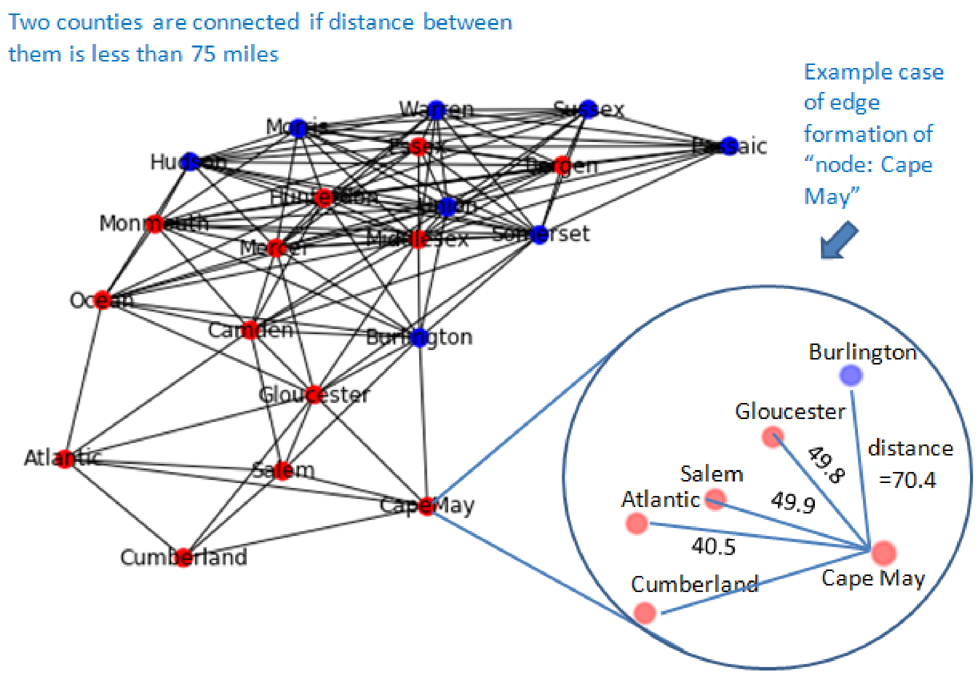


**Figure 7.1:** Procedure in constructing edges of the graph. Red nodes indicate training points and blue nodes testing points.

4. If number of patients greater than 2600 - class D.

#### 7.2.1.2 Results

Out of 8 counties used for the prediction, 7 were accurately predicted. The best result was with the *diffusion* filter.

# Chapter 8

# Conclusions and Future Works

The data in the form of graphs have wide applications in many domains of science and technology. Four methods were developed for analysing different types of graph data and their efficiency was tested in the classification tasks. These methods are also equally applicable in the case of regression as well.

In the first approach, graph embedding techniques were formulated using a collection of graph properties such as edges, shortest paths, and subtree patterns. Each embedding was considered as a separate view of the graph and they were jointly optimized in a MKL framework. The individual views were assigned weights and they were helpful to analyze the effectiveness of the corresponding embedding process. The representation capability of each view was analyzed by designing an R-convolution kernel and effectiveness of multi-view approach was empirically verified.

In the second approach, kernels were designed using the relatively less explored area viz optimal assignment kernel framework. The kernels were formulated as an assignment problem between the nodes of the argument graphs and efficient computation algorithms were developed using the hierarchy concept in the OA framework. We compared the kernels based on the R-convolution and the OA frameworks and made useful observations. The first two methods are applicable to a dataset that has a collection of graph in which each having a set of node and edge labels.

In the third approach, graph kernels were developed for attributed graphs. A product graph is formulated from which the structurally similar regions of argument graphs can be found out in terms of the edges that preserve the neighborhood. A kernel was proposed that utilize the label and attribute information separately and was found to be effective. Methods were proposed to compute the kernel value in the WL iterations recursively from the product graph. The proposed methods were applied in image classification and gave promising results.

The fourth approach is for the graph data that is in the form of a large network whose nodes are needed to be classified. For this, a framework was formulated to design filters of spectral graph convolution neural networks based on the regularization theory in graphs characterized by its Laplacian. We proposed a new set of filters, identified the state-of-the-arts as its special cases and formulated the relation between the filters and support vector kernels. The state-of-the-art architecture of SGCNs were reviewed in this context and deduced useful directions for its further improvement.

## 8.1 Future Works

Based on our contribution to the field, there are a few related works that can be done. They are briefly discussed in this section.

### 8.1.1 Optimal assignment kernel for attributed graphs

The proposed OA kernel designs based on node assignment can be extended to the case of attributed graphs. In this case, efficient base kernels that satisfy the strong kernel criterion has to be formed for processing the vector information.

The challenge in designing OA kernels is in the design of base kernels. In the case of attributed graphs, additional complexity is involved as the base kernel needs to process the vector information. Hence an explicit sorting of kernel values will be required to make a strong kernel to satisfy (2.4) which makes the kernel design costly. With an efficient computation strategy to avoid this sorting can enable application of OA kernels in the case of attributed graphs.

### 8.1.2 Exploring neighborhood preserving property in image processing

The concept of neighborhood preserving property can be useful in images in which the objects can be distinguished with the neighborhood structures. In this direction, methods can be developed to convert an image to graph, and computationally efficient feature extraction methods.

It is possible to convert grayscale digit images into a graph structure as explained in Section 5.3.7. But advanced techniques are required to convert a color picture into a graph structure. For this purpose, techniques like image segmentation built on top of graph partitioning problem can be considered. To define the attribute information over the nodes also require further research.

### 8.1.3 Introducing a regularizer term in SGCN loss function

The state-of-the-art SGCN architectures are optimized only for the loss related to the learning task. However, the filtering characteristics of the filter used also need to be considered and for this purpose a regularizer term can be introduced.

The filter in existing SGCNs are chosen without considering their filtering characteristics. Instead of this, they are chosen as a localization and information aggregating mechanism over the nodes. The network learning is having a direct influence on the filtering properties but the loss function optimized concerns only about the downstream learning task. So there lies a possibility of adding an explicit regularizer that can make a trade-off between desired filtering behavior and loss function optimization for the learning task.

### 8.1.4 Filter banks and attention mechanism to improve SGCN architecture

This is an alternative to address the requirement of regularizer term explained in Section 8.1.3. Since it is difficult to find the desired filtering characteristic required for a learning problem, a collection of filters each having unique filtering properties can be employed. The domain knowledge can also be incorporated in choosing the right filter combinations.

The filter bank can also be formulated in an attention mechanism where the individual node embedding may be attended from the output of multiple filters. These output are also some embeddings in their own capacity and hence each nodes will have a collection of embeddings. To find the final embedding, this collection may be incorporated into an appropriate attention mechanism.

# Bibliography

[1] R. R. Dipert, "The mathematical structure of the world: The world as graph," *The Journal of Philosophy*, vol. 94, no. 7, pp. 329–358, 1997.

[2] D. Haussler, "Convolution kernels on discrete structures," Citeseer, Tech. Rep., 1999.

[3] S. Sun, "A survey of multi-view machine learning," *Neural Computing and Applications*, vol. 23, no. 7-8, pp. 2031–2038, 2013.

[4] C. Xu, D. Tao, and C. Xu, "A survey on multi-view learning," *arXiv preprint arXiv:1304.5634*, 2013.

[5] N. M. Kriege, P.-L. Giscard, and R. Wilson, "On valid optimal assignment kernels and applications to graph classification," in *Advances in Neural Information Processing Systems*, 2016, pp. 1623–1631.

[6] A. Rakotomamonjy, F. R. Bach, S. Canu, and Y. Grandvalet, "Simplemkl," *Journal of Machine Learning Research*, vol. 9, no. Nov, pp. 2491–2521, 2008.

[7] K. M. Borgwardt and H.-P. Kriegel, "Shortest-path kernels on graphs," in *Fifth IEEE International Conference on Data Mining (ICDM'05)*. IEEE, 2005, pp. 8–pp.

[8] N. Kriege and P. Mutzel, "Subgraph matching kernels for attributed graphs," in *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, 2012, pp. 1015–1022.

[9] H. Fröhlich, J. K. Wegner, F. Sieker, and A. Zell, "Optimal assignment kernels for attributed molecular graphs," in *Proceedings of the 22nd international conference on Machine learning*. ACM, 2005, pp. 225–232.

[10] B. Weisfeiler and A. Leman, "The reduction of a graph to canonical form and the algebra which appears therein," *NTI, Series*, vol. 2, 1968.

[11] S. F. Mousavi, M. Safayani, A. Mirzaei, and H. Bahonar, “Hierarchical graph embedding in vector space by graph pyramid,” *Pattern Recognition*, vol. 61, pp. 245–254, 2017.

[12] K. Riesen and H. Bunke, “Graph classification based on vector space embedding,” *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 23, no. 06, pp. 1053–1081, 2009.

[13] J. Gibert, E. Valveny, and H. Bunke, “Graph embedding in vector spaces by node attribute statistics,” *Pattern Recognition*, vol. 45, no. 9, pp. 3072–3083, 2012.

[14] M. M. Luqman, J.-Y. Ramel, J. Lladós, and T. Brouard, “Fuzzy multilevel graph embedding,” *Pattern Recognition*, vol. 46, no. 2, pp. 551–565, 2013.

[15] N. Sidère, P. Héroux, and J.-Y. Ramel, “A vectorial representation for the indexation of structural informations,” in *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*. Springer, 2008, pp. 45–54.

[16] R. C. Wilson, E. R. Hancock, and B. Luo, “Pattern vectors from algebraic graph theory,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 7, pp. 1112–1124, 2005.

[17] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[18] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” *arXiv preprint arXiv:1301.3781*, 2013.

[19] Q. Le and T. Mikolov, “Distributed representations of sentences and documents,” in *International Conference on Machine Learning*, 2014, pp. 1188–1196.

[20] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal, “graph2vec: Learning distributed representations of graphs,” *arXiv preprint arXiv:1707.05005*, 2017.

[21] A. Narayanan, M. Chandramohan, L. Chen, Y. Liu, and S. Saminathan, “subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs,” *arXiv preprint arXiv:1606.08928*, 2016.

[22] A. Grover and J. Leskovec, “node2vec: Scalable feature learning for networks,” in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2016, pp. 855–864.

[23] D. Nguyen, W. Luo, T. D. Nguyen, S. Venkatesh, and D. Phung, “Learning graph representation via frequent subgraphs,” in *Proceedings of the 2018 SIAM International Conference on Data Mining*. SIAM, 2018, pp. 306–314.

[24] P. Yanardag and S. Vishwanathan, “Deep graph kernels,” in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 1365–1374.

[25] N. Shervashidze, P. Schweitzer, E. J. v. Leeuwen, K. Mehlhorn, and K. M. Borgwardt, “Weisfeiler-lehman graph kernels,” *Journal of Machine Learning Research*, vol. 12, no. Sep, pp. 2539–2561, 2011.

[26] R. I. Kondor and J. Lafferty, “Diffusion kernels on graphs and other discrete input spaces,” in *ICML*, vol. 2, 2002, pp. 315–322.

[27] A. J. Smola and R. Kondor, “Kernels and regularization on graphs,” in *Learning theory and kernel machines*. Springer, 2003, pp. 144–158.

[28] A. J. Smola, B. Schölkopf, and K.-R. Müller, “The connection between regularization operators and support vector kernels,” *Neural networks*, vol. 11, no. 4, pp. 637–649, 1998.

[29] T. Gärtner, P. Flach, and S. Wrobel, “On graph kernels: Hardness results and efficient alternatives,” in *Learning Theory and Kernel Machines*. Springer, 2003, pp. 129–143.

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

[31] H. Kashima, K. Tsuda, and A. Inokuchi, “Marginalized kernels between labeled graphs,” in *ICML*, vol. 3, 2003, pp. 321–328.

[32] P. Mahé, N. Ueda, T. Akutsu, J.-L. Perret, and J.-P. Vert, “Extensions of marginalized graph kernels,” in *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004, p. 70.

[33] H. Morgan, "The generation of a unique machine description for chemical structures-a technique developed at chemical abstracts service." *Journal of Chemical Documentation*, vol. 5, no. 2, pp. 107–113, 1965.

[34] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H.-P. Kriegel, "Protein function prediction via graph kernels," *Bioinformatics*, vol. 21, no. suppl_1, pp. i47–i56, 2005.

[35] Z. Zhang, M. Wang, Y. Xiang, Y. Huang, and A. Nehorai, "Retgk: Graph kernels based on return probabilities of random walks," in *Advances in Neural Information Processing Systems*, 2018, pp. 3964–3974.

[36] A. Feragen, N. Kasenburg, J. Petersen, M. de Bruijne, and K. Borgwardt, "Scalable kernels for graphs with continuous attributes," in *Advances in Neural Information Processing Systems*, 2013, pp. 216–224.

[37] J. Ramon and T. Gärtner, "Expressivity versus efficiency of graph kernels," in *First international workshop on mining graphs, trees and sequences*. Citeseer, 2003, pp. 65–74.

[38] P. Mahé and J.-P. Vert, "Graph kernels based on tree patterns for molecules," *Machine learning*, vol. 75, no. 1, pp. 3–35, 2009.

[39] L. Bai, L. Rossi, Z. Zhang, and E. Hancock, "An aligned subtree kernel for weighted graphs," in *International Conference on Machine Learning*, 2015, pp. 30–39.

[40] G. Da San Martino, N. Navarin, and A. Sperduti, "Tree-based kernel for graphs with continuous attributes," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 7, pp. 3270–3276, 2017.

[41] W. Ye, Z. Wang, R. Redberg, and A. Singh, "Tree++: Truncated tree based graph kernels," *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[42] T. Horváth, T. Gärtner, and S. Wrobel, "Cyclic pattern kernels for predictive graph mining," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2004, pp. 158–167.

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

[44] F. Aziz, A. Ullah, and F. Shah, "Feature selection and learning for graphlet kernel," *Pattern Recognition Letters*, 2020.

[45] F. Costa and K. De Grave, "Fast neighborhood subgraph pairwise distance kernel," in *Proceedings of the 26th International Conference on Machine Learning*. Omnipress, 2010, pp. 255–262.

[46] F. Orsini, P. Frasconi, and L. De Raedt, "Graph invariant kernels," in *Proceedings of the 24th International Conference on Artificial Intelligence*. AAAI Press, 2015, pp. 3756–3762.

[47] L. Bai and E. R. Hancock, "Fast depth-based subgraph kernels for unattributed graphs," *Pattern Recognition*, vol. 50, pp. 233–245, 2016.

[48] F. Johansson, V. Jethava, D. Dubhashi, and C. Bhattacharyya, "Global graph kernels using geometric embeddings," in *Proceedings of the 31st International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, 2014.

[49] M. Neumann, R. Garnett, C. Bauckhage, and K. Kersting, "Propagation kernels: efficient graph kernels from propagated information," *Machine Learning*, vol. 102, no. 2, pp. 209–245, 2016.

[50] B. Gaüzere, L. Brun, and D. Villemin, "Two new graphs kernels in chemoinformatics," *Pattern Recognition Letters*, vol. 33, no. 15, pp. 2038–2047, 2012.

[51] L. Bai, L. Rossi, A. Torsello, and E. R. Hancock, "A quantum jensen–shannon graph kernel for unattributed graphs," *Pattern Recognition*, vol. 48, no. 2, pp. 344–355, 2015.

[52] L. Xu, X. Jiang, L. Bai, J. Xiao, and B. Luo, "A hybrid reproducing graph kernel based on information entropy," *Pattern Recognition*, vol. 73, pp. 89–98, 2018.

[53] L. Xu, L. Bai, X. Jiang, M. Tan, D. Zhang, and B. Luo, "Deep rényi entropy graph kernel," *Pattern Recognition*, p. 107668, 2020.

[54] B. Gaüzere, P.-A. Grenier, L. Brun, and D. Villemin, "Treelet kernel incorporating cyclic, stereo and inter pattern information in chemoinformatics," *Pattern Recognition*, vol. 48, no. 2, pp. 356–367, 2015.

[55] M. Togninalli, E. Ghisu, F. Llinares-López, B. Rieck, and K. Borgwardt, "Wasserstein weisfeiler-lehman graph kernels," in *Advances in Neural Information Processing Systems*, 2019, pp. 6439–6449.

[56] R. Kondor and H. Pan, "The multiscale laplacian graph kernel," in *Advances in Neural Information Processing Systems*, 2016, pp. 2990–2998.

[57] B. Rieck, C. Bock, and K. Borgwardt, "A persistent weisfeiler-lehman procedure for graph classification," in *International Conference on Machine Learning*, 2019, pp. 5448–5458.

[58] M. Neumann, N. Patricia, R. Garnett, and K. Kersting, "Efficient graph kernels by randomization," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2012, pp. 378–393.

[59] C. Morris, N. M. Kriege, K. Kersting, and P. Mutzel, "Faster kernels for graphs with continuous attributes via hashing," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 2016, pp. 1095–1100.

[60] F. Aiolli, M. Donini, N. Navarin, and A. Sperduti, "Multiple graph-kernel learning," in *2015 IEEE Symposium Series on Computational Intelligence*, 2015, pp. 1607–1614.

[61] M. Donini, N. Navarin, I. Lauriola, F. Aiolli, and F. Costa, "Fast hyperparameter selection for graph kernels via subsampling and multiple kernel learning," in *ESANN*, 2017.

[62] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," *arXiv preprint arXiv:1312.6203*, 2013.

[63] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," *arXiv preprint arXiv:1506.05163*, 2015.

[64] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in neural information processing systems*, 2016, pp. 3844–3852.

[65] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[66] B. Xu, H. Shen, Q. Cao, K. Cen, and X. Cheng, "Graph convolutional networks using heat kernel for semi-supervised learning," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. AAAI Press, 2019, pp. 1928–1934.

[67] Q. Li, X.-M. Wu, H. Liu, X. Zhang, and Z. Guan, "Label efficient semi-supervised learning via graph filtering," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9582–9591.

[68] B. Xu, H. Shen, Q. Cao, Y. Qiu, and X. Cheng, "Graph wavelet neural network," *arXiv preprint arXiv:1904.07785*, 2019.

[69] R. Li, S. Wang, F. Zhu, and J. Huang, "Adaptive graph convolutional neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[70] C. Zhuang and Q. Ma, "Dual graph convolutional networks for graph-based semi-supervised classification," in *Proceedings of the 2018 World Wide Web Conference*, 2018, pp. 499–508.

[71] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein, "Cayleynets: Graph convolutional neural networks with complex rational spectral filters," *IEEE Transactions on Signal Processing*, vol. 67, no. 1, pp. 97–109, 2018.

[72] R. Liao, Z. Zhao, R. Urtasun, and R. S. Zemel, "Lanczosnet: Multi-scale deep graph convolutional networks," *arXiv preprint arXiv:1901.01484*, 2019.

[73] M. Belkin, P. Niyogi, and V. Sindhwani, "Manifold regularization: A geometric framework for learning from labeled and unlabeled examples," *Journal of machine learning research*, vol. 7, no. Nov, pp. 2399–2434, 2006.

[74] M. Belkin, I. Matveeva, and P. Niyogi, "Regularization and semi-supervised learning on large graphs," in *International Conference on Computational Learning Theory*. Springer, 2004, pp. 624–638.

[75] D. Zhou and B. Schölkopf, "A regularization framework for learning from graph data," in *ICML 2004 Workshop on Statistical Relational Learning and Its Connections to Other Fields (SRL 2004)*, 2004, pp. 132–137.

[76] X. Zhu, Z. Ghahramani, and J. D. Lafferty, “Semi-supervised learning using gaussian fields and harmonic functions,” in *Proceedings of the 20th International conference on Machine learning (ICML-03)*, 2003, pp. 912–919.

[77] J. Weston, F. Ratle, H. Mobahi, and R. Collobert, “Deep learning via semi-supervised embedding,” in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 639–655.

[78] H. Nt and T. Maehara, “Revisiting graph neural networks: All we have is low-pass filters,” *arXiv preprint arXiv:1905.09550*, 2019.

[79] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, “Simplifying graph convolutional networks,” in *Proceedings of the 36th International Conference on Machine Learning*, 2019, pp. 6861–6871.

[80] J. Klicpera, S. Weißenberger, and S. Günnemann, “Diffusion improves graph learning,” in *Advances in Neural Information Processing Systems*, 2019, pp. 13 333–13 345.

[81] S. Li, D. Kim, and Q. Wang, “Beyond low-pass filters: Adaptive feature propagation on graphs,” *arXiv preprint arXiv:2103.14187*, 2021.

[82] M. Balcilar, G. Renton, P. Héroux, B. Gaüzère, S. Adam, and P. Honeine, “Analyzing the expressive power of graph neural networks in a spectral perspective,” in *International Conference on Learning Representations*, 2020.

[83] H. Chang, Y. Rong, T. Xu, W. Huang, S. Sojoudi, J. Huang, and W. Zhu, “Spectral graph attention network,” *arXiv preprint arXiv:2003.07450*, 2020.

[84] D. Bo, X. Wang, C. Shi, and H. Shen, “Beyond low-frequency information in graph convolutional networks,” *arXiv preprint arXiv:2101.00797*, 2021.

[85] G. Fu, Y. Hou, J. Zhang, K. Ma, B. F. Kamhoua, and J. Cheng, “Understanding graph neural networks from graph signal denoising perspectives,” *arXiv preprint arXiv:2006.04386*, 2020.

[86] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, “Graph attention networks,” *arXiv preprint arXiv:1710.10903*, 2017.

[87] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *Thirty-Second AAAI conference on artificial intelligence*, 2018.

[88] X. Zhang, H. Liu, Q. Li, and X.-M. Wu, "Attributed graph clustering via adaptive graph convolution," *arXiv preprint arXiv:1906.01210*, 2019.

[89] F. Gama, J. Bruna, and A. Ribeiro, "Stability properties of graph neural networks," *arXiv preprint arXiv:1905.04497*, 2019.

[90] R. Kaspar and B. Horst, *Graph classification and clustering based on vector space embedding*. World Scientific, 2010, vol. 77.

[91] B. Schölkopf, A. J. Smola, F. Bach *et al.*, *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[92] J. Shawe-Taylor and N. Cristianini, *Kernel methods for pattern analysis*. Cambridge university press, 2004.

[93] H. Feng and T.-S. Chua, "A bootstrapping approach to annotating large image collection," in *Proceedings of the 5th ACM SIGMM international workshop on Multimedia information retrieval*, 2003, pp. 55–62.

[94] Y. Li, B. Geng, Z.-J. Zha, D. Tao, L. Yang, and C. Xu, "Difficulty guided image retrieval using linear multiview embedding," in *Proceedings of the 19th ACM international conference on Multimedia*, 2011, pp. 1169–1172.

[95] H. Feng, R. Shi, and T.-S. Chua, "A bootstrapping framework for annotating and retrieving www images," in *Proceedings of the 12th annual ACM international conference on Multimedia*, 2004, pp. 960–967.

[96] A. Salim, S. Shiju, and S. Sumitra, "Effectiveness of representation and length variation of shortest paths in graph classification," in *International Conference on Pattern Recognition and Machine Intelligence*. Springer, 2017, pp. 509–516.

[97] C.-C. Chang and C.-J. Lin, "Libsvm: a library for support vector machines," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3, p. 27, 2011.

[98] A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch, “Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity,” *Journal of medicinal chemistry*, vol. 34, no. 2, pp. 786–797, 1991.

[99] H. Toivonen, A. Srinivasan, R. D. King, S. Kramer, and C. Helma, “Statistical evaluation of the predictive toxicology challenge 2000–2001,” *Bioinformatics*, vol. 19, no. 10, pp. 1183–1193, 2003.

[100] I. Schomburg, A. Chang, C. Ebeling, M. Gremse, C. Heldt, G. Huhn, and D. Schomburg, “Brenda, the enzyme database: updates and major new developments,” *Nucleic acids research*, vol. 32, no. suppl_1, pp. D431–D433, 2004.

[101] P. D. Dobson and A. J. Doig, “Distinguishing enzyme structures from non-enzymes without alignments,” *Journal of molecular biology*, vol. 330, no. 4, pp. 771–783, 2003.

[102] N. Wale, I. A. Watson, and G. Karypis, “Comparison of descriptor spaces for chemical compound retrieval and classification,” *Knowledge and Information Systems*, vol. 14, no. 3, pp. 347–375, 2008.

[103] C. Cortes, P. Haffner, and M. Mohri, “Rational kernels: Theory and algorithms,” *Journal of Machine Learning Research*, vol. 5, no. Aug, pp. 1035–1062, 2004.

[104] J. J. Sutherland, L. A. O’brien, and D. F. Weaver, “Spline-fitting with a genetic algorithm: A method for developing classification structure- activity relationships,” *Journal of chemical information and computer sciences*, vol. 43, no. 6, pp. 1906–1915, 2003.

[105] K. Kersting, N. M. Kriege, C. Morris, P. Mutzel, and M. Neumann, “Benchmark data sets for graph kernels,” 2016. [Online]. Available: http://graphkernels.cs.tu-dortmund.de

[106] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.

[107] D. Grattarola and C. Alippi, “Graph neural networks in tensorflow and keras with spektral [application notes],” *Comp. Intell. Mag.*, vol. 16, no. 1, p. 99–106, Feb. 2021. [Online]. Available: https://doi.org/10.1109/MCI.2020.3039072

[108] F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model cnns," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017, pp. 5425–5434.

[109] F. R. Chung, *Spectral graph theory*. American Mathematical Soc., 1997, no. 92.

[110] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE signal processing magazine*, vol. 30, no. 3, pp. 83–98, 2013.

[111] F. Girosi, M. Jones, and T. Poggio, "Regularization theory and neural networks architectures," *Neural computation*, vol. 7, no. 2, pp. 219–269, 1995.

[112] R. A. Willoughby, "Solutions of ill-posed problems (an tikhonov and vy arsenin)," *SIAM Review*, vol. 21, no. 2, p. 266, 1979.

[113] X. Zhu and A. B. Goldberg, "Introduction to semi-supervised learning," *Synthesis lectures on artificial intelligence and machine learning*, vol. 3, no. 1, pp. 1–130, 2009.

[114] D. K. Hammond, P. Vandergheynst, and R. Gribonval, "Wavelets on graphs via spectral graph theory," *Applied and Computational Harmonic Analysis*, vol. 30, no. 2, pp. 129–150, 2011.

[115] N. Tremblay, P. Gonçalves, and P. Borgnat, "Chapter 11 - design of graph filters and filterbanks," in *Cooperative and Graph Signal Processing*, P. M. Djurić and C. Richard, Eds. Academic Press, 2018, pp. 299–324. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128136775000110

[116] F. M. Bianchi, D. Grattarola, L. Livi, and C. Alippi, "Graph neural networks with convolutional arma filters," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2021.

[117] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in neural information processing systems*, 2017, pp. 1024–1034.

[118] Z. Yang, W. W. Cohen, and R. Salakhutdinov, “Revisiting semi-supervised learning with graph embeddings,” in *Proceedings of the 33rd International Conference on International Conference on Machine Learning-Volume 48*, 2016, pp. 40–48.

[119] B. Perozzi, R. Al-Rfou, and S. Skiena, “Deepwalk: Online learning of social representations,” in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 701–710.

[120] Q. Lu and L. Getoor, “Link-based classification,” in *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, 2003, pp. 496–503.

[121] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[122] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 249–256.

[123] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, “Tensorflow: Large-scale machine learning on heterogeneous distributed systems,” *arXiv preprint arXiv:1603.04467*, 2016.

[124] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, “Graphsaint: Graph sampling based inductive learning method,” in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=BJe8pkHFwS

[125] B. Rozemberczki, O. Kiss, and R. Sarkar, “Karate club: An api oriented open-source python framework for unsupervised learning on graphs,” ser. CIKM ’20. New York, NY, USA: Association for Computing Machinery, 2020, p. 3125–3132. [Online]. Available: https://doi.org/10.1145/3340531.3412757

[126] B. Rozemberczki and R. Sarkar, “Characteristic functions on graphs: Birds of a feather, from statistical descriptors to parametric models,” ser. CIKM ’20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1325–1334. [Online]. Available: https://doi.org/10.1145/3340531.3411866

[127] B. Rozemberczki, C. Allen, and R. Sarkar, “Multi-Scale attributed node embedding,” *Journal of Complex Networks*, vol. 9, no. 2, 05 2021, cnab014. [Online]. Available: https://doi.org/10.1093/comnet/cnab014

[128] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[129] H. Yang, K. Ma, and J. Cheng, "Rethinking graph regularization for graph neural networks," *arXiv preprint arXiv:2009.02027*, 2020.

[130] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, "Measuring and relieving the over-smoothing problem for graph neural networks from the topological view," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 3438–3445.

[131] F. Frasca, E. Rossi, D. Eynard, B. Chamberlain, M. Bronstein, and F. Monti, "Sign: Scalable inception graph neural networks," *arXiv preprint arXiv:2004.11198*, 2020.

[132] X. Wang, M. Zhu, D. Bo, P. Cui, C. Shi, and J. Pei, "Am-gcn: Adaptive multi-channel graph convolutional networks," in *Proceedings of the 26th ACM SIGKDD International conference on knowledge discovery & data mining*, 2020, pp. 1243–1253.

[133] X. Gao, W. Dai, C. Li, J. Zou, H. Xiong, and P. Frossard, "Message passing in graph convolution networks via adaptive filter banks," *arXiv preprint arXiv:2106.09910*, 2021.

[134] Y. Wang, Z. Hu, Y. Ye, and Y. Sun, "Demystifying graph neural network via graph filter assessment," 2019.

[135] S. Pan, J. Wu, X. Zhu, G. Long, and C. Zhang, "Task sensitive feature exploration and learning for multitask graph classification," *IEEE Transactions on Cybernetics*, vol. 47, no. 3, pp. 744–758, 2017.

# List of Publications

## Journals

1. Asif Salim, S. S. Shiju, and S. Sumitra. "Neighborhood Preserving Kernels for Attributed Graphs." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45.1 (2022): 828-840.

2. Asif Salim, and S. Sumitra. "Spectral Graph Convolutional Neural Networks in the Context of Regularization Theory." *IEEE Transactions on Neural Networks and Learning Systems* (2022): 1-12, doi:10.1109/TNNLS.2022.3177742

3. Asif Salim, S. S. Shiju, and S. Sumitra. "Graph kernels based on optimal node assignment." *Knowledge-Based Systems* 244 (2022): 108519.

4. Asif Salim, S. S. Shiju, and S. Sumitra. "Design of multi-view graph embedding using multiple kernel learning." *Engineering Applications of Artificial Intelligence* 90 (2020): 103534.

## Conference

1. Asif Salim, Shiju S. S., and S. Sumitra. "Effectiveness of representation and length variation of shortest paths in graph classification." *In International Conference on Pattern Recognition and Machine Intelligence*, pp. 509-516. Springer, Cham, 2017.

# Appendix A

# From Chapter 4

## A.1 Proof of optimal assignment of $\tilde{K}_{ONA}(G, G')$

**Theorem A.1.** $\tilde{K}_{ONA}(G, G') = \sum\limits_{l_i \in \Sigma_{WL}} k_{ns_2}(g_{l_i}, g'_{l_i})$ *is an optimal assignment kernel.*

*Proof.* First we prove that corresponding to a WL label $l_i$, $k_{ns_2}(g_{li}, g'_{li})$ is an optimal assignment kernel. Consider,

$$k_{ns_2}(g_{l_i}, g'_{l_i}) = \sum_i \left\langle (M_{g_{l_i}}(i,.))^T, M'_{g_{l_i}}(i,.)^T \right\rangle$$

Define the bijective function

$$B_{l_i} : g_{l_i} \to g'_{l_i}, \text{ where } B_{l_i}(v) = v' \text{ if } \sigma(v) = \sigma(v')$$

With this $k_{ns_2}$ can be written as,

$$k_{ns_2}(g_{l_i}, g'_{l_i}) = \sum_{(v, B_{l_i}(v))} \left\langle \mathcal{V}_v, \mathcal{V}_{B_{l_i}(v)} \right\rangle = \sum_{(v, B_{l_i}(v))} k_{l_i}(v, B_{l_i}(v))$$

where $\mathcal{V}_v$, $\mathcal{V}_{B_{l_i}(v)}$ are the vector representation as explained in Section 4.1.2.2 and for each $l_i \in \Sigma_{WL}$, $k_{l_i} : g_{l_i} \times g'_{l_i} \to \mathbb{R}$ is a valid kernel. $k_{l_i}$ is a strong kernel since its range set $\left\{0, \left\langle (M_{g_{l_i}}(i,.))^T, M'_{g_{l_i}}(i,.)^T \right\rangle \right\}$ is of cardinality two [5].

Define,

$$B : \tilde{V} \to \tilde{V}' \text{ where } B(v) = B_{l_i}(v) \text{ if } l_{WL}(v) = l_i$$

.

$B$ is a bijection as each $B_{l_i}$s are bijective functions. Now

$$\tilde{K}_{ONA}(G, G') = \sum_{(v, B_(v))} \left\langle \mathcal{V}_v, \mathcal{V}_{B_{l_i}(v)} \right\rangle .$$

Hence $\tilde{K}_{ONA}(G, G')$ is an optimal assignment kernel.

□

## A.2 Computing $\tilde{K}_{ONA}$ using hierarchy

**Theorem A.2.** $\tilde{K}_T(G, G') = \tilde{K}_{ONA}(G, G') \; \forall \; G, G' \in \mathcal{G}$.

*Proof.*

$$\begin{aligned}\tilde{K}_T(G, G') &= \sum_{i=1}^{|V_T|} \min\left(G_V(i), G'_V(i)\right) \times \left(N_T(i)\right)^2 \\ &= \sum_{i=1}^{|V_T|} \min\left(G_V(i), G'_V(i)\right) \times \|\mathcal{V}_i\|^2\end{aligned}$$

where $\mathcal{V}_i$ is the vector representation of the WL label corresponding to $i^{th}$ node in $T$ as explained in Section 4.1.2.2.

$G_V(i)$, and $G'_V(i)$ are equal to the number of non-dummy nodes in $g_i$, and $g'_i$ respectively. Therefore,

$$\min\left(G_V(i), G'_V(i)\right) \times \|\mathcal{V}_i\|^2 = \sum_k \left\langle (M_{g_i}(k,.)), (M'_{g_i}(k,.)) \right\rangle$$

where $(M_{g_i}(k,.)$ and $(M'_{g_i}(k,.))$ are the $k^{th}$ row of $M_{g_i}$ and $M'_{g_i}$ respectively.

Hence

$$\sum_{i=1}^{|V_T|} \min\left(G_V(i), G'_V(i)\right) \times \|\mathcal{V}_i\|^2 = \sum_{i=1}^{|V_T|} \sum_k \left\langle (M_{g_i}(k,.)), (M'_{g_i}(k,.)) \right\rangle$$

Now $V_T = \bigcup_{j=1}^{h} \bigcup_{i=1}^{\left|\Sigma_{WL}^j\right|} \{l_i\}$, where $l_i$ is the $i^{th}$ element of $\Sigma_{WL}^j$. Therefore

$$\begin{aligned}\sum_{i=1}^{|V_T|}\sum_{k}\big\langle(M_{g_i}(k,.)),(M'_{g_i}(k,.))\big\rangle &= \sum_{j=1}^{h}\sum_{l_i\in\Sigma_{WL}^{j}}\sum_{k}\big\langle(M_{g_i}(k,.)),(M'_{g_i}(k,.))\big\rangle \\ &= \sum_{j=1}^{h}\sum_{l_i\in\Sigma_{WL}^{j}} k_{ns_2}(g_{l_i},g'_{l_i}) = \tilde{K}_{ONA}(G,G').\end{aligned}$$

□

# Appendix B

# From Chapter 6

## B.1 Regularization in graphs, support vector kernels and spectral GCNN filters

The support vector kernel $k : X \times X \to \mathbb{R}$ is considered as a similarity measure between a pair of data points in a space $X$. Support vector kernels can be formulated by solving the self-consistency condition ([28]), $\langle k(x,.), Pk(x',.)\rangle = k(x,x')$ where $P$ is the regularization operator. Analyzing this, Smola et.al [28] found that given a regularization operator $P$, there exist a support vector kernel $k$ that minimize the regularized risk functional,

$$R_{reg}[f] = R_{emp} + \frac{\lambda}{2}\|Pf\|^2, \tag{B.1}$$

that also enforce flatness (determined by $P$) in the feature space or Reproducing Kernel Hilbert Space (RKHS) of functions. They also found that given a support vector kernel $k$, regularization operator $P$ can be found out such that a regularization network [28] using $P$ is equivalent to a support vector machine that uses the kernel $k$. Note that $R_{emp}$ is the empirical loss function and $\lambda$ is a hyper-parameter.

Smola et.al [27] used the above concepts to design support vector kernels on graphs. As shown in (6.2), graph Laplacian ($\tilde{L}$) can be used to define a smoothness functional on graphs that aids in designing regularization operators. They proved that *if $H$ is the image of $\mathbb{R}^n$ under $P \in \mathbb{R}^{n\times n}$ (a positive semidefinite regularization matrix), then $H$ whose dot product is defined as $\langle f, Pf\rangle$ is a RKHS and the corresponding support vector kernel is defined as $k(i,j) = [P^{-1}]_{ij}$, where $P^{-1}$ denotes the psuedo-inverse if $P$ is not invertible.*

Now if we consider the case of SGCN filters, we can observe that if the parameters $\theta$s associated with the filter definition maintains positive definiteness of the matrix $\mathcal{F} = \big(r(\tilde{L})\big)^{-1}$, then the filter can be considered as a valid and equivalent support vector kernel that solves the regularized risk functional in (B.1). The regularization behavior induced by $P$ in (B.1) can be attributed to the corresponding *regularization function*, $r(\lambda)$.

## B.2 Ablation studies on architectures

The proposed filters and that of the state-of-the-arts have experimented with the following architectures. The base architecture followed is the one proposed by Kipf et.al [65]. It consists of two graph convolution (GC) layers. We tried models with one, two, and three GC layers and also the models with one GC layer followed by one and two dense layers. The details of the models are explained in the following sections. We assume $c_n$ is the number of classes in which the nodes can belong.

**Model with one GC layer:** It takes the form

$$Z = \text{softmax}(\mathcal{F}(\tilde{L})\ \Theta),$$

where $\mathcal{F}(\tilde{L}) \in \mathbb{R}^{n\times n}$ is the filter, $X \in \mathbb{R}^{n\times d}$ is the input feature matrix, $\theta^{(1)} \in \mathbb{R}^{d\times c_1}$ is the filter parameters of the first layer ($c_1$ is the number of filters) and $\theta^{(2)} \in \mathbb{R}^{c_1\times c_n}$ is the filter parameters of the second layer.

**Model with two GC layers:** The model computes

$$Z = \text{softmax}(\mathcal{F}(\tilde{L})\ \text{ReLU}(\mathcal{F}(\tilde{L})X\Theta^{(1)})\Theta^{(2)}),$$

where $\mathcal{F}(\tilde{L}) \in \mathbb{R}^{n\times n}$ is the filter, $X \in \mathbb{R}^{n\times d}$ is the input feature matrix, $\theta^{(1)} \in \mathbb{R}^{d\times c_1}$ is the filter parameters of the first layer ($c_1$ is the number of filters) and $\theta^{(2)} \in \mathbb{R}^{c_1\times c_n}$ is the filter parameters of the second layer.

**Model with three GC layers:** It takes the form

$$Z = \text{softmax}\big[\mathcal{F}(\tilde{L})\big(\text{ReLU}(\mathcal{F}(\tilde{L})O_1\Theta^{(2)})\big)\Theta^{(3)}\big],$$

where $O_1 = \text{ReLU}(\mathcal{F}(\tilde{L})X\Theta^{(1)})$ is the output of first layer in which $\mathcal{F}(\tilde{L}) \in \mathbb{R}^{n\times n}$ is the filter, $X \in \mathbb{R}^{n\times d}$ is the input feature matrix and, $\theta^{(1)} \in \mathbb{R}^{d\times c_1}$ is the filter parameters of first layer ($c_1$ is the number of filters) which is a GC layer. Similarly, $\theta^{(2)} \in \mathbb{R}^{c_1\times c_2}$ is the filter parameters of the second layer ($c_2$ is the number of filters) and $\theta^{(3)} \in \mathbb{R}^{c_2\times c_n}$ is the filter parameters of the third layer.

**Model with one GC layer followed by one dense layer (GD1):** The output of the model is

$$Z = \text{softmax}(\text{ReLU}(\mathcal{F}(\tilde{L})X\Theta^{(1)})\Theta^{(2)}),$$

where $\mathcal{F}(\tilde{L}) \in \mathbb{R}^{n\times n}$ is the filter, $X \in \mathbb{R}^{n\times d}$ is the input feature matrix, $\theta^{(1)} \in \mathbb{R}^{d\times c_1}$ is the filter parameters of the first GC layer ($c_1$ is the number of filters) and $\theta^{(2)} \in \mathbb{R}^{c_1\times c_n}$ is the parameters of the second layer which is dense.

**Model with one GC layer followed by two dense layers (GD2):**

Output is computed as

$$Z = \text{softmax}\Big(\text{ReLU}\big(\text{ReLU}(O_1\Theta^{(2)})\Theta^{(3)}\big)\Big),$$

$O_1 = \text{ReLU}(\mathcal{F}(\tilde{L})X\Theta^{(1)})$ where $\mathcal{F}(\tilde{L}) \in \mathbb{R}^{n\times n}$ is the filter, $X \in \mathbb{R}^{n\times d}$ is the input feature matrix, $\theta^{(1)} \in \mathbb{R}^{d\times c_1}$ is the filter parameters of first layer ($c_1$ is the number of filters) which is GC layer. $\theta^{(2)} \in \mathbb{R}^{c_1\times c_2}$ is the parameters of the second layer and $\theta^{(2)} \in \mathbb{R}^{c_2\times c_n}$ is the parameters of the third layer which are dense.

### B.2.1 Results and observations

The results of the models are tabulated in Table B.1. Note that the results reported are the best after hyper-parameter tuning and tuning the layer size among the set $\{16, 32, 64, 128\}$. The results reported are average of 10 runs corresponding to the random seeds selected at random.

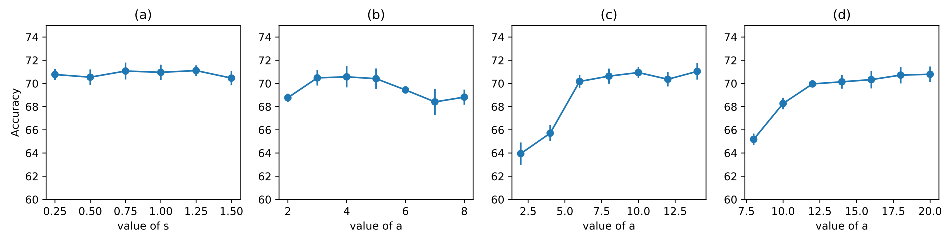


**Figure B.1:** Accuracy variation with hyper-parameters in Citeseer dataset. (a) Diffusion, (b) 1-step RW, (c) 2-step RW, (d) 3-step RW

If we compare the models only with GC layers, we can see that model with two layers of GC performs best (the results reported in the article corresponds to this model). The performance of the three-layer GC model is better than one-layer GC model. The variance of the accuracy of the ChebyNet in the three-layer GC model is very high while that of others is low.

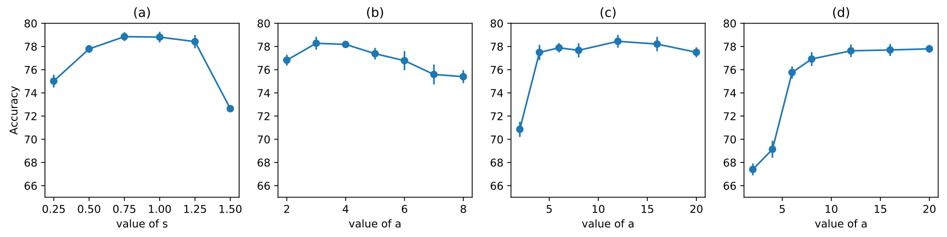


**Figure B.2:** Accuracy variation with hyper-parameters in Pubmed dataset. (a) Diffusion, (b) 1-step RW, (c) 2-step RW, (d) 3-step RW

The performance of the GD1 model is in par with the three-layer GC model for most of the methods but the performance of GD2 is the lowest among all models. Also, the variance of GD2 models is high compared with other models. From the experiments, it can be seen that the model with two layers of GC is the most robust one.

## B.3 Effects of hyper-parameter tuning in Citeseer and Pubmed datasets

For the experiments, the network with two layers of GC having 32 number of filters is used and for *diffusion* filter, the value of $K$ is taken as 3. The analysis in the case of the Cora dataset is already provided in the manuscript.

The variation in accuracy against hyper-parameters of the proposed filters applied to the Citeseer dataset is given in Figure B.1. In the case of *diffusion*, the accuracy increases till a threshold point, and then it decreases slowly. Similar is the trend of variation of parameter $a$ in *1-step RW* filter. In the case of *2-step* and *3-step RW* filters, accuracy increases as the value of $a$ increases but after a threshold point, accuracy variation is minimal. For the Pubmed dataset as well, the observations are similar as shown in Figure B.2.

**Table B.1:** Classification accuracy (in percentage ± standard deviation) for the models.

| Model | Methods | Cora | Citeseer | Pubmed |
|---|---|---|---|---|
| GCN one layer | GCN | 72.59 ± 1.01 | 65.78 ± 1.05 | 72.17 ± 0.49 |
| | ChebyNet | 71.38 ± 0.72 | 62.90 ± 1.38 | 71.57 ± 0.56 |
| | GraphHeat | 64.89 ± 0.63 | 59.94 ± 1.47 | 63.31 ± 0.41 |
| | Diffusion | 77.35 ± 0.79 | 65.56 ± 1.08 | 72.05 ± 0.62 |
| | 1-step RW | 75.18 ± 0.56 | 63.58 ± 1.32 | 71.70 ± 0.53 |
| | 2-step RW | 75.06 ± 0.52 | 62.98 ± 1.37 | 71.11 ± 0.37 |
| | 3-step RW | 65.42 ± 0.34 | 63.02 ± 1.48 | 71.28 ± 0.64 |
| | Cosine | 66.84 ± 0.75 | 59.35 ± 1.74 | 65.79 ± 0.69 |
| GCN two layer | GCN | 81.78 ± 0.64 | 70.73 ± 0.53 | 78.48 ± 0.58 |
| | ChebyNet | 82.16 ± 0.74 | 70.46 ± 0.70 | 78.24 ± 0.43 |
| | GraphHeat | 81.38 ± 0.69 | 69.90 ± 0.50 | 75.64 ± 0.64 |
| | Diffusion | 83.12 ± 0.37 | 71.17 ± 0.43 | 79.20 ± 0.36 |
| | 1-step RW | 82.36 ± 0.34 | 71.05 ± 0.34 | 78.74 ± 0.27 |
| | 2-step RW | 82.51 ± 0.22 | 71.18 ± 0.59 | 78.64 ± 0.20 |
| | 3-step RW | 82.56 ± 0.24 | 71.21 ± 0.63 | 78.28 ± 0.36 |
| | Cosine | 75.53 ± 0.52 | 67.29 ± 0.64 | 75.52 ± 0.53 |
| GCN three layer | GCN | 78.26 ± 6.58 | 67.36 ± 3.85 | 75.14 ± 3.92 |
| | ChebyNet | 64.28 ± 23.48 | 53.57 ± 17.24 | 59.02 ± 14.98 |
| | GraphHeat | 77.89 ± 2.37 | 68.51 ± 1.37 | 74.80 ± 1.32 |
| | Diffusion | 81.01 ± 0.72 | 68.37 ± 1.24 | 76.36 ± 1.28 |
| | 1-step RW | 80.96 ± 0.56 | 68.81 ± 2.05 | 75.76 ± 1.06 |
| | 2-step RW | 81.02 ± 0.71 | 68.85 ± 1.46 | 75.89 ± 1.40 |
| | 3-step RW | 81.12 ± 0.69 | 69.57 ± 1.83 | 76.66 ± 1.23 |
| | Cosine | 75.66 ± 1.65 | 66.07 ± 0.80 | 74.59 ± 1.00 |
| GCN one layer + one dense layer (GD1) | GCN | 79.61 ± 0.62 | 68.27 ± 1.84 | 76.42 ± 0.78 |
| | ChebyNet | 78.73 ± 1.45 | 67.16 ± 0.85 | 74.33 ± 0.34 |
| | GraphHeat | 69.48 ± 0.73 | 61.46 ± 0.94 | 69.52 ± 0.38 |
| | Diffusion | 80.92 ± 0.72 | 70.24 ± 0.88 | 76.04 ± 0.70 |
| | 1-step RW | 79.88 ± 0.60 | 69.34 ± 0.93 | 76.25 ± 0.58 |
| | 2-step RW | 79.21 ± 0.77 | 70.01 ± 0.87 | 76.02 ± 0.44 |
| | 3-step RW | 80.26 ± 0.62 | 70.11 ± 0.90 | 76.81 ± 0.44 |
| | Cosine | 72.61 ± 0.82 | 55.46 ± 1.06 | 72.82 ± 0.76 |
| GCN one layer + two dense layer (GD2) | GCN | 32.64 ± 17.07 | 30.38 ± 6.07 | 67.75 ± 12.69 |
| | ChebyNet | 26.35 ± 4.57 | 31.58 ± 6.39 | 61.30 ± 13.96 |
| | GraphHeat | 28.11 ± 8.22 | 31.33 ± 6.07 | 59.58 ± 13.99 |
| | Diffusion | 48.03 ± 19.42 | 30.73 ± 6.62 | 63.60 ± 11.28 |
| | 1-step RW | 29.00 ± 10.33 | 26.91 ± 6.86 | 64.56 ± 17.43 |
| | 2-step RW | 31.28 ± 12.86 | 29.36 ± 5.92 | 65.04 ± 12.08 |
| | 3-step RW | 33.76 ± 9.39 | 27.79 ± 5.48 | 65.39 ± 11.83 |
| | Cosine | 24.57 ± 5.15 | 28.51 ± 7.48 | 65.48 ± 13.51 |