# Learning on Relational Data: Prototype-Based Classification of Attributed Graphs

vorgelegt von

**M.Sc. (Engg.)**
**Shankar Deepak Srinivasan**
**aus Chennai, Indien**

von der Fakultät IV - Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften
- Dr. rer. nat. -

Genehmigte Dissertation

Tag der wissenschaftlichen Aussprache: 19. August 2011

Promotionsausschuss:
Vorsitzender: Prof. Dr. O. Hellwich
Berichter: Prof. Dr. K. Obermayer
Berichterin: Prof. Dr. B. Hammer

Berlin, 2011
D 83

# Acknowledgements

I thank my *Doktorvater* Prof. Klaus Obermayer for his advice and support. I wish to thank Brijnesh J. Jain for the many discussions, critical arguments and support over the last three years, Johannes Mohr, Prof. Manfred Opper for their comments and insights, Mr. Francois-Xavier Dupé and Prof. Luc Brun for providing the SHAPES dataset. I am grateful to members of the NI group both past and present for providing an atmosphere of support and enthusiasm. Special thanks to Max Göttner for running the experiments on SHAPES and COIL datasets. I would like to thank the efforts of the NI Sekretariat- Ms. Camilla Bruns and Ms. Gabriele Rösler and BCCN administration - especially Ms. Margret Franke, Dr. Vanessa Casagrande and Dr. Daniela Pelz (formerly at BCCN Berlin) in helping me handle all manner of bureaucratic and official issues. The financial support provided by BCCN, Berlin in the form of Doctoral scholarship is gratefully acknowledged.

Finally, nothing would have possible without the love and support of my family. The inadequacy of these words notwithstanding- thank you.

# Contents

# List of Figures

# Chapter 1

# Introduction

## 1.1 Preface

### 1.1.1 Graph based representations

Structural pattern recognition concerns itself with the task of analyzing and learning patterns that are not normally representable by features of a fixed length in an Euclidean space. Instances of such patterns include combinatorial entities such as strings, point patterns, trees and more generally attributed graphs. Such representations occur naturally in many contexts out of which we specifically mention the following

-Bioinformatics, where Protein structures are represented as graphs in terms of their secondary structure elements or as contact maps representing their tertiary structure [6, 67].

-Chemoinformatics, where chemical molecules are represented as attributed graphs with nodes representing atoms along with atomic information as attributes and edges denoting bonds between atoms attributed by edge type and details [48].

-Computer vision, where part based models are used to represent objects as a set of parts with relevant attributes interconnected by edges denoting intra-part relationships. This field of research also provides an important class of problems relating to classifying shapes. Shapes are commonly represented as graphs or trees. Hence, learning tasks on shapes involve pattern analysis of graphs and trees representing them [52, 63].

While graph based representations have enormous potential in terms of the wide variety of patterns they can describe with a significant level of complexity, there are however difficulties associated with handling them, primarily due to the following reasons

-The "space of graphs" has no mathematical structure unlike the space of vectors. This means that even the simplest of operations such as addition of graphs, sample mean of a set of graphs are not easy to define.

-Computational complexity. In order to assess the similarity of two graphs, we need to find a correspondence between their nodes and edges or in some cases, determine if they are isomorphic. It is yet unclear if determining whether two graphs are isomorphic is solvable in Polynomial time or is NP complete. Determining the optimal matching between nodes of two graphs consists of Maximum common subgraph isomorphism problem as a special case. The latter is known to be NP-hard [22]. Hence, we need approximate algorithms which run efficiently while also providing effective solutions. Even then, matching two graphs is computationally intensive [14].

### 1.1.2 Approaches & Challenges

Broadly, there are three schools of thought for developing machine learning and pattern recognition algorithms on combinatorial structures (of which we will take graphs as an instance). The first is graph kernels, which has been quite popular recently [43, 68]. The framework for this idea emerges from the more general convolution kernels proposed by Haussler [32]. A structured object is decomposed into its constituent parts and a kernel between two such objects is defined as the convolution of (positive semi-definite) measures on the constituent parts. Some examples of such kernels include random walk kernel, product graph kernel and marginalized graph kernels. They have been applied successfully to learning problems in bio- and chemo-informatics [5, 51].

Another class of machine learning algorithms could be developed on combinatorial structures using only the notion of distance- a quantity that measures how similar two objects are. It is possible to endow the set of combinatorial structures with a proximity measure. The term proximity is used in a generic manner- it

could refer to similarity, distance or any other measure that quantitatively describes relationship between different objects in the set. For feature vectors, any $\mathcal{L}^n$ metric could be used depending on applications. For strings and sequences, there exists a measure of dissimilarity measured by the edit operations required to transform one element of the space to another. A suitable modification called graph edit distance is proposed for trees and graphs [8, 9].

With a distance measure defined, it is possible to carry out learning tasks directly in the original space of graphs. For instance, to classify the data it is possible to use the k-Nearest Neighbor classifier with the underlying dissimilarity measure. Graepel et. al. [25], propose classification algorithms directly on the proximity values by embedding the data into a pseudo-Euclidean space, where linear classifiers are designed. Related efforts include embedding the original set along with distance measure into Hilbert spaces, Banach spaces and as a special case into Euclidean space itself [69].

The third class of algorithms uses spectral methods to embed graphs into a vector space. Spectral graph theory [12] deals with understanding the connection between structural properties of graphs and the eigensystem of its adjacency matrix $A$ or the closely related Laplacian matrix $L$ defined as $L = D - A$, where $D$ is the diagonal matrix whose entries are given by $D_{ii} = \sum_j A_{ij}$. The Laplacian is positive semi-definite. Hence, it is possible to decompose it as $L = \sum_i \lambda_i e_i e_i^T$, where $\lambda \geq 0$ are the eigenvalues and $e_i$ are the eigenvectors. The eigenvectors are then utilized to embed the graphs into a feature vector space, where clustering and classification algorithms are applied [50, 70].

However, the techniques mentioned above are not without problems. It is not clear how to define graph kernels for graphs with arbitrary feature vector attributes. A similar argument holds for spectral methods as well, which require node and edge attributes to be scalars. The process of defining eigenvalues and eigenvectors is not defined for graphs with arbitrary node and edge attributes. In many applications, the dissimilarity measure considered may not lend itself for embedding into Euclidean spaces. For instance, if the property of symmetry and sub-additivity do not hold for a proximity measure, it is not possible to embed the set into an Euclidean space in a natural fashion. If the space that is embedded into lacks sufficient mathematical structure, then it is not possible to develop a full

repertoire of analytical concepts to develop more extensive algorithms. Moreover, even after a suitable embedding is proposed into another space, there is still the problem of losing the original representations.

Moreover, in applications, there is a need to come up with concepts such as sample mean of a set of combinatorial structures for algorithms such as central clustering, component analysis or external expert interpretation. It is not possible to define these terms using graph kernels, as they can only operate on a pair of graphs without any explicit individual feature representation. An embedding of the data into a simplified "flat" vector space (either by spectral embedding or otherwise) causes loss of information, which means reconstruction of structural entitites from the embedded space is not possible and also leads to degradation of performance.

In order to overcome these shortcomings in a principled fashion, Jain et al. [39, 37], propose to embed the space of combinatorial structures with a metric $(\mathcal{X}, d)$ into a quotient space $(\mathcal{X}', d')$ of an Euclidean space. The embedding is isometric- distance relations are preserved in the quotient spaces, and isostructural- different representations of same structure are projected onto the same element in the quotient space. The elements of an equivalence class are different vector representations of the same structure, thus retaining structural information. In this locally Euclidean (quotient) space, analytical concepts such as distance and angle between combinatorial structures are defined and also gradients of functions defined on them. With these definitions, pattern recognition algorithms for tasks such as clustering and classification are developed. This is the mathematical framework for the Chapters 3, 4 in this thesis

## 1.2 Overview of this thesis

### 1.2.1 Scope

The aim of this thesis is to develop prototype based learning methods for classifying attributed graphs. The motivation for prototype learning algorithms in graph domain are the following

-There already exist successful classes of prototype learning algorithms for

feature vectors- Learning Vector Quantization [44]. They are conceptually simple in that the algorithms require fundamentally only the concept of a differentiable distance measure. The formalism of structure spaces provides us with the means to define a metric between graphs and to derive an analogue of its gradient. Thus, we can extend LVQ algorithms directly to structered domain.

-Crammer et.al. [16], in their seminal contribution show that LVQ algorithms belong to a family of maximum margin classifiers. Hence, there is a strong theoretical motivation to extend these algorithms to graphs.

-Prototype based methods solve naturally the multi-class problem without resorting to simplifications or reduction to many two-class classification problems

-The availability of prototypes for further applications. For instance, it may be necessary in some applications to define a set of "typical elements" or prototypes of a class. Moreover, it has been demonstrated that prototypes could be used to embed the original patterns into a feature (vector) space [54]. In this view, prototypes generate "kernels" in a feature space, where classification performance could be improved, as we show in this thesis

-It is possible to define prototypes for a wide variety of node and edge attributes. This is not possible for classification methods using graph kernels, or spectral methods.

### 1.2.2 Organization

The organization of this thesis is as follows-

In Chapter 2, we review the theory of Structure Spaces. This provides the analytical concepts needed for Chapters 3 and 4.

In Chapter 3, we develop a class of prototype based classification algorithms called Learning Graph Quantization (LGQ) algorithms. The classifiers are parametrized by a set of prototypes with class labels. The class label of a new graph is predicted to be the class label of the nearest prototype (NPC) according to the nearest neighbor rule. The goal of learning is to find a set of prototypes that best predicts the class labels of graphs from the underlying distribution. With the notion of metric and subgradient defined, a novel class of algorithms are proposed to learn the prototypes using a subgradient descent procedure analogous

to Learning Vector Quantization in the domain of feature vectors.

In Chapter 4, we use the prototype based methods to embed the graphs into a vector space. Consider the dissimilarity of the graphs to each prototype as a feature in a (feature) vector space. In such a view, the prototypes generate a kernel, which in turn generate features for classification. We demonstrate that the prototypes learnt by the class of LGQ algorithms generate features that are well suited for classification. In experiments, they improve the state-of-the art performance on multiple datasets. We also evaluate the quality of prototypes for embedding, using bounds from statistical learning theory.

In Chapter 5, a class of probabilistic methods is proposed for classifying attributed graphs. Within the framework of random attributed graphs, we propose an online algorithm to estimate the parameters, using concepts from Information Geometry. The resulting random graph is chosen as a prototype and a formula for defining likelihood is proposed with suitable independence assumptions. The graph set is embedded into a feature space defined by log-likelihood, where classifiers are learnt. It is demonstrated that likelihood is a feature for classification.

Chapters 3, 4, 5 comprise the original scientific contributions of this thesis.

Chapter 6 is the concluding chapter.

A detailed description of the datasets used in this thesis is presented in Appendix A. Appendix B discusses in detail the Graduated Asssignment algorithm for matching attributed graphs, which is used with minor variations throughout this thesis.

# Chapter 2

# Structure Spaces

In this chapter, we describe the theoretical foundations and methodology underpinning this thesis. We begin by reviewing the fundamentals of the theory of Structure Spaces- a framework for defining analytical and geometrical concepts such as angle between combinatorial structures, gradients of scalar functions defined on them. This enables us to develop learning algorithms directly in structural domain. As a first application, central clustering algorithms are considered in view of their importance to prototype learning algorithms later.

## 2.1 Introduction

The nature of this chapter is one of review. Hence, we follow the original contributions by Jain et. al. [39, 37], closely throughout this chapter with the following organization: In Section 2, the theory of Structure Spaces is reviewed in detail, followed by the discussion of central clustering algorithm in Section 3. In Section 4 we present the experimental details along with setting and results, followed by the concluding Section 5.

## 2.2 Structure Spaces

### 2.2.1 Setting

We will use attributed relational graphs (referred in this thesis as attributed graphs or simply as graphs) as the model for combinatorial structures because of its widespread applications and also because other representations such as point patterns, trees are subsumed by it. Recall that attributed graph is a tuple $\mathcal{G} : (\mathbf{V}, \mathbf{E}, \alpha)$ with a set of vertices $\mathbf{V}$, set of edges $\mathbf{E} \in \mathbf{V} \times \mathbf{V}$ and $\alpha : \{\mathbf{V}, \mathbf{E}\} \rightarrow \mathbb{R}$. The attribute values are assumed to be scalars only for simplicity. The graph is then represented by its weighted adjacency matrix $\boldsymbol{X}$, in a representation space $\mathcal{X}$, whose elements are given by the attribute function $\alpha$.

The representation $\boldsymbol{X}$ is determined by a particular ordering of vertices. A different ordering of vertices results in another adjacency matrix $\boldsymbol{X'}$. The relation between $\boldsymbol{X}$ and $\boldsymbol{X'}$ is given by the permutation matrix $\boldsymbol{P}$, which maps the node ordering corresponding to the former representation to the latter. That is,

$$\boldsymbol{X'} = \boldsymbol{P}^T \boldsymbol{X} \boldsymbol{P} : \boldsymbol{P} \in \mathcal{T}$$

where $\mathcal{T}$ is the set of all permutation matrices. Since the set of all permutations of an order forms an algebraic group, their corresponding representations (i.e. the set of all permutation matrices of order $n$) form a group as well. Any two representations $\boldsymbol{X}, \boldsymbol{X'} \in \mathcal{X}$ are said to be equivalent, $\boldsymbol{X} \sim \boldsymbol{X'}$, if $\exists \boldsymbol{P} \in \mathcal{T}$, such that above equivalence relation holds. Hence, the set of permutations $\mathcal{T}$ partitions the representation space $\mathcal{X}$ into orbits consisting of all elements belonging to the equivalence class of elements. The elements of an equivalence class have different representations of the same graph structure.

A graph $X$ with $n$ nodes has $n!$ possible node orderings, of which not all of them may be distinct depending upon the symmetry of graphical structure. The equivalence class of $X$ consists of all its possible representations $\boldsymbol{X}$ and is denoted by $[\boldsymbol{X}]$. Since the representation space is the set of matrices which is a linear space, we could treat it as a vector space of dimension $\mathbb{R}^{n \times n}$, partitioned by the set of permutations into equivalence classes, the elements of which denote the

same structure with different representations. The quotient set

$$\mathcal{X}_{\mathcal{T}} = \mathcal{X}/\sim = \bigcup [\boldsymbol{X}]$$

is called $\mathcal{T}$- space over the representation space $\mathcal{X}$.

Notation: Normal capital letters $X$ denote the abstract structure i.e. elements from the quotient space $\mathcal{X}_{\mathcal{T}}$. The notation $\boldsymbol{X}$ denotes a particular representation. Depending on the context, we use expressions like $\boldsymbol{x} \in X$ which means $\boldsymbol{x} \in [\boldsymbol{x}]$ i.e. $\boldsymbol{x}$ is a particular chosen representation of the graph structure $X$.

### 2.2.2 Analysis on $\mathcal{T}$- spaces

We now define analytical concepts such as inner product and distance measures between structures, which will be computed in the $\mathcal{T}$- space. A $\mathcal{T}$- space over $\mathcal{X}$ is the orbit space $\mathcal{X}_{\mathcal{T}} = \mathcal{X}/\mathcal{T}$ of all orbits $\boldsymbol{x} \in \mathcal{X}$ under the action of $\mathcal{T}$. We introduce a mapping $\mu$,

$$\mu : \mathcal{X} \to \mathcal{X}_{\mathcal{T}}$$

called membership function that sends any vector representation (from the representation space) to the abstract structure they describe.

As mentioned earlier, the representation space is a vector space. Consider a mapping $f : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ therein. This mapping induces a mapping in $\mathcal{X}_{\mathcal{T}}$ corresponding to a pointwise extrema in the following manner:

$$F^* : \mathcal{X}_{\mathcal{T}} \times \mathcal{X}_{\mathcal{T}} \to \mathbb{R}, (X, Y) \to \max\{f(\boldsymbol{x}, \boldsymbol{y}) : \boldsymbol{x} \in X, \boldsymbol{y} \in Y\}$$

$$F_* : \mathcal{X}_{\mathcal{T}} \times \mathcal{X}_{\mathcal{T}} \to \mathbb{R}, (X, Y) \to \min\{f(\boldsymbol{x}, \boldsymbol{y}) : \boldsymbol{x} \in X, \boldsymbol{y} \in Y\}$$

As indicated, the above definitions of $F^*$ and $F_*$ called maximizer and minimizer respectively, are motivated by concept of standard inner product and metric. Consider the standard inner product in Euclidean space $< ., . >$. It induces a "structural" inner product in the $\mathcal{T}$- space as below,

$$< ., . >^* : (X, Y) \to \max\{< \boldsymbol{x}, \boldsymbol{y} > : \boldsymbol{x} \in X, \boldsymbol{y} \in Y\}$$

The map $< ., . >^*$ is positive definite and symmetric. It follows directly from the definition. However, it is not strictly linear because of the following reasons: addition is not a well-defined operation in the quotient space, even if an equivalent structure were to be found which could be interpreted as sum of structures (in terms of their mean structure, which is defined as the structure equidistant to both) the operation $< X + Y, Z >^*$ depends on the pointwise maximum, which may occur for different representations for $X$ and $Y$. The operator is sub-linear, i.e.

$$< X + Y, Z >^* \leq < X, Z >^* + < Y, Z >^*$$

This definition of inner product gives rise to a norm of structure which is,

$$\| X \|_* = < X, X >^*$$

The above definition of "structural" norm is invariant under choice of representation, because

$$\| \boldsymbol{P}^T \boldsymbol{X} \boldsymbol{P} \|_* = \| X \|_* = < X, X >^*$$

as $P$ is a permutation matrix, which is orthonormal and hence preserves norms.

The minimizer $F_*$ is used to define distance measure in $\mathfrak{T}$- space in the following manner:

$$D_*(., .) : (X, Y) \rightarrow \min\{d(\boldsymbol{x}, \boldsymbol{y}) : \boldsymbol{x} \in X, \boldsymbol{y} \in Y\}$$

where $d(., .)$ is (Euclidean) distance measure in the representation space. Also, the distance is related to the structural dot product via

$$D_*^2(X, Y) = < X, X >^* + < Y, Y >^* - 2 < X, Y >^*$$

We also note that this definition of distance follows from set theoretical viewpoint as well. Every structure is identified with an equivalence class in the representation space. Hence, if a representation for one structure $X$ were to be fixed (say $\boldsymbol{x}$), then its distance to another set $Y$ is by definition of set distance

$$d(\boldsymbol{x}, Y) = inf_{\boldsymbol{y} \in Y} d(\boldsymbol{x}, \boldsymbol{y})$$

This distance function is indeed a metric as turns out to be symmetric, non-negative and satisfies triangle inequality. We note that the distance and inner product for graphs defined here is applicable for graphs with arbitrary node and edge feature attributes, by stacking the attribute feature vectors to form an extended adjacency matrix.

### 2.2.3 Functions on $\mathcal{T}$- spaces

In order to develop learning algorithms on structures, we need to set up a cost function which has to be minimized. The minimization algorithms proceed by update along the direction of greatest descent i.e. update in a direction opposite to gradient. Hence, this motivates us to define gradients of functions defined on $\mathcal{T}$- spaces.

Let $\mathcal{X}_{\mathcal{T}}$ be the $\mathcal{T}$- space over $\mathcal{X}$. A $\mathcal{T}$ mapping is then a mapping $f_{\mathcal{T}} : \mathcal{X}_{\mathcal{T}} \rightarrow \mathcal{Y}$. A $\mathcal{T}$ mapping is a $\mathcal{T}$ function if $\mathcal{Y} \subseteq \mathbb{R}$. Consider a map on the representation space $f : \mathcal{X} \rightarrow \mathcal{Y}$, which induces a map $f_{\mathcal{T}}$ on elements $[\boldsymbol{X}] \in \mathcal{X}_{\mathcal{T}}$. If $f_{\mathcal{T}}$ is constant on $[\boldsymbol{x}]$, then $f$ is defined to be $\mathcal{T}$- invariant. Conversely, a $\mathcal{T}$ mapping induces a "pullback" map $f$ on representation space $\mathcal{X}$, which is $\mathcal{T}$- invariant related by $f = f_{\mathcal{T}} \circ \mu$. The "pullback" map is referred to as representation function.

A $\mathcal{T}$ function is $\mathcal{T}$ differentiable at $X \in \mathcal{X}_{\mathcal{T}}$, if its representation function is differentiable at any of its arbitrary representation. This implies that $\mathcal{T}$- differentiability is independent of a representation and hence is well defined. Let $f : \mathcal{X} \rightarrow \mathbb{R}$ be the representation function of a $\mathcal{T}$ function $F$. Let $f$ be differentiable at $\boldsymbol{x} \in \mathcal{X}$ with gradient $\nabla f(\boldsymbol{x})$. The gradient map induces a "push forward" ($\mathcal{T}$- gradient) map on $\mathcal{X}_{\mathcal{T}}$ as below,

$$\nabla F(X) = \mu(\nabla f(\boldsymbol{x}))$$

From the definitions, the following observations are immediate: $\mathcal{T}$- gradient is independent of the choice of representation, it points to the direction of steepest ascent, the necessary and sufficient condition for local extrema remains analogous

as in the representation space, i.e. $\nabla F(X) = 0_{\mathcal{T}}$ for $X$ being an extremum.

Let $f$ be a locally Lipschitz continuous function between representation spaces $\mathcal{X} \to \mathcal{Y}$, i.e.

$$d_{\mathcal{Y}}(f(\boldsymbol{x}, \boldsymbol{x'})) \leq d_{\mathcal{X}}(x, x')$$

If $\mathcal{Y} \subseteq \mathbb{R}$, by Rademacher's theorem, $f$ is Frechet-differentiable everywhere except a set of Lebesgue measure zero. Also, one can define for it subgradient [13] as below,

$$\partial f(\boldsymbol{x}) = \{c : f(\boldsymbol{x}) - f(\boldsymbol{x}_0) \geq c(x - x_0) \forall x \in \mathcal{X}\}$$

At points where $f$ is differentiable, subgradient reduces to a gradient, i.e. $\partial f(\boldsymbol{x}) = \nabla f(\boldsymbol{x})$.

*Optimal alignment*- Given two structures $X, Y \in \mathcal{X}_{\mathcal{T}}$, $\mathcal{A}(X, Y)$ denotes the set of all possible alignments of $X, Y$. A particular vector representation (which is a result of a chosen aligning or ordering of nodes) is said to be an optimal alignment if it minimizes the graph distance (which we define as pointwise minimizer over all possible node orderings) i.e. $\boldsymbol{x}', \boldsymbol{y} \in \mathcal{A}(X, Y)$ is optimal if $\boldsymbol{x}' = \text{argmin}_{\boldsymbol{x} \in \mathcal{X}} \parallel \boldsymbol{x} - \boldsymbol{y} \parallel^2$.

## 2.3 Central clustering

There are some approaches towards extending concept of unsupervised learning such as central clustering and self organizing maps to the domain of structured data. In the case of self-organizing maps for strings, Kohonen et al., [46] propose that the generalized median of the sample set be used to update the winning unit and its topological neighbors. On a similar note, Günter et al., [26] propose a model of SOM for graphs where they use their concept of weighted graph means based on the graph-edit distance in the prototype updation step. Hammer et al., [28] formulate a scheme for processing structured representations by means of recursive representations for the given context. Hagenbuchner et al., [27] propose a self-organizing map that operates for structured data using an unfolding procedure used in recurrent networks.

A notable effort to define the central measure of a set of graphs is that of median and generalized median graphs by Jiang et al., [42]. Given a set of graphs $S : \{g_1, .., g_n\}$, the median graph is defined as $g_{med} = argmin_{g \in S} \sum_{i=1}^{n} d(g, g_i)$. The generalized median is defined as $argmin_{g \in U} \sum_{i=1}^{n} d(g, g_i)$, where $U$ is the set of all graphs that could be contructed with the set of nodes and edges. The computation of generalized median uses the heuristic genetic algorithm on the search space. However, the graph edit distance used in this computation is not differentiable. Hence, it is not possible to use these measures in a theoretically rigorous manner to define central measures.

*Sample mean for a set of attributed graphs-* The concept of sample mean is crucial to the development of central clustering algorithms. Apart from its natural utility for pattern recognition tasks, it has applications in multiple alignment of structures and visualization [36]. In applications, we would like the mean graph to give a concise summary of properties of samples in the set. For example, if the graphs have no attributes i.e. the adjacency matrix has only ones if an edge is present, then the sample mean should give the fraction of edge occurences. Consider the definition of sample mean for feature vectors- it is the vector that is central in the set, in the sense that it minimizes the sum of squares of distances to all elements in the set. Extending the analogy to graphs, we define the sample mean of a set of graphs $\mathcal{S} = \{X_1, .., X_n\} \subseteq \mathcal{X}_{\mathcal{T}}$ as a minimizer of the following cost function,

$$F(X) = \sum_i D(X_i, X)^2, X \in \mathcal{X}_{\mathcal{T}}$$

where $D$ is the distance function on $\mathcal{X}_{\mathcal{T}}$. As the choice of graph metric we choose is Lipschitz, the cost function for sample mean is locally Lipschitz in terms of the mean. It is thus possible to come up with a gradient descent rule to minimize the cost as follows,

$$\boldsymbol{m_k} = (1 - \eta)\boldsymbol{m}_{k-1} + \eta \boldsymbol{x_k}$$

where, $\eta$ is the decreasing step size (typically $\frac{1}{n}$ for step $n$), $(\boldsymbol{m}_{k-1}, \boldsymbol{x}_k) \in \mathcal{A}(M_{k-1}, X_k)$ is the optimal alignment. Thus, we have reformulated the sample mean problem as one of continuous optimization instead of a combinatorial

optimization problem.

We now turn to the first application of theory of structure spaces to structural pattern recognition- central clustering algorithm for attributed graphs . [39]. Clustering is a fundamental problem in pattern recognition. Moreover, the concept of cluster centres is heuristically close to the concept of prototypes as an "ideal representative of the set". Learning algorithms such as LVQ family of algorithms typically begin with prototypes initialized to cluster centres.

Given a set $\mathcal{S} = \{X_1, .., X_n\} \subseteq \mathcal{X}_{\mathcal{T}}$, the goal of central clustering is to find a set $\mathcal{Y} = \{Y_1, .., Y_k\} \subseteq \mathcal{X}_{\mathcal{T}}$ (referred to as cluster centres) such that the cost function

$$F(M) = \frac{1}{n} \sum_i \sum_j m_{ij} D(\mathcal{X}_i, \mathcal{Y}_j)$$

is minimized. The assignments are given by the membership matrix $M$, such that

$$m_{ij} \in [0, 1], \sum_{j=1}^{k} m_{ij} = 1, \text{for } i = 1, .., n$$

Each term in the cost function is locally Lipschitz. Hence, the total cost is locally Lipschitz in terms of the cluster centres. Hence, we adapt the K-means algorithm [4] to the domain of structures in the following manner.

**Algorithm 1** Structural K-means algorithm

**Input:**
training set $\mathcal{S} = \{X_1, \ldots, X_n\} \subseteq \mathcal{X}_{\mathcal{T}}$
**Procedure**:
Choose $k$, initial prototypes $\mathcal{Y} = \{Y_1, \ldots, Y_k\} \subseteq \mathcal{X}_{\mathcal{T}}$
Repeat until termination
Assign data to nearest cluster centre
Recompute cluster centres $Y_k = \text{Sample mean}\{X_{Y_k}\}$
**Return:** set $\mathcal{Y}$ of prototypes

The expectation E step consists of assigning patterns to their nearest cluster centres. The M- step involves updating the prototypes, to the sample mean

of their respective graphs. This process continues until there is no change in assignments or when alignments do not change the cost function significantly [40].

## 2.4 Experiments

### 2.4.1 Datasets

We carry out central clustering algorithm for three datasets- Fingerprint, GREC and Skolnick dataset. A detailed description of the datasets is found in Appendix A. A brief summary is given below.

| Dataset | #(graphs *tr, va, te*) | #(classes) | avg.(nodes) | max(nodes) |
|---|---|---|---|---|
| Fingerprint | 500, 300, 2000 | 4 | 5.4 | 26 |
| GREC | 286, 286, 528 | 22 | 22.5 | 25 |
| Skolnick | 40 | 5 | 158.4 | 255 |

Table 2.1: Summary of main characteristics of the data sets.

### 2.4.2 Algorithms

For computing optimal alignments and graph distances, we use the graduated assignment algorithm [23]. This algorithm is discussed in Appendix B. This algorithm places the problem of attributed graph matching within the context of nonlinear optimization. In order to avoid getting trapped in a local minima, a continuation method is adopted- i.e. a series of cost functions (in terms of match matrix) parametrized by a control variable is minimized. This control parameter is analogous to the temperature in an annealing process. At every step, the match matrix is estimated and the soft-assign condition is enfoced by Sinkhorn's procedure. The node and edge compatibilities are set to the inner product of their respective attribute vectors. The algorithm returns a doubly stochastic match matrix and a discrete match matrix is recovered by applying Munkres' algorithm. The permutation matrix is then used to align the smaller graph with the larger one.

### 2.4.3 Setting and results

The purpose of central clustering is to reflect the ground truth, which in this case are the class memberships/labels. The choice of $k$ is made by examination of the variabilities of graph size within a dataset that is best suited to reflect the unknown cluster distribution. To that end, we adopt the following methodology: Initially, the dataset is analyzed for graph sizes. For every graph size, if the number of graphs having that size exceeds a fraction of the dataset, a cluster centre is initialized with the sample mean of all graphs having the corresponding number of nodes. Then, in order to prune the number of clusters, we combine classes incrementally, as long as there is no significant number of mismatch of class labels. This process is done classwise, using a subset of training set to validate the parameter at every step. This yields the optimal value of cluster centres to be $31, 37$ and $5$ for Fingerprint, GREC and Skolnick datasets respectively.

| Fingerprint | GREC | Skolnick |
|---|---|---|
| 0.73 | 0.86 | 1 |

Table 2.2: Rand index

The Rand index is presented in Table 2.2. It is clear that for Protein contact maps, central clustering correctly categorizes the maps into 5 clusters.

*Cluster centres as prototypes*- Following Riesen et al. [59], who use median graphs as central elements in a class to embed the graphs into a feature space, we use the cluster centres to define a feature space embedding of graphs by computing the distance of a graph to the cluster centres. We embed each graph $X \in \mathcal{S}$ in $\mathbb{R}^k$ given a cluster centre set $\mathcal{W} = \{W_1, .., W_k\}$ in the following fashion

$$X \rightarrow (d(X, W_1), ..., d(X, W_k))$$

In the resulting feature space, we use Support Vector Machines [19] to classify the test data.

The results of this approach is presented in Table 2.3, where the are compared with k-Nearest Neighbour (kNN) directly in the graph domain with graph edit distance [59]. We note here that the number of prototypes (cluster centres) are

| Method | Fingerprint | GREC |
|---|---|---|
| Cluster centres + SVM | 76.8 | 94.7 |
| kNN | 76.7 | 95.5 |

Table 2.3: Classification accuracy obtained by linear SVM in the feature space generated by cluster centres

less compared to kNN which effectively uses a large prototype set (the entire training set) to get similar performance. This implies that the number of graph matching operations needed to be performed when using a compact cluster centre set to embed the graphs is much less than the graph matching operations needed to classify in the kNN algorithm.

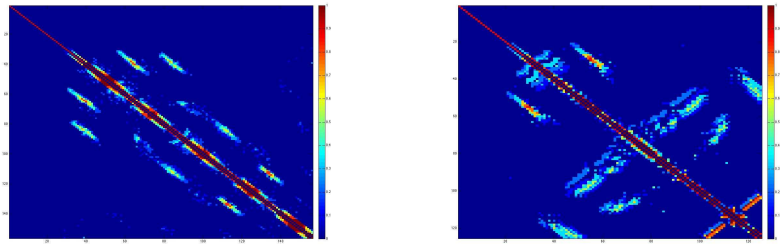
The advantages of using central clustering algorithms over other techniques such as pairwise clustering are

- number of graph matching operations. Pairwise algorithms require $\frac{N(N-1)}{2}$ matchings for $N$ patterns, while k-means needs approximately $M \times N \times K$ matching operations where $M$ is the number of iterations. For datasets with few cluster centres, we have observed that k-means algorithm converges quickly. This means that $K \times M \ll N$ and lesser graph matchings have to be solved.

- the availability of cluster centres. The cluster centres are available for further applications such as multiple alignment, which is equivalent to solving the sample mean problem or prototypes to be used for classification tasks.

## 2.5 Conclusion

In this chapter, we have reviewed the theory of structure spaces, which enables us to define distance measure between attributed graphs and subgradients of functions that are defined on them. These concepts are extremely crucial in order to develop pattern recognition algorithms on combinatorial structures. As a first application, we consider the problem of central clustering of attributed graphs. The cluster centres are conceptually related to representative elements or prototypes of a set. The clustering algorithm alternates until convergence, assignment of a graph pattern to a cluster and subsequent estimation of cluster



(a) Che-Y like family (b) Cuperdoxin like family

Figure 2.1: Representative cluster centres for Skolnick dataset
The off-diagonal elements show the contacts between residues with their relative frequencies denoted by the colour bar

centres. The analytical concepts needed for the computations are provided by the framework of structure spaces, as the notion of graph edit distance in the current literature is not differentiable. The central clustering algorithm has been shown to be an effective indicator of ground truth. In the subsequent chapters, we use the formalism developed here to devise effective prototype learning algorithms, with central clustering used in the initialization phase.

# Chapter 3

# Learning a set of prototypes for classifying attributed graphs

In this chapter, we propose a novel class of algorithms for learning a set of prototypes for classifying attributed graphs. We use the framework of "Structure Spaces" to define a suitable metric between attributed graphs. Subsequently, we derive a prototype update rule based on the sub-derivative of the metric analogous to Learning Vector Quantization algorithms for feature vectors. Two flavors of such an algorithm are presented, which are then evaluated on datasets from the IAM graph dataset repository.

## 3.1 Introduction

Given a set of attributed graphs belonging to different classes, we would like to learn a classifier that maps the elements of the dataset to class labels. In prototype based classification schemes, the classifier is parametrized by a set of labeled prototypes, and the classification is done in the Nearest neighbour framework. There are many advantages of such a scheme- conceptual simplicity, easy extension to multi-class problems, and an interpretable model (the set of prototypes), rather than a black-box classifier model. As we show later, the prototypes could also be used embed the dataset into a dissimilarity space, by considering the distance between the dataset and every element of the prototype

set as a feature. This finds particular utility in graph-based representations- the availability of powerful techniques for classification and clustering in vector spaces means that it is quite gainful to embed the attributed graphs into a feature space.

In this chapter, we consider the problem of learning a set of prototypes for classification purposes. We begin by reviewing the Learning Vector Quantization algorithm and its variants, especially for non-vectorial data. We discuss then the extension of these algorithms for attributed graphs, by deriving a subgradient type update rule for the prototypes. These algorithms, referred to as Learning Graph Quantization algorithms are tested on multiple datasets from IAM Graph Database Repository and their performance is compared to reference and state-of-the-art classification algorithms [41]. We also discuss some generalization properties of the algorithms. Part of the material presented in this chapter has been published [41]. The experiments on COIL and SHAPE datasets were carried out by Göttner [24].

## 3.2 Learning Vector Quantization and its variants

Learning Vector Quantization (hereafter referred to as LVQ) algorithms are a popular class of algorithms to generate prototypes of a set of feature vectors. The algorithms, first proposed by Kohonen [44], find applications in domains as diverse as spreech recognition, image analysis, signal processing and robotics. The fundamental idea is to estimate directly, the decision surfaces using a gradient type learning rule as below.

Given a dataset $(X_i, y_i) \in \mathbb{R}^n \times \mathcal{C}$, and a set of prototypes $(Y_k, c_k) \in \mathbb{R}^n \times \mathcal{C}$, the latter induces a tessellation of $\mathbb{R}^n$ into Voronoi cells consisting of all points that are closer to one prototype than the other. The Voronoi cell for a prototype $Y_p$ is given as,

$$X \in \mathbb{R}^n : d(X, Y_p) < d(X, Y_r), \forall r \neq p$$

Thus every labeled prototype has a polygonal cell around it, the edges of which represent decision boundaries with different classes. LVQ aims to learn the

classifier by directly adjusting the Voronoi cells and hence the class boundaries.

The training phase begins by first initializing the prototypes to the codebooks generated by obtained by Vector Quantization (VQ) [19] or Self Organizing Map (SOM) algorithms. Alternatively, the initialization could also be done by a clustering algorithm or by utilizing prior expert knowledge. In the next stage, the elements of the sample dataset are presented sequentially. For every presentation, the closest prototype is found which is then updated according to the following rule

$$Y_{c_{t+1}} = Y_{c_t} \pm \nabla_{Y_c} d(X, Y_c)$$

depending upon if the prototype vector corresponds to the same class as the presented data or not. The other prototypes stay unaffected.The learning rate $\eta$ is non-negative and non- increasing satisfying the following constraints, [7]

$$\sum \eta = \infty, \sum \eta^2 < \infty$$

Kohonen [45] suggests initializing the learning rate as $\eta = 0.01$ and then monotonically decreasing it at every step. The algorithm terminates when either the learning rate vanishes or the prototypes converge; whichever is earlier. The following point needs to be mentioned- LVQ is a "fine-tuning algorithm" - the prototypes should be initialized close to the stable equilibrium in order for the algorithm to converge [44, 47] .

Extensions have been developed for LVQ algorithms with general similarity measures that need not be symmetric nor satisfy triangle inequality [30]. Hammer et.al. [31], show that by using a scaled distance measure $d(x, y) = \lambda_i(x_i - y_i)^2$, rather than normal Euclidean metric, it is possible to better process data with non-homogeneous input dimensions and hence achieve better generalization performance. In fact, choosing a similarity measure is akin to kernelizing the standard LVQ algorithms; hence generalization bounds from statistical learning theory can be established in this case[29].

With respect to non-vectorial data, Sumervuo et. al., [65] have constructed LVQ algorithms for variable length sequences. It is accomplished by setting up a distance function between sequences and defining a weighted average of se-

quences. Also, the notion of prototypes of a set of attributed graphs has been examined. Marini et al., [53] describe a constructive approach to build the prototypes of a class of geometrical shapes represented by attributed graphs. Their scheme comprises two steps: computing the common subgraph occuring in the class members and computing a suitable set of editing operations. Initially, a representative is chosen for every class and all common subgraphs are computed between all elements of the class and the representative. Then, nodes and edges are modified (inserted or deleted) depending upon the other elements of the class successively followed by transforming attribute values. Using a similar approach, Cordella et al., [15] determine the set of maximally general prototypes of a class of shapes represented as attributed graphs. The above approaches depend highly on the nature of datasets in terms of graph and attribute types. They are also extremely dependent on the choice of representative which often is carried out manually. Moreover, the approaches are heuristic, in the sense that it is not clear if they minimize a cost function or the error rate.

The primary motivation for this chapter follows this line of reasoning- LVQ algorithms are widely used in the domain of feature vectors for prototype learning; there are no corresponding versions in the domain of graphs. Hence, using the framework of structure spaces, we propose an extension of LVQ algorithms to attributed graphs and evaluate is performance on multiple datasets.

## 3.3 Distance measure between attributed graphs and prototype update rule

Consider the standard LVQ algorithm:

**Algorithm 2** Prototype update in LVQ

Repeat until convergence
Present pattern $(X_i, y_i)$
Find the winner; Nearest Prototype (NP)$Y_c$ (according to distance measure $d$, commonly Euclidean)
Update NP according to, $Y_{c_{t+1}} = Y_{c_t} \pm \nabla_{Y_c} d(X, Y_c)$

The challenges in extending LVQ to the domain of attributed graphs are the following; defining a distance metric between the objects, and formulating a prototype update rule. For feature vectors in Euclidean space, the update rule is quite simply pushing/pulling the prototypes along the direcion of steepest change, which is the gradient.

### 3.3.1 Graph metric

The problem of defining a graph metric has already been considered in Chapter 2. Here, we present a brief review of the concepts.

On a quotient space $\mathcal{X}_{\mathcal{T}}$ over an underlying Euclidean space $\mathcal{X}$, consider graph distance functions $d : \mathcal{X}_{\mathcal{T}} \times \mathcal{X}_{\mathcal{T}} \to \mathbb{R}_+$ of the form

$$d(X, Y) = \min \left\{ \|\boldsymbol{x} - \boldsymbol{y}\|^2 \, : \, \boldsymbol{x} \in X, \boldsymbol{y} \in Y \right\}.$$

It has already been shown to be a metric. A pair $\boldsymbol{x}, \boldsymbol{y} \in X \times Y$ is an *optimal alignment* if $d(X, Y) = \|\boldsymbol{x} - \boldsymbol{y}\|^2$. $\mathcal{A}(X, Y)$ denotes the set of all optimal alignments of $X$ and $Y$.

### 3.3.2 Generalized differentiability

The next step is to define a concept of gradient of the graph metric. It should be pointed out that the graph metric that we defined earlier is not differentiable; hence it is necessary to use concepts from non smooth optimization [13].

**Definition 1.** *A function $f : \mathbb{R}^n \to \mathbb{R}$ is called **generalized-differentiable (GD)** if there exists point-set mapping $\delta f : x \in \mathbb{R}^n \to \delta f(x) \subset \mathbb{R}^n$, where $\delta f(x)$ are bounded, convex and closed with the following decomposition*

$$f(y) = f(x) + < g, y - x > + o(x, y, g),$$

*where $\frac{o(x,y,g)}{y-x} \to 0$, uniformly and $g \in \delta f(y)$.*

The following propositions accompany the definition of GD functions.

1. GD functions are continuous
2. Continuously differentiable functions are GD

3. The class of GD functions is closed under finite class operations like maximum, minimum and superposition.

4. GD functions satisfy local Lipschitz condition

$$\frac{\mid f(y) - f(x) \mid}{\mid y - x \mid} \leq K$$

The point-set mapping defined is called the subgradient of $f$ at $x$. The elements of the set are referred to as generalized gradients. When the function $f$ is differentiable at a point $x$, then the subgradient (set) has only one element, namely the derivative. Hence, it might be interpreted that subgradients generalize the notion of derivatives at non-differentiable points of a function.

In our definition of graph distance, we have a pointwise minima of a set of differentiable functions. If $f = max\{f_0, f_1, ..., f_m\}$, then determination of subgradient of $f$ proceeds by finding $k$, such that $f = f_k$ and choosing a subgradient of $f_k$ at that point. More generally, the subgradient is $\delta f(x) = Co\{f_i(x) : f(x) = f_i(x)\}$, where $Co$ denotes the convex hull.

With the above definitions, we can define the subgradient of our graph metric to be,

$$\delta d(X, Y) = 2(\boldsymbol{x} - \boldsymbol{y}_*)$$

where $(\boldsymbol{x}, \boldsymbol{y}_*)$ is the optimal alignment in the set of all alignments $\mathcal{A}(\mathcal{X}, \mathcal{Y})$.

With the notions of graph metric and gradient defined, we can now proceed to extend LVQ algorithms to graph domain.

## 3.4 Learning Graph Quantization

The motivation of Learning Graph Quantization (LGQ) algorithms is to construct a classifier $c : \mathcal{X}_\mathcal{T} \rightarrow \mathcal{C}$ that maps graphs from $\mathcal{X}_\mathcal{T}$ to class labels from a finite set $\mathcal{C}$. The classifier is parametrized by a set of $k$ prototypes $Y_1, ..., Y_k \in \mathcal{X}_\mathcal{T}$ with class labels $c_1, ..c_k \in \mathcal{C}$. We predict the class label $c(x)$ of a previously unseen graph by assigning to it the class label of the Nearest Prototype (NP). The goal of learning is to determine the prototypes that is suited to the classification task.

### 3.4.1 Learning Graph Quantization

Given that $\mathcal{S} = \{(X_i, y_i)\} \subseteq \mathcal{X}_{\mathcal{T}} \times \mathcal{C}$ is a training set, consisting of $n$ input graphs, together with class labels $y_i \in \mathcal{C}$. The algorithm initializes $k$ prototypes $\mathcal{Y} = \{(Y_j, c_j)\}$, such that each class is represented by at least one prototype. During the training phase, a randomly chosen example $\{(X, y)\} \in \mathcal{S}$ is presented which then modifies the closest prototype graph $Y_X$. If the class labels of $X$ and $Y_X$ agree, then the prototypes are pulled towards the sample along the direction of steepest change i.e. the subgradient. If $Y_X$ does not belong to the same class as that of $X$, it is pushed away along the direction determined by the subgradient.

To update the closest prototype $Y_X$, the algorithm first selects an optimal alignment $(\boldsymbol{x}, \boldsymbol{y_x}) \in \mathcal{A}(X, Y)$. Then, it modifies $\boldsymbol{y_x}$ according to the following rule

$$\boldsymbol{y_x} \leftarrow \begin{cases} \boldsymbol{y_x} + \eta(\boldsymbol{x} - \boldsymbol{y_x}) & y = c_x \\ \boldsymbol{y_x} - \eta(\boldsymbol{x} - \boldsymbol{y_x}) & y \neq c_x \end{cases},$$

where $\eta$ is a monotonically decreasing learning rate following the guidelines of stochastic optimization. The updated vector representation projects to the updated graph prototype. The learning continues until convergence has been reached or the learning rate vanishes. Algorithm 2 summarizes the procedure.

### 3.4.2 Learning Graph Quantization 2.1

In contrast to LGQ1, LGQ2.1 algorithm updates the two closest prototypes $Y_X^1$ and $Y_X^2$ according to the presented training example $(X, y) \in \mathcal{S}$. The algorithm modifies the prototypes $Y_X^1$ and $Y_X^2$ if the following conditions hold:

1. The pattern is misclassified; and $Y_X^2$ has the same class label as $X$

2. The sample $X$ falls within a "window" around the decision border i.e. $d(X, Y_X^2) < (1 + \delta) d(X, Y_X^1)$

The update rule, is then given by

$$\boldsymbol{y_x^2} \leftarrow \begin{cases} \boldsymbol{y_x^2} + \eta(\boldsymbol{x} - \boldsymbol{y_x^2}) \\ \boldsymbol{y_x^1} - \eta(\boldsymbol{x} - \boldsymbol{y_x^1}) \end{cases},$$

**Algorithm 3** Learning Graph Quantization 1

**Input:**
training set $\mathcal{S} = \{(X_1, y_1), \ldots, (X_n, y_n)\} \subseteq \mathcal{X}_\mathcal{T} \times \mathcal{C}$
**Procedure**:
1. choose initial prototypes $\mathcal{Y} = \{(Y_1, c_1), \ldots, (Y_k, c_k)\} \subseteq \mathcal{X}_\mathcal{T} \times \mathcal{C}$
2. choose vector representations $\boldsymbol{y}_1 \in Y_1, \ldots, \boldsymbol{y}_k \in Y_k$
3. **repeat** until termination
   - 3.1. randomly select a training example $(X, y) \in \mathcal{S}$
   - 3.2. let $Y_X = \arg\min_{Y \in \mathcal{Y}} d(X, Y)$
   - 3.3 choose optimal alignment $(\boldsymbol{x}, \boldsymbol{y}_{\boldsymbol{x}}) \in \mathcal{A}(X, Y_X)$
   - 3.4. determine learning rate $\eta > 0$
   - 3.5. update according to the rule

$$\boldsymbol{Y}_{\boldsymbol{x}} \leftarrow \begin{cases} \boldsymbol{y}_{\boldsymbol{x}} + \eta\,(\boldsymbol{x} - \boldsymbol{y}_{\boldsymbol{x}}) & \text{if } y = c_X \\ \boldsymbol{y}_{\boldsymbol{x}} - \eta\,(\boldsymbol{x} - \boldsymbol{y}_{\boldsymbol{x}}) & \text{if } y \neq c_X \end{cases}$$

**Return:** set $\mathcal{Y}$ of prototypes

The window width $\delta$ is initialized to value of 0.2 and decreased monotonically. If the conditions above do not hold, the update step does nothing.

### 3.4.3 Generalization bounds and margins

Crammer et al., [16] show that LVQ algorithms belong to a family of maximal margin algorithms. The definition of margin here is not that of "sample margin" (i.e. the distance between a data instance and classification boundary), but that of *hypothesis margin* which is defined as the distance the classifier can be shifted without changing any class labels. The hypothesis margin is given by,

$$\theta = (d(x, y_-) - d(x, y_+))/2$$

where $y_+$ and $y_-$ are the nearest prototypes with correct and incorrect labels respectively.

The only condition we require in deriving $\theta$ is that the distance measure be a metric. Since, the distance measure between attributed graphs satisfies the condition, the above expression for $\theta$ is valid. Analogous to the loss function set up for LVQ1, a hinge loss function [19] with slope parameter $\beta$ could be set up

for LGQ1 as,

$$L(\theta) = (1 - \beta\theta)_+$$

The online algorithm for minimizing the above loss function (with the normal gradient replaced by the sub-derivative) is identical to the prototype update rule in LGQ1. The loss function for LGQ2.1 is given by the following,

$$L(\theta) = min(2, (1 - \beta\theta)_+)$$

The generalization bounds derived in [16] apply,

$$R_h \leq R_{emp} + \sqrt{\frac{8}{m}(d_{VC} \log^2 \tfrac{32m}{\theta^2} + \log \tfrac{4}{\delta})}$$

where $\delta$ is the confidence interval, $m$ is the number of samples and $d_{VC}$ is the $VC$ dimension given by,

$$d_{VC} = \min (N + 1, \frac{64R^2}{\theta^2})2k^C \log (ek^2)$$

Here $C$ is the number of classes, $k$ is the number of prototypes per class, and $N$ is the dimension of embedding space,

$$N = \mathcal{G}l_v + \mathcal{G}(\mathcal{G} - 1)l_e$$

where $\mathcal{G}$ is the size of largest graph in the sample set, $l_v, l_e$ denote the dimensions of node and attribute features respectively.

## 3.5 Experiments

To assess the performance of the Learning Graph Quantization algorithms, we conducted a set of experiments on the datasets from the IAM Graph Database Repository described in [57] and on the Shape dataset [20]. Three datasets from the IAM graph dataset were chosen as benchmarking datasets for the following reasons- these three datasets have on average less number of nodes, hence it is easier to use them for parameter exploration and the wide class of algorithms

available makes it possible to evaluate the proposed LGQ algorithms for a thorough study.

### 3.5.1 Description of the datasets

We choose the following datasets to test and validate the performance of the proposed algorithms. Three are chosen from the standard IAM Graph database repository (Fingerprint, GREC and Letter (HIGH)) and one each from the COIL and SHAPE datasets. More details are presented in the appendix.

| Dataset | #(graphs *tr, va, te*) | #(classes) | avg(nodes) | max(nodes) | avg(edges) |
|---|---|---|---|---|---|
| Letter (HIGH) | 750, 750, 750 | 15 | 4.7 | 8 | 3.1 |
| GREC | 286, 286, 528 | 22 | 11.5 | 25 | 12.2 |
| Fingerprint | 500, 300, 2000 | 4 | 5.42 | 26 | 4.42 |
| COIL-DEL | 96,20, 40 | 4 | 13.3 | 40 | 49.4 |
| SHAPES | 108 | 5 | 11.7 | 22 | 14.1 |

Table 3.1: Summary of main characteristics of the data sets.

### 3.5.2 Experimental setup

*Setting of LGQ algorithms-* The performance of LGQ algorithms depends upon the initialization of prototypes. The number of prototypes is preferably not too high because of the following reasons: there is a degradation of performance associated with large number of prototypes per class which is explained in terms of generalization ability [16]; more prototypes imply more graph distance computations which blows up the computational complexity and hence the runtime of the algorithm.

We initialized the prototypes in a class-wise manner as described below, in accordance with the suggestion by Kohonen [44] to initialize the prototypes by a Vector Quantization type procedure. For every class, we applied the $k$-means algorithm for graphs. To determine the optimal number $k$ and initialize the clusters, we partitioned the dataset according to the graph sizes. Each cell of the partition forms a cluster, if it contains at least $m$ graphs. We optimize the

number $m$ with respect to the validation set. The central clustering algorithm then gives the cluster centres which are then initial prototypes.

The algorithms were iterated $n_{iter} = 100$ times through the training set. The annealing schedule was chosen to be $\eta_t = 0.01(1 - t/n_{iter})$ at every step $t$. For LGQ2.1, the window width was chosen to be 0.1 for the Letter and GREC datasets and 0.2 for the Fingerprint dataset. The low value of $n_{iter}$ is to reduce the runtime of the algorithm and it has been observed that the prototypes converge well within this limit.

*Graph distance computations-* For computing optimal alignments and graph distances, we use the graduated assignment algorithm [23] followed by Munkres' algorithm as a post-processing step in order to determine the most optimal matching (Refer Appendix B).

*Experimental protocol-* We run LGQ1 and LGQ2.1 on the training set of each dataset 10 times. We assess the quality of model obtained by evaluating the results on the validation set. The parameters and run showing the best validation set performance is chosen as a model for the test set. We denote the algorithms by their respective initials *LGQ* and *LGQ*2.1.

### 3.5.3 Comparison with state-of-the-art techniques

The following classification algorithm is chosen for comparison study [57]

*k- Nearest Neighbour (kNN)-* kNN is the conceptually simplest algorithm for classifying any set of objects, between which a dissimilarity measure could be defined. An unknown (test) object is assigned to the class, which is determined by the majority of its nearest neighbours. The value of $k$ is chosen based on the training and validation set. We choose the results obtained by the kNN procedure as the benchmark for the following reasons: it is a prototype based classification paradigm; unlike the other procedures it works directly on the domain of graphs.

### 3.5.4 Results & Discussion

The results of LGQ and LGQ2.1 algorithms is presented in Table 3.2 along with reference classification algorithm (kNN).

| Method | Letter | Fingerprint | GREC | COIL-DEL | SHAPES |
|---|---|---|---|---|---|
| kNN | 82 | 76.7 | 95.5 | 87.5 | 77.6 |
| *LGQ* | 80.9 | 79.2 | 94.7 | 90 | 79.7 |
| *LGQ2.1* | 83.7 | 82.2$^\star$ | 97.3 | 92.5 | 80.4$^\star$ |

Table 3.2: Comparing classification accuracy

The number of prototypes chosen for the Letter and GREC datasets are 2 per class. For the Fingerprint dataset, 30 prototypes were chosen for all the four classes. The kNN algorithm has $k$ chosen as 3 for the Fingerprint dataset, 5 for the Letter and GREC dataset. The symbol $\star$ indicates that the improvement of result is statistically significant [18] compared to kNN (resampled paired t-test for SHAPES dataset and Z-test for all other datasets) with the level of significance set to 0.05.

The first observation that we make is the LGQ algorithms (except LGQ on the Letter dataset) performs better than the reference kNN classifiers. We also note that the kNN needs to compute the distance to every element in the training (then choose the $k$ nearest neighbours). For the LGQ algorithms, once the prototypes have been learnt offline (in the training phase), the distance to a compact training set is all that needs to be calculated. Hence, in real time (online) classification tasks, there is a large reduction in the computational complexity, as much less number of graph distance computations need to be performed. In addition, empirical observations show that LGQ algorithms are quite robust against noisy data and the prototype set it yields summarizes the structural and attribute distributions. kNN classifiers do not share both these advantages.

It is also seen that LGQ2.1 performs uniformly better than standard LGQ algorithm on all the datasets. This is explained in terms of the update rule of LGQ2.1 which is able to approximate the Bayesian decision surface than LGQ. This behavior is analogous to the behaviour of its counterpart in the vector spaces[44].

## 3.6 Conclusion

In this chapter, we have proposed a novel class of prototype learning algorithms in the domain of attributed graphs. The theoretical framework for the algorithm is provided by the structure spaces formalism, which allows for the concept of a metric between graphs and a gradient of the metric. With these definitions, we extend the LVQ algorithms to the domain of graphs, using the graph metric and the prototype learning rule given by the direction of steepest change of the metric (subgradient). The algorithms referred to as Learning Graph Quantization (LGQ) have two versions- LGQ1 and LGQ2.1, analogous to LVQ1 and LVQ2.1 respectively. The algorithms are tested on five datasets, where they improve upon the performance of the standard kNN classifier on four of the datasets, including statistically significant improvement on two of them. We also note that the analysis of generalization performance of LVQ algorithms applies in our case as well due to the definition of metric. In the next Chapter, we will examine the suitability of these prototypes to generate a dissimilarity space, where powerful classifiers such as Support Vector Machines could be applied, thus giving better classification performance.

# Chapter 4

# Dissimilarity representations of attributed graphs

In graph based representations, it is common to embed attributed graphs into a feature vector space to carry out tasks such as classification and clustering. A standard approach to embed the patterns is to choose a set of prototype graphs and assign every pattern to a vector in the dissimilarity space spanned by the prototype set, whose components are given by distances to the prototypes. We use the class of prototype learning algorithms to generate the dissimilarity space representations of the graphs, where we classify the dataset. The efficiency of this technique is demonstrated by experiments on datasets from the IAM Graph Database Repository.

## 4.1 Introduction

In order to classify attributed graphs, a standard approach is to embed them into a vector space, where some of the established algorithms such as support vector machines, Bayes' classifier or neural networks could be used. A common method to embed the graph dataset is to choose a set of prototype graphs and consider the distance of the graphs to each prototype as a feature in the dissimilarity (vector) space [59]. The utility of this method relies on the observation that dissimilarity is an effective feature for classification [54]. There are two approaches for developing

good classification strategies in this scheme.

1. Choose a large set of prototypes and embed the graphs into a high-dimensional feature space spanned by distances to the prototypes. Select the features that are most useful for classification [10]. We note that in this case, selection strategy is done on the set of already generated features.

2. Choose a compact set of prototypes such that they effectively represent the class they belong to, thus giving rise to a good dissimilarity space representation. This implies that we are learning the features (*feature generation*) in constrast to the feature selection procedure mentioned earlier.

We consider the problem of learning the prototypes that is suited to the classification task in the resulting dissimilarity space. There have been some attempts (when the original patterns themselves are feature vectors) to explore good prototype optimization schemes, such as using LVQ, mixture of Gaussians to learn the prototypes [49]. However, there is no analytical motivation underpinning this approach. This problem is even more acute in the domain of attributed graphs, as even the most basic mathematical tools necessary to develop learning algorithms are lacking. Hence, most of the attempts to develop good prototype selection algorithms are heuristic.

A few of the schemes commonly used [59] are

-choosing the most central graphs (Median, k Centres Prototype Set Selection) in the dataset. The underlying concept of such an approach is that different choice of prototypes optimize different clustering cost functions, thus giving the most "central" elements. For instance, median graphs minimize heuristically a clustering cost function with $\mathcal{L}^1$ norm, while mean graphs correspond to optima of $\mathcal{L}^2$ norm.

-graphs uniformly distributed in the dataset (Spanning Prototype Selection). The principle here is to choose prototypes such that they represent the data distribution in a uniform manner. The motivation here is to choose prototype graphs such that they represent also the "atypical" (outlier) elements as well. A simple strategy is to choose initially the median graph of the dataset and subsequently choose graphs that are far away from the already existing set of graphs.

-completely random choice of prototype graphs.

These procedures could be carried out in a classwise or class independent fashion.

However, there are two shortcomings associated with this technique. It has been observed that when the dataset is highly diverse or corrupted by noise, it is hard to choose a compact prototype set. The second is that embedding is uncoupled from the subsequent problem at hand- classification. Intuitively, an embedding could be much more effective, if it were to simplify the following classification problem. As has been demonstrated [17, 56], supervised embedding taking into account class labels acts as an useful pre-processing step, yielding improved classification results and also enables construction of a classifier that has low computational complexity.

In this chapter, we propose to use the prototypes obtained by the class of Learning Graph Quantization (LGQ) algorithms to generate a dissimilarity representation of graphs. Although LGQ algorithms are themselves classification algorithms, there are a couple of advantages in using the prototypes obtained by the LGQ algorithms to embed the graphs in a feature space, where classification is carried out- improvement in classification performance and less computational complexity. We hypothesize that in the domain of attributed graphs, prototypes that are learnt in a supervised manner using class labels give rise to better dissimilarity space representations suited to classification tasks. To that end, we use the prototypes obtained by LGQ algorithms and a concomitant class of generalized LGQ algorithms for embedding [38]. We show that classification results hence obtained compare well against state-of-the-art approaches.

## 4.2 Prototype based embedding

In this section, we present an analytical approach to learn the prototypes to generate the feature space spanned by dissimilarity values. The approach relies on the concept of a generalized differentiable distance measure, thus making the analysis presented here valid for feature vectors (with a differentiable distance measure such as Euclidean distance) and attributed graphs with the graph metric (along with its subgradient) as discussed in Chapter 2. The aim is to motivate the case for using the prototypes obtained by Learning Graph Quantization algorithms

(or LVQ algorithms for feature vectors) to generate the dissimilarity space.

### 4.2.1 Analysis on Dissimilarity Space

Given a pattern set $\mathcal{S}$ from an arbitrary space $\mathcal{X}$, with class labels belonging to set $\mathcal{C}$ given by label function $l(.) : \mathcal{X} \rightarrow \mathcal{C}$ and a set of prototypes $\mathcal{Y} = \{Y_1, .., Y_k\} \subseteq \mathcal{X}$, also with class labels belonging to $\mathcal{C}$, we embed each pattern $X \in \mathcal{S}$ in $\mathbb{R}^k$ in the following fashion

$$X \rightarrow (d(X, Y_1), ..., d(X, Y_k)) \tag{4.1}$$

where $d(.,.)$ is a dissimilarity measure.

The choice of prototypes determines the embedding of patterns into a dissimilarity space (also abbreviated as DSS in this chapter), which in turn has a bearing on classifier performance. The objective is to choose the prototypes such that the embedded data shows robust class coherence while achieving effective interclass separation.

We now assume that the classifiers in the DSS are a set of hyperplanes. This is merely an assumption to simplify the analysis. In practise, one uses non-linear classifiers (such as typically SVM's with Gaussian or Polynomial kernels) to achieve superior classification results. Every class with label $i \in \mathcal{C}$ has a corresponding class region $\mathcal{H}_i$ in the dissimilarity space, linear in the coefficients and in the features given by embedding described by Eq. 4.1. The DSS is partitioned by linear *discriminant functions* into different class regions, with each region corresponding to a class label $i$.

$$\mathcal{H}_i : \sum_{p=1}^{k} \gamma_{ip} d(X, Y_p) \tag{4.2}$$

where $\gamma_{ip}$ are the coefficients which determine the linear classifier belonging to some class $i$. For every pattern $X$, the set of prototypes $\mathcal{Y}$ could be split into two subsets- one subset of prototypes having the same label as that of data referred to as $\mathcal{Y}(X)$ and another subset consisting of prototypes having labels different than the data, $\mathcal{Y}'(X)$. We use notation such as $p \in \mathcal{Y}(X)$ to mean that prototype $Y_p$ has the same label as data $X$, i.e. $Y_p \in \mathcal{Y}(X)$. We rewrite the discriminant

function in Eq. 4.2 for the class $i$ as below

$$\mathcal{H}_i : \sum_{p \in \mathcal{Y}(X)} \gamma_{ip} d(X, Y_p) + \sum_{p \in \mathcal{Y}'(X)} \gamma_{ip} d(X, Y_p) \tag{4.3}$$

We now require that the discriminant function $\mathcal{H}_i$ corresponding to the label of the pattern $X$ be minimized i.e.

$$min( \sum_{p \in \mathcal{Y}(X)} \gamma_{ip} d(X, Y_p) + \sum_{p \in \mathcal{Y}'(X)} \gamma_{ip} d(X, Y_p)) \tag{4.4}$$

for $i = l(X)$.

The class information is incorporated in the following manner- the coefficients corresponding to the prototype term having the same class label as the data is positive and the coefficients corresponding to the prototypes having different class label as the data is negative. We rewrite Eq. 4.4 in the following manner

$$min( \sum_{p \in \mathcal{Y}(X)} \alpha_{ip} d(X, Y_p) - \sum_{p \in \mathcal{Y}'(X)} \beta_{ip} d(X, Y_p)) \tag{4.5}$$

where,

$$\alpha_{ip}, \beta_{ip} \geq 0 \tag{4.6}$$

for $i = l(X)$

Moreover, to prevent numerical instabilities from occurring, we impose the following constraint.

$$\sum_{p \in \mathcal{Y}(X)} \alpha_{ip}^2 = 1, \sum_{p \in \mathcal{Y}'(X)} \beta_{ip}^2 = 1, \forall i \in \mathcal{C} \tag{4.7}$$

The non-negativities of $\alpha_{ip}$ and $\beta_{ip}$ have the effect of pushing the prototypes belonging to the same class as that of a pattern towards it, while repelling away the prototypes belonging to different classes. If the constraint in Eq. 4.6 were not present, then the minimization of cost function would result in all prototypes being pushed towards the pattern irrespective of the class labels, analogous to an unsupervised clustering algorithm.

The constraints in Eq. 4.7 ensure the boundedness at every update step. We now need to derive a prototype update rule for minimizing the cost given by Eq. 4.5 summed over all patterns $X$ in the set. Such a rule would update every prototype uniformly irrespective of its distance from the data. In order to avoid modifying prototypes that are far away from the pattern (especially repelling prototypes of other classes which are far already), we introduce a tuning function (analogous to one introduced by Kohonen [44]), that is "active" (non-zero) for some of the prototypes (say for the two nearest prototypes to the pattern) and vanishes otherwise. The tuning function is dependent upon both the prototypes and data pattern. Ideally, it should be active only for few prototypes close to the data.

The modified cost function for a single pattern $X$ to be minimized now becomes,

$$\sum_{p\in\mathcal{Y}(X)} \alpha_p d(X,Y_p)\Delta_\alpha - \sum_{p\in\mathcal{Y}'(X)} \beta_p d(X,Y_p)\Delta_\beta \tag{4.8}$$

Henceforth, implicitly, $\alpha_p, \beta_p$ denote the coefficients $\alpha_{ip}, \beta_{ip}$ for $i = l(X)$. Similarly $\Delta_\alpha, \Delta_\beta$ are functions of $X, Y_p$.

The cost for embedding the training set is given by,

$$E_{\mathcal{S}}(\mathcal{Y},\alpha,\beta) = \sum_{m=1}^{n}\Big(\sum_{p\in\mathcal{Y}(X_m)} \alpha_p d(X_m,Y_p)\Delta_\alpha - \sum_{p\in\mathcal{Y}'(X_m)} \beta_p d(X_m,Y_p)\Delta_\beta\Big) \tag{4.9}$$

under the constraints

$$\sum_{p\in\mathcal{Y}(X)} \alpha_p^2 = 1,\ \sum_{p\in\mathcal{Y}'(X)} \beta_p^2 = 1, \alpha_p, \beta_p \geq 0 \tag{4.10}$$

### 4.2.2 Prototype update rule

If the original patterns are attributed graphs, then the distance measure is given by the graph metric introduced in Chapter 2. If the patterns are feature vectors, the distance measure is chosen as Euclidean distance. Then, applying stochastic subgradient descent to minimize the cost function results in the following update

rule for the prototypes at step $t$ for the discriminant function belonging to class $i = l(X)$,

$$\begin{aligned} Y_{p_{t+1}} &= Y_{p_t} - \eta\alpha_p\Delta_\alpha\partial d(X, Y_{p_t}); p \in \mathcal{Y}(X) \\ Y_{p_{t+1}} &= Y_{p_t} + \eta\beta_p\Delta_\beta\partial d(X, Y_{p_t}); p \in \mathcal{Y}'(X) \end{aligned} \tag{4.11}$$

where, $\eta$ is the learning rate corresponding to the step size in the gradient descent. $\partial d(X, Y_{p_t})$ refers to subgradient which coincides with the gradient with respect to $Y$ for feature vectors with Euclidean distance.

In order to calculate the normalized factors $\beta_p, \alpha_p$, we set up the Lagrangian for the cost function incorporating the constraints as follows,

$$\begin{aligned} L(X, \mathcal{Y}) = &\sum_{p\in\mathcal{Y}(X)} \alpha_p d(X, Y_p)\Delta_\alpha - \sum_{p\in\mathcal{Y}'(X)} \beta_p d(X, Y_p)\Delta_\beta \\ &+\lambda_1(\sum_{p\in\mathcal{Y}(X)} \alpha_p^2 - 1) + \lambda_2(\sum_{p\in\mathcal{Y}'(X)} \beta_p^2 - 1) \end{aligned} \tag{4.12}$$

where $\lambda_1, \lambda_2$ are the Lagrange multipliers.

Setting the derivatives in Eq. 4.12 with respect to $\beta_p, \alpha_p$ to vanish yields,

$$\begin{aligned} \alpha_p &= -\frac{d(X, Y_p)\Delta_\alpha}{2\lambda_1}; p \in \mathcal{Y}(X) \\ \beta_p &= \frac{d(X, Y_p)\Delta_\beta}{2\lambda_2}; p \in \mathcal{Y}'(X) \end{aligned} \tag{4.13}$$

We now solve for $\lambda_1, \lambda_2$. Plugging Eq. 4.13 in the constraints given by Eq. 4.10

$$\begin{aligned} \lambda_1 &= -\sqrt{\sum_{p\in\mathcal{Y}(X)} d^2(X, Y_p)\Delta_\alpha^2} \\ \lambda_2 &= \sqrt{\sum_{p\in\mathcal{Y}'(X)} d^2(X, Y_p)\Delta_\beta^2} \end{aligned} \tag{4.14}$$

Hence, the coefficients in the update rule in Eq. 4.11 are given by,

$$\alpha_p = \frac{d(X,Y_p)\Delta_\alpha}{\sqrt{\sum_{p\in\mathcal{Y}(X)} d^2(X,Y_p)\Delta_\alpha^2}}; p \in \mathcal{Y}(X)$$
$$\beta_p = \frac{d(X,Y_p)\Delta_\beta}{\sqrt{\sum_{p\in\mathcal{Y}'(X)} d^2(X,Y_p)\Delta_\beta^2}}; p \in \mathcal{Y}'(X) \tag{4.15}$$

The Eqs. 4.11, 4.15 are the formulas for prototype update. For different functional values of $\Delta_\alpha, \Delta_\beta$, different prototype learning rules emerge.

Consider the following scenario. $\Delta_\alpha = 1$ only for the nearest prototype (from the same class) if it is not the nearest prototype overall and zero otherwise. Similarly, $\Delta_\beta = 1$ only for the nearest prototype (from the different class) if it is not the nearest prototype overall and zero otherwise. Then, the prototype learning rule is identical to LVQ2.1/LGQ2.1. This follows from Eq. 4.15, where $\alpha_p = \beta_p = 1$ for the nearest correct and incorrect prototypes respectively and zero for all other prototypes.

If $\Delta_\alpha = 0$ throughout, i.e. the prototypes belonging to the same class as the pattern are never updated and $\Delta_\beta = 1$ for only the nearest incorrect prototype, which is also the nearest prototype overall, the prototype update rule is similar to LVQ1/LGQ1. This is because the term $\beta_p = 1$ only for $p = inf_{p\in\mathcal{Y}'(X)}(d(X,Y_p))$ and vanishes for every other other $p$.

### 4.2.3 Embedding based on Generalized Learning Graph Quantization algorithms

Recently, extensions have been proposed for Generalized Learning Vector Quantization algorithms- Generalized Learning Vector Quantization(GLVQ) and Generalized Relevance Learning Vector Quantization (GRLVQ) in the domain of graphs [60, 31]. These algorithms are referred to as Generalized Learning Graph Quantization(GLGQ) and Generalized Relevance Learning Graph Quantization (GRLGQ) respectively [38]. We present a short review here.

The GLGQ algorithm aims at minimizing the following cost function

$$E_{\mathcal{S}}(\mathcal{Y}) = \sum_{m=1}^{n} f(\mu(d(X_m, \mathcal{Y})) \tag{4.16}$$

On account of its smooth, monotonic nature and its widespread use, sigmoidal function is chosen for $f(.)$ defined as $sgd(x) = (1 + e^{-x})^{-1}$. $\mu$ is defined as $\frac{d_{NCP} - d_{NIP}}{d_{NCP} + d_{NIP}}$ where, $d_{NCP}$ and $d_{NIP}$ denotes the distance to the nearest correct prototype and the nearest incorrect prototype. Such a definition is chosen considering its boundedness and convergence properties. It must be pointed out that the function $sgd(.)$ is bounded, hence the prototypes converge.

If the original patterns are feature vectors in $\mathbb{R}^n$, a natural choice for a differentiable dissimilarity measure $d(.,.)$ is the square of the Euclidean norm. For graphs, we use the definition of graph metric. Again, the cost function is generalized differentiable as it is a composition of generalized differentiable functions. Hence, a subgradient descent rule to update the nearest correct prototype ($y_{NCP}$) and nearest incorrect prototype ($y_{NIP}$) gives the following equations

$$\begin{aligned} \Delta \boldsymbol{y_{NCP}} &= \eta \frac{sgd'(\mu(x_i)) d_{NIP}}{(d_{NCP} + d_{NIP})^2} (\boldsymbol{x_i} - \boldsymbol{y_{NCP}}) \\ \Delta \boldsymbol{y_{NIP}} &= -\eta \frac{sgd'(\mu(x_i)) d_{NCP}}{(d_{NCP} + d_{NIP})^2} (\boldsymbol{x_i} - \boldsymbol{y_{NIP}}) \end{aligned} \tag{4.17}$$

where, $(\boldsymbol{x}, \boldsymbol{y_{NCP}}) \in \mathcal{A}(X, Y_{NCP})$ and $(\boldsymbol{x}, \boldsymbol{y_{NIP}}) \in \mathcal{A}(X, Y_{NIP})$.

Hammer et. al. [31], propose to kernelize the GLVQ algorithms by modifying the distance function $d$ to include a weight for every component as $d(x, y) = \sum \lambda_i (x_i - y_i)^2$. This implies that every component/feature has a weight which associates to it how important or relevant it is for classification. In this case, the prototype and weight update rule corresponds to the learning scheme called Generalized Relevance Learning Vector Quantization. An analogous formulation has been proposed to graphs as well referred to as Generalized Relevance Learning Graph Quantization algorithm (GRLGQ). For the GRLGQ algorithm, the update equations given by Eq. 4.17 hold. There is an additional rule for updating the coefficients $\Lambda := \{\lambda_i\}$

$$\Delta\Lambda_{NCP} = -\eta \frac{sgd'(\mu(x_i))d_{NIP}}{(d_{NCP}+d_{NIP})^2}(x_i - y_{NCP})^2$$
$$\Delta\Lambda_{NIP} = -\eta \frac{sgd'(\mu(x_i))d_{NCP}}{(d_{NCP}+d_{NIP})^2}(x_i - y_{NIP})^2 \quad (4.18)$$

We use these algorithms to learn the prototypes and then embed the graphs into a dissimilarity space using the prototypes so learnt.

## 4.3 Experiments

The goal of the experiments are two-fold- to compare the classification performance based on the prototype embedding scheme discussed before with state-of-the-art results and to assess the improvement of classification results by embedding the graphs into dissimilarity space spanned by the prototypes compared to using the prototypes directly for classification within the Nearest Prototype (NP) framework.

### 4.3.1 Data and settings

To assess the performance of the proposed prototype based embedding, we conducted a set of experiments on the datasets from the IAM Graph Database Repository [57]. The characteristics of the dataset are summarized in Table 4.1.

| Dataset | #(graphs *tr, va, te*) | #(classes) | avg(nodes) | max(nodes) | avg(edges) |
|---|---|---|---|---|---|
| Letter (HIGH) | 750, 750, 750 | 15 | 4.7 | 8 | 3.1 |
| GREC | 286, 286, 528 | 22 | 11.5 | 25 | 12.2 |
| Fingerprint | 500, 300, 2000 | 4 | 5.42 | 26 | 4.42 |

Table 4.1: Summary of main characteristics of the data sets.

We generate the prototypes by using the family of prototype learning algorithms viz. LGQ1, LGQ2.1, GLGQ and GRLGQ. The algorithms were applied to the training set of each dataset 10 times. We then chose the model that performs best on the validation set. In the next step, we embedded the graphs into

a dissimilarity space (DSS) based on the distance to the prototype set. In the DSS, we classified the dataset using Support Vector Machines with Gaussian and Polynomial kernels whose parameters were optimized by cross validation.

**Setting of the LGQ algorithms** : We initialized the prototypes classwise by assigning just one prototype per class, typically class mean. The algorithms were iterated $n_{iter} = 100$ times through the training set. The annealing schedule was chosen to be $\eta_t = 0.01(1 - t/n_{iter})$ at every step $t$. For LGQ2.1, the window width was chosen to be 0.1 for the Letter and GREC datasets and 0.2 for the Fingerprint dataset.

We chose the following state-of-the-art classification techniques for comparison.

*k- Nearest Neighbour (kNN)*- Reference system [57]

*Similarity Kernels*- Similarity kernels [58] are defined which transform graph distances $d(g, g')$ to a similarity measure $e^{-\gamma d(g,g')^2}$, which is an indefinite kernel for a subsequent SVM classifier. The parameter $\gamma$ is the meta parameter that needs to be determined based on performance on the validation set. The standard training paradigm is applied to the SVM classifier in the next stage. This procedure is referred to as *SK + SVM*.

*Lipschitz embedding*- Given a dataset of graphs $\mathcal{G} : \{g_1, .., g_m\}$, we define a set $\mathcal{S} : \{\mathcal{P}_1, .., \mathcal{P}_n\}$ consisting of $n$ subsets of $\mathcal{G}$, called *reference set* for embedding. The Lipschitz embedding with respect to $\mathcal{P}$, $\phi_{\mathcal{S},f} : \mathcal{G} \rightarrow \mathbb{R}^n$ is defined as, $\phi_{\mathcal{S},f}(g) = (f(g, \mathcal{P}_1), .., f(g, \mathcal{P}_n))$, where $f$ could be the minimum, maximum or mean distance of the graph to elements in the set $\mathcal{P}_n$. The meta paremeters here are the size of the reference set. The optimal parameters are chosen in conjunction with optimizing the SVM classifier in the next stage. This procedure is referred to as *LE + SVM* [58].

*SVM recursive feature selection*- This approach uses the most significant features depending upon how crucial they are for classification. Consider the decision surface of a linear SVM $f(x) =< w, x > +b, w, x \in \mathbb{R}^n, b \in \mathbb{R}$. The influence a feature $x_i$ exerts on classification depends on weight $w_i$. Hence, a SVM is trained, the features ranked based on their corresponding weights and low ranking features recursively eliminated until an optimal number of features remain. This

procedure is referred to as *Feat. sel. (svm)* [10].

*Feature selection through component analysis*- Principal Component Analysis (or its nonlinear extension kernel Principal Component Analysis) is a popular technique for projecting original data into a space spanned by eigenvectors of its covariance matrix. The most directions (or components) with high variances correspond to higher eigenvalues. Now a feature ranking strategy is implemented by choosing dimensions which are crucial for classification. This procedure is referred to as *Feat. sel. (kpca)* [10].

### 4.3.2 Results

| Method | Letter | Fingerprint | GREC |
|---|---|---|---|
| kNN | 82 | 76.7 | 95.5 |
| SK + SVM | 79.1 | 41 | 94.9 |
| LE + SVM | 92.5 | 82.8 | 96.8 |
| Feat. sel. (svm) | **92.8** | 81.7 | 92.2 |
| Feat. sel. (kpca) | 90.3 | 82.6 | 91.6 |
| *LGQ1 + emb.* | 83.7 | 81.6 | **97.6**$^{\star,\dagger}$ |
| *LGQ2.1 + emb.* | 88.7 | 81.7 | 96.9 |
| *GLGQ + emb.* | 89.5 | **85.3**$^{\star,\dagger,\ddagger}$ | 97 |
| *GRLGQ + emb.* | 87.1 | 84.7 | 97.2 |

Table 4.2: Comparing classification accuracy

The symbol $\star$ indicates that the improvement of result is statistically significant [18] compared to kNN, and † denotes a statistically significant improvement compared to feature selection algorithms (Feat. sel. (svm/pca)) and ‡ indicates improvement over family of state-of-the-art embedding algorithms (SK/LE + SVM) at 0.05 level of significance. The best result overall is typed in boldfront.

We observe overall that prototypes obtained through LGQ algorithms give rise to a feature space embedding that shows significant improvement over state-of-the-art techniques on Fingerprint and GREC datasets. The results so obtained have been the best so far and shows statiscally significant improvement.

On closer examination,the following observations are made- the class of embedding procedures is significantly better than kNN on all the datasets. This is

due to the fact that there is a process of supervised learning in the feature (dissimilarity) space after the features have been learnt in contrast to kNN, where only the parameter $k$ is chosen based on training observations. As the best result over the test set for all *LE + SVM, SK + SVM* classifiers and a family of feature selection methodology is chosen, the comparison is clearly biased towards the same. Even so, the results obtained using the prototypes obtained by GLGQ algorithm and LGQ1 algorithm betters the results on Fingerprint and GREC datasets respectively. Similar improvement of results have been obtained comparing with feature selection algorithms using svm and kpca.

The most important observation is that family of LGQ algorithms with just one prototype per class yields prototypes which give better results in the embedded feature space than state-of-the art techniques, which demand a much larger prototype set. We place this observation in the context of graph generation process which involves distorting a few base patterns (which are ideally the prototypes) to produce the entire dataset and the ability of LGQ algorithms to approximate the base graphs. This also has the additional advantage of less computational complexity as the number of prototypes is low (just one per class), which reduces extensively the number of graph matching operations. The training cycle for SVM's in the feature space is not computationally expensive as there exist computationally efficient algorithms for that purpose.

Additionally, embedding the dataset into the dissimilarity space improves the classification performance compared to using the prototypes as classifiers directly in the graph domain using the Nearest Prototype framework (ref. Table (4.3)). This is due to the fact that a combination of all dissimilarities due to the entire prototype set has a more powerful discriminative ability than just a single dissimilarity feature corresponding to the nearest prototype. This is more so for the LGQ1 and LGQ2.1 algorithms where there is increase for all the datasets (where $\star$ denotes a statistically significant improvement at level of 0.05). We postulate this happens because there is a limit to how much learning could be done overall- Algorithms such as LGQ1 and LGQ2.1 do not learn as much as GLGQ and GRLGQ in the graph space, so there is a possibility for learning in the subsequent embedding space, while the latter class learns more in the graph space leaving less scope for learning in the dissimilarity space, potentially over

Table 4.3: Classification accuracy in embedded space (EMB) and Nearest Prototype framework (NPC)

| Algorithm | Scheme | Letter | Fingerprint | GREC |
|---|---|---|---|---|
| LGQ1 | NPC | 80.3 | 80.6 | 87.8 |
| | EMB | 83.7 | 81.6 | 97.6$^\star$ |
| LGQ2.1 | NPC | 85.6 | 80.5 | 93.7 |
| | EMB | 88.7 | 81.7 | 96.9$^\star$ |
| GLGQ | NPC | 88.1 | 84.1 | 97.7 |
| | EMB | 89.5 | 85.3 | 97 |
| GRLGQ | NPC | 85.7 | 83.3 | 97.3 |
| | EMB | 87.1 | 84.7 | 97.2 |

learning as the slight degradation of classification accuracy for the GREC dataset would seem to indicate. However, more analytical and qualitative studies have to be conducted to help better understand this observation. One possible approach might be to compare the generalization performance of LGQ classifiers directly with the classifiers in the dissimilarity space. On a related note, we motivate the classification results due to embedding from different LGQ algorithms through concepts from statistical learning theory in the next section.

## 4.4 Analysis based on statistical learning theory

*Number of features*

We begin by analyzing the optimal size of prototype set for embedding. The number of features (or the dimensionality of embedding feature space), is preferably as low as possible, as seen from the following relation [66].

$$R(\alpha) = R_{emp}(\alpha) + \sqrt{\frac{VC(H)(\frac{2m}{VC(H)} + 1) + ln(\frac{4}{\delta})}{m}} \quad (4.19)$$

$R_{emp}$ is the empirical error. For a set of linear classifiers in the DSS,

$$VC(H) = k + 1 \tag{4.20}$$

where $k$ is the dimension of the DSS which is equal to the number of prototypes.

Hence, in order for the VC dimension to be less, the number of prototypes should be as less as possible. Hence, it is advisable to choose just one prototype per class in order to reduce generalization error. It is also noted here that such a choice is advantageous from the point of view of computational complexity.

*Prototypes as defining kernels*

The prototypes define a kernel which embeds the graphs into a feature space. Consider the mapping induced by the prototype set of the graphs into a feature space as given by,

$$\phi(x) \rightarrow (d(x, y_1), .., d(x, y_p)) \tag{4.21}$$

with the kernel defined as,

$$\begin{aligned} K(x_i, x_j) &= \phi(x_i)^T \phi(x_j) \\ &= \sum d(x_i, y_p) d(x_j, y_p) \end{aligned} \tag{4.22}$$

As has been pointed out by Chapelle et al. [11], the ratio $\frac{R^2}{\gamma^2}$, (R2-W2 ratio) [66] is a suitable measure for evaluating the "goodness" of a kernel, where $R$ is the radius of the minimum enclosing ball of data in the feature space, and $w$ is the inverse of margin i.e. $w^2 = \frac{1}{\gamma^2}$. Informally, it could be stated that lower the ratio, better the kernel (assuming of course that empirical error is equal, which we observed in experiments as well).

We observe that R2-W2 ratio compares favorably to GLGQ algorithm on Letter and Fingerprint datasets, for which these prototypes also give the best results in the dissimilarity space. For GREC dataset, the best ratio is obtained for LGQ1 algorithm, which also gives the best classification performance.

In order to get a better insight, we look at the radius of the minimum enclosing

| Algorithm | Letter | Fingerprint | GREC |
|---|---|---|---|
| LGQ1 | 18 | 10.8 | 2.3 |
| LGQ2.1 | 6.7 | 8.4 | 15.3 |
| GLGQ | 6.3 | 3.7 | 11.3 |
| GRLGQ | 7.8 | 5.6 | 7.6 |

Table 4.4: R2-W2 ratio

ball (MEB) in the dissimilarity space:

| Algorithm | Letter | Fingerprint | GREC |
|---|---|---|---|
| LGQ1 | 12.5 | 4.9 | 6.8 |
| LGQ2.1 | 15.9 | 5.3 | 12.1 |
| GLGQ | 12 | 5.1 | 6.6 |
| GRLGQ | 21.4 | 5.2 | 7.7 |

Table 4.5: Radius of minimum enclosing ball

The radius is comparable for LGQ1 and GLGQ algorithms. For the GLGQ algorithm, this effect manifests itself in the low R2-W2 value as well. The low value of radius for LGQ1 algorithm is best understood in terms of the prototype update, which adjusts only the winning prototype. However, this means that the margin is low in this case. Among algorithms that simultaneously adjust multiple prototypes, GLGQ yields both the lowest radius and R2-W2 ratio.

## 4.5 Conclusion

Dissimilarity based representation is a very useful technique to embed attributed graphs into a feature space to carry out classification tasks. The state-of-the-art techniques construct the dissimilarity space by choosing the prototypes in a fashion, overlooking the subsequent classification problem. In this chapter, we have investigated classification of the graph set in the space generated by prototypes learnt using a family of learning algorithms. We motivate this by setting up a cost function for the prototype set taking into account the class labels. The online algorithm for optimizing this cost function subsumes Learning Vector Quantization

algorithms as a special case and in the domain of attributed graphs corresponds to the class of Learning Graph Quantization algorithms for special functional values. Classifiers in the dissimilarity space spanned by the prototypes so obtained achieve a high accuracy rate, with a considerably small prototype set and a standard, principled initialization procedure. We also consider classification in the space spanned by prototypes obtained by two generalized learning algorithms-Generalized Learning Graph Quantization and Generalized Relevance Learning Graph Quantization algorithms.

In order to better understand the classification results, we consider the R2-W2 ratio of the classifiers in the dissimilarity space. Lower the bound, better the generalization performance. It is seen that bounds are lowest for GLGQ algorithm for two of the three datasets, corresponding exactly to its superior classification results. Further research should focus on the two following directions

-establish a clear analytical link between minimizing the bounds and learning algorithms

-learning prototypes directly based on minimizing $\frac{R^2}{\gamma^2}$ ratio directly binds the prototype optimization procedure with maximizing classifier margin. Such a scheme would result in a classifier with the most optimal generalization performance.

# Chapter 5

# Probabilistic models of attributed graphs

In this chapter we propose a new approach towards developing a class of probabilistic methods for classifying attributed graphs. The key concept is a *random attributed graph*, which is defined as an attributed graph whose nodes and edges are annotated by random variables. Every node/edge has two probabilistic variables associated with it- structural probability, which describes a probability that the node/edge occurs in an instantiation of the random graph and the probability distribution over the attribute values. We then develop estimates of structural and attribute probability distributions. The likelihood of a random attributed graph to generate an outcome graph is used as a feature for classification.

## 5.1 Introduction

The analysis of a dataset of patterns represented by attributed graphs is a challenging problem and is closely related to carrying out machine learning tasks such as clustering and classification. Developing probabilistic models for attributed graphs allows us to statistically describe a set of attributed graph data and also enables us to develop machine learning algorithms to deal with them. Wong et al. [72, 71], propose a concept of random graph, which takes into account structural and contextual probabilities. An instantiation (outcome) of a random graph is an

attributed graph, which enables the characterization of an ensemble of outcome graphs with a probability distribution. Sole-Ribalta et al. [64], generalize the idea of random graph to structure described random graphs (SDRG), with node and edge value distributions. Algorithms have been proposed where random graphs are used for classification by first synthesizing random graph as a model for every class and assigning data to a class depending upon which model has highest likelihood of having generated it. This framework has also been adopted by Seong et al. [62], to develop an incremental clustering algorithm for attributed graphs, Sengupta et al. [61], to efficiently organize large structural modelbases for quick retrieval. There are two features of such a definition that are quite noteworthy- the fact that both the structural and contextual probabilities are considered, under suitable independence assumptions the estimation of these parameters could be simplified and this definition gives us the ability to deal with a wide variety of attribute values.

The present chapter aims to use the notion of probabilistic modeling of attributed graphs to develop new techniques for classification within this framework. Hence, we consider the following two-fold theme: first, we develop a probabilitic model based clustering algorithm, where the cluster centres are random graphs. We then consider the cluster centre graphs as prototypes. It has been demonstrated that [54], given any set of objects and a prototype set, it is possible to consider the distance of an object to every element of the prototype set as a feature for classification. We then show that for a set of graphs and prototype set consisting of random graph set, the likelihood of each random (prototype) graph generating the graphs is a feature for classification. In this feature space spanned by the likelihood values, we apply Support Vector Machines for classification, thus demonstrating the utility of random attributed graphs for classification.

This chapter is organized as follows: in Section 2 we give a brief review of the theory of random attributed graphs subsequently deriving the clustering cost function. Subsequently, we derive an online density estimate for the node and edge attributes. The motivation and concepts for using random attributed graphs as a prototype for feature generation is presented in Section 3; Section 4 discusses experimental results; concluding remarks and directions for future research comprise Section 5.

## 5.2 Model based clustering for attributed graphs

### 5.2.1 Random attributed graphs- A review

A random graph is a graph whose nodes and edges are finite probability distributions. Each outcome of a random graph is a labeled graph and a morphism of the labeled graph into the random graph. The morphism specifies for each vertex (or edge) of the outcome graph which vertex (or edge) of the random graph generated it. The probability space of random graphs should be such that, the outcomes are attributed graphs with specified morphism relations and is complete. The definitions in this section follow the original contribution in [72] closely.

Technically, the random graph $\mathfrak{G} = (\mathfrak{V}, \mathfrak{E})$ is defined to be such that:

1. Each vertex $\mathfrak{v} \in \mathfrak{V}$ and edge $\mathfrak{e} \in \mathfrak{E}$ is a finite probability distribution
2. $\forall \mathfrak{e} \in \mathfrak{E}$, $p(\mathfrak{e} = \phi | \sigma(\mathfrak{e}) = \phi) = 1$
3. The space of joint distribution of all random nodes and random edges is complete

Notation: Elements of the random attributed graph are represented by $\mathfrak{fraktur}$ script.

Condition 2 ensures that an edge can occur in an outcome only if both its ends (terminal nodes, given by $\sigma(\mathfrak{e})$) occur. Completeness means that the space is indeed a (standard) probability space. Consider the probability space of the joint distribution. This space is the probability space of attributed graphs and every outcome is an attributed graph.

Let $G = (V, E)$ be an outcome graph. A morphism $\mu : V \rightarrow \mathfrak{V}$ and $\nu : E \rightarrow \mathfrak{E}$ specifies the structural mapping between the random graph and its outcome. Thus, an outcome of a random graph is specified by the tuple $(V, E, \gamma)$, where $\gamma = (\mu, \nu)$. It is to be noted that the mappings $\mu$ and $\nu$ are into and the inverse mappings $\mu'$ and $\nu'$ are such that some elements could be mapped to $\phi$, i.e. $\mu'(\mathfrak{v}) = \phi$ if no morphism exists. The probability of an outcome graph is then the probability of its joint outcome described by the following,

$$p_{\mathfrak{G}}(G, \gamma) = prob\{(v = \mu'(\mathfrak{v}), \forall \mathfrak{v} \in \mathfrak{V}, \alpha(v) = \alpha_i), (e = \nu'(\mathfrak{e}), \forall \mathfrak{e} \in \mathfrak{E}, \beta(e) = \beta_i)\} \tag{5.1}$$

where $\alpha(v)$ is the node attribute function that assigns attribute for every node and $\alpha_i$ denotes the particular node attribute value. $\beta(e), \beta_i$ are the corresponding edge attribute function and values respectively.

We make the following assumptions- node occurences are independent, and edge occurences depend only on the nodes that the edge is incident to. The basis for our assumptions are that they give rise to a computationally tractable model, with a factorizable probability density function and the node and edge occurences contribute independently to the density function, which is the case in many applications.

Then, we can simplify Eq.(5.1) to,

$$p_{\mathfrak{G}}(G,\gamma) = \prod_{\mu'(\mathfrak{v})\neq\phi} p(\mathfrak{v}) \prod_{\mu'(\mathfrak{v})=\phi} q(\mathfrak{v}) \prod_{\nu'(\mathfrak{e})\neq\phi} p(\mathfrak{e}) \prod_{\nu'(\mathfrak{e})=\phi} q(\mathfrak{e})$$
$$\prod_{\mu'(\mathfrak{v})\neq\phi} prob(\alpha(\mu'(\mathfrak{v}) = \alpha_v)) \prod_{\nu'(\mathfrak{e})\neq\phi} prob(\beta(\nu'(\mathfrak{e}) = \beta_e)) \tag{5.2}$$

where $p(\mathfrak{v})$ denotes a probability that the node $\mathfrak{v}$ occurs and $q(\mathfrak{v}) = 1 - p(\mathfrak{v})$ is the probability that $\mathfrak{v}$ doesn't occur. Similar notation has been adopted for the edges as well. We note that formula in 5.2 decomposes the probability of an attributed graph instance as the product of probability of nodes/edges of generating random graphs that occur in the outcome, "not occurence" probability of nodes/edges that are absent in the outcome, and the probability of the occuring nodes/edges to assume their respective attribute values.

The total probability density function for a random attributed graph with continuous attribute values is then given as below,

$$p_{\mathfrak{G}}(G,\gamma) = \prod_{\mathfrak{v}\in\mathfrak{V}} (\mathfrak{1}_{\mathfrak{v}} p_{\mathfrak{v}} \mathcal{F}_{\mathfrak{v}}(x_{\mathfrak{v}}) + (1 - \mathfrak{1}_{\mathfrak{v}}) q_{\mathfrak{v}}) \prod_{\mathfrak{e}\in\mathfrak{E}} (\mathfrak{1}_{\mathfrak{e}} p_{\mathfrak{e}} \mathcal{F}_{\mathfrak{e}}(x_{\mathfrak{e}}) + (1 - \mathfrak{1}_{\mathfrak{e}}) q_{\mathfrak{e}}) \tag{5.3}$$

where $\mathfrak{1}$ is the indicator variable equal to identity for the node/edge occurences and null otherwise and $\mathcal{F}$ is the attribute value distribution for nodes/edges. Werecall that by our assumptions, node occurence probabilities are independent of each other. Therefore, we model them each by Bernoulli distributions. We also

model edge occurence probabilities by Bernoulli distributions, with the constraint that edges can occur only if both of their end nodes occur.

### 5.2.2 Model based clustering of attributed graphs

The estimation of structural parameters of a random graph given a dataset follows from maximizing the likelihood: the node and edge occurence probabilities of random graphs are set to those values which maximize the likelihood of the dataset being generated due to the random graph. Therefore, we see that the total cost function is,

$$E(\{G_i\}, \mathfrak{W}) = \sum_i \ln p(\mathfrak{W}(G_i)) \tag{5.4}$$

where $\ln p(\mathfrak{W}(G_i))$ is the likelihood that random graph $\mathfrak{W}$ generates the graph $G_i$. The node and edge attributes are in many cases given by feature vectors. Hence, we assume the attribute vectors to be generated by Gaussian distributions whose means and covariances are to be determined. This reflects the process by which graph datasets are obtained in many applications: a few base graphs are corrupted by variable levels of noise due to change in state or measurement errors. Also, this model provides for online density estimation. These assumptions, however do not hold for discrete node and edge attribute variables, in which case a kernel/histogram density estimator could be applied or graph datasets which consists of graphs that have drastic attribute dissimilarities even within a single class.

The likelihood function is therefore,

$$p(\mathfrak{W}(G_i)) = \prod_{\mu'(\mathfrak{v})\neq\phi} p(\mathfrak{v}) \prod_{\mu'(\mathfrak{v})=\phi} q(\mathfrak{v}) \prod_{\nu'(\mathfrak{e})\neq\phi} p(\mathfrak{e}) \prod_{\nu'(\mathfrak{e})=\phi} q(\mathfrak{e}) \prod_{\mu'(\mathfrak{v})\neq\phi} \mathcal{N}_{\mathfrak{v}}(x_{\mathfrak{v}}) \prod_{\nu'(\mathfrak{e})\neq\phi} \mathcal{N}_{\mathfrak{e}}(x_{\mathfrak{e}}) \tag{5.5}$$

and the log-likelihood is then,

$$\ln(p(\mathfrak{W}(G_i))) = \sum_{\mu'(\mathfrak{v})\neq\phi} \ln p(\mathfrak{v}) + \sum_{\mu'(\mathfrak{v})=\phi} \ln q(\mathfrak{v}) + \sum_{\nu'(\mathfrak{e})\neq\phi} \ln p(\mathfrak{e}) + \sum_{\nu'(\mathfrak{e})=\phi} \ln q(\mathfrak{e}) + \sum_{\mu'(\mathfrak{v})\neq\phi} (C - \frac{1}{2}\ln(|\Sigma_{\mathfrak{v}}|) - \frac{1}{2}(x-\mu_{\mathfrak{v}})^T\Sigma_{\mathfrak{v}}^{-1}(x-\mu_{\mathfrak{v}}) + \sum_{\nu'(\mathfrak{e})\neq\phi} (C - \frac{1}{2}\ln(|\Sigma_{\mathfrak{e}}|) - \frac{1}{2}(x-\mu_{\mathfrak{e}})^T\Sigma_{\mathfrak{e}}^{-1}(x-\mu_{\mathfrak{e}}) \tag{5.6}$$

where $C$ is a generic symbol for constant terms that arise.

Initially, we maximize the cost function with respect to the node and edge occurence probabilities $p(\mathfrak{v})$ and $p(\mathfrak{e})$. As the node occurences are independently modelled by Bernoulli distributions, the ML estimate is the fraction of its occurences in the dataset [4]

$$p(\mathfrak{v}) = \frac{n_{\mathfrak{v}}}{N} \tag{5.7}$$

where $n_{\mathfrak{v}}$ is the number of occurences of node $\mathfrak{v}$ in the sample set. Similar estimates hold good for the edges except that edge occurence probabilities are normalized by their respective node probabilities (accounting for the fact that the edges cannot occur if any of their end nodes do not occur).

### 5.2.3 Estimating the density function of node and edge attributes

We now consider the problem of estimating means and covariances of node and edge attribute distributions. It is possible to derive gradient descent update rule for the means and covariance matrices. The vanilla gradient descent where the means and covariances are updated in the direction of the gradient (as it is assumed to the steepest direction) is not ideal, as it ignores the geometry of the underlying probability space. Therefore, we use Natural gradient descent to estimate the mean and covariance online[34, 2].

Natural gradient descent is a modification of the gradient descent procedure

which takes into account the geometry of the underlying manifold by incorporating a corrective term given by the Riemannian metric tensor. The equations for updating the means and covariances in the direction of the natural gradient are given by,

$$\mu_{\mathfrak{v}_{t+1}} = \mu_{\mathfrak{v}_t} + \eta G_{\mathfrak{v}}^{-1} \nabla_{\mu_{\mathfrak{v}}} ln(p) \tag{5.8}$$

where, $G_{\mathfrak{v}}$ is the Riemannian metric tensor and $\eta G_{\mathfrak{v}}^{-1} \nabla_{\mu_{\mathfrak{v}}} lnp$ is the direction of steepest change (rather than $\nabla_{\mu_{\mathfrak{v}}} lnp$ as given by "normal" gradient).

Similarly, the gradient descent in the direction of covariance matrices is given by,

$$\Sigma_{\mathfrak{v}_{t+1}} = \Sigma_{\mathfrak{v}_t} + \eta H_{\mathfrak{v}}^{-1} \nabla_{\Sigma_{\mathfrak{v}}} ln(p) \tag{5.9}$$

where, $H$ is the Riemannian metric tensor and $\eta H_{\mathfrak{v}}^{-1} \Sigma_{\mu_{\mathfrak{v}}} lnp$ is the direction of steepest change (rather than $\nabla_{\Sigma_{\mathfrak{v}}} lnp$ as given by "normal" gradient).

The Riemannian metric tensor in the space of mean vectors $G$ is given by the Fisher information matrix,

$$G_{\mathfrak{v}} = \mathbb{E}((\nabla_{\mu_{\mathfrak{v}}} \mathcal{F}_{\mathfrak{v}})(\nabla_{\mu_{\mathfrak{v}}} \mathcal{F}_{\mathfrak{v}})^T) \tag{5.10}$$

Plugging in the expression for the derivative of $\mathcal{F}_{\mathfrak{v}}$, we get the following expression for $G$.

$$G_{\mathfrak{v}} = \Sigma_{\mathfrak{v}}^{-1} \tag{5.11}$$

Hence, the online update equation for the Gaussian means is given as below,

$$\mu_{\mathfrak{v}_{t+1}} = \mu_{\mathfrak{v}_t} + \eta(x_v - \mu_{\mathfrak{v}_t}) \tag{5.12}$$

Similarly, the metric tensor in the space of covariance matrices is defined as,

$$H_{\mathfrak{v}} = \mathbb{E}((\nabla_{\Sigma_{\mathfrak{v}}} \mathcal{F}_{\mathfrak{v}})(\nabla_{\Sigma_{\mathfrak{v}}} \mathcal{F}_{\mathfrak{v}})^T) \tag{5.13}$$

which after expanding the terms simplifies to,

$$H_{\mathfrak{v}} = -\frac{\Sigma_{\mathfrak{v}}^{-2}}{4} + \frac{\Sigma_{\mathfrak{v}}^{-4}}{4}\mathbb{E}((x_{\mathfrak{v}} - \mu_{\mathfrak{v}})(x_{\mathfrak{v}} - \mu_{\mathfrak{v}})^T(x_{\mathfrak{v}} - \mu_{\mathfrak{v}})^T(x_{\mathfrak{v}} - \mu_{\mathfrak{v}})) \tag{5.14}$$

The term within the expectation operator is by definition the fourth order central moment (kurtosis), which for Gaussian is calculated to be $3\Sigma_{\mathfrak{v}}^2$. Substituting it in Eq. (5.14) gives the following formula for $H$,

$$H_{\mathfrak{v}} = \frac{\Sigma_{\mathfrak{v}}^{-2}}{2} \tag{5.15}$$

Plugging the expression for $H$ from Eq. (5.15), in the online update rule based on Natural gradient descent Eq. (5.9) gives the following online equation for updating covariance matrices,

$$\Sigma_{\mathfrak{v}_{t+1}} = (1 - \eta)\Sigma_{\mathfrak{v}_t} + \eta(x_v - \mu_{\mathfrak{v}_t})(x_v - \mu_{\mathfrak{v}_t})^T \tag{5.16}$$

## 5.3 Random graphs as prototypes

Prototype based classification schemes are widespread in the domain of attributed graphs. The key idea is to embed the graphs into a vector space in the following manner. Given a set of graphs $\{(X_i, y_i)\}$, we synthesize a set of prototype graphs $\mathcal{C} : \{(C_i, y_i)\}$ such that every graph $X_i$ is embedded in $\mathbb{R}^k$ as,

$$X_i \rightarrow (d(X_i, C_1), .., d(X_i, C_k)) \tag{5.17}$$

where $d(X_i, C_1)$ is a dissimilarity measure between the graphs and prototypes. The choice of prototypes influences the distance measure and hence the dissimilarity space. To illustrate, when the prototype graphs are chosen to be set median or means or cluster centres, it is clear as to how the distance is calculated. However, what is the distance measure when we choose random graphs as prototypes ?

The key lies in defining the Kullback-Liebler divergence between the probability density of random prototype graph $\mathfrak{W}$ and the true (hidden) probability

distribution $\mathbf{q}$ [33, 19].

$$KL(\mathbf{q} \parallel (\mathbf{p}(\mathfrak{W})) = -\int \mathbf{q} \ln \frac{\mathbf{p}(\mathfrak{W})}{\mathbf{q}} = \\ -\int \mathbf{q} \ln(\mathbf{p}(\mathfrak{W})) + \int \mathbf{q} \ln(q) \tag{5.18}$$

The unknown probability distribution $\mathbf{q}$ is represented by $\delta(g-g_i)$, where $\delta(.)$ is the Dirac delta function at every data sample $g_i$. Hence,

$$\int \delta(g-g_i) \ln(\mathbf{p}(\mathfrak{W})) = \ln(\mathbf{p}(\mathfrak{W}_{\mathfrak{g}_i})) \tag{5.19}$$

which is the log-likelihood that the random graph $\mathfrak{W}$ generates the outcome $g_i$. Hence, likelihood (or more precisely its logarithm) could be used as a feature for classification naturally in the dissimilarity/distance representation framework.

*To summarize, when the prototype of a set of attributed graphs is also an attributed graph (set median, cluster centre etc..) dissimilarity between the prototypes and graphs is a feature for classification. When the prototypes are themselves random graphs, likelihood is a feature for classification.*

The essential definitions and concepts for a probabilistic model for classifying attributed graphs have been setup. We thus decribe the scheme in the following manner- Given a dataset of graphs representing patterns belonging to different classes, sythesize first random attributed graphs acting as a model/prototype for each class. The largest graph (i.e. the graph with maximum number of nodes) is initialized as prototype classwise. We then present every graph in the training set, align them with the corresponding prototype and update the node and edge occurence (structural) probabilities. The means and covariances are also updated according to the formulae in Eq. (5.12)-(5.16). Once the parameters of the random prototype graphs are determined, we embed the dataset into a feature space by calculating the log-likelihood between every graph in the dataset and every element in the prototype set. We point out two notable features of this scheme-

1. The size of the prototypes are bound by the size of the largest graph in the dataset.

2. The number of graph matching operations during the parameter estimation stage is $N$, the size of the training set; once the prototype random graphs are sythesized, the training set (with $N$ samples) and the test set (with $M$ samples) have to be embedded in the likelihood space. This needs another $(N+M) \times K$ graph matching operations.

## 5.4 Experiments

### 5.4.1 Algorithmic details

*Matching attributed graphs with RAG-* The problem of aligning random graphs with each of the sample graph and the likelihood calculation involve attributed graph matching. We adopt again the graduated assignment algorithm described in Appendix B [23] with a suitable compatibility function for this purpose. To recall, this algorithm minimizes a cost function as a function of match matrix over all possible matchings by an iterative procedure which estimates match matrix at every step and normalizes it.

**Algorithm 4** Graduated assignment algorithm

Repeat until $\beta > \beta_f$
$Q_{ai} = \sum_b \sum_j M_{bj} C_{aibj}$
$M_{ai} = exp(\beta Q_{ai})$
Perform Sinkhorn normalization

The matching is determined by node and edge compatibilites $C_{aibj}$, which measures their respective similarities structurally and attribute value wise. In determining the morphism between random attributed graphs and outcome graphs, the node compatibility function between node $a$ of the random graph and $i$ of the sample attributed graph is set according to the following,

$$C_{\mathfrak{a}i} = p_{\mathfrak{a}} e^{-(x_i - \mu_{\mathfrak{a}})^T \Sigma^{-1} (x_i - \mu_{\mathfrak{a}})} \tag{5.20}$$

where $p_i$ is the probability of node $i$ occuring, $x_i$ is its attribute value, $\mu_a$ the mean of the distribution of node $a$. Similar choice is made for edge compatibility

functions as well. The choice of this compatibility function is motivated as follows: The definition of probability of an outcome graph depends on morphism Eq. (5.1). We would like to determine the morphism such that the probability of an outome of a random attributed graph is maximum, which corresponds to minima of cost function $-\sum_{\mathfrak{a}}\sum_{i}\sum_{\mathfrak{b}}\sum_{j} M_{\mathfrak{a}i}M_{\mathfrak{b}j}C_{\mathfrak{a}i\mathfrak{b}j}$, where $M_{\mathfrak{a}i}$ denotes matching $\mathfrak{a}$ of random graph with node $i$ of the outcome graph. Hence, setting compatibility values as defined in Eq. (5.20) gives the "optimal" morphism.

*Classification procedure*- Once the random graphs have been synthesized classwise, the dataset was embedded into a feature space by calculating the log-likelihood of graphs beng generated by the prototype random graphs. In the feature space, various classifiers were learned on the training set and validated (by performance on the validation set or by cross-validation on the training set). The classifier exhibiting best validation performance was used to classify the test data. All classification experiments were done using PyML software [3].

### 5.4.2 Synthetic datasets

We first analyzed the performance of this algorithm on synthetic datasets. We consider a dataset consisting of 200 graphs in training and test set belonging to two classes. The dataset is generated by considering distortions of two base graphs classwise at different levels viz. $5\%, 10\%, 15\%, 20\%$. Node and edge attributes are generated according to a normal distribution. The noise according to the specified distortion level is added which modifies node and edge occurences and also their respective attributes. The nodes are then randomly permuted. The dataset is then divided uniformly into training and test sets. The classification scheme described in this chapter is referred to as $RAG+LF$ (Random Attributed Graph model + Likelihood as a Feature). The standard $k-$ Nearest Neighbour algorithm ($kNN$)in the graph domain is chosen as the benchmark classifier [19].

The classifiers are evaluated on the basis of the Area under the ROC curve ($AUC$) [21], Blue for $RAG + LF$ and Red for $kNN$. Higher the value of $AUC$, the more "ideal" the classifier. As is seen, for low values of distortion, $RAG+LF$ family of classifiers give near ideal performance. The following table compares the results against $kNN$ classifier for the best-performing value of $k = 3$.

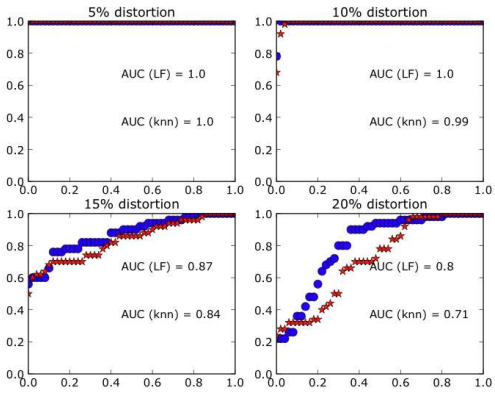



Figure 5.1: Classifier ROC plots for different distortion levels

Table 5.1: Classification results % on the synthetic datasets

| Distortion | 5 | 10 | 15 | 20 |
|---|---|---|---|---|
| RAG + LF | 97 | 97 | 81 | 74 |
| kNN | 95 | 84 | 72 | 56 |

### 5.4.3 Experimental results on IAM Graph datasets

A set of experiments were conducted on datasets from the IAM graph dataset repository. We choose two datasets for evaluation- Letter (HIGH) and Fingerprint, on account of the large training and test set consisting of a homogeneous selection of graphs of different sizes. A third simplified dataset referred to as Fingerprint(AW) is generated by considering only two of the four classes Arch (A) and Whorl (W) of the fingerprint dataset. The purpose here is to examine the performance of the approach on a two-class problem with significant graph size variations (as graphs belonging to Whorl class is bigger than graphs in Arch class). There are 770 graphs in this dataset.

The state-of-the-art techniques chosen are kNN (chosen as Reference system) [57], embedding based on Similarity Kernels (SK + SVM), embedding based on Lipschitz Embedding (LE+SVM) [58], and Structurally-described random graphs (SDRG) [64]. The approach proposed here is referred to as Random Attributed Graph model + Likelihood as Feature (**RAG + LF**). SK+SVM and LE+SVM

refer to a family of related classifiers whose best results are chosen. The family of prototype learning methods in the domain of graphs are not considered.

Table 5.2: Classification results % on IAM datasets

| Method | Letter(HIGH) | Fingerprint(AW) | Fingerprint |
|---|---|---|---|
| kNN | 82 | 91.8 | 76.6 |
| SK + SVM | 79.1 | - | 41 |
| LE + SVM | 92.5 | - | 82.8 |
| SDRG | 64.3 | - | - |
| **RAG + LF** | 75.7 | 95.9 | 78.2 |

The following observations are made- the results compare well for the Fingerprint dataset overall, and for the Letter (HIGH) dataset compares well with SK + SVM and is superior to SDRG; although kNN yields good results overall, it faces the computationally challenging task of choosing k. For SK + SVM and LE + SVM, the task of choosing effective prototype set and calculating the graph-edit distance between the dataset and prototype set is expensive as well and offers no analytical insight. The approach presented here is fast as it involves estimating the parameters of random graph model analytically and needs far less graph matching operations corresponding to generating only one class prototype model. The prototypes also give a good summary of node and edge occurence probabilities in the dataset and probability distributions of their attributes. Learning the classifier happens in the feature space where there exist fast packages for SVM's and other algorithms. The performance for dataset with structurally distinct graphs (Fingerprint(AW)) is also seen to be good, as would be expected. More prototype random graphs could be setup for every class at the cost of speed, but with good choice, better classification results could be obtained.

## 5.5 Conclusions & future directions

This work builds upon the notion of random graph models with applications in structural pattern recognition with the following contributions- with independence assumptions a random attributed graph is represented as a joint random variable in its node and edge occurences and of their respective attribute values, an analytical method to estimate the different probability distributions of a random graph model as a prototype given an ensemble of attributed graphs is presented using a maximum likelihood procedure, the utility of the random graph as a prototype is shown by using the likelihood of an outcome graph as a feature for classification. The proposed approach is validated on few datasets and its effectiveness is shown.

The direction of further work aims to build on the following themes- first, a method to derive a class of probabilistic clustering and classification algorithms needs to be investigated. This means that the random prototype graph is learned from the dataset in a procedure akin to a standard quantization type scheme. Second, is there a way to tie the classifiers in the feature space directly with the learning of prototypes? To elaborate, it is important to investigate the link between type/family of classifiers on the feature space (due to likelihood) with how the random prototypes are estimated/learned. This would help to integrate probabilistic learning in the domain of graphs with discriminative methods for classification in the subsequent likelihood space. Lastly, the foundations of the random graph definitions need to be explored- although node and edge independence is useful in that it allows an easy analytical estimation of model parameters, it is too strong an assumption. Is there a way to model dependencies of nodes and edges and their attributes (node/edge co-occurences)? Such an understanding would help enormously in probabilistic sub-structure analysis methods and also give possibly superior classification and clustering algorithms.

# Chapter 6

# Summary & Outlook

To place the work contained here in proper perspective, this thesis aims to answer the question: How can prototypes of a set of graph patterns be defined in a principled manner? We have proposed algorithms to learn prototypes of a set of attributed graphs, broadly falling into supervised and unsupervised learning techniques.

Learning Graph Quantization (LGQ) is a class of supervised prototype learning algorithms that is analogous to Learning Vector Quantization. During the learning phase, the algorithm updates nearest prototypes after observing a new training sample. The theoretical background is provided by embedding the set of graphs into the quotient space of an Euclidean space. With this isostructural and isometric embedding, we define a metric between graphs and derivatives of functions defined on them. In experiments, we see that LGQ algorithms with a principled initialization procedure shows good classification performance, improving upon the performance of the standard k- Nearest Neighbour (kNN) algorithm with a graph-edit distance. Another notable advantage is also the compact prototype set, which means that during the testing phase the number of computationally intensive graph matching operations is reduced sharply.

We then use the prototypes obtained by LGQ algorithms to embed the graphs into a feature space. This is accomplished by considering the distance of a graph to a prototype as a feature in a vector space. In this approach, the prototypes define a kernel with an explicit feature space representation. In this feature space, classifiers are trained and used to classify the test patterns. Thus, we may

adopt the view that there is learning in the original structural domain followed by learning in the (feature) vector domain. In experiments, we observe that it compares favorably to state-of-the-art classification schemes, which uses a large prototype set and then picks out the most informative features for classification. We also note a relationship between the classification performance and the VC-dimension in the feature space induced by the prototypes.

The last part deals with unsupervised learning of prototypes by treating it as a density estimation problem. We adopt the random attributed graph model- a probabilty density function over node and edge occurences and attributes. Using independence assumptions, the parameters are then estimated online- by learning algorithm involving the natural gradient. Once the random graph parameters are estimated, the set of graphs are embedded into the space spanned by their likelihood values and classifiers are learnt. The advantages here are observed to be- robustness against noise for kNN classifiers on synthetic datasets, the compact prototype set, and the lesser number of graph matching operations.

Further research relating to this thesis are in two directions.

-The first is with respect to the problem of learning prototypes which are most suitable for classification in the dissimilarity space. This is achieved by directly minimizing the ratio of radius of minimum enclosing ball to the margin of the classifier in the feature space. Both of these quantities depend on the kernel matrix, which in turn is a function of the prototypes. In this scheme, we learn the prototypes with the most optimal generalization performance.

- The random attributed graph model we use relies on independence assumptions to estimate the parameters. This is simplistic as we lose the node, edge attribute probability relationships involving more nodes and edges. A possible solution is to model node and edge attributes and occurences with a graphical model and use inference algorithms to estimate them.

The major constraint faced when dealing with structural data is the computational complexity of graph matching. In spite of using an efficient approximation algorithm to solve the problem, we have observed experiments involving such data take much longer compared to machine learning experiments on feature vectors. In order to overcome this disadvantage, it is desirable to develop embedding techniques for graphs into vector spaces so that the computational complexity is

reduced, while also trying to avoid loss of structural information. Moreover, it is always useful to try different heuristics and algorithms to simplify the graph matching procedure while not losing out on performance.

# Appendix A

We present here a short description of the datasets used in experiments in this thesis.

## LETTER (HIGH)

The Letter (HIGH) dataset consists of distorted letter drawings from the Roman alphabet that consist only of straight lines (A, E, F, H, I, K, L, M, N, T, V, W, X, Y, Z). The graphs are uniformly distributed over 15 classes. A prototype letter drawing is manually constructed for each class, which is then converted to prototype graphs by representing lines by undirected edges and endpoints of lines by nodes. The nodes are annotated by a two-dimensional vector indicating its position relative to a reference coordinate system. The graph set is then distorted by applying high strength distortion to the prototype graphs. Ths size of training, test and validation set are 750 each.

## GREC

The GREC dataset consists of drawings of symbols from architectural and electronic drawings. The images occur at different distortion strengths. Depending upon the distortion level, erosion, dilation or other morphological operations were applied. The result is thinned to obtain lines of one pixel width. Finally, graphs are extracted from the resulting denoised images by tracing the lines from end to end and detecting intersections and corners. Ending points, corners, intersections

and circles are represented by nodes and annotated by two-dimensional attribute denoting its position. The nodes are connected by edges labeled as line or arc. An additional attribute specifies the angle with respect to the horizontal direction or the diameter in case of arcs. The size of training, test and validation are 286, 528 and 286 respectively.

## Fingerprint

This dataset represents fingerprint images of the NIST-4 database from four classes *arch, left, right* and *whorl*. Images of the fingerprints are converted into graphs by filtering the images and extracting regions that are relevant, which are then binarized and further processed by denoising and thinning. The ending and the bifurcation points of the resulting skeletonized regions are represented by vertices. Additional vertices are inserted in regular intervals between ending points and bifurcation points. Undirected edges are inserted to link vertices that are directly connected through a ridge in the skeleton. Each vertex is labeled with a two-dimensional attribute vector specifying its position. The size of training, test and validation set are 500, 2000 and 300 respectively.

## COIL-DEL

Columbia Object Image Library (COIL-100) is a database of colour images of 100 objects. The objects are placed at the centre of a turntable. The turntable was rotated through 360 degrees and an image was taken every 5 degrees of rotation- 72 images per object, 7200 images in total. The images were then size and intensity normalized. In order to extract graph representation of objects, the corners are extracted using Harris corner detection algorithm. The images are then segmented using Delauney triangulation with the corner points. The result of the triangulation is then converted into a graph by representing lines by undirected edges and terminal points of lines as nodes. The nodes are attributed with a two-dimensional position vector and edges are unattributed. A subset of the database for every 15 degrees of rotation is used as a training set (2400

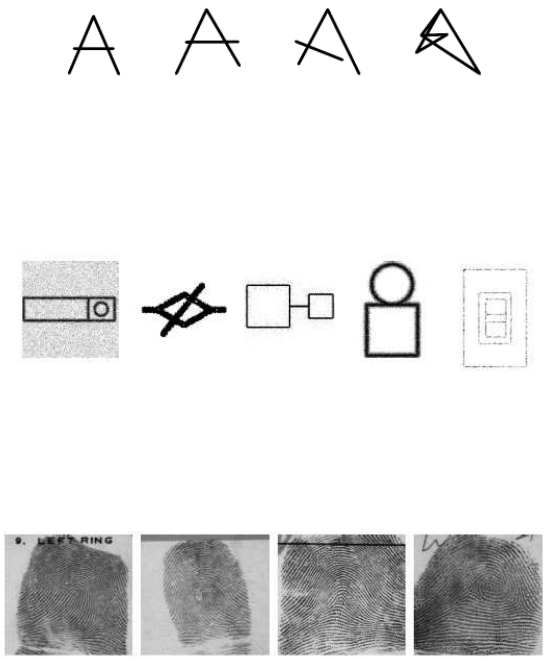



Figure 1: IAM graph dataset repository

graphs). The test and validation set number 1000 and 500 graphs in all. These graphs are uniformly distributed over all classes. Out of the database, a subset of 32 objects that are hard to recognize has been identified, bounded by red boxes [55]. We choose four of the 32 objects (2 spherical fruit like shapes and 2 toy car like shapes, indexed by class number 1, 3, 18, 41 starting from 0 at the topmost left) for the purpose of this thesis empirically based on interclass similarities and graph sizes. The size of training, test and validation set are 96, 40 and 20 respectively. More details on experimental settings and dataset are to be found in [24].

## SHAPES

Shapes are skeletonized by using the medial axis, which is defined as the centres of circles with maximum radius. The points where there is change in shape structure- end points, junction points and points with large change in slope of the radius function along the medial axis are considered as the nodes for the graph. The edges of the graph correspond to skeleton's branches between two nodes. Edge attribute are an approximation of the perimeter of the boundary which contributes to the formation of the edge, normalized by the approximated perimeter of the whole shape. Node attributes give the distance between the node position and the gravity center of the shape divided by the square of the shape area.

We considered the dataset provided by Dupé et al. [20]. We considered five classes of objects, cars, children, animals and tools. There are 24 graphs in every class except children which has only 12 graphs. The grouping was based on their morphological similarity and context of these shapes occuring in applications. As the dataset is small, we use the resampled paired t-test [18] with 20 trials [24].

## Protein contact maps

Contact maps are used to represent the tertiary structure of Protein molecules. They consist of a set of residues which are vertices of a graph, with two vertices

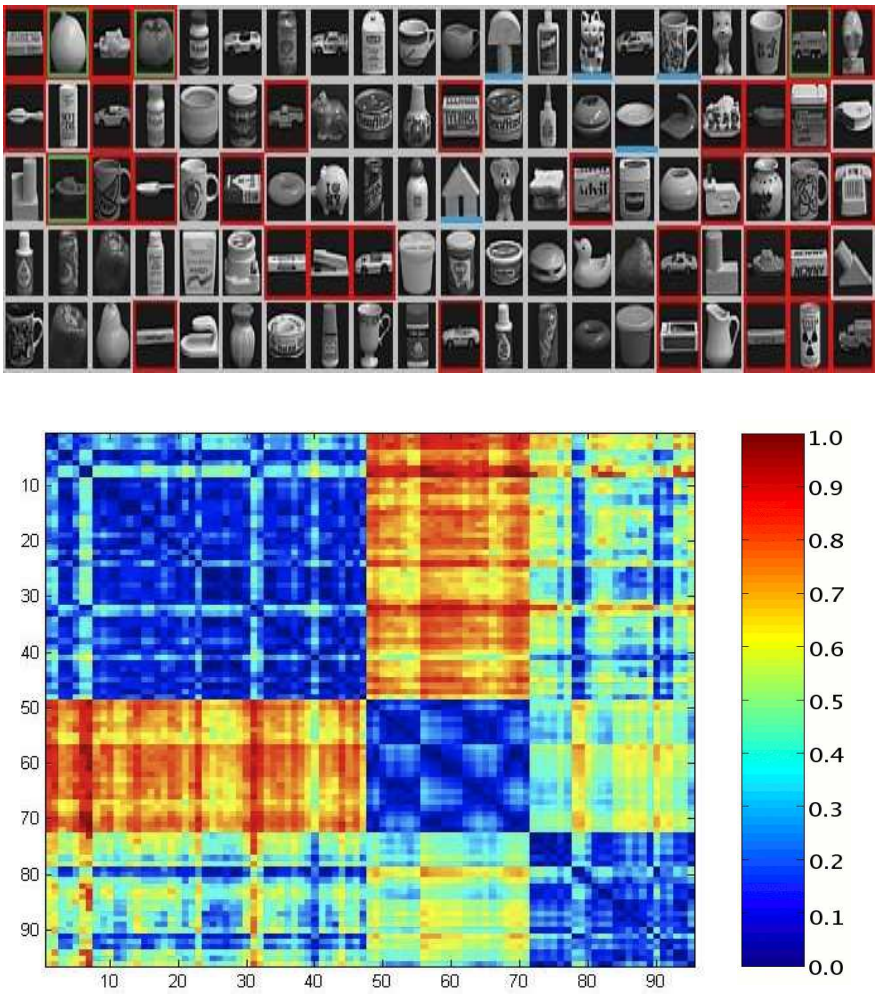



Figure 2: COIL-100 dataset along with distance matrix of objects 1, 3, 18, 41

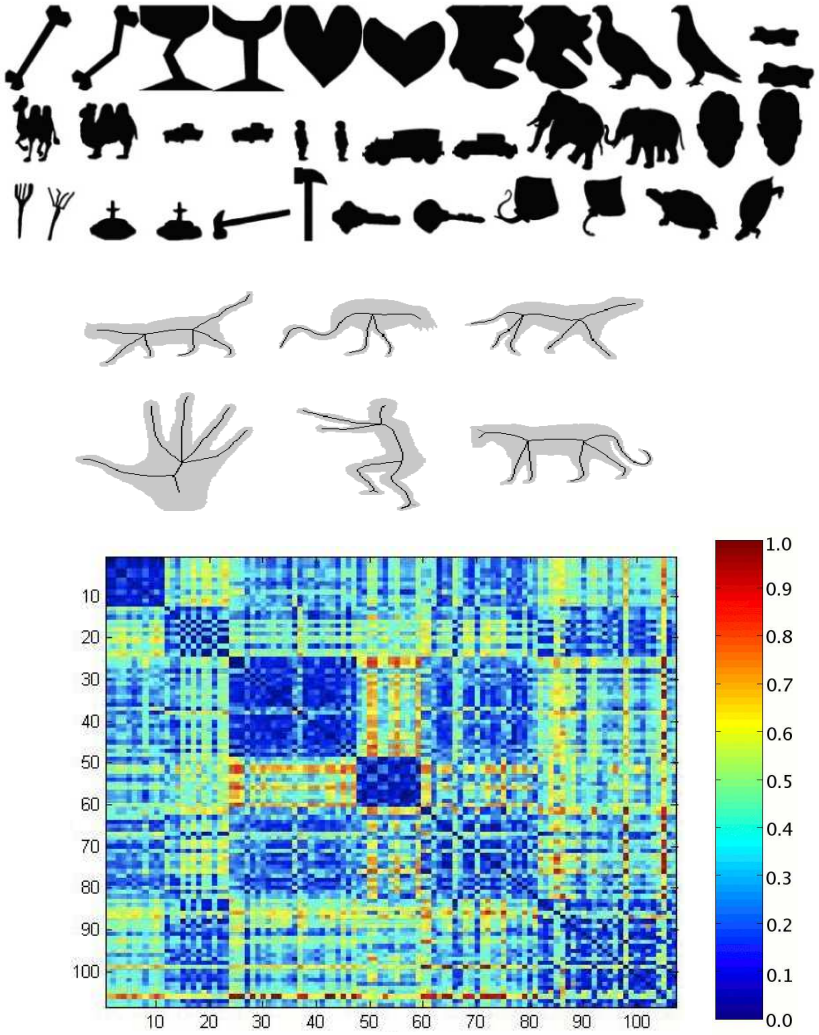



Figure 3: Shape dataset
Typical shapes (Top), Skeleton of some sample shapes (Second row), Distance matrix between shapes (Bottom)

being connected by an edge (called as contact) if they are spatially close (lesser than some threshold). Skolnick data sets consist of 40 contact maps belonging to five different superfamilies viz. Che Y-like, Microbial ribonucl., Cuperdoxins, Triosephosphateisomerase (TTM), Ferritin. The mean number of residues ranges from 100 for Microbial ribonucl. family to 250 for TTM.

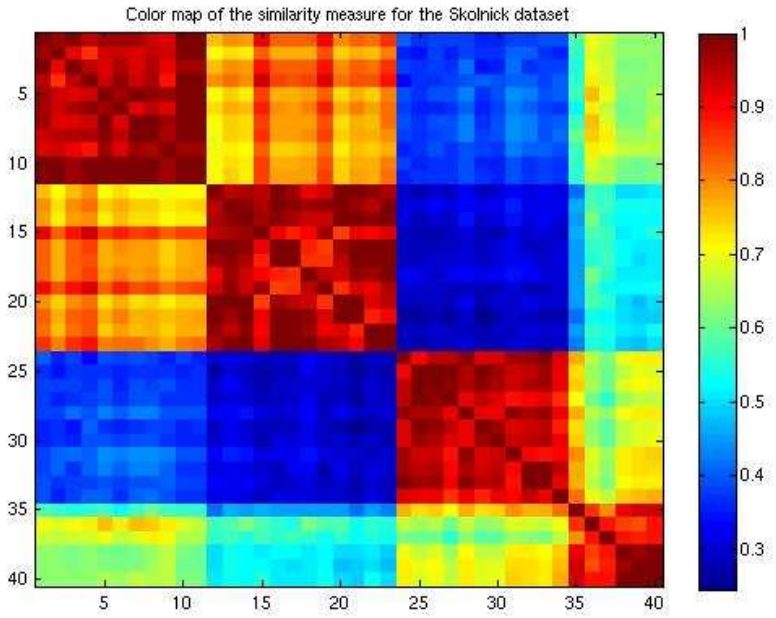



Figure 4: Distance matrix for contact maps for Skolnick dataset

# Appendix B

We discuss the graduated assignment algorithm for graph matching, which is used throughout this thesis [23]. Determining graph distance and subgradient involve a graph matching problem in order to find the optimal alignment. However, this problem is NP-complete, so we have to resort to "good" approximate suboptimal solutions. We use the graduated assignment algorithm on account of its speed, quality of solutions and ability to handle a wide variety of attributes.

The basic idea behind the algorithm is to reframe the problem of graph matching in terms of nonlinear optimization by converting the discrete variable into continuous ones. This avoids the problem of getting trapped into a local minima. At every step, the match matrix is estimated and normalized so as to enforce the constraint that every row and column of match matrix sums to one, while a control parameter is varied.

Th match matrix $M$ should be chosen such that, the following cost function is minimized for two graphs $X$ and $Y$ with entries $X_{ab}, Y_{ij}$ respectively

$$E(M) = -\frac{1}{2}\sum_{a=1}^{N_x}\sum_{i=1}^{N_y}\sum_{b=1}^{N_x}\sum_{j=1}^{N_y} M_{ai}M_{bj}C(X_{ab}, Y_{ij})$$

with the constraints $\forall a, \sum_{a=1}^{N_x} M_{ai} \leq 1; \forall i, \sum_{i=1}^{N_y} M_{ai} \leq 1; M_{ai} \in \{0,1\}$. $X_{ab}, Y_{ij}$ could be $\{0,1\}$ or could consist of features represented as vectors, strings. The choice of compatibility matrix depends on the nature of graphs- its connectivity and attributes. A common choice is,

$$C(X_{ab}, Y_{ij}) = \begin{cases} 0, \text{ if } X_{ab} \text{ or } Y_{ij} = \phi \\ f(X_{ab}, Y_{ij}) \text{ otherwise} \end{cases}$$

where $f(X_{ab}, Y_{ij})$ is a functional mapping. For example, $f$ is commonly chosen to be normalized inner product if the attributes themselves are feature vectors. If the attributes are scalars then either their product or difference normalized to zero is a suitable compatibility measure. It is seen that the matching cost can deal with graphs with a wide variety of attributes. Heuristically, the compatibility matrix is chosen such that its expected value is zero in order to improve the quality of matching.

The algorithm then proceeds to convert the matching problem into an assignment problem in the following manner. The objective function is expanded as a Taylor series around the initial $M_0$ as,

$$-\frac{1}{2}\sum_{a=1}^{N_x}\sum_{i=1}^{N_y}\sum_{b=1}^{N_x}\sum_{j=1}^{N_y} M_{ai}M_{bj}C(X_{ab}, Y_{ij}) = -\frac{1}{2}\sum_{a=1}^{N_x}\sum_{i=1}^{N_y}\sum_{b=1}^{N_x}\sum_{j=1}^{N_y} M^0_{ai}M^0_{bj}C(X_{ab}, Y_{ij})$$

- $\sum_{a=1}^{N_x}\sum_{i=1}^{N_y} Q_{ai}(M_{ai} - M^0_{ai})$

where,

$$Q_{ai} = -\frac{\partial E}{\partial M_{ai}} = \sum_{b=1}^{N_x}\sum_{j=1}^{N_y} M^0_{bj}C(X_{ab}, Y_{ij})$$

The assignment term $\sum_{a=1}^{N_x}\sum_{i=1}^{N_y} Q_{ai}M_{ai}$ needs to be maximized. Also, the inequality constraints in the original cost function are converted to equality constraints by introducing slack variables. The continuous values of the match matrix are then converted into discrete variables by using softmax and by a Sinkhorn procedure which normalizes rows and columns alternatively until the matrix converges to a doubly stochastic matrix. The algorithm is summarized below:

We adopt the values as proposed in the literature [23]. $\beta_0 = 0.5, \beta_f = 10$ and $M$ is initialized to approximately the diagonal matrix of the corresponding

**Algorithm 5** Graduated assignment algorithm

**Initialize** $\beta, M$
  While $\beta \leq \beta_f$
    Repeat till $M$ converges
      $Q_{ai} = -\frac{\partial E}{\partial M_{ai}}$
      $M_{ai} = exp(\beta Q_{ai})$
        Repeat Sinkhorn loop until $M$ converges to doubly stochastic matrix
          Normalize across rows
          Normalize across columns
**Return:** $M$

order. The Sinkhorn loop is repeated until 30 times if convergence does not occur earlier.

The graduated assignment algorithm returns a doubly stochastic matrix with dominant entries row and column wise. However, the values are still not 1/0. Hence, in order to convert it to an actual permutation matrix, we could

- heuristically use a cleanup procedure by setting the largest value in row and column to be 1 and set the other entries to 0.

- use algorithms which solve the assignment problem, as the match matrix entries could be viewed as scores of assigning nodes of one graph with another. A computationally efficient method to solve it is the Munkres algorithm [1], a polynomial time algorithm widely used in combinatorial optimization. We use Munkres algorithm as the post-processing step as we have empirically observed it to yield good matching solutions than the simple cleanup procedure mentioned earlier.

- use post-processing cleanup step taking into account the nature of the datasets and associated constraints. This holds for the Protein contact maps (Skolnick) dataset, where the matching must preserve the order of nodes i.e. the node assignments shall not cross [35]. We therefore use dynamic programming to recover the most optimal matching.

# . APPENDIX B

# References


[1] M. Aigner. *Discrete Mathematics*. American Mathematical Society, Boston, MA, USA, 2007.

[2] S-I. Amari. Natural gradient works efficiently in learning. *Neural Comput.*, **10**:251–276, February 1998.

[3] A. Ben-Hur. PyML- A Python Machine Learning package. Technical report, 2008.

[4] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[5] K. M. Borgwardt, C. S. Ong, S. Schoenauer, S. V. N. Vishwanathan, A. J. Smola, and H. P. Kriegel. Protein function prediction via Graph Kernels. In *Intelligent Systems in Molecular Biology*, **21**, pages 47–56, 2005.

[6] K.M. Borgwardt. *Graph Kernels*. PhD thesis, Ludwig Maximilians Universität, Fakultät für Mathematik, Informatik und Statistik, 2007.

[7] L. Bottou. Stochastic gradient learning in neural networks. In *Proceedings of Neuro-Nîmes 91*, Nimes, France, 1991. EC2.

[8] H. Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recogn. Lett.*, **18**:689–694, August 1997.

[9] H. Bunke and G. Allerman. Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters*, **1**:245–253, 1983.

[10] H. Bunke and K. Riesen. Improving vector space embedding of graphs through feature selection algorithms. *Pattern Recognition*, **44**:1928–1940, 2011.

[11] O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee. Choosing multiple parameters for Support Vector Machines. *Mach. Learn.*, **46**:131–159, March 2002.

[12] F. R. K. Chung. *Spectral Graph Theory*. American Mathematical Society, 1997.

[13] F. H. Clarke. *Optimization and Nonsmooth Analysis*. Wiley Publishers, 1983.

[14] D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, **18**:265–298, 2004.

[15] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. Learning structural shape descriptions from examples. *Pattern Recognition Letters*, **23**:1427–1437, 2002.

[16] K. Crammer, R. Gilad-bachrach, A. Navot, and N. Tishby. Margin analysis of the LVQ algorithm. In *Neural Information Processing Systems*, pages 462–469, 2002.

[17] D. de Ridder, O. Kouropteva, O. Okun, M. Pietikinen, and R. P. W. Duin. Supervised Locally Linear Embedding, 2003.

[18] T. G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Comput.*, **10**:1895–1923, October 1998.

[19] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2000.

[20] F.-X. Dupé and L. Brun. Tree covering within a graph kernel framework for shape classification. In *Proceedings of the 15th International Conference on Image Analysis and Processing*, ICIAP '09, pages 278–287, Berlin, Heidelberg, 2009. Springer-Verlag.

[21] T. Fawcett. ROC graphs: Notes and practical considerations for data mining researchers, 2003.

[22] M.R. Garey and D.S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[23] S. Gold and A. Rangarajan. A Graduated Assignment algorithm for graph matching. *IEEE Trans. Pattern Anal. Mach. Intell.*, **18**:377–388, April 1996.

[24] M. Göttner. *Evaluating LGQ algorithms for shape and object classification problems*. Bachelor thesis, Technische Universität, Berlin, 2011.

[25] T. Graepel, R. Herbrich, P. Bollmann-sdorra, and K. Obermayer. Classification on pairwise proximity data. In *Neural Information Processing Systems*, pages 438–444, 1999.

[26] S. Günter and H. Bunke. Self-organizing map for clustering in the graph domain. *Pattern Recognition Letters*, **23**:405–417, 2002.

[27] M. Hagenbuchner, A. Sperduti, and A.C. Tsoi. A self-organizing map for adaptive processing of structured data. *IEEE Transactions on Neural Networks*, **14**:491–505, 2003.

[28] B. Hammer, A. Micheli, and A. Sperduti. Adaptive contextual processing of structured data by recursive neural networks: A survey of computational properties. *Springer Studies in Computational Intelligence, Volume 77, Springer Verlag*, 2007.

[29] B. Hammer, M. Strickert, and T. Villmann. On the generalization ability of GRLVQ networks. *Neural Process. Lett.*, **21**:109–120, April 2005.

[30] B. Hammer, M. Strickert, and T. Villmann. Supervised neural gas with general similarity measure. *Neural Process. Lett.*, **21**:21–44, February 2005.

[31] B. Hammer and T. Villmann. Generalized Relevance Learning Vector Quantization. *Neural Netw.*, **15**:1059–1068, October 2002.

[32] D. Haussler. Convolution Kernels on discrete structures. Technical report, 1999.

[33] J. Hollmen, V. Tresp, and O. Simula. A self-organizing map for clustering probabilistic models. In *Proceedings of the Ninth International Conference on Artificial Neural Networks (ICANN99*, pages 946–951. IEEE, 1999.

[34] A. Honkela, M. Tornio, T. Raiko, and J. Karhunen. Natural conjugate gradient in variational inference. In *Neural Information Processing*, pages 305–314, Berlin, Heidelberg, 2008. Springer-Verlag.

[35] B. J. Jain and M. Lappe. Joining softassign and dynamic programming for the contact map overlap problem. In *Proceedings of the 1st international conference on Bioinformatics research and development*, BIRD'07, pages 410–423, Berlin, Heidelberg, 2007. Springer-Verlag.

[36] B. J. Jain and K. Obermayer. On the sample mean of graphs. In *IJCNN*, pages 993–1000, 2008.

[37] B. J. Jain and K. Obermayer. Structure Spaces. *J. Mach. Learn. Res.*, **10**:2667–2714, December 2009.

[38] B. J. Jain and K. Obermayer. Generalized Learning Graph Quantization. In *GbRPR*, pages 122–131, 2011.

[39] B. J. Jain and F. Wysotzki. Central clustering of attributed graphs. *Mach. Learn.*, **56**:169–207, June 2004.

[40] B.J. Jain and K. Obermayer. Graph Quantization. *Computer Vision and Image Understanding*, **115**[7]:946–961, 2011.

[41] B.J. Jain, S.D. Srinivasan, A. Tissen, and K. Obermayer. Learning Graph Quantization. In *SSPR/SPR*, **6218** of *Lecture Notes in Computer Science*. Springer, 2010.

[42] X. Jiang, A. Mnger, and H. Bunke. On median graphs: Properties, algorithms, and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **23**:1144–1151, 2001.

[43] H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized Kernels between labeled graphs. In *ICML*, pages 321–328, 2003.

[44] T. Kohonen, editor. *Self-Organizing Maps*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 3rd edition, 2001.

[45] T. Kohonen, J. Hynninen, J. Kangas, J. Laaksonen, and K. Torkkola. LVQ PAK: The Learning Vector Quantization Program Package. 1996.

[46] T. Kohonen and P. Somervuo. Self-organizing maps of symbol strings. *Neurocomputing*, **21**:19–30, 1998.

[47] A. LaVigna. *Nonparametric Classification using Learning Vector Quantization*. PhD thesis, Department of Electrical Engineering, University of Maryland, 1989.

[48] A. R. Leach and V. J. Gillet. *An Introduction to Chemoinformatics*. Dordrecht Kluwer Academic Publishers, 2003.

[49] M. Lozano, J. M. Sotoca, J. S. Sanchez, F. Pla, E. Pkalska, and R. P. W. Duin. Experimental study on prototype optimisation algorithms for prototype-based classification in vector spaces. *Pattern Recogn.*, **39**:1827–1838, October 2006.

[50] B. Luo, R. C. Wilson, and E. R. Hancock. Spectral embedding of graphs. *Pattern Recognition*, **36**:2213–2230, 2003.

[51] P. Mahe, N. Ueda, T. Akutsu, J.L. Perret, and J.P. Vert. Graph Kernels for molecular Structure-Activity relationship analysis with support

vector machines. *Journal of Chemical Information and Computer Sciences*, **45**:939–951, 2005.

[52] R. Marfil, F. Escolano, and A. Bandera. Graph-based representations in pattern recognition and computational intelligence. In *IWANN (1)*, pages 399–406, 2009.

[53] S. Marini, M. Spagnuolo, and B. Falcidieno. Structural shape prototypes for the automatic classification of 3d objects. *IEEE Comput. Graph. Appl.*, **27**:28–37, July 2007.

[54] E. Pekalska, R. P. W. Duin, and P. Paclk. Prototype selection for dissimilarity-based classifiers. *Pattern Recognition*, **39**:189–208, 2006.

[55] M. Pontil and A. Verri. Support Vector Machines for 3d object recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, **20**:637–646, June 1998.

[56] R. Raich, J. A. Costa, and A. O. Hero. On dimensionality reduction for classification and its application. In *in IEEE Int. Conf. Acoust., Speech. Signal Processing*, pages 917–920, 2006.

[57] K. Riesen and H. Bunke. IAM graph database repository for graph based pattern recognition and machine learning. In *Proceedings of the 2008 Joint IAPR International Workshop on Structural, Syntactic, and Statistical Pattern Recognition*, SSPR & SPR '08, pages 287–297, Berlin, Heidelberg, 2008. Springer-Verlag.

[58] K. Riesen and H. Bunke. Graph classification by means of Lipschitz embedding. *IEEE Transactions on Systems, Man, and Cybernetics*, **39**:1472–1483, 2009.

[59] K. Riesen, M. Neuhaus, and H. Bunke. Graph embedding in vector spaces by means of prototype selection. In *Proceedings of the 6th IAPR-TC-15 international conference on Graph-based representations in pattern recognition*, GbRPR'07, Alicante, pages 383–393, Berlin, Heidelberg, 2007. Springer-Verlag.

[60] A. Sato and K. Yamada. Generalized Learning Vector Quantization. In *Neural Information Processing Systems*, pages 423–429, 1995.

[61] K. K. Sengupta and K. L. Boyer. Organizing large structural modelbases. *IEEE Trans. Pattern Anal. Mach. Intell.*, **17**:321–332, April 1995.

[62] D. S. Seong, H. S. Kim, and K. H. Park. Incremental clustering of attributed graphs. *IEEE Transactions on Systems, Man and Cybernetics*, **23**:1399 –1411, Sep/Oct 1993.

[63] K. Siddiqi, A. Shokoufandeh, S.J. Dickinson, and S.W. Zucker. Shock graphs and shape matching. In *Proceedings of the Sixth International Conference on Computer Vision*, ICCV '98, pages 222–229, Washington, DC, USA, 1998. IEEE Computer Society.

[64] A. Solé-Ribalta and F. Serratosa. A structural and semantic probabilistic model for matching and representing a set of graphs. In *Proceedings of the 7th IAPR-TC-15 International Workshop on Graph-Based Representations in Pattern Recognition*, GbRPR '09, pages 164–173, Berlin, Heidelberg, 2009. Springer-Verlag.

[65] P. Somervuo and T. Kohonen. Self-organizing maps and Learning Vector Quantization for feature sequences. *Neural Process. Lett.*, **10**:151–159, October 1999.

[66] V. Vapnik. *Statistical Learning Theory*. John Wiley and Sons, 1998.

[67] M. Vassura, L. Margara, P.D. Lena, F. Medri, P. Fariselli, and R. Casadio. Reconstruction of 3d structures from Protein contact maps. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, **5**:357–367, 2008.

[68] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt. Graph Kernels. *J. Mach. Learn. Res.*, **11**:1201–1242, August 2010.

[69] U. von Luxburg and O. Bousquet. Distance-based classification with Lipschitz functions. In *Computational Learning Theory*, pages 314–328, 2003.

[70] R. C. Wilson, E. R. Hancock, and B. Luo. Pattern vectors from algebraic graph theory. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **27**:1112–1124, 2005.

[71] A. K. C. Wong and D. E. Ghahraman. Random graphs: Structural-contextual dichotomy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **2**:341–348, 1980.

[72] A. K. C. Wong and M. You. Entropy and distance of random graphs with application to structural pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **7**:599–609, 1985.