Universitat Autònoma de Barcelona

Vector Space Embedding of Graphs via Statistics of Labelling Information

A dissertation submitted by **Jaume Gibert Domingo** at Universitat Autònoma de Barcelona to fulfil the degree of **Doctor en Informàtica**.

Bellaterra, June 2012

This document was typeset by the author using LATEX 2 .

The research described in this book was carried out at the Computer Vision Center, Universitat Autònoma de Barcelona.

ISBN:

Printed by

Al Jordi i a la Roser,
que ho han fet i ho segueixen fent possible

al Pere i al Robert,
que junts ens hem fet grans

Agraıments

Des d aquestes línies vull agraır profundament l ajut inestimable de dues persones clau en la gènesis, el desenvolupament i la resolució d aquesta tesi doctoral. L Ernest, per un costat, que des del primer dia va confiar en mi i que durant aquests quatre anys que hem treballat junts sempre m ha deixat fer la meva, tant com he volgut. En molts moments ell ha sigut sense cap mena de dubte el principal responsable de que les coses tiressin endavant. Per altra banda, el Horst. Els seus savis consells i la seva extraordinària experiència han sigut tan clarificadors com determinants del que es pot trobar en el text que teniu entre mans. Moltes gràcies a tots dos.

I per descomptat vull agraır-vos a tots els que m heu acompanyat d una manera o altra en aquest camí. Aquells que llegiu aquestes línies però segurament no teniu cap interès en el que pugui dir més enllà d aquesta pàgina.

Barcelona, juny de 2012.

Resum

El reconeixement de patrons és la tasca que pretén distingir objectes entre diferents classes. Quan aquesta tasca es vol solucionar de forma automàtica un pas crucial és el com representar formalment els patrons a l ordinador. En funció d aquests formalismes, podem distingir entre el reconeixement estadístic i l estructural. El primer descriu objectes com un conjunt de mesures col·locats en forma del que s anomena un vector de característiques. El segon assumeix que hi ha relacions entre parts dels objectes que han de quedar explícitament representades i per tant fa servir estructures relacionals com els grafs per codificar la seva informació inherent. Els espais vectorials són una estructura matemàtica molt flexible que ha permès definir diverses maneres eficients d analitzar patrons sota la forma de vectors de característiques. De totes maneres, la representació vectorial no és capaç d expressar explícitament relacions binàries entre parts dels objectes i està restrigida a mesurar sempre, independentment de la complexitat dels patrons, el mateix nombre de característiques per cadascun d ells. Les representacions en forma de graf presenten la situació contrària. Poden adaptar-se fàcilment a la complexitat inherent dels patrons però introdueixen un problema d alta complexitat computational, dificultant el disseny d eines eficients per al procés i l anàlisis de patrons.

Resoldre aquesta paradoxa és el principal objectiu d aquesta tesi. La situació ideal per resoldre problemes de reconeixement de patrons seria el representar-los fent servir estructures relacionals com els grafs, i a l hora, poder fer ús del ric repositori d eines pel processament de dades del reconeixement estadístic. Una solució elegant a aquest problema és la de transformar el domini dels grafs en el domini dels vectors, on podem aplicar qualsevol algorisme de processament de dades. En altres paraules, assignant a cada graf un punt en un espai vectorial, automàticament tenim accés al conjunt d algorismes del món estadístic per aplicar-los al domini dels grafs. Aquesta metodologies s anomena *graph embedding*.

En aquesta tesi proposem de fer una associació de grafs a vectors de característiques de forma simple i eficient fixant l atenció en la informació d etiquetatge dels grafs. En particular, comptem les freqüències de les etiquetes dels nodes així com de les aretes entre etiquetes determinades. Tot i la seva localitat, aquestes característiques donen una representació prou robusta de les propietats globals dels grafs. Primer tractem el cas de grafs amb etiquetes discretes, on les característiques són sencilles de calcular. El cas continu és abordat com una generalització del cas discret, on enlloc de comptar freqüències d etiquetes, ho fem de representants d aquestes. Ens trobem que les representacions vectorials que proposem pateixen d alta dimensionalitat i correlació entre components, i tractem aquests problems mitjançant algorismes de selecció de característiques. També estudiem com la diversitat de diferents representacions pot ser explotada per tal de millorar el rendiment de classificadors base en el marc d un sistema de múltiples classificadors. Finalment, amb una extensa evaluació experimental mostrem com la metodologia proposada pot ser calculada de forma eficient i com aquesta pot competir amb altres metodologies per a la comparació de grafs.

Abstract

Pattern recognition is the task that aims at distinguishing objects among different classes. When such a task wants to be solved in an automatic way a crucial step is how to formally represent such patterns to the computer. Based on the different representational formalisms, we may distinguish between *statistical* and *structural pattern recognition*. The former describes objects as a set of measurements arranged in the form of what is called a feature vector. The latter assumes that relations between parts of the underlying objects need to be explicitly represented and thus it uses relational structures such as graphs for encoding their inherent information. Vector spaces are a very flexible mathematical structure that has allowed to come up with several efficient ways for the analysis of patterns under the form of feature vectors. Nevertheless, such a representation cannot explicitly cope with binary relations between parts of the objects and it is restricted to measure the exact same number of features for each pattern under study regardless of their complexity. Graph-based representations present the contrary situation. They can easily adapt to the inherent complexity of the patterns but introduce a problem of high computational complexity, hindering the design of efficient tools to process and analyse patterns.

Solving this paradox is the main goal of this thesis. The ideal situation for solving pattern recognition problems would be to represent the patterns using relational structures such as graphs, and to be able to use the wealthy repository of data processing tools from the statistical pattern recognition domain. An elegant solution to this problem is to transform the graph domain into a vector domain where any processing algorithm can be applied. In other words, by mapping each graph to a point in a vector space we automatically get access to the rich set of algorithms from the statistical domain to be applied in the graph domain. Such methodology is called *graph embedding*.

In this thesis we propose to associate feature vectors to graphs in a simple and very efficient way by just putting attention on the labelling information that graphs store. In particular, we count frequencies of node labels and of edges between labels. Although their locality, these features are able to robustly represent structurally global properties of graphs, when considered together in the form of a vector. We initially deal with the case of discrete attributed graphs, where features are easy to compute. The continuous case is tackled as a natural generalization of the discrete one, where rather than counting node and edge labelling instances, we count statistics of some representatives of them. We encounter how the proposed vectorial representations of graphs suffer from high dimensionality and correlation among components and we face these problems by feature selection algorithms. We also explore how the diversity of different embedding representations can be exploited in order to boost the performance of base classifiers in a multiple classifier systems framework. An extensive experimental evaluation finally shows how the methodology we propose can be efficiently computed and compete with other graph matching and embedding methodologies.

Contents

Agraıments **i**

Resum **iii**

Abstract **v**

1 Introduction **1**

1.1 Overview of Pattern Recognition . . . 1

1.1.1 Statistical Pattern Recognition . . . 2

1.1.2 Structural Pattern Recognition . . . 3

1.1.3 From structural to statistical . . . 5

1.2 Objective of this work . . . 7

1.3 Contributions and organization . . . 7

2 Graph Matching **9**

2.1 Graphs and subgraphs . . . 9

2.2 Exact Graph Matching . . . 10

2.3 Error-tolerant Graph Matching . . . 12

2.3.1 Graph Edit Distance . . . 13

2.4 Graph Kernels . . . 14

2.5 Graph Embeddings . . . 16

2.6 Discussion . . . 17

3 Embedding of Discretely Attributed Graphs via Label Frequencies **19**

3.1 Embedding of Graphs via Label Frequencies . . . 19

3.1.1 Basic Procedure . . . 19

3.1.2 Adding Edge Attributes . . . 20

3.1.3 Formal Definition . . . 21

3.1.4 Node-based *vs* Edge-based Features . . . 22

3.2 On the Correlation with Edit Distance . . . 23

3.2.1 Edit Cost Function . . . 24

3.2.2 Distance Correlation . . . 25

3.2.3 Classifiers Correlation . . . 27

3.2.4 Discussion . . . 29

3.3 The Embedding Space under different Metrics . . . 29

3.4 Higher order edge-based features . . . 31

3.4.1 k-length Paths . . . 31

3.4.2 Experimentation . . . 32

4 Statistics on Node Label Representatives for the Embedding of Continuously Attributed Graphs 35
4.1 From discrete to continuous: node attribute representatives 35
4.1.1 Hard version 36
4.1.2 The fuzzy version 37
4.1.3 Discussion 39
4.2 Selection of representatives 39
4.2.1 Spanning prototypes 39
4.2.2 kMeans algorithm 40
4.2.3 Fuzzy kMeans 40
4.2.4 Mixture of Gaussians 40
4.2.5 Summary 41
4.3 The embedding space under different metrics 42
4.4 The effect of the representative set 42
4.4.1 Experimental results 43
4.4.2 Discussion 46

5 Feature Selection 49
5.1 Feature selection 49
5.1.1 Ranking methods 50
5.1.2 PCA-based methods 51
5.2 Experimental evaluation 52
5.2.1 Ranking methods 52
5.2.2 PCA-based methods 56
5.3 Discussion 59

6 Ensembles of Classifiers 61
6.1 Multiple Classifiers on Graphs 61
6.2 Multiple Classifier Methods 62
6.3 Classifier Selection 63
6.4 Experimental evaluation 63
6.4.1 Ensembles for discrete attributed graphs 64
6.4.2 Ensembles for continuous attributed graphs 66
6.5 Discussion 69

7 Experimental Evaluation 71
7.1 Discrete Attributed Graphs Classification 71
7.1.1 Reference Systems 71
7.1.2 Computational Time Analysis 72
7.1.3 Results 74
7.1.4 Discussion 76
7.2 Continuous Attributed Graphs Classification 76
7.2.1 Reference Systems 77
7.2.2 Results 78
7.2.3 Discussion 83
7.3 Graph Clustering under Domain-specific Embedding: color RAGs 84
7.3.1 Motivation 84
7.3.2 Color Naming based Embedding 85
7.3.3 Experimental validation on the ICPR Graph Embedding Contest . . . 87
7.3.4 Contest Participants 90
7.3.5 Results 91

7.3.6 Discussion 92

8 Conclusions 95
8.1 Summary of the Contributions 96
8.2 Future Perspective 99

A Graph Data 103
A.1 Discrete Attributed Graphs 103
A.1.1 Object Image Datasets 103
A.1.2 Molecule Datasets 105
A.2 Continuous Attributed Graphs 106
A.2.1 IAM Database Repository 106
A.2.2 ICPR 2010 Graph Embedding Contest 109

B Scatter Plots of the Correlation between Graph Edit Distance and the L_1 distance for the Embedding Features 111

C Study of Different Distances for the Continuous Attributed Graphs 121

List of Publications 129

Bibliography 131

List of Tables

1.1 Statistical *vs* Structural Pattern Recognition. 6

5.1 Feature statistics for the different ranking methods under different embedding configurations. 55

5.2 Feature statistics for the different PCA-based methods under different embedding configurations. 58

6.1 Ensemble distributions for the discrete datasets. 66

6.2 Ensemble distributions for the continuous datasets. 69

7.1 Discrete attributed graph datasets. Classification results on the test sets. Accuracy rates given in %. 74

7.2 Performance of the ensembles of classifiers for the discrete attributed graph datasets. Single classifier results are shown only if $k = 1$. Multiple classifier results are shown only if the resulting ensemble is compounded of more than 1 base classifier. 76

7.3 kNN results on the test sets of the continuous attributed graph datasets for the different embedding configurations. The best result for each dataset is shown bold face. 80

7.4 SVM results on the test sets of the continuous attributed graph datasets for the different embedding configurations. The best result for each dataset is shown bold face. 80

7.5 kNN results on the test sets of the continuous attributed graph datasets after feature selection. The best result for each dataset is shown bold face. 82

7.6 SVM results on the test sets of the continuous attributed graph datasets after feature selection. The best result for each dataset is shown bold face. 82

7.7 Performance of the ensembles of classifiers for the continuously attributed graph datasets. Results are shown only if the resulting ensemble is compounded of more than 1 base classifier. The best result for each dataset is shown bold face. 83

7.8 Generic *vs* color-based representations. Clustering results on the train sets by the use of different distance measures. The geometric mean of the results on the different datasets is also shown. The best results are shown bold face. . . 88

7.9 Generic *vs* color-based representations. Clustering results on the train sets by the use of different kernel functions. The geometric mean of the results on the different datasets is also shown. 90

7.10 Clustering results on the test sets. C-index under the Euclidean distance. Comparison with the contest participants. The best results are shown bold face. . . . 91
7.11 Clustering results on the test sets. Comparison of the C-index under the use of different distances. The best results are shown bold face. . . . 92
7.12 Clustering results on the test sets. Comparison of the C-index under the use of different kernel functions. . . . 93

A.1 Characteristics of the discrete object image datasets. Size of the training (tr), validation (va) and test (te) sets, the number of classes for each dataset (#classes), the average number of nodes and edges ($\pm V$ /$\pm E$), the maximum number of nodes and edges (max V /max E) and the size of the node (L_V) and the edge (L_E) label alphabets. . . . 105
A.2 Characteristics of the molecule datasets. Size of the training (tr), validation (va) and test (te) sets, the number of classes for each dataset (#classes), the average number of nodes and edges ($\pm V$ /$\pm E$), the maximum number of nodes and edges (max V /max E) and the size of the node (L_V) and the edge (L_E) label alphabets. . . . 106
A.3 Characteristics of the IAM repository datasets. Size of the training (tr), validation (va) and test (te) sets, the number of classes (#Cls), the average number of nodes and edges (An/Ae) and the maximum number of nodes and edges (Mn/Me) . . . 109
A.4 Characteristics of the ICPR 2010 Graph Embedding Contest datasets. Size of the sets, the number of classes (#Cls), the average number of nodes and edges (An/Ae) and the maximum number of nodes and edges (Mn/Me) . . . 110

List of Figures

1.1 Different relational structures. 3

1.2 Examples of graph-based representations. 4

2.1 A graph and two subgraphs. 10

2.2 An edit path between two graphs. Node labels are represented by different colors. 13

3.1 Two non-isomorphic graphs with the same vectorial representation counting node label appearances. 20

3.2 Two edges with different attributes joining equally labelled nodes. 21

3.3 Two graphs for which the L_1 distance on the node-based features is equivalent to the edit distance when neither edge costs nor node substitutions are considered. 23

3.4 Correlation maps (3D plot and projection) in an ideal situation where the embedding distances would perfectly emulate the edit distances. 25

3.5 Correlation values as a function of the weighting parameters. 26

3.6 Classifier performances. Edit distance as a function of ρ. L_1 embedding as a function of α. Scatter plots of the accuracies of all pairs $\rho = \alpha$. 28

3.7 Study of the use of different vectorial metrics in the embedding spaces. kNN classifier on the two different version of the embedding of discretely attributed graphs. 30

3.8 A path of length k between two nodes with labels A and B 31

3.9 kNN results on the sets of features for discretely attributed graphs as a function of the length of paths considered in graphs. 32

4.1 Four graphs whose node attributes from $\mathbb{R}^2$ are all around one of the four representatives (0 0) (0 1) (1 0) (1 1) , and their corresponding discrete versions after assigning each node to an element in the set of representatives. 37

4.2 Study of the effect of distances for the Digits database using a kNN classifier. 43

4.3 Validation results for the different databases of letters. Accuracy rates of a kNN classifier in conjunction with a ϵ^{2} distance on the validation set. The horizontal axis shows different choices of the size of the representative set. The hard configurations of the proposed embedding are depicted in the figures on the left. Soft configurations are shown on the right column of figures. 44

4.4 Validation results for the GREC, Digits, Fingerprints and COIL databases. Accuracy rates of a kNN classifier in conjunction with a ϵ^2 distance on the validation set. The horizontal axis shows different choices of the size of the representative set. The hard configurations of the proposed embedding are depicted in the figures on the left. Soft configurations are shown on the right column of figures. 45

5.1 Validation results for some configurations on the Fingerprint and Digits databases. Accuracy rates of a kNN classifier in conjunction with a ϵ^2 distance on the validation set. First and third rows show the behavior when keeping a relative number of components. Dots on the curves show the best configuration. Second and fourth rows plot the same curves in absolute terms of the selected features. 53
5.2 Fractions of variance for different choices of the representative set on the GREC dataset. 56

6.1 ϵ^2 SVMs. Left column: accuracy as a function of the length of the paths considered for the edge-based features. Right column: ensembles of the classifiers. 65
6.2 ϵ^2 SVMs. Left column: accuracy as a function of the size of the representative set. Right column: ensembles of the classifiers. 67
6.3 ϵ^2 SVMs. Left column: accuracy as a function of the size of the representative set. Right column: ensembles of the classifiers. 68

7.1 Average times in milliseconds (vertical axis, logarithmic scale) of the test stage of both the reference and the proposed systems. The length of the ticks on the top of the bars show the variance around the mean values. 73
7.2 Distributions of the graphs nodes in the RGB space. (a) Original color of each node. (b) kMeans clusters for $k = 10$. (c) Color naming distribution. 86
7.3 Clustering performance of the generic versions of the embedding as a function of the size of the set of representatives. Results for different vectorial measures. 89

A.1 Examples of images in the object datasets. 104
A.2 Object segmentation for graph extraction. 104
A.3 Examples of molecules from the AIDS dataset. 105
A.4 Synthetic prototype of the letter A (left). One instance of the same class for each of the distortion levels (low, medium, high). 107
A.5 An example of each of the distortion levels in different symbols of the GREC dataset. 107
A.6 Examples of each of the classes of the Digits dataset. 108
A.7 Fingerprints Dataset: one example of each class. From left to right: *left*, *right*, *arch* and *whorl*. 108
A.8 The 100 COIL objects. 109

B.1 ALOI Dataset: Scatter plots of the Graph Edit Distance versus the L_1 distance on the Embedding Features. Results for $\rho = 0\ 0\ 2\ 0\ 4$. 112
B.2 ALOI Dataset: Scatter plots of the Graph Edit Distance versus the L_1 distance on the Embedding Features. Results for $\rho = 0\ 6\ 0\ 8\ 1$. 113
B.3 ODBK Dataset: Scatter plots of the Graph Edit Distance versus the L_1 distance on the Embedding Features. Results for $\rho = 0\ 0\ 2\ 0\ 4$. 114

B.4 ODBK Dataset: Scatter plots of the Graph Edit Distance versus the L_1 distance on the Embedding Features. Results for $\rho = 0\ 6\ \ 0\ 8\ \ 1$. 115
B.5 AIDS Dataset: Scatter plots of the Graph Edit Distance versus the L_1 distance on the Embedding Features. Results for $\rho = 0\ \ 0\ 2\ \ 0\ 4$. 116
B.6 AIDS Dataset: Scatter plots of the Graph Edit Distance versus the L_1 distance on the Embedding Features. Results for $\rho = 0\ 6\ \ 0\ 8\ \ 1$. 117
B.7 MUTAG Dataset: Scatter plots of the Graph Edit Distance versus the L_1 distance on the Embedding Features. Results for $\rho = 0\ \ 0\ 2\ \ 0\ 4$. 118
B.8 MUTAG Dataset: Scatter plots of the Graph Edit Distance versus the L_1 distance on the Embedding Features. Results for $\rho = 0\ 6\ \ 0\ 8\ \ 1$. 119

C.1 Study of the effect of distances for the Letter LOW database using a kNN classifier. 122
C.2 Study of the effect of distances for the Letter MEDIUM database using a kNN classifier. 123
C.3 Study of the effect of distances for the Letter HIGH database using a kNN classifier. 124
C.4 Study of the effect of distances for the GREC database using a kNN classifier. 125
C.5 Study of the effect of distances for the Fingerprints database using a kNN classifier. 126
C.6 Study of the effect of distances for the COIL database using a kNN classifier. 127

Chapter 1

Introduction

1.1 Overview of Pattern Recognition

According to the dictionary, a *pattern* is *a particular,* ***recognizable*** *way in which something is done or organized* . Also *a particular physical form or arrangement* . Or even *a design made up of an arrangement of lines or shapes, especially one in which the same shape is repeated at regular intervals over a surface* [1]. As marked bold face, a basic property of patterns is that they should be recognizable, this is, one should be able to tell the difference between distinct patterns. Indeed, every day, we encounter such a situation plenty of times. For example, when the alarm bell wakes us up in the morning and we watch the clock, we recognize the numbers and are able to tell what time it is. When we open the fridge to prepare our breakfast, we are able to identify the juice bottle and pick it up instead of, for instance, a beer can. Also, when we leave home towards work, we can easily tell which is the key to our entrance door among all keys in our keyring so we can lock the door after us. Recognizing patterns is thus an activity we do all the time and is actually one of the major abilities of human beings. In fact, solving the problem of distinguishing and recognizing patterns is so important for us that our cognitive system has evolved in a very sophisticated way in order to be able to efficiently and properly get optimal solutions for its resolution.

Over the years, we have learnt to solve this problem by *learning from examples.* Facing the same situation over and over gives us intuitive ways to deal with our daily life and tackle straightforward problems and so, by eating every day with forks, knifes and spoons, we just know without making any hard decision which is the object we have to use whenever we need to cut a steak. What it is not that easy, though, is to methodically formalize the process under which we recognize the knife and the spoon. In other words, we know the knife is sharp and the spoon is rounded and so we easily distinguish the tools, but the concepts of *sharp* and *rounded* are not trivially defined in an algorithmic manner, so if we wanted a machine to do this task for us or any other similar task we would need to find the optimal ways to tell it how to understand such concepts. And this is far from having a basic solution.

The pattern recognition community has been trying for the last decades to use machines for solving tasks similar to the one just described [45]. Regardless of the complexity of the tackled problems, several successful results have been obtained. Relevant examples include optical character recognition [5, 77], mail sorting [77, 107], text categorization [85, 148, 170], handwritten text recognition [13, 105, 185], writer identification [143, 144], molecular structure-activity relationship analysis [19, 112], fingerprint recognition [80, 119, 183], face detection in still images [71, 171], and many others. In many of these cases, imitating our own way of learning from examples, the methods employed are based on feeding the machine with data so it can be able to generalize and give proper answers to the problem under consideration.

Machine learning tries to emulate what we humans have experienced over the years. Again, back to the same illustrative example, by showing several instances of knifes and spoons to the machine, one assumes the machine will be able to learn the main characteristics that makes knifes different from spoons and spoons different from knifes. Formally, a mathematical formulation that models the learning process is the following. Let $\mathcal{X}$ be the set of all possible objects under consideration for instance, all utensils we have in our kitchen drawer , and let $\mathcal{Y} = \quad y_1 \qquad y_k$ be the finite discrete set of class labels among which we aim at distinguishing the patterns in $\mathcal{X}$ let us say, *forks*, *spoons* and *knifes*. In supervised learning[1] a set of pattern examples $\mathcal{T}$ together with their corresponding class label is provided, $\mathcal{T} = \ (\mathbf{x}_i \ y_i)\ _{i=1 \quad N}$. Usually this set is referred to as the training set. The main goal is to infer a function $f : \mathcal{X} - \quad \mathcal{Y}$ that assigns patterns to classes. This is, a decision rule able to tell whether a given tool is either a fork, a knife or a spoon. The mapping f is usually called a *classifier*.

A good classifier should of course be able to correctly classify all patterns in the training set. We face the risk, though, that such a classifier fits too much to the training data in the sense that it is not able to assign correctly the category of new or *unseen* elements $\mathbf{x} \quad \mathcal{X} \quad \mathbf{x}_i \ _{i=1 \quad N}$, usually known as *test* elements. Such situation is usually called as *overfitting*. The opposite situation, *underfitting*, is the case where the classifier is not discriminative enough for the training patterns and so the boundaries between different categories are not properly defined. Machine learning researchers thus face the problem of finding the correct balance between properly classifying training data and being able to generalize to new unseen patterns.

On top of these things, it is obvious that a key issue in the learning process, and more particularly, in the design of the classifiers, is the nature of the space of patterns, this is, the way in which the space of patterns $\mathcal{X}$ is formally and mathematical presented to the machine. In particular, the way patterns are codified should be able to capture all the relevant information that makes them distinguishable ones from the others, so that the evetual classifier easily detects the inherent characteristics of each pattern category. Based on how this situation is tackled we may divide the field of pattern recognition into two main branches, namely, statistical pattern recognition where feature vectors are used as the representational paradigm for patterns and structural pattern recognition where relational structures like graphs are employed for this purpose.

1.1.1 Statistical Pattern Recognition

The most common way to present the space of patterns to the machine is by using what are called *feature vectors*. In particular, a set of n measurements are computed for a given pattern $\mathbf{x} \quad \mathcal{X}$ and so each pattern is represented as a point[2] $\mathbf{x} = (x_1 \qquad x_n) \quad \mathbb{R}^n$ in an n-dimensional Euclidean space. For instance, our kitchen utensils could be described as 2-dimensional vectors where components regard a measure of roundness and a measure of

[1]Other learning strategies are there in the literature. Unsupervised learning is the case where input data comes without a predefined category and semi-supervised learning is the case where only some examples are provided together with its class label.

[2]Note the abuse of the terminology. Mathematically, we would distinguish among points and vectors in a vector space. For the current discussion and for the construction of the whole pattern recognition apparatus, these terms are interchangeable.

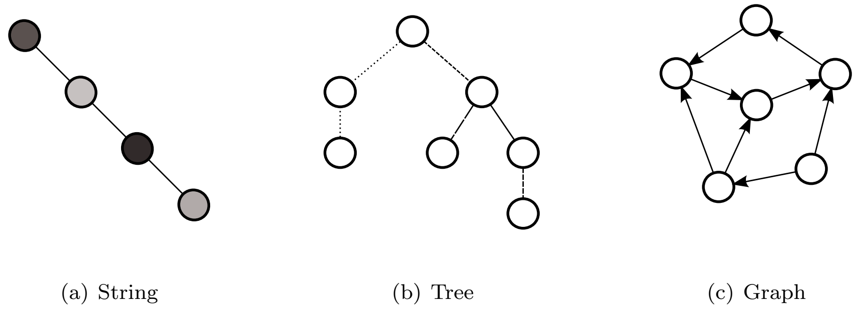

(a) String (b) Tree (c) Graph

Figure 1.1: Different relational structures.

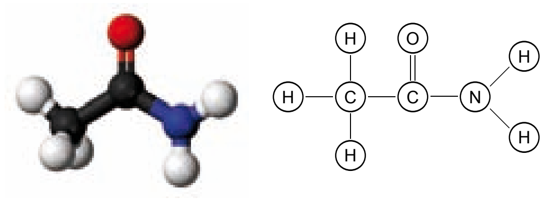
sharpness of each utensil, and thus ideally, in such a space, points representing spoons should be separated from those representing knifes.

Representing patterns as points in a vector space provides us with a very fancy situation for the comparison of the elements under study. Operations such as the distance between points, the mean of a set of points or even the variance, are easily defined and are cheap to compute. More generally, the fact that vector spaces are a very flexible and malleable mathematical construction has allowed the research community to come up with many efficient and sophisticated algorithms for the problem of pattern analysis [14, 45].

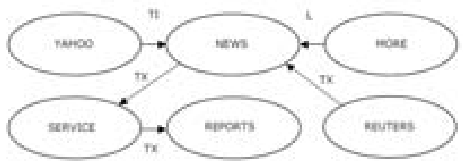
In any case, although its popularity and successful results, not everything is completely convenient when using feature vectors as the representational paradigm of patterns. For instance, there is a major restriction in the fact that all patterns, regardless of their complexity, should be represented using the same exact number of features. In other words, in a same problem, a very simple pattern should be measured as many times as a rather more complicated one, when probably such situation is unnecessary or undesired. A second major problem is the lack of an explicit relationship among parts of the objects. In many settings, it would be interesting to describe an inherent structure within the parts of the patterns under consideration —for instance, a relative spatial location between them or, more generally, whether or not two parts of the object interact with each other— and this is not directly applicable by the use of feature vectors. Relational structures such as strings, trees or graphs fit under a representational paradigm that do not suffer from these two described drawbacks —they actually solve these situations in a straightforward manner—, and pattern analysis under these scenarios is known as structural pattern recognition.

1.1.2 Structural Pattern Recognition

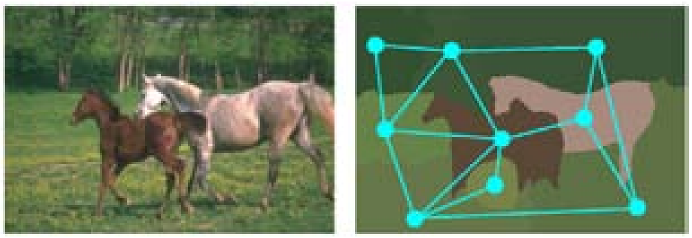
Graphs are an abstract formalism for data structures, widely used in computer science. A graph is composed of a finite set of nodes and a set of edges linking these nodes. Nodes and edges can store numerical or symbolic information called labels, and edges can have a direction. Strings are a particular case of graphs where a starting node is connected to only one node, which at the same time is connected to a third node and so forth, until a last node is reached. Trees are another case of graphs where nodes are connected by exactly only one path. In this thesis we will not distinguish among these different structures and will work with general graphs. An example of each of these relational structures are depicted in Fig. 1.1. For instance, Fig. 1.1(a) shows a string with labelled nodes (each shade degree is a different label) and undirected and unlabelled edges, Fig. 1.1(b), a tree with unlabelled



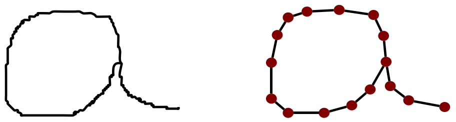
(a) A molecule and its graph representation



(b) A web document as a graph



(c) An image and its segmentation graph



(d) A character and its graph representation

Figure 1.2: Examples of graph-based representations.

nodes and undirected and labelled edges (each line dashed pattern is a different label), and finally, Fig. 1.1(c), a graph with unlabelled nodes and directed and unlabelled edges.

The power of graphs in terms of their ability to represent patterns has attracted a lot of attention in the past decades and has been applied to many fields [22, 32, 33, 67, 87]. For instance, molecular compounds have a natural structure in the form of undirected graphs, where atoms are represented by nodes labelled with the corresponding atomic element and edges regard the covalent bonds between atoms and are labelled with the corresponding covalent number. Fig. 1.2(a) shows the 3D representation of the acetamide molecule and its corresponding graph representation. Bioinformatics and chemoinformatics have widely used graphs as the main paradigm for molecule representation with great success [16, 112, 127]. Web documents, the world wide web and, generally, computer networks, are also a good example where a graph representation suits perfectly. A standard representation of a web document as a graph is shown in Fig. 1.2(b) (image taken from [141]). Topics and web links among them are represented as a graph. Generally, computers constitute nodes of a graphs and the connections between computers are represented by edges. Exploiting the information of these graphs is what the web content mining and network analysis fields commonly carry out [41,42,141,142]. Computer vision is nowadays one of the major computer science research lines and it has also served itself of graph-based representations. Image classification and segmentation are just a few examples of where computer vision may have served itself of relational structures [44, 50, 68, 110]. Extracted from [68], in Fig. 1.2(c), an image is shown together with a graph representing its segmentation regions. Finally, character and symbol recognition has also used graphs as their representational paradigm, showing profitable results [34, 106, 138]. As a simple example, in Fig. 1.2(d) a handwritten character is shown with a typical graph representation where nodes and edges are extracted along the stroke.

Most recently, there has been a great eruption of interest in using relational structures to characterize the complexity of certain networks [8, 48]. Rather than comparing graph-based representations of patterns one-to-one, large graphs are used to represent the interaction of certain entities. For instance, protein-protein interaction networks can be used to comprehend the relationship between the function of different biological organisms. The study of brain connectivity can also take advantage of complexity network characterization as a form of graph analysis.

Despite the broad applicability of graph-based representations, there is an important drawback concerning their use. The very same complexity regarding their representational capabilities is responsible of making the processing and analysis of graphs a hard and costly task. In particular, just a few mathematical structure is there available in the graph domain. More precisely, the unordered nature of the nodes in a graph and the fact that the number of nodes in graphs may differ from one graph to another in the same problem toughens the comparison of patterns under graph-based representations. What were straightforward operations for the case of feature vectors, they now become expensive and arduous tasks. Notions like the mean or the variance of a set of graphs are far from being trivial to define.

In contrast to the case of feature vectors where the mathematical flexibility of vector spaces allows to come up with efficient and powerful machine learning algorithms, in the case of graphs we are very limited in this sense. In comparison to the statistical domain, we lack a set of sophisticated tools for the analysis of patterns and this drastically restricts our endeavour to successfully use graph-based representations for pattern recognition. We thus encounter a paradoxical situation. We have at our disposal plenty of machinery for the analysis of patterns represented under feature vectors, but such representations are weak in terms of they representational power. Then, whenever we find another data structure that is able to overcome the limitations of the feature vectors, we discover that a repository of tools for the analysis of patterns is missing. This paradox is illustrated in Table 1.1. The main objective of this thesis is to shed some light on this puzzle by trying to bridge the gap between the statistical and the structural pattern recognition worlds.

1.1.3 From structural to statistical

The ideal scenario for solving pattern recognition problems would be to represent patterns under graph-based representations, and further analyse these patterns using any of the several machine learning algorithms developed under the rich statistical framework that feature vectors offer. Several attempts have been done towards this objective [26]. Two large families of works can be found in the literature, namely, ***graph kernels***, where graphs are implicitly

Table 1.1: Statistical *vs* Structural Pattern Recognition.

	Statistical Pattern Recognition	Structural Pattern Recognition
Representational power	✗	✓
Wealth of algorithmic tools	✓	✗

mapped into a hidden feature space and similarities between them are expressed in terms of the scalar product between their corresponding image vectors, and ***graph embeddings***, where an explicit formulation of such a map is proposed and thus any operation between graphs can be performed in terms of their corresponding image vectors.

To detail the first one, we must describe   although briefly   the set of learning algorithms known as *kernel machines*. Let us go back now into the statistical pattern recognition world. In many cases, the set of steps that define the algorithm to build up a classifier need not to explicitly present the feature vectors themselves but, instead, the similarity between pairs of patterns in terms of the scalar product between their corresponding feature vectors. Formally, the construction of a classifier f does not generally depend on feature vector representations but on a dot product $\cdot \ \cdot_{\mathcal{X}}$ in the pattern vector space, $f \sim f(\ \cdot \ \cdot_{\mathcal{X}})$.

Together with this property of the classifier construction, we grab the notion of a kernel function from the functional analysis domain. A kernel function is a symmetric function mapping pairs of objects to real numbers, $\kappa : \mathcal{X} \times \mathcal{X} \quad \mathbb{R}$. Such a kernel is called *valid* when is positive definite. It turns out that, for a given valid kernel κ, there always exist a Hilbert space $\mathcal{H}$ and a mapping $\nu : \mathcal{X} \quad \mathcal{H}$ such that the kernel function between two patterns from $\mathcal{X}$ is equal to the scalar product between the corresponding mappings by ν, this is

$$\kappa(x_1 \ x_2) = \ \nu(x_1) \ \nu(x_2) \ _{\mathcal{H}} \tag{1.1}$$

Intuitively, kernel functions are the similarity functions in terms of scalar product of some implicitly embedded patterns in a hidden   in the sense of implicit   space $\mathcal{H}$. This important theorem allows those mentioned algorithms that only depend on scalar products between pattern vectors to be *kernelized*. This is, the scalar product may be substituted by a valid kernel function and the mathematical foundation of the algorithm steps remain intact. We call these algorithms kernel machines. Examples of kernel machines are the nearest neighbour rule, principal component analysis, support vector machines, kMeans clustering, and many others [147, 151].

Let us try not to lose the thread. We were describing ways in which statistical-based machine learning algorithms may be applicable to graph-based representations. As a matter of fact, although we have explained kernel machines from the statistical pattern recognition point of view, we have not put any assumption on which should be the space of patterns under consideration in the definition of what a valid kernel is. Consequently, if we are able to define proper valid kernel functions for graph-based representations, the set of all kernel machines becomes available for the graph domain. In other words, by defining proper similarity functions between graph instances, we will be able to directly apply kernel machines to our graph-based representations of patterns. Families of graph kernels will be reviewed in detail in Section 2.4 of this thesis.

As we mentioned before, there is another group of ways to make machine learning algorithms applicable to the graph domain. This one   known as graph embeddings   is rather more intuitive than the one just described. In particular, the goal of graph embeddings is to extract numerical features from graphs and arrange them in the form of a feature vector. And that is to say, assign to each graph a point in a vector space. By this methodology, we clearly get access to all learning algorithms that are available since we do nothing but converting the situation of graphs into that of feature vectors. Of course, the problem arises of which graph features should be extracted in order to lose as less structural information as possible, which is precisely the main benefit of graph representations with respect to feature vectors. Section 2.5 will review some attempts to make the described transition. Finally, let us note that it exists a clear relation between these two manners of resolving the gap between the statistical and structural worlds. As soon as pattern vectors are assigned to graph-based representations, any valid kernel on vectorial entities will also define a symmetric positive definite function for graphs.

1.2 Objective of this work

As it will be described in the next chapter, most of the graph embedding methodologies still suffer from a high computational complexity. Given the richness of the graph representations, it is of course not a trivial task to efficiently extract features that regard their whole representational power. In any case, the effort of transiting from graphs to vectors would not be worth if we would still have to perform costly operations for getting a convenient vectorial representation of graphs.

The main objective of this thesis is to define ways for embedding graphs into vector spaces that make the transition within a reasonable number of operations and without losing the abilities of graphs to properly represent structured patterns. In particular, we aim at describing a way to embed any type of graphs discrete and continuous attributed ones into vector spaces that overcome the efficiency problem that most of the embedding methodologies carry with them and such that classification problems under these representations are successfully solve. For this to be accomplished, we define embedding features that regard statistics of labelling information. More precisely, the features proposed in this work check for the frequency of a certain label to appear as a node label in a given graph and also the frequency of a certain relation between nodes with two specific labels. These measurements are rather local concerning the topology of the graphs under study, but the arrangement of all of them in the form of a feature vector makes the proposed methodology, as proven with an extensive experimental evaluation, a robust way to analyse patterns and to efficiently compete with other state-of-the-art approaches for the characterization of graphs in terms of feature vectors.

1.3 Contributions and organization

The contributions and the rest of this thesis are organized as follows. We will first stablish the notation that is going to be used throughout this work regarding graphs and graph comparison. We will also go over the main classical graph comparison methodologies divided into exact and inexact graph matching , putting special emphasis on *graph edit distance* as its major representative. This particular graph matching approach will serve us as a comparative baseline in many situations regarding the validation of our embedding methodology. Finally, as stated above, the literature concerning graph kernels and graph embeddings will be properly reviewed. These are the contents that constitute the body of Chapter 2.

The embedding of graphs into vector spaces will be initially tackled for the case of discrete attributed graphs. We propose to compute a set of features regarding the occurrence and co-occurrence of node labels in the graphs. After formalizing the computation of such features, they will be related to a specific form of graph edit distance by which we will start to discover the benefits of our embedding proposal for the problem of graph characterization and classification. A discussion regarding the metric of the vectorial space where our vectors life follows this part. We will discuss all these contents in Chapter 3, where we finally will try to redefine the original features in terms of paths travelled along the graph edges and then generalize this idea to give more robust and diverse graph embedding representations.

Whenever we encounter graphs whose node labels are continuous attributes, this is, values from $\mathbb{R}^d$, the previous methodology in not directly applicable since occurrence and co-occurrence of continuous labels is not feasible. In Chapter 4 we will study the transition from the discrete methodology proposed in the previous chapter to the case of continuous attributed graphs. The idea underlying the methodology from then on is to select a set of representative elements for the node labels of the graphs under study and extract statistics of these representatives instead of the labels themselves. We will present essentially two versions of this transition by either a *hard* assignment or a *soft* assignment from node labels to representatives. We will discuss several ways to select the set of representatives that will eventually lead to a vast and diverse set of vectorial representations of graphs. The metric of the vectorial space will also be experimentally discussed for the continuous case. We will then finish this chapter by experimentally validate the different versions of the proposed methodologies.

It turns out that some problems regarding high dimensionality, correlation among features and redundancy might appear in the feature vectors of our embedding proposal. In order to tackle these problems, in Chapter 5 we will apply feature selection algorithms to the embedding representations we propose in the previous chapter. We will experimentally evaluate which is the effect of applying such algorithms for selecting relevant features that allow to perform on the same classification levels by using only the actual most discriminative features.

As we will observe, both the length of the considered paths in the discrete attributed graphs and the size of the set of representatives for the case of continuous attributed graphs are parameters that offer some diversity in the embedding representations. Based on such diversity we will be able to construct different base classifiers. In Chapter 6 we will use these base classifiers and combine them in a multiple classifier systems framework in order to extract the best of our vectorial representations and eventually boost the performance of the classification problem.

A rigorous experimental evaluation will be carried out in Chapter 7. In particular, we will put our methodologies in comparison to classical graph matching approaches and to a another graph embedding algorithm. The problems of graph classification and graph clustering will be successfully and efficiently solved. In this chapter we will present the results we have obtained.

Chapter 8 will conclude this thesis by summarizing the main contributions of the work presented hereafter. It will also points out the future research directions that this thesis may open. Several datasets of graphs will be used along this work. Some of them are publicly available and some others have been created by ourselves. Appendix A carefully describes all of them. Finally, Appendixes B and C will present figures that result from the validation of some stages of this work.

Chapter 2

Graph Matching

In the previous introduction of this thesis we have stressed the main advantages of graph representations over feature vectors in terms of their representational power. In this chapter, we formalize the definition of graph and subgraph and review the common ways of graph comparison. Based on how the assignment of nodes of one graph are matched to nodes of the other graph, these ways are divided into two main categories, namely, exact and inexact graph matching. The former wants to answer whether two graphs are equivalent both at the structure and at the labelling levels. The latter tries to find an assignment between nodes that is able to cope with structural errors. We also review modern approaches for graph processing, such as graph kernels, where similarity measures between graphs are sought, and graph embeddings, where an explicit feature vector representation of graphs is defined.

2.1 Graphs and subgraphs

There exist several ways to define a graph. The following one is general enough to include all types of graphs used in this thesis and all discussions concerning related works.

Definition 2.1 (Graph) *Let L_V and L_E be two labelling sets. A graph g is a 4-tuple $g = (V \ E \ \mu \ \nu)$, where V is a finite set of nodes, $E \subseteq V \times V$ the set of edges, $\mu : V \quad L_V$ the node labelling function assigning a label from L_V to each node in the graph and $\nu : E \quad L_E$ the edge labelling function assigning a label from L_E to each edge in the graph.*

The number of nodes of a graph g is denoted by $\ g\ $. Edges of a graph are usually identified by the nodes they link. An edge $e \quad E$ can thus be represented by $e = (u \ v)$ where $u \ v \quad V$. Edges may be defined in a directed way and then u is called the source node and v the target node. In this case the graph is called *directed*. A graph is called *undirected* when such direction of edges is not really defined, or technically, when for all $e = (u \ v) \quad E$ there always exist $e' = (v \ u) \quad E$ such that $\nu(e) = \nu(e')$. All graphs considered in this work are undirected.

Based on the definitions of the labelling sets and the labelling functions we also have different types of graphs. For instance, a graph whose labelling sets are sets of discrete attributes[1] is called a *discrete attributed graph*. Whenever L_V and L_E are subsets of $\mathbb{R}^d$ for any $d \geq 1$, we call the graph *continuously attributed*. The labelling sets can also include a

[1]The words *label* and *attribute* are used indistinctly throughout all chapters of this thesis.

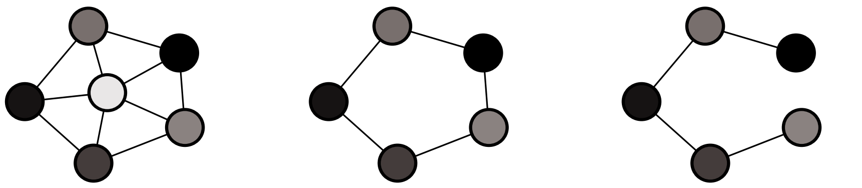

(a) Graph (b) Induced subgraph (c) Non-induced subgraph

Figure 2.1: A graph and two subgraphs.

special attribute ϵ called the null label. If all nodes and edges are labelled with this attribute the graph is called *unattributed.* Combinations of these settings can also occur. For instance, a graph with continuous attributed nodes and unlabelled edges or with discrete attributed nodes and continuous attributed edges. In this thesis we will work with different types of graphs, each dataset used is thoroughly described in Appendix A.

An important related concept is that of *subgraph* of a graph. Intuitively, a subgraph is a part of a graph preserving the structure and the labelling information.

Definition 2.2 (Subgraph) *Let $g_1 = (V_1, E_1, \mu_1, \nu_1)$ and $g_2 = (V_2, E_2, \mu_2, \nu_2)$ be two graphs. The graph g_1 is a subgraph of g_2, and we denote $g_1 \subseteq g_2$ if the following conditions hold*

- $V_1 \subseteq V_2$,
- $E_1 = E_2 \cap (V_1 \times V_1)$,
- $\mu_1(u) = \mu_2(u)$, *for all* $u \in V_1$
- $\nu_1(e) = \nu_2(e)$, *for all* $e \in E_1$

From this definition, a subgraph of a graph can be understood as the graph resulting of removing some nodes ($V_2 \setminus V_1$) of the graph and all their incident edges. In this case the subgraph is called induced. If, however, the second condition of the definition is substituted by $E_1 \subseteq E_2$, not only the incident edges to the deleted nodes are removed but also some other edges can be removed. In this case the subgraph is called non-induced. In Fig. 2.1 we depict these cases. A graph is shown (2.1(a)) together with an induced subgraph (2.1(b)) and a non-induced subgraph (2.1(c)).

2.2 Exact Graph Matching

The process of evaluating the similarity between two graphs is usually known as graph matching. In particular, exact graph matching aims at deciding whether two graphs — or parts of two graphs— are identical in both the topology or structure and the labelling information. As in all subfields of mathematics the equality of two entities is established through a bijection between them. In this case, we refer to this bijective function as *graph isomorphism* and define it as follows.

Definition 2.3 (Graph isomorphism) *Let $g_1 = (V_1, E_1, \mu_1, \nu_1)$ and $g_2 = (V_2, E_2, \mu_2, \nu_2)$ be two graphs. A graph isomorphism between g_1 and g_2 is a one-to-one correspondence between the node sets of the two graphs $f : V_1 \to V_2$ such that the labels of nodes are preserved under the bijection, this is,*

- *$\mu_1(u) = \mu_2(f(u))$ for all $u \quad V_1$,*

and, in both directions of the mapping, the adjacency of nodes is also preserved together with the labelling, this is

- *for all edge $e = (u \ v) \quad E_1$ there exists an edge $e' = (f(u) \ f(v)) \quad E_2$ between the images of the nodes by the mapping such that $\nu_1(e) = \nu_2(e')$,*
- *and for all edge $e = (u \ v) \quad E_2$ there exists an edge $e' = (f^{-1}(u) \ f^{-1}(v)) \quad E_1$ between the inverses of the nodes by the mapping such that $\nu_2(e) = \nu_1(e')$.*

Two graphs are called isomorphic if there exists a graph isomorphism between them.

Isomorphic graphs share the exact same structure and labelling information. In order to check whether two graphs are isomorphic, one has to find a one-to-one mapping between the node sets of both graphs so that the topology is preserved and the node and edge label correspondences are coherent.

Tree search algorithms are typically used for checking the graph isomorphism condition. The search tree is organized in such a way that mappings of nodes of the first graph to nodes of the second graph are explored until either the edge structure does not match or the labelling constraints are not fulfilled. Whenever all nodes of the first graph can be mapped to all nodes of the second graph without infringing the topology and labelling restrictions, these two graphs are deemed isomorphic. Examples of this strategy with their particular variants are [34 36,100,145,164]. The main problem of all of them is its high computational complexity, which is usually exponential in the number of nodes of the involved graphs. A different and interesting approach for graph isomorphism verification is based on the algebraic group theory. For each graph, the automorphism group is built and characterized in terms of a canonical form. By checking the equality of these canonical forms one is able to tell whether the graphs are isomorphic or not [115].

Particularly related to the problem of graph isomorphism is that of subgraph isomorphism. Intuitively, the problem of subgraph isomorphism is that of finding an isomorphism between a graph and a subgraph of it.

Definition 2.4 (Subgraph isomorphism) *Let $g_1 = (V_1 \ E_1 \ \mu_1 \ \nu_1)$ and $g_2 = (V_2 \ E_2 \ \mu_2 \ \nu_2)$ be two graphs. An injective function between the node sets of the two graphs $f : V_1 \quad V_2$ is called a subgraph isomorphism from g_1 to g_2 if there exist a subgraph $g \subseteq g_2$ such that f is a graph isomorphism between g and g_1.*

In plain words, a subgraph isomorphism indicates whether a graph is contained as a subgraph of a larger graph. The tree search approaches described above for the problem of graph isomorphism are also applicable to the problem of subgraph isomorphism.

Graph and subgraph isomorphisms are a rather restrictive way for matching graphs. They either answer whether two graphs are identical or not, but do not account for partial similarities between the two involved entities. In other words, two graphs or a graph and a part of it must be identical in terms of their structure and labels, when for instance, two graphs with most, but not all, of their nodes and structure being equal should be considered similar to some extent. Consider for instance the case where two graphs are exactly the same but only differ in the label of one node. Under the isomorphism paradigm these two graphs are not considered similar since a one-to-one correspondence is not possible to be established between the nodes of the graphs and thus this methodology fails to account for the obvious similarity there exists between these two graphs.

These deficiencies gave birth to the concepts of the largest common subgraph and the minimum common subgraph of two graphs. The *maximum common subgraph* of two graphs is the common subgraph of two graphs with the maximum number of nodes. Clearly, the larger the maximum common subgraph $mcs(g_1\ g_2)$ between two graphs is, the more similar the graphs are. The maximum common subgraph of graphs g_1 and g_2 can be understood as the intersection between the two graphs. Analogously, the union of the graphs defines the *minimum common supergraph* of two graphs, $MCS(g_1\ g_2)$. Intuitively, it is the smaller graph containing both g_1 and g_2 as subgraphs. Plenty of work is there available concerning the computation and the use of the maximum common subgraph and the minimum common supergraph for graph matching, and particularly, for defining several distance metrics on graphs [20, 23, 104, 114, 174]. In particular, a distance metric can be defined by relating the size of the maximum common subgraph of two graphs and the size of the largest one [27],

$$d(g_1\ g_2) = \frac{mcs(g_1\ g_2)}{\max(\ g_1 \quad g_2\ )} \tag{2.1}$$

or by relating the sizes of the minimum common supergraph and the maximum common subgraph [52],

$$d(g_1\ g_2) = \ MCS(g_1\ g_2) \ - \ mcs(g_1\ g_2) \tag{2.2}$$

In pattern recognition problems, the graph extraction process usually leads to graph-based representations of patterns encoding some level of distortion. In particular, graph labels and topology may be affected in such a way that the underlying represented objects have very different representations even if they belong to the same category. In these cases, not even the graph similarity measures induced by the maximum common subgraph and the minimum common supergraph are adequate since they require the graphs to have, at some extent, identical parts to regard high similarity values. This situation has led the structural pattern recognition community to devote quite some effort to come up with graph matching methodologies that are tolerant to structural and labelling differences in the graphs. These *inexact* or *error-tolerant* graph matching approaches calibrate the possibilities of the assignment of nodes between graphs under a wider perspective. We review them in the next section.

2.3 Error-tolerant Graph Matching

In order to allow the graph matching framework to cope with real world problems where graph-based representations usually suffer from distortion, one should tolerate to a certain degree errors in the matching of nodes and edges. In particular, instead of just demanding whether or not two node labels are the same, we would better check the similarity between these labels and permit some degree of dissimilarity. In this case, mappings between nodes and edges whose labels differ become possible. The main idea here would be to assign a high penalty to those mappings that link nodes with different labels and a low penalty to those linking similar nodes. This can also be applied for the topology of the graphs, and allow the mappings to disregard the edge structure of graphs by penalizing in a higher degree those cases where the edge structure is preserved at a lower degree. Thus, generally speaking, error-tolerant graph matching approaches aim at finding mappings between graphs such that these penalties or costs are minimized.

Different families of inexact graph matching are there in the literature. For instance, tree search algorithms are also applicable to cope with structural errors. In contrast to the exact graph matching situation, in this case the search is usually directed by the cost of the partial matching obtained at each time and by a heuristic estimate of the matching of the remaining nodes [32]. Relevant examples of this methodology are [10 12, 46, 150]. Genetic algorithms can also be applicable to graph matching [6, 38, 156, 160, 175]. Their main advantage over other techniques is that they can handle huge search spaces by encoding possible solutions of the node assignments in the forms of chromosomes. Nevertheless, their non-deterministic solutions make them not suitable in many cases. Spectral approaches have also been used to treat this problem [28, 95, 109, 110, 153, 165, 179]. The idea behind these methods is based on the fact that an eigendecomposition of any matrix regarding the topology of graphs for example, the adjacency or the Laplacian matrices is invariant under node permutations. Then, similar graphs are assumed to have similar eigenvectors and eigenvalues, and they are used for graph comparison. Relaxation labelling techniques cast the problem of graph matching as a non-linear optimization problem [30, 54, 66, 76, 92, 117, 180]. In particular, it is formulated as a labelling problem, where Gaussian probabilities model the compatibility between the assignment of each node of one graph to all the nodes of the other graphs, which at their time are expressed in terms of a set of labels. Other approaches have also been used for the problem of inexact graph matching. Artificial neural networks [51, 56, 81, 159], the Expectation Maximization algorithm [53, 108], random walks on graphs [62, 135] or even approximate least-squares [167, 168] are some examples among many other.

Graph edit distance is an error-tolerant graph matching approach that can deal with arbitrary graphs and has been proved to achieve robust results in many applications [4, 119, 136]. Because of these reasons, in many places of this thesis, it will serve as a reference method to compare the proposed methodologies with. We devote the next section to explain this approach in detail.

2.3.1 Graph Edit Distance

The main idea of graph edit distance is to define a dissimilarity measure between two given graphs by taking into account the minimum number of structural transformations that are needed to convert one graph into the other. Basic edit operations are defined in terms of insertion, deletion and substitution of nodes and edges. The concept of edit distance was originally proposed for the case of string matching [103, 173]. It was then generalized first to trees [149] and finally to graphs [21, 47, 139, 163].

Formulation

Given two graphs g_1 and g_2, the edit distance methodology, seeks for a sequence of edit operations e_1 e_2 e_n, consisting of the deletion of some nodes and edges of the first graph g_1, substituting some other nodes and edges of g_1 by those of g_2, and maybe also inserting the nodes and edges that are needed to obtain the second graph g_2. Such a sequence of operations is called an edit path and there is obviously an infinite number of edit paths that transform graph g_1 into graph g_2. For instance, one could always remove all nodes and edges of the first graph and then insert all nodes and edges of the second graph. This procedure is in fact an edit path transforming g_1 into g_2, but it might not be regarding the actual structural similarity between both graphs in a proper way. In Fig. 2.2 we show an edit path between two graphs that consists on a node deletion (with the corresponding deletion

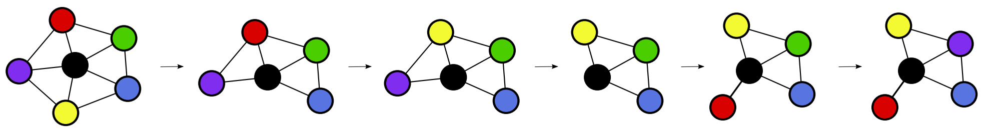


Figure 2.2: An edit path between two graphs. Node labels are represented by different colors.

of its adjacent edges), a node substitution, another node deletion (again together with the deletions of its adjacent edges), a node insertion and finally a node substitution.

In order to find the most appropriate edit path out of all existing ones, edit costs are assigned to each of the edit operations, corresponding to the strength of each operation. Such edits costs are usually given in terms of a cost function. Defining such a function is a key issue in the application of graph edit distance and several ways have been proposed to tackle the problem [120, 121]. The basic idea is to define them in such a way that there exists an inexpensive edit path between two similar graphs and an expensive one between two dissimilar graphs. Formally, the edit distance between two graphs is thus defined as the cost of the edit path that has the minimum cost among all the existing paths, this is,

Definition 2.5 (Graph Edit Distance) *Given two graphs $g_1 = (V_1\ E_1\ \mu_1\ \nu_1)$ and $g_2 = (V_2\ E_2\ \mu_2\ \nu_2)$, the edit distance between g_1 and g_2 is defined as*

$$d(g_1\ g_2) = \min_{e \in \mathcal{E}(g_1\ g_2)} \sum_{i=1}^{n} c(e_i) \tag{2.3}$$

where $e = (e_1 \qquad e_n)$ is an edit path from the set $\mathcal{E}(g_1\ g_2)$ of all existing edit paths from g_1 to g_2, and $c(e_i)$ indicates the cost of the edit operation e_i.

Computation

Computing the edit distance between two graphs is a problem of high computational complexity. Typical solutions are based on exploring all possible mappings from the sets of nodes and edges of one graph to those of the other graph [21, 139]. Such procedures have an exponential complexity in the number of nodes of the graphs and thus they are restricted to graphs with just a small number of nodes. Suboptimal methods for computing the edit distance of graphs have been widely studied and they allow larger graphs as input. Commonly, suboptimal methods are based on local search in the graphs reducing the search space of possible node-to-node mappings [15, 158]. In [86], the distance between graphs with unlabelled edges can be efficiently computed by a linear programming approach. Moreover, in order to speed up the edit distance computation, two modifications of an optimal algorithm are presented in [123]. The idea is to split the graphs into smaller subgraphs and transform the problem into that of finding an optimal match between the sets of subgraphs by dynamic programming.

Finally, in [133], a suboptimal algorithm based on bipartite graph matching is presented. An approximate solution of graph edit distance is provided by means of solving the assignment problem of nodes of one graph to nodes of the other in the following way. A cost matrix regarding the substitution of the local structure of every node of the source graph by the local structure of every other node in the target graph is built. Then the optimal assignment is extracted by the Munkres algorithm [116] and thus an edit path can be inferred. This suboptimal approach has been proved to obtain results similar to the optimal edit distance while reducing the computation time from exponential to cubic. As already said above, in several parts of this thesis, graph edit distance will act as a baseline methodology. All graph edit distance computations that are reported hereafter will be done by using this suboptimal approach.

2.4 Graph Kernels

Graph kernels are a rather modern form for graph processing. Similarity measures between graphs are sought in terms of positive definite symmetric functions. Plenty of work has been done towards the definition of valid kernel functions for the purpose of graph comparison. In particular, we can split the literature into different broad families, namely, diffusion kernels, dissimilarity-based kernels, convolution kernels and substructure finding kernels.

Diffusion kernels are those graph kernels that are defined in terms of a base similarity measure between the input graphs in order to build a kernel matrix [88, 94, 98, 99, 157, 169]. In fact, such methodology does not require patterns to be represented by graph-based representations, but as long as a symmetric similarity measure for graphs is provided, this can also be applied to graphs. Let $\mathcal{G}$ be a set of N graphs and $\delta : \mathcal{G} \times \mathcal{G} \quad \mathbb{R}$ a symmetric similarity measure defined on this set of graphs. The matrix $\Delta = (\delta_{ij})_{N \times N}$ of the similarities of all pairs of graphs in $\mathcal{G}$ can be converted into a positive definite kernel matrix by the use of the exponential diffusion kernel [94]

$$K = \sum_{k=0}^{\infty} \frac{1}{k!} \lambda^k \Delta^k = \exp(\lambda \Delta) \tag{2.4}$$

or the use of the von Neumann diffusion kernel [88]

$$K = \sum_{k=0}^{\infty} \lambda^k \Delta^k \tag{2.5}$$

The main idea behind diffusion kernels is to take profit of the base similarity measure by considering not only pairs of similar graphs, but also all those graphs that are similar to a given pair. With regard to their computation, the weighting factor λ^k makes similarities for a large enough k to be insignificant and thus their computation is not necessary.

Related to the diffusion kernels are those kernel functions that take advantage of having a dissimilarity measure between graphs and turns them into a similarity function. For instance, given a dissimilarity function such as the edit distance $d : \mathcal{G} \times \mathcal{G} \quad \mathbb{R}$, one is tempted to turn the dissimilarity values by a monotonically decreasing transformation. For instance, being $d(g_1 \;\; d_2)$ the edit distance of graphs g_1 and g_2, a similarity measure may be defined by [122]

$$\kappa(g_1 \;\; g_2) = \exp(-\gamma \cdot d(g_1 \;\; g_2)) \tag{2.6}$$

where $\gamma > 0$. This similarity measure is generally not positive definite, but in some works, it has been suggested that non-valid kernel functions are also usable in conjunction with kernel machines if some conditions are fulfilled [65, 122].

Convolution kernels are a family of kernels that try to infer similarity measures between graphs by decomposing the graphs into smaller parts for which a similarity measure can be computed. The assumption behind these kernel functions is that these similarity measures for simpler parts of the graphs are easier to infer than those for the whole graphs. Once the similarities between the simpler parts are computed, convolution operations may turn them into a valid kernel function for graphs [70, 176, 177].

The next and last family of graph kernels is the broader one and it is based on how similar the distribution of different types of substructures between two graphs are. A first approach is the random walk kernel, originally proposed in [58], where the idea is to measure the similarity of graphs by counting the number of random walks two graphs share in common. Other types of substructures are also proposed for the sake of graph similarity. For instance, shortest paths in graphs [18], subtrees [128], cycles [73, 74] or even simple subgraphs called *graphlets* [59, 152].

The novelty of the random walk idea is the fact that it has a straightforward computation in terms of the product graph between two graphs. The product graph of two graphs is the graph whose node set is the subset of the cartesian product of the two corresponding sets of nodes, composed by those pairs of nodes sharing the same label. Two nodes of the product graph are linked with an edge if the corresponding nodes were also linked in the original graphs. Clearly, a walk in the product graph of two graphs is a common walk between them. Thus, evaluating the number of common walks of two graphs turns into evaluating the number of walks of a single graph. If we let $A_\times$ be the adjacency matrix of the product graph $G_\times = (V_\times \;\; E_\times \;\; \mu_\times \;\; \nu_\times)$ of graphs g_1 and g_2, the random walk kernel can be computed by

$$\kappa(g_1 \;\; g_2) = \sum_{i \;\; j=1}^{|V_\times|} \left[\sum_{n=0}^{\infty} \lambda^n A_\times^n \right]_{ij} \tag{2.7}$$

since exponentiating the adjacency matrix A of any graph turns into a matrix A^n telling how many walks of length n are there between each pair of nodes of the graph. Some extensions and modifications has been made to this idea in order to increase the expressiveness and ease the computation of the random walk kernel. In [17], the kernel is extended to graphs with continuous labels and allows walks to be matched in terms of a similarity measure and not just when they share the exact same label sequence. Also, a probability-based approach is proposed in [89], where the probabilities of each label sequence to be generated by a random walk constitute the implicit features of the hidden feature space in which graphs are mapped. Outstanding works summarizing the main ideas and approaches of graphs kernels are [57, 172].

2.5 Graph Embeddings

As already stated in the introductory chapter of this thesis, graph embedding methodologies aim at defining a set of numerical features by which every graph in an arbitrary domain of graphs is assigned to a point in a vector space. Formally, a graph embedding is a mapping $\nu : \mathcal{G} \quad \mathbb{R}^n$ from the set of graphs into an n-dimensional Euclidean space. In this section, we review the literature concerning graph embedding methodologies in order to picture the general framework where our proposal belongs to.

The major work on graph embedding in vector spaces can be found in the field of spectral graph theory [31]. This field tries to characterize the structure of graphs in terms of the eigenvectors of matrices regarding their topology for instance, the adjacency or the Laplacian matrices . For instance, in [110], the authors extract embedding features from the leading eigenvectors eigenmodes of the adjacency matrix of graphs. In particular, unary and binary features are computed in terms of, for example, the leading eigenvalues or the eigenmode volumes and the entries of the intermode adjacency matrix or the intermode distances. Features are arranged in the form of a feature vectors and principal component analysis [82], independent component analysis [78] and multidimensionality scaling [37] are applied for the purpose of graph clustering and graph visualization. Spectral features from the Laplacian matrix are proposed in [179]. In particular, elementary symmetric polynomials are sampled on the eigenmodes of the Laplacian matrix and thus node permutation invariant features are obtained and arranged in the form of pattern vectors. A rather recent work is the one in [130]. The Ihara zeta function which characterizes graphs in terms of their prime cycles is used to extract informative features. In this work, the Ihara zeta function is proved to be the quasi characteristic polynomial of the adjacency matrix of the associated oriented line graph of a given graph, and the coefficients of such polynomial are used as features for the eventual embedding representation, which lead to remarkable clustering results. Other interesting spectral approaches can be found in [28, 95, 153]. Spectral approaches present some disadvantages. First of all, the eigen-decomposition of the related matrices is very sensitive to structural errors, leading to potential confusions in the embedding representations. On top of that, since these approaches only account for the structure of the graphs, they are restricted to work with node-unlabelled graphs.

Related to the spectral approaches, miscellaneous works are also available. For instance, using differential geometry concepts, in [137] the relationship between the Laplace-Beltrami operator and the graph Laplacian is used to embed the nodes of a graph into a Riemannian manifold. Inspired in Isomap [162], in [181] geodesic distances between nodes of a given tree are computed and multidimensionality scaling is applied in order to embed such nodes in a vector space where spectral methods are eventually used.

A rather different family of graph embedding approaches is the one based on (dis)similarity measures of graphs. Given a dissimilarity matrix between every pair of graphs in a given set of graphs, a naive approach for graph embedding would be to apply multidimensionality scaling to this matrix in order to obtain those vectors that best reproduce the original distance between them. Related to this, in [84], graph features are extracted out of a dissimilarity matrix obtained by comparisons of node signatures of graphs [83]. The most relevant work in this family is the one proposed in [134]. Based on the dissimilarity representation formalism [124, 125], a graph is embedded into a vector space where each component is regarding the (edit) distance to a given graph prototype. This methodology can be easily related to the graph edit distance since bounds between the corresponding distances are straightforward to derive. Moreover, it can handle arbitrary graphs since graph edit distance is able to do so and it has been reported remarkable results on graph classification and clustering problems. Nevertheless, the fact that this methodology depends on a computationally hard task such as the edit distance and that selecting the suitable set of prototypes is of major importance makes the methodology an arduous task to be validated.

A final family of approaches slightly related to the embedding methodology we propose in this thesis is that of computing frequencies of substructures as features for the eventual vectorial representation. For the problem of symbol recognition, [154, 155] propose to compute the frequency of certain substructures that are found in some training data. Also, in the field of chemoinfomatics, the works [79, 96] assign to every molecule a feature vector whose components are the frequencies of appearance of specific knowledge-dependent substructures in the graph. The embedding methodology proposed in this theses is rather more general and simpler than these ones since it does not impose any domain dependency and it is based on counting substructures of order 1 and 2 nodes and edges of the graphs based on the labelling information they store.

2.6 Discussion

Along this chapter we have visited several ways for the comparison of graph structures. Classically, graphs are compared by seeking an assignment of nodes of one graph to node of the other. Exact techniques aim at perfect match between nodes and edges of the involved graphs thus restricting their use for real world problems. At most, only distance measures are possible to define by the use of the maximum common subgraph and the minimum common supergraph. Error-tolerant graph matching methodologies allow node-to-node assignments to cope with real world problems by lightening the labelling and topological constraints and permitting non-isomorphic graphs to be deemed as highly similar. The main problem all these approaches exhibit is besides their restricted use as we have discussed their computational complexity, which is usually exponential in the number of nodes of the compared graphs.

Modern ways to compare graphs are kernels and embeddings. They try to evaluate how similar graphs are by either implicitly mapping graphs into hidden vector spaces where the scalar product can be regarded as a kernel function or by explicitly representing graphs as feature vectors where comparisons can be made by the use of all statistical-based analysis tools. We also observe some drawbacks in these approaches. As it happens in the graph matching methodologies, some of them are still of high computational complexity for instance, computing the edit distance to a set of graph prototype or they are restricted to be use on rather simple graphs eigen-decompositions do not care about labelling information of nodes . In the forthcoming chapters of this thesis we try to propose a graph embedding methodology that is capable to process as much types of graphs as possible, and to do it in a very efficient and natural way.

Chapter 3

Embedding of Discretely Attributed Graphs via Label Frequencies

In this chapter, we propose a set of graph features for discretely attributed graphs. They constitute the embedding components of the proposed methodology and will inspire the features for the continuously attributed case. We initially formalize their computation and then relate these features to a specific form of graph edit distance that will reveal how well the embedding methodology is able to reproduce the original distances between graphs in the graph domain if certain conditions are fulfilled. We also discuss the metric that should be use for these vectorial representations of graphs and, finally, give some hints on how the proposed features could be generalized to obtain more structurally global representations.

3.1 Embedding of Graphs via Label Frequencies

In this section, we start by outlining the main ideas underlying the proposed embedding methodology and give a formal description of the computation of the corresponding embedding features for discretely attributed graphs. In particular, we define the embedding of a graph into a vector space in terms of occurrences and co-occurrences of node labels.

3.1.1 Basic Procedure

Given a discretely attributed graph $g = (V\ E\ \mu)$ without edge attributes and with node alphabet $L_V = \quad l_1\ l_2 \qquad l_n$, a simple vectorial representation of g is, for instance, the one that takes, as each component of the vector, the number of times each node label appears in the graph, *i.e.*,

$$\mathbf{x}_g = (\#(l_1\ g)\ \#(l_2\ g) \qquad \#(l_n\ g)) \tag{3.1}$$

where $\#(l_i\ g)$ refers to the frequency that l_i happens to be the label of a node in graph g. For example, both graphs g_1 and g_2 in Fig. 3.1 have node alphabets $L_V = \quad A\ B\ C$ and they both have two labels A, one B and one C. Their respective vectorial representation in this form would be $\mathbf{x}_{g_1} = \mathbf{x}_{g_2} = (2\ 1\ 1)$.

In the simple example of Fig. 3.1, one gets the same vectorial representation from both graphs, although the graphs differ in their edge structure. Thus, more components should be added to the vectors in order to make this representation more discriminative. This can be achieved by considering not only the node labelling frequencies but also the frequencies

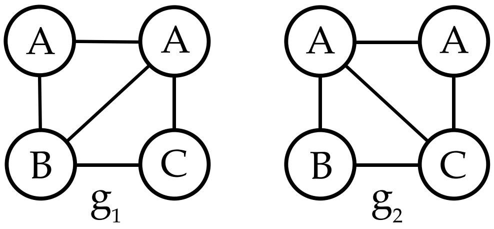


Figure 3.1: Two non-isomorphic graphs with the same vectorial representation counting node label appearances.

of the structural links between any two different nodes according to their corresponding attributes. More precisely, the vector representation (3.1) is enriched by $\mathcal{O}(n^2)$ components of the form

$$\#(l_i \leftrightarrow l_j, g), \tag{3.2}$$

counting how many edges between every pair of node labels occur in a given graph. With this information at hand, the vectors $\mathbf{x}_{g_1}$ and $\mathbf{x}_{g_2}$ in the example above will no longer be equal because the features $\#(A \leftrightarrow B, g)$ and $\#(A \leftrightarrow C, g)$ are, in fact, different. In the following order,

$$\begin{aligned}\mathbf{x}_g = (&\#(A,g), \#(B,g), \#(C,g),\\ &\#(A \leftrightarrow A, g), \#(A \leftrightarrow B, g), \#(A \leftrightarrow C, g),\\ &\#(B \leftrightarrow B, g), \#(B \leftrightarrow C, g), \#(C \leftrightarrow C, g)),\end{aligned} \tag{3.3}$$

the vectors $\mathbf{x}_{g_1}$ and $\mathbf{x}_{g_2}$ become

$$\begin{aligned}\mathbf{x}_{g_1} &= (2,1,1,1,2,1,0,1,0),\\ \mathbf{x}_{g_2} &= (2,1,1,1,1,2,0,1,0).\end{aligned}$$

Obviously, these two vectors are now a more proper representation of the graphs in Fig. 3.1.

With respect to the complexity of the construction of such a vector, it is worth mentioning the fact that all features that are being considered can be obtained by just visiting the nodes of the graph and their respective entries in the adjacency matrix. In fact, given a graph with n nodes and m edges, the complexity of constructing such a vectorial representation is only $\mathcal{O}(n+m)$. By contrast, other embedding methods look for paths or cycles in the graphs with the same label sequence [79, 96] and these procedures —and others based on finding substructures in the graphs— require for more expensive operations. In particular, the embedding based on (3.1) and (3.2) can be seen as a simple particular case of them, in which we look for paths of length 0 (labels of the nodes) and paths of length 1 (edges between nodes with specific labels).

3.1.2 Adding Edge Attributes

The example given above is built on graphs whose edges are unattributed. Let us now assume that graphs have discrete labels on their edges and that the edge alphabet is $L_E = \{a, b, \ldots, z\}$. In this situation, we should take into account those edges that are attributed with different labels but link nodes with the same attributes. In the example of Fig. 3.2, both graphs have only two nodes and one edge. The topology and their node labels are the same, but the edge labels differ.

Figure 3.2: Two edges with different attributes joining equally labelled nodes.

Consequently, the relation $\#(A \leftrightarrow B, g)$ between nodes with labels A and B should be redefined so as to distinguish between different labels of the corresponding edges. The edge on the left graph of Fig. 3.2 will count as an appearance of the relation between nodes with labels A and B, having edge label a, this is

$$\#\left([A \leftrightarrow B]_a, g\right),$$

while the edge on the right graph will count for the relation between nodes with label A and B, having edge label b, this is

$$\#\left([A \leftrightarrow B]_b, g\right).$$

3.1.3 Formal Definition

To formalize the distinction of edges with different labels, consider a set of graphs $\mathcal{G} = \{g_1, \ldots, g_N\}$, with $g_i = (V_i, E_i, \mu_i, \nu_i)$ being the ith graph in the set with labelling alphabet L_{V_i} for the nodes and L_{E_i} for the edges. We assume that all graphs in $\mathcal{G}$ have the same labelling alphabets, this is $L_{V_i} = L_{V_j}$ and $L_{E_i} = L_{E_j}$ for all $i, j \in \{1, \ldots, N\}$. We do not assume, however, that each node and edge label occurs in each graph. Let $L_V = \{\alpha_1, \ldots, \alpha_p\}$ and $L_E = \{\omega_1, \ldots, \omega_q\}$ be the common labelling alphabets.

For each graph $g = (V, E, \mu, \nu) \in \mathcal{G}$, we define p unary features measuring the number of times each label in L_V appears in the graph, this is

$$U_i = \#(\alpha_i, g) = |\{v \in V \mid \alpha_i = \mu(v)\}|. \tag{3.4}$$

For the edges we will distinguish two different scenarios. We will construct features in the case where edges of the graphs remain unattributed and features in the case of attributed edges. Binary features for edge unattributed graphs are defined by

$$\begin{aligned} B_{ij} &= \#(\alpha_i \leftrightarrow \alpha_j, g) \\ &= |\{e = (u, v) \in E \mid \alpha_i = \mu(u) \wedge \alpha_j = \mu(v)\}|, \end{aligned} \tag{3.5}$$

and binary features for edge attributed graphs are defined by

$$\begin{aligned} B_{ij}^k &= \#\left([\alpha_i \leftrightarrow \alpha_j]_{\omega_k}, g\right) \\ &= |\{e = (u, v) \in E \mid \alpha_i = \mu(u) \wedge \alpha_j = \mu(v) \wedge \omega_k = \nu(e)\}|. \end{aligned} \tag{3.6}$$

Note that, since graphs are undirected, in both cases the features are symmetric, this is, $B_{ij} = B_{ji}$ and $B_{ij}^k = B_{ji}^k$ for all $i, j \in \{1, \ldots, p\}$. We can then just consider half of them and always assume that $i \leq j$. This results in defining $\frac{1}{2} \cdot p \cdot (p+1)$ binary features for the edge unattributed case and $\frac{1}{2} \cdot q \cdot p \cdot (p+1)$ for the edge attributed one.

These two sets of edge features —based on whether we consider the attributes on edges or not— give rise to two different vectorial representations of a given graph from $\mathcal{G}$.

Definition 3.1 (Graph Embedding) *Given a graph $g \quad \mathcal{G}$, let ${}_n(g)$, ${}_e^1(g)$ and ${}_e^2(g)$ be the vectors*

$$ {}_n(g) = \Big(\; U_i \;\; {}_{1 \le i \le p} \Big) \tag{3.7} $$

$$ {}_e^1(g) = \Big(\; B_{ij} \;\; {}_{1 \le i \le j \le p} \Big) \tag{3.8} $$

$$ {}_e^2(g) = \left(\left\{ B_{ij}^k \right\}_{1 \le i \le j \le p}^{1 \le k \le q} \right) \tag{3.9} $$

where U_i, B_{ij} and B_{ij}^k are defined in Eqs. (3.4), (3.5) *and* (3.6), *respectively. For the edge unattributed case, the embedding of a graph $g \quad \mathcal{G}$ into a vector space is defined as the vector*

$$ {}_1(g) = [\quad {}_n(g) \quad {}_e^1(g)] \tag{3.10} $$

and for the edge attributed case as

$$ {}_2(g) = [\quad {}_n(g) \quad {}_e^2(g)] \tag{3.11} $$

Clearly, the embedding features we have defined in Eqs. (3.4), (3.5) and (3.6) are quite local regarding the topology of graphs. In particular, they describe the local structure of every graph in terms of how frequent a simple substructure   a node with a certain label or an edge with a given label between two given node labels   occurs in the graph. Bringing together all these pieces of local information in the form of a large feature vector will eventually lead to potentially good descriptors of the global structure of graphs.

3.1.4 Node-based *vs* Edge-based Features

In Definition 4.1 we gather together in the same vectorial representation both the node-based features (Eq. (3.4)) and the edge-based ones (Eqs.(3.5) or (3.6)). This is, both sets of features are assumed to have the same impact in the embedding representation. Such an assumption might be too hard in some cases. For instance   and as we will see later in some situations   , the edge topology of certain graphs is less relevant than the node labelling information, although the structural side of graphs still needs to be considered for a proper embedding characterization.

This situation can be tackled using a weighting factor that balances the strength of node-based features in the representation with respect to that of the edge-based ones. We thus introduce the weighted version of the embedding methodology in the following definition.

Definition 3.2 (Weighted Graph Embedding) *Given a graph $g \quad \mathcal{G}$, let ${}_n(g)$, ${}_e^1(g)$ and ${}_e^2(g)$ be defined as in Eqs.* (3.7), (3.8) *and* (3.9), *respectively. For the edge unattributed case, the weighted version of the embedding of a graph $g \quad \mathcal{G}$ into a vector space is defined as the vector*

$$ {}_1^{\alpha}(g) = [\, (1-\alpha) \cdot \quad {}_n(g) \quad \alpha \cdot \quad {}_e^1(g) \,] \tag{3.12} $$

and for the edge attributed case as

$$ {}_2^{\alpha}(g) = [\, (1-\alpha) \cdot \quad {}_n(g) \quad \alpha \cdot \quad {}_e^2(g) \,] \tag{3.13} $$

where $\alpha \quad [0 \;\; 1]$. This is, we concatenate the node-based features and the edge-based features with a weighting factor for each of the components.

With respect to the computation of the distance between two graphs g_1 and g_2 under this weighted version of the embedding, let us note that we could either use any vectorial distance between vectors ${}_1^{\alpha}(g_1)$ and ${}_1^{\alpha}(g_2)$ or, equivalently, we could first measure the same vectorial distance between the corresponding parts, ${}_n(g_1)$ with ${}_n(g_2)$ and ${}_e^1(g_1)$ with ${}_e^1(g_2)$ and finally weight the resulting distances[1]. Formally, given $d(\cdot \quad \cdot)$ any (component-

[1]Obviously, the same holds for φ_e^2.

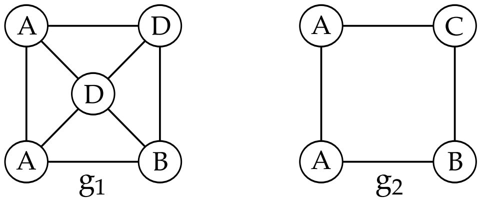


Figure 3.3: Two graphs for which the L_1 distance on the node-based features is equivalent to the edit distance when neither edge costs nor node substitutions are considered.

wise) vectorial distance, we are saying that

$$\begin{aligned} D(g_1\ \ g_2) &= d(\ \ ^{\alpha}_{1}(g_1)\ \ \ ^{\alpha}_{1}(g_2)) \\ &= (1-\alpha)\cdot d(\ \ _n(g_1)\ \ \ _n(g_2)) + \alpha \cdot d(\ \ ^{1}_{e}(g_1)\ \ \ ^{1}_{e}(g_2)) \end{aligned} \tag{3.14}$$

3.2 On the Correlation with Edit Distance

A desired property of any generic graph embedding scheme is that it should be able to reproduce the original distribution of patterns in the graph domain. In other words, distances between objects in the graph domain should be preserved in the corresponding embedding space. For instance, for the dissimilarity space embedding proposed in [134] it has been shown that the graph edit distance between two graphs is an upper bound of the Euclidean distance between the corresponding vectorial maps. Similarly, in [130], the Ihara coefficients have been experimentally shown to be a set of features with distances that correlate linearly with the edit distance.

In this section we experimentally check how distances between the embedding representations defined in the previous sections are able to reproduce the corresponding distances in the graph domain, in particular to those of graph edit distance. Our starting point is motivated by the following observation. Using an edit cost function in the edit distance computation that disregards both node substitutions and force instead node insertions and deletions and assumes null cost for edge operations, the resulting edit path turns into that of deleting nodes of the source graph that are not present in the target graph and inserting nodes of the target graph that are not present in the source graph. This particular edit path translates into checking the difference in the number of each node label between the two graphs. This is, the L_1 distance between vectors $_n(g_1)$ and $_n(g_2)$. We recall here that for two vectors $x\ \ y\ \ \ \mathbb{R}^n$, the L_1 distance is defined as

$$d_{L_1}(x\ \ y) = \sum_{i=1}^{n} \ x_i - y_i \tag{3.15}$$

See, for example, the graphs in Fig. 3.3. Assuming the set of node labels is $L_V = \ A\ \ B\ \ C\ \ D\ $, we have the following node-based embedding features,

- $_n(g_1) = (2\ \ 1\ \ 0\ \ 2)$,
- $_n(g_2) = (2\ \ 1\ \ 1\ \ 0)$,

and thus the corresponding L_1 distance is

- $d_{L_1}(\ \ _n(g_1)\ \ \ _n(g_2)) = 0 + 0 + 1 + 2 = 3$.

Note that this would be the case of computing the distance between the whole embedding representations (Eq. (3.14)) for the case of $\alpha = 0$.

For the case of the edit distance, with an edit cost function as the one described above, this is, assigning cost 1 to node insertions and deletions, a much higher cost to node substitutions and null cost to edge operations, we would have the following edit path and the corresponding edit distance

Edit operation	Cost
• Deletion of the central node with label D in g_1	1
• Deletion of its adjacent edges	0
• Deletion of the top-right node with label D in g_1	1
• Deletion of its adjacent edges	0
• Insertion of the top-right node with label C in g_2	1
• Insertion of its adjacent edges	0
– Total cost (edit distance)	3

We see how, under such an edit cost function, the edit distance is equivalent to the L_1 distance of the node-based embedding part of the whole embedding representation. In the following sections, we will see how the correlation of the embedding distances as a function of α behaves with respect to a general formulation of this way of computing the edit distance, where edge costs are also taken into account. We first formalize the edit cost function and then present some results.

3.2.1 Edit Cost Function

As stated above, we want now to work with an edit cost function that penalizes substitutions of nodes and edges with different labels, forcing the (sub)optimal path found by the edit distance computation to, first, delete the source node (or edge) and then insert the target node (or edge). Formally, deleting or inserting a given node (or edge) has cost 1, while substituting them has at least twice that cost if the corresponding labels are different. We assume null cost of substituting two nodes (or edges) with the same label. In order to weight the node operations against those on the edges usually one introduces a parameter $\rho \quad [0\ 1]$ and multiply the node costs by $1 - \rho$ and the edge ones by ρ. The resulting cost function is summarized as follows:

- Edit costs for nodes:

$$c(u \quad \epsilon) = c(\epsilon \quad v) = 1 - \rho$$

$$c(u \quad v) = \begin{cases} 0 & \text{if } \mu(u) = \mu(v) \\ 2 \cdot (1 - \rho) & \text{otherwise} \end{cases}$$

- Edit costs for edges:

$$c(e_1 \quad \epsilon) = c(\epsilon \quad e_2) = \rho$$

$$c(e_1 \quad e_2) = \begin{cases} 0 & \text{if } \nu(e_1) = \nu(e_2) \\ 2 \cdot \rho & \text{otherwise} \end{cases}$$

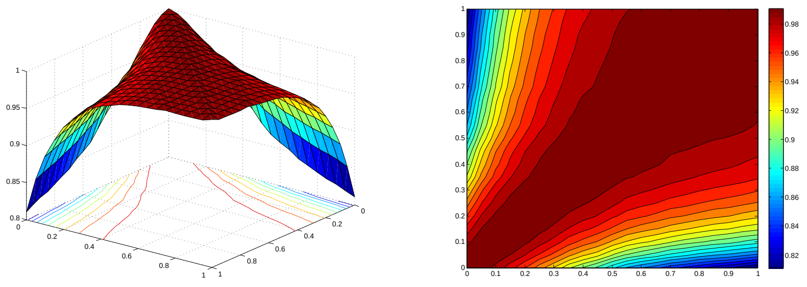
The motivational example introducing this section corresponds to the case where edge costs were disregarded, this is, when $\rho = 0$. In particular, we have observed that whenever edge costs are disregarded ($\rho = 0$) in the edit distance computation and when only the node-based embedding features are considered in the embedding distance computation ($\alpha = 0$) these two ways of computing the distance between graphs are equivalent. The following section will check for the correlation of these two way of computing distances between graphs as a function of ρ and α.

3.2.2 Distance Correlation

Given a pair of values (ρ α), we compute the set of all pairwise graph distances X_ρ and Y_α using parameter value ρ for the edit distances and parameter value α for the embedding distance. For a fairer analysis, since graph edit distance takes edge labels into account, we here use the $\frac{\alpha}{2}$ embedding representation (Eq. (3.13)), where edge labels are also considered. From these two sets of distance values, X_ρ and Y_α, we extract the correlation coefficient by

$$C_{(\rho\ \alpha)} = \frac{cov(X_\rho\ Y_\alpha)}{\varphi_{X_\rho}\varphi_{Y_\alpha}} \tag{3.16}$$

where $cov(X_\rho\ Y_\alpha)$ is the covariance between distributions X_ρ and Y_α, and φ_{X_ρ} and φ_{Y_α} are the corresponding standard deviations. In particular, for the pair $(\rho\ \alpha) = (0\ 0)$ we know the correlation coefficient will be equal to 1 since in this case both ways of computing distances are equivalent. We compute the correlation coefficient for all pairs $(\rho\ \alpha)\quad [0\ 1]^2$ and plot the corresponding 3D functions and correlation maps. The desired effect is shown in Fig. 3.4. Each axis would represent the weighting parameter of one of the ways of computing the distances. These artificial plots show the ideal situation where similar values of both parameters would make the two sets of distances to behave exactly the same (high correlation) and dissimilar values would make them behave differently (low correlation).

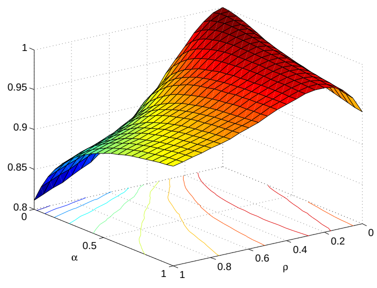


(a) (b)

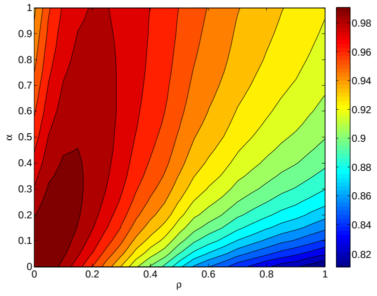
Figure 3.4: Correlation maps (3D plot and projection) in an ideal situation where the embedding distances would perfectly emulate the edit distances.

We use the four discretely attributed graph datasets described in Appendix A.1. In particular, the two object datasets, the ALOI and OBDK ones, and the two molecule collections, the AIDS and MUTAG ones. We show the distance correlation results in Fig. 3.5. The corresponding scatter plots between both sets of distance numbers are shown in Appendix B.

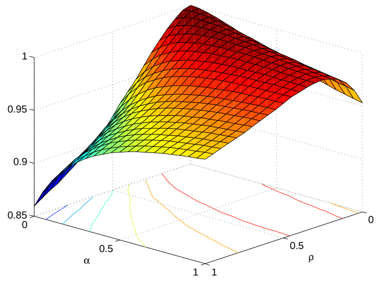
First of all, we note how values close to $(\rho\ \alpha) = (0\ 0)$ have, both in the object and molecule datasets, a high correlation coefficient. This confirms that the embedding features under the L_1 metric replicate the edit distances when node information is considered as more relevant than that of edges. If this is the case in the underlying application, we suggest



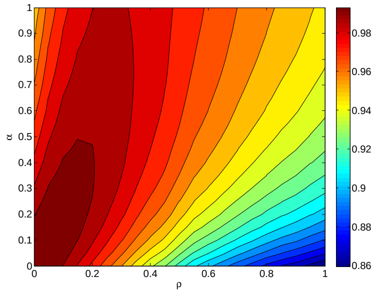
(a) ALOI, 3D



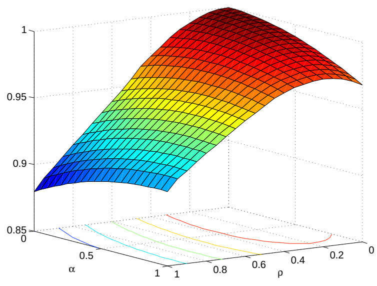
(b) ALOI, Correlation map



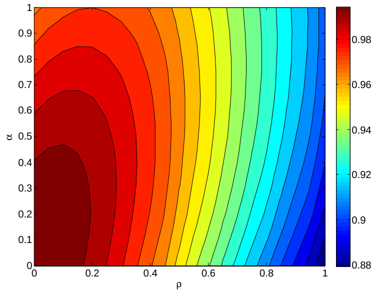
(c) ODBK, 3D



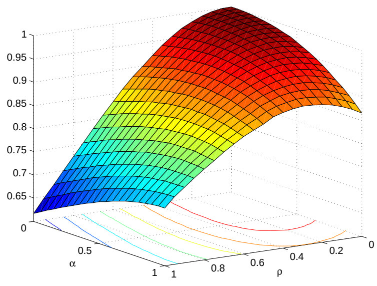
(d) ODBK, Correlation map



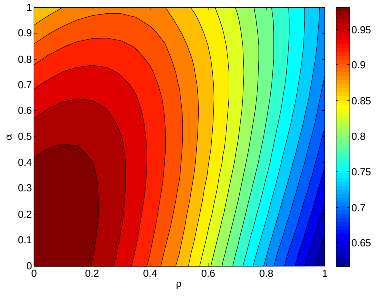
(e) AIDS, 3D



(f) AIDS, Correlation map



(g) MUTAG, 3D



(h) MUTAG, Correlation map

Figure 3.5: Correlation values as a function of the weighting parameters.

to use the attribute statistics based graph embedding rather than working with graph edit distance because, first, the relative graph distribution is well maintained and, second, the computational efficiency is much higher.

In Fig. 3.5 we can also observe the biased effect of the correlation values with respect to the ideal case (see Fig. 3.4). The explanation for this behavior is the fact that the edge-based embedding features still keep quite some information of the node labels. In particular, the co-occurrence of a certain pair of node labels at the end of an edge tells that these particular node labels do appear in the graph. Therefore, it is clear that considering edge-based features only, the embedding representation still keeps information about the node attributes. Indeed, the correlation of the embedding distances for $\alpha = 1$ is maximized by values ρ 0 2, suggesting that 80% of the node information in the graph domain is still considered for the embedding representation when only edge-based features are considered.

In Fig. 3.5, when both ρ and α tend to 1, low correlation values result. This might be explained by the fact that the edit distance computation looks for an edit path that completely disregards the information of the nodes. Thus, since edge-based embedding features still keep some of this information, the behavior of distances in both domains becomes different.

Also worth noting is the shape of the correlation regions. This shape is more ellipse-like in the molecule datasets than in the object datasets. This fact has an interpretation in terms of how much important is the actual structural configuration of graphs in each dataset. In the molecule datasets edge information is more salient than in the object datasets. The more weight we put on the edge-based features (α 1) the faster the correlation values for ρ 0 2 descend, which means that edge-based features are less correlated with the node information in the graph domain and thus we should put more attention on the edges.

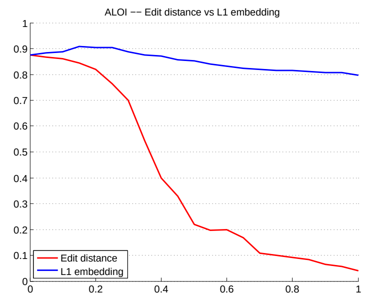
3.2.3 Classifiers Correlation

Another way to check how well the edit distances are emulated by the embedding features is to see how a distance based classifier performs. In particular, we use a kNN classifier with both ways of computing the distances between graphs and look for the differences in their performances. In particular, we compare the performance of both classifiers for all values $\rho = \alpha$. In Fig. 3.6 we show the corresponding classification curves on the validation sets for all values of ρ and α and the corresponding scatter plots of the accuracies of both classifiers. We also compute the correlation coefficient for these scatter plots.

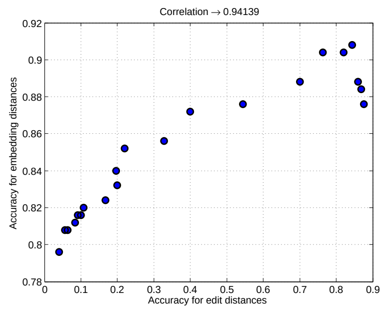
For the object datasets we observe how the edit distance tends to drop its performance as we give more weight to the edge costs. This confirms that in these datasets edge structure is less important than in the molecule s case, where this effect is not found. Correspondingly, the embedding curve in the object dataset does not drop that rapidly since, as we said before, edge-based features keep some information of the node characteristics of graphs.

With respect to the performance of the classifiers in the molecules dataset, we observe how both curves obtain their higher results for intermediate values of the parameters, thus confirming that here edge have higher importance than in the objects case. It is also interesting to see how the case $\alpha = 0$ gives worse result than that of $\alpha = 1$, but $\rho = 0$ better than $\rho = 1$. Again, this is possible because edge-based embedding features are quite informative with respect to the node labelling information while edit distances without any attention on node labels provide less meaningful edit paths.

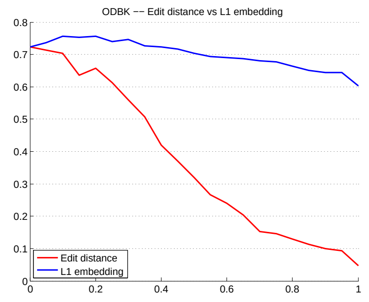
With regards to the correlation values of the classifiers, we observe how the object datasets obtain high correlation between classifiers while the molecules encounter the contrary situation, very low values. This result is explained by the same reason we have been discussing. The fact that the embedding features correlate with edit distance whenever the



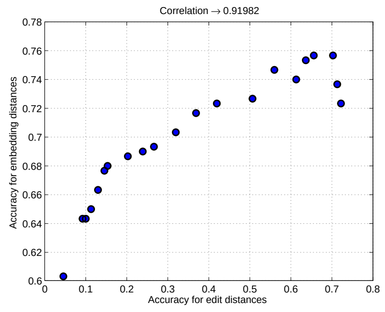
(a) ALOI, validation curves



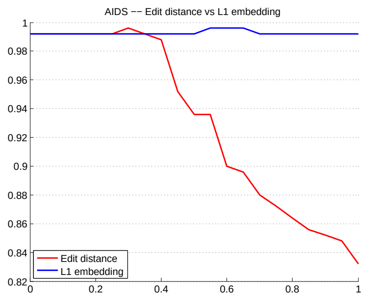
(b) ALOI, scatter plot



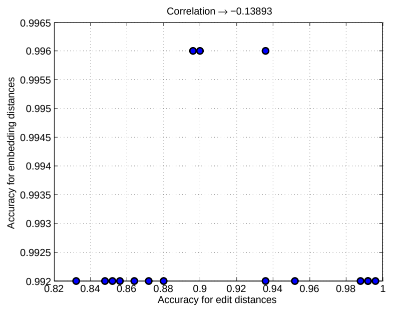
(c) ODBK, validation curves



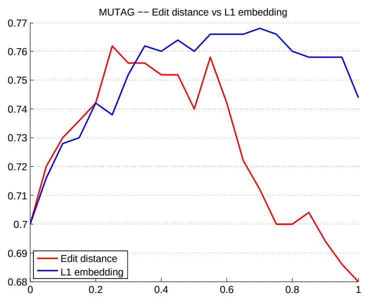
(d) ODBK, scatter plot



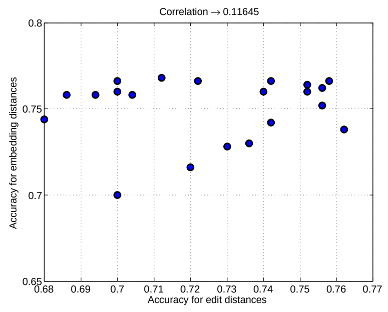
(e) AIDS, validation curves



(f) AIDS, scatter plot



(g) MUTAG, validation curves



(h) MUTAG, scatter plot

Figure 3.6: Classifier performances. Edit distance as a function of ρ. L_1 embedding as a function of α. Scatter plots of the accuracies of all pairs $\rho = \alpha$.

node information of graphs is actually relevant makes the classifiers perform in similar ways. On the other hand, the molecules need some more attention on the edge structure and here both the edit distance and the embedding features give different versions of the actual graph distributions in the corresponding domains.

In any case, we also observe how the embedding features are able to outperform the results of the edit distances in these particular classification setting. All in all, and beside their higher efficiency, whenever the node information is relevant for the problem under study, we suggest the use of the attribute statistics based features since the distances between them highly correlate with the edit distances and so a proper understanding of what goes on in the graph domain can be given. Although distances do not correlate whenever edge structure seems to be more relevant   and thus a similar understanding of the situation in the graph domain cannot be extracted   , the embedding representation also provides better classification results that those obtained with the edit distance, and therefore its use is also recommended.

3.2.4 Discussion

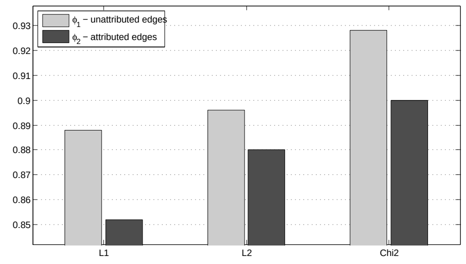
In this section we have shown how   under certain circumstances   the graph embedding scheme we propose for discretely attributed graphs is able to reproduce the same graph distribution that is obtained under the graph edit distance. In particular, given a set of graphs where the node labelling information is relevant for a proper characterization of them, we have seen how the L_1 distance between the proposed embedding features highly correlated with the edit distances. On the other hand, in other situations where the topological information is rather more relevant, the embedding distance do not correlate with those of the edit distance methodology, although the performance of a distance-based classifier is still better since the edge-based embedding feature still keep quite some information of the labelling of nodes in the graphs.

On top of these things, the validation curves of the kNN classifier presented in the previous section reveal an interesting property about the proposed embedding methodology. It is the fact that, in all cases, the optimal value of the parameter α is an intermediate value in the range [0 1]. It is closer to 0 for the object datasets (where node information is much more relevant) and closer to 1 for the molecule collections (where the structure of graphs demands more attention). From now on, in order to avoid the validation of this parameter for every particular case, we will assume the same weight for both the node and the edge-based features and keep on working with the unweighted version of the embedding (Definition 4.1). We should of course keep this in mind and investigate it in the future.

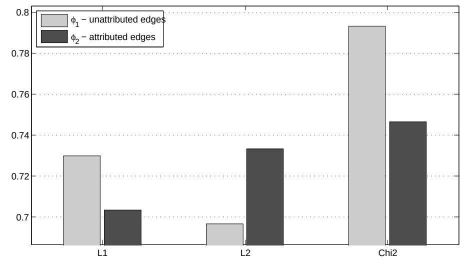
3.3 The Embedding Space under different Metrics

In the previous section we have shown that it exists a relation between graph edit distance and the L_1 distance on the proposed embedding features. In particular, we have shown that this vectorial metric on the embedding formulations is equivalent to the edit distance under some conditions. Nevertheless, placing ourselves only in the embedding space without thinking on the corresponding relation with the graph domain, this might not be the vectorial metric that extracts the best out of the features we are exploring. We here propose to investigate the effect of the vectorial metric used on the distribution of graphs under the embedding methodology.

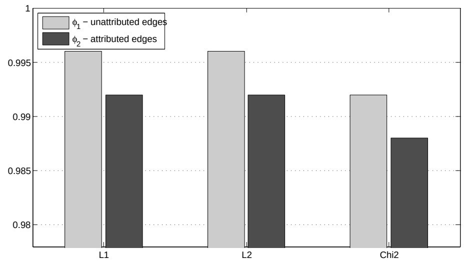
In particular, by using both versions of the unweighted embedding (the one that does not consider edge labels and the one that does), we will evaluate the effect of the distance



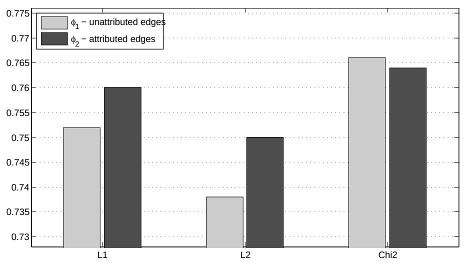
(a) ALOI



(b) ODBK



(c) AIDS



(d) MUTAG

Figure 3.7: Study of the use of different vectorial metrics in the embedding spaces. kNN classifier on the two different version of the embedding of discretely attributed graphs.

in the embedding features in terms of a distance-based classifier such as the kNN. Besides the L_1 distance used above (Eq. (3.15)), we also use the Euclidean distance due to its wide applicability in pattern recognition applications. The Euclidean or L_2 distance is defined by

$$d_{L_2}(x\ \ y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{3.17}$$

Apart from these two cases of the L_p metric, we will also test the embedding representations under a histogram-based distance. In fact, our representations are histograms over node and edge labels and thus it seems reasonable to use such kind of metrics. In particular we use the ϵ^2 distance since it has been broadly used in pattern recognition and computer vision problems [184] and, as we will see, it offers optimum results. It is defined as

$$d_{\omega^2}(x\ \ y) = \frac{1}{2}\sum_{i=1}^{n}\frac{(x_i - y_i)^2}{(x_i + y_i)} \tag{3.18}$$

As we just mentioned, using the embedded graphs under $_1$ and $_2$ representations, we use a kNN classifier with the three mentioned vectorial metrics on the validation sets of the same four discrete datasets used along this chapter so far. The results of this experiment are shown in Fig. 3.7.

As we have already announced, the ϵ^2 distance provides the best results in most of the cases. The fact that it is computing the sum of a relative proportion among components makes it, in general, more adequate for the comparison between these type of vectors. In

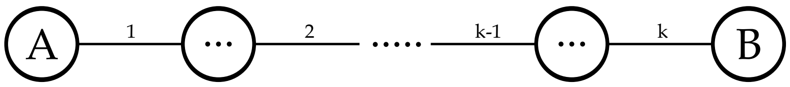


Figure 3.8: A path of length k between two nodes with labels A and B

particular, the ϵ^2 distance is discounting the effect of small changes for large values of the components. In other words, the difference between two components with values 1 and 10 is the same as between components with values 1001 and 1010 but it seems clear that, under these vectorial representations of graphs, the difference between the small values should have a higher impact than that for larger ones, since it is more expected that two graphs having values 1 and 10 in the same component are from different categories than those having the same absolute difference but much larger values. We thus may conclude that the embedding spaces are non-Euclidean and that the L_p distances will not offer, in general, as good results as a histogram-based metric. We will in fact encounter this situation all along this thesis: the ϵ^2 distance is, from the set of metrics that have been experimentally tested, the one that usually provides the best results, even with continuously attributed graphs or in clustering scenarios.

As a collateral effect of the previous experiment, we also discover an interesting thing. No matter the metrics used for comparing the vectorial representations of graphs, the version where edge attributes are disregarded, $_1$, usually offers better results than the one where edge attributes are certainly taken into account. A plausible explanation for this fact is that the $_2$ representation breaks the topological structure of graphs into many small pieces that eventually are not enough to represent the global structure of graphs. On its side, the $_1$ embedding does not consider edge labels, but it is less sparse than the other one and it can thus provide a more strong representation of the graph structures. From now on, unless otherwise indicated, we will discard edge attributes.

3.4 Higher order edge-based features

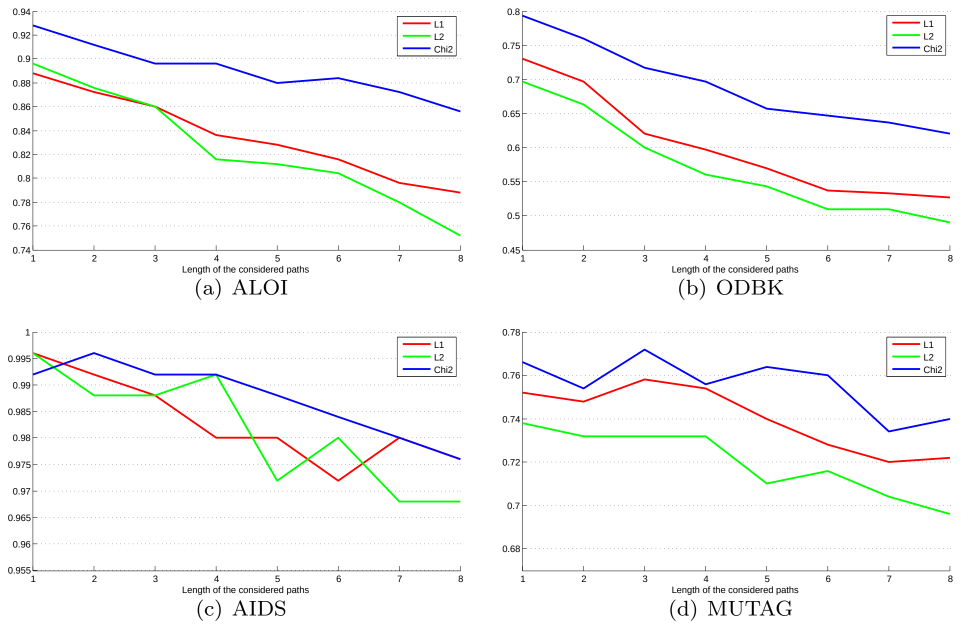
The edge-based features defined, used and discussed so far are rather local concerning the graph topology. By gathering them together we obtain a rich representation able to cope with the structure of graphs. In this section, we are concerned on finding more structurally global features that will eventually provide better vectorial representations of graphs. In particular we discuss how the edge-based features can by understood as the simple particular case of finding paths of length 1 in the graphs, and thus we propose to generalize the edge-based embedding features to be occurrence of paths of length k, for $k \geq 1$.

3.4.1 k-length Paths

The unweighted version of the adjacency matrix of a graph g is the matrix that tells whether there exists an edge between a pair of nodes. This is, the matrix $M \quad 0\ 1^{n\times n}$ is defined in the following way:

$$M_{ij} = \begin{cases} 1 & \text{if } (v_i \ \ v_j) \quad E \\ 0 & \text{otherwise.} \end{cases} \tag{3.19}$$

Another point of view of this matrix is that it accounts for the number of paths of length 1 between every two nodes. Under this terminology, it is clear that the edge-based features for the edge unattributed case defined in the previous sections are the number of paths of length



(a) ALOI (b) ODBK (c) AIDS (d) MUTAG

Figure 3.9: kNN results on the sets of features for discretely attributed graphs as a function of the length of paths considered in graphs.

1 that exist between two nodes with two specific labels. We now propose to use features that account for longer paths, this is, to count the number of paths of length k that exist between two nodes with labels, for example, A and B. Given a graph $g = (V \ E)$ with the nodes set $V = \ v_1 \qquad v_n$, a path of length k in g is a set of $k+1$ consecutive nodes linked by edges. In Fig. 3.8 we show an example of a path of length k between two nodes with labels A and B.

An interesting property of the adjacency matrix is that, when it is exponentiated, it generalizes the fact that it accounts for paths of length 1 and so M^k is the matrix regarding the number of paths of length k that exists between every two nodes. We here use these new form of the edge-based features. The computation of such a set of features is straightforward. At least, as simple as it was in the previous case. The 1-length features (Eq. (3.5)) are built by summing up those entries of the adjacency matrix M that correspond to the connection between points with two certain labels. In this case, we use the matrix M^k instead.

Once these features are computed, the embedding representation we use is analogous to the one proposed above. We concatenate the unary features counting occurrences of node labels with the higher order edge-based features.

3.4.2 Experimentation

To evaluate the impact of the length of features considered in these higher order representations, we apply a kNN classifiers on the validation sets of the discretely attributed graph datasets. We use different values for the parameter k. On top of that, and following the same methodology as in Section 3.3, we use the same three vectorial metrics to assess the performance of the classifier, this is, we use the L_1, L_2 and ϵ^{2} distances. The results are shown in Fig. 3.9.

As already announced before, the ϵ^2 distance is the one that performs best with respect to the other considered distances. No matter the dataset that is used or even the length of the paths considered for the features, the ϵ^2 curves are in most cases above the other curves.

A deeper analysis of the results can be made by differentiating the object from the molecule datasets. In the case of the object datasets, it turns out that the 1-length path features always provide better results than longer paths. This is not happening for the molecules datasets, where 2-length (AIDS) or 3-length (MUTAG) features offer better results than the original ones. This can be explained by the fact that object datasets are more populated with edges than molecule datasets (see Tables A.1 and A.2) and thus computing longer path features generates much more noise in the object datasets than in the molecule cases.

In fact, these sets of features we are considering in this section still need to be refined quite in depth. Several situations might occur where, for instance, an edge is traversed several times in the same path and thus it creates redundant information that weakens the final representation. Self-loops, or going back and forth using the same nodes and edges should be avoided in order to properly represent graphs under this setting.

Results are not significantly better than the simple 1-length case. Nevertheless, we have shown a way that is able to create a population of vectors for each graph that might be used to represent both local (shorter paths) and global (longer paths) structural information. We will no longer use this representation except in Chapter 6, where such a diversity of representations will be exploited in order to build several classifiers of graphs and combine them in a multiple classifiers framework in order to boost the performance of the classification strategy.

Chapter 4

Statistics on Node Label Representatives for the Embedding of Continuously Attributed Graphs

The transition of the embedding methodology from discretely attributed graphs to continuously attributed graphs is covered in this chapter. Continuous attributes of nodes are discretized by assigning to a set of representatives. The embedding is thus defined by statistics on these representatives. The intuition behind this procedure is to assume a topological model for each graph category. Then, observed graphs would be variations of these models. Therefore, we first undo possible deformations from such models by the assignment step and then apply the embedding procedure described in the previous chapter for discretely attributed graphs.

We initially define the embedding for the continuous case. We explore two different ways of performing the transition to the discrete domain, namely, the hard and the fuzzy versions of the embedding. Then, we evaluate the suitability of different metrics in the vector space for comparing the vector representations of graphs. We finally discuss the effect of the set of representatives in a classification scenario.

4.1 From discrete to continuous: node attribute representatives

In this chapter, we consider the case of graphs where the set of node labels is $L_V = \mathbb{R}^d$ and edges remain unattributed. The proposed methodology is based on the idea that the more similar two node labels are the more likely they should be considered the same and thus the more they should count for the same feature. In other words, the nodes of the graphs can be assigned to some *representatives* of the node labels in terms of the Euclidean distance, and then only statistics of these representatives are counted as features in the vectorial representation of the graph. As representatives of a set we choose those points not necessarily in the set that best represent the whole set, just as the classical cluster analysis techniques do.

Based on this idea, we present two versions of the node-based and edge-based features for continuously attributed graphs. The first one is called the *hard* version and is directly related to the discretely attributed graph versions. In some sense, continuous node labels are just substituted with a discrete value. The second version is more able to cope with deformations in the graph models and is called *fuzzy* version. Instead of assigning each node label to a discrete value, nodes are distributed to all representatives with a certain degree of probability for each of them.

4.1.1 Hard version

We now formally describe the new version of the graph features and the corresponding embedding representation. Suppose we are given a set of N graphs $\mathcal{G} = \quad g_1 \qquad g_N \quad$. Each of these graphs has nodes with continuous attributes. This is, for all $i \qquad 1 \qquad N \quad$, the set of nodes attributes is $L_{V_i} = \mathbb{R}^d$. Let $\mathcal{P} \wedge \mathbb{R}^d$ be the set of all node labels in all the graphs of $\mathcal{G}$. Furthermore, let $\mathcal{W} = \quad w_1 \qquad w_n \quad$ be a set of n representatives of all vectors in $\mathcal{P}$. In Section 4.2 we describe different approaches for defining such sets of vectors. Elements in $\mathcal{W}$ do not necessarily belong to $\mathcal{P}$. The *node to representative* map is a function assigning every node of a given graph to the vector in the representative set which is, among all elements in the set of representatives, the closest to the label of the node. Formally, for every graph, we have

$$\begin{aligned} \lambda_h : V - \quad & \mathcal{W} \\ v \ - \quad & \lambda_h(v) = \operatorname*{argmin}_{w_i \in \mathcal{W}} \quad \mu(v) - w_i \quad {}_2 \end{aligned} \tag{4.1}$$

where $\quad \cdot \quad {}_2$ stands for the Euclidean metric.

Using this function, we can redefine the concept of frequency of a specific label used in the previous chapter by checking how many nodes have been assigned to this specific label. This is, given a graph $g = (V \ E \ \mu)$ and a representative set $\mathcal{W}$, the frequency of a representative $w_i \quad \mathcal{W}$ is

$$U_i = \#(w_i \ g) = \quad v \quad V \ \ w_i = \lambda_h(v) \tag{4.2}$$

Similarly, the frequency of a specific relation between two representatives is defined as

$$\begin{aligned} B_{ij} &= \#(w_i \qquad w_j \ g) \\ &= \quad (u \ v) \quad E \ \ w_i = \lambda_h(u) \qquad w_j = \lambda_h(v) \end{aligned} \tag{4.3}$$

It is important to notice the symmetry of the features $\#(w_i \qquad w_j \ g)$ as long as the involved graph g is undirected. Because of this symmetry, we will consider each such feature only once instead of counting both $\#(w_i \qquad w_j \ g)$ and $\#(w_j \qquad w_i \ g)$.

Definition 4.1 (Graph Embedding) *Let $\mathcal{G}$ be a set of graphs as defined above. Given a set of node representatives $\mathcal{W} = \quad w_1 \qquad w_n \quad$, we define the embedding of a graph g into a vector space as the vector*

$$_{\mathcal{W}}(g) = (U_1 \qquad U_n \ B_{11} \qquad B_{ij} \qquad B_{nn}) \tag{4.4}$$

where $1 \leq i \leq j \leq n$, $U_i = \#(w_i \ g)$ and $B_{ij} = \#(w_i \qquad w_j \ g)$.

Let us illustrate this definition with an example. In Fig. 4.1, all the nodes of the four depicted graphs in the top row have attribute values that are close to either one of the following points: $\mathcal{W} = \quad w_1 \ w_2 \ w_3 \ w_4 \quad = \quad (0 \ 0) \ (0 \ 1) \ (1 \ 0) \ (1 \ 1) \quad$. These four points are a natural set of representatives for the whole set of node labels of the four different graphs. By using this set of representatives, we would have four different node labels and, for example,

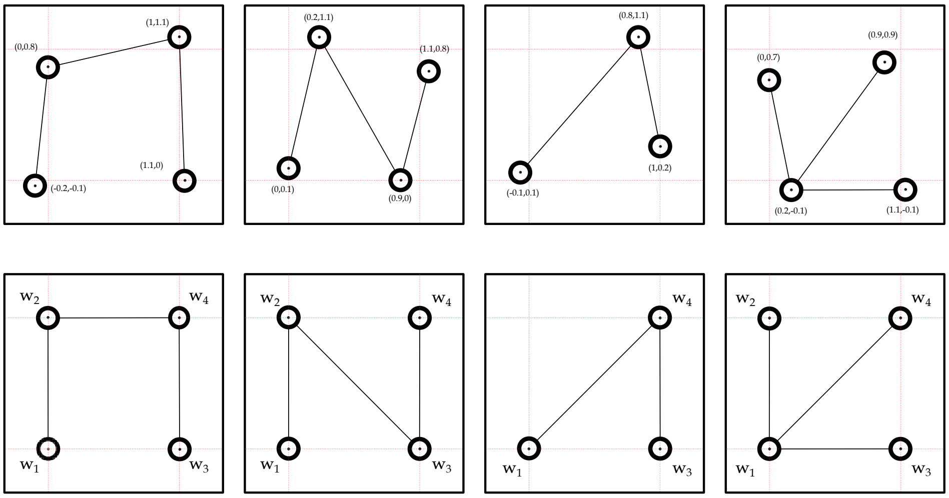


Figure 4.1: Four graphs whose node attributes from $\mathbb{R}^2$ are all around one of the four representatives $\{(0,0),(0,1),(1,0),(1,1)\}$, and their corresponding discrete versions after assigning each node to an element in the set of representatives.

the node of the leftmost graph with label $(0, 0.8)$, will count as an appearance of its closest representative $w_2 = (0,1)$. Regarding the relations between labels, the B_{ij} features, the edge in the second graph connecting the nodes with labels $(0.2, 1.1)$ and $(0.9, 0)$, would count as an appearance of the relation between the representatives $w_2 = (0,1)$ and $w_3 = (1,0)$, which are the corresponding nearest representatives to the node attributes. In particular, if g_1 is the leftmost graph and g_4 the rightmost one, given $\mathcal{W}$ their embedding representations assuming the order $\varphi_{\mathcal{W}}(g) = (U_1, U_2, U_3, U_4, B_{11}, B_{12}, B_{13}, B_{14}, B_{22}, B_{23}, B_{24}, B_{33}, B_{34}, B_{44})$ are

- $\varphi_{\mathcal{W}}(g_1) = (1,1,1,1,0,1,0,0,0,0,1,0,1,0)$
- $\varphi_{\mathcal{W}}(g_2) = (1,1,1,1,0,1,0,0,0,1,0,0,1,0)$
- $\varphi_{\mathcal{W}}(g_3) = (1,0,1,1,0,0,0,1,0,0,0,0,1,0)$
- $\varphi_{\mathcal{W}}(g_4) = (1,1,1,1,0,1,1,1,0,0,0,0,0,0)$

4.1.2 The fuzzy version

In noisy situations, it might be the case that a node label is between two representatives, and there is no clear rule telling us to which representative the node should be assigned. Also, a node label could be far away from all representatives such that no element of $\mathcal{W}$ is actually a proper representative of the node. These are typical situations where a *soft* rather than a *hard* assignment can be beneficial. *Fuzzy* clustering assigns to every node a certain degree of belongingness to every cluster, rather than saying that a node is assigned to just one representative [45, 166]. In this work, we also address this situation and propose a fuzzy version of the graph vectorization procedure.

While Definition 4.1 is still our basic embedding approach, the features U_k and U_{ij} will be redefined. Eq. (4.1) is no longer usable for the assignment of nodes to representatives and every node will be given a certain degree of assignment to every representative. Formally, we define the function

$$\begin{aligned}\lambda_s : V - \quad & S_n^+ \wedge \mathbb{R}^n \\ v \ - \quad & \lambda_s(v) = (p_1(v) \qquad p_n(v))\end{aligned} \tag{4.5}$$

where $p_i(v) = P(v \sim w_i)$ is the probability of the node v being assigned to the representative w_i and S_n^+ is the positive orthant of the L_1-hypersphere in $\mathbb{R}^n$. In other words, we put these degrees of belongingness under a probability framework in order to ease the methodology, while requiring that $p_i(v) \geq 0$ and $\sum_{i=1}^n p_i(v) = 1$.

In this situation, the appearance frequency of a certain representative, Eq. (4.2), has to be reformulated in terms of the probabilities of belongingness for all nodes in the graph. This leads to

$$U_i = \#(w_i \ \ g) = \sum_{v \in V} p_i(v) \tag{4.6}$$

When there is an edge between two nodes in the graph, $(u \ \ v) \qquad E$, the fact that nodes are assigned to representatives according to Eq. (4.5), makes it unclear how one should compute the features U_{ij}. Assume we have available the corresponding fuzzy assignment representations of the source and the target nodes:

$$\begin{aligned}\lambda_s(u) &= (p_1(u) \qquad p_n(u)) \\ \lambda_s(v) &= (p_1(v) \qquad p_n(v))\end{aligned}$$

From these two vectors of probabilities we need to find out how much the edge $(u \ \ v)$ is contributing to the relation between two representative points. We handle this situation using two different approaches.

1) The first one is the naive and conservative approach in which the edge only contributes to the relations of the representatives with maximum probability. This is, if w_t is the representative to which the node u has maximum probability of belongingness and w_s the corresponding one for the node v, then the edge $(u \ \ v)$ will only count as a relevant relation between the representatives w_t and w_s. Formally,

$$B_{ij} = \#(w_i \qquad w_j \ \ g) = \sum_{(u \ \ v) \in E} \delta_{ij}(u \ \ v) \tag{4.7}$$

where

$$\delta_{ij}(u \ \ v) = \begin{cases} 1 & \text{if } w_i = \underset{w_k \in \mathcal{W}}{\operatorname{argmax}} \ p_k(u) \\ & \quad \text{and } w_j = \underset{w_k \in \mathcal{W}}{\operatorname{argmax}} \ p_k(v) \\ 0 & \text{otherwise.} \end{cases} \tag{4.8}$$

This procedure for constructing the B_{ij} features will be referred to the *max* assignment, since we only take into account the edge relation from the most (maximum) probable representative of one node to the most (maximum) probable representative of the other one.

2) The second approach will keep considering all possible assignments from one node to the other, and thus, and edge $(u \ \ v) \qquad E$ will contribute to all relations between any two representatives. In particular, we call this approach the *all* assignment method since all probabilities are taken into account. In particular, we define

$$\begin{aligned}B_{ij} &= \#(w_i \qquad w_j \ \ g) \\ &= \sum_{(u \ \ v) \in E} p_i(u)p_j(v) + p_j(u)p_i(v)\end{aligned} \tag{4.9}$$

The intuition behind (4.9) is based on walks of length 1 on the graph. The edge $(u\ v)$ E of a graph g will contribute to the relation w_i w_j the amount of probability of travelling from the part of u assigned to w_i to the part of v assigned to w_j, this is, $p_i(u)p_j(v)$. Then, since we work with undirected graphs, we should also consider the path back and aggregate the probability of travelling from the part of v which is assigned to w_i to the part of u assigned to w_j, *i.e.*, $p_j(u)p_i(v)$.

4.1.3 Discussion

As already stated above, the main idea behind this methodology is that similar nodes will count for the same feature with similar weights. We assume there is an intrinsic topological model for every category in a given database of graphs and by this approach we believe that we can extrapolate such model, via undoing possible deformations on the different instances of a certain class.

This way, it seems reasonable that the methodology is going to perform well on graphs whose continuous attributes of the nodes describe positions on the plane. Indeed, as we will see and discuss later (Section 4.4), the results confirm this hypothesis since we work on graphs of this kind and the results are quite satisfactory. Moreover, this point is reinforced by the fact that in those cases where we have severe deformations, the proposed methodology does not perform properly, and this is because such high degree of distortion makes it impossible to discover the intrinsic model for each class. On top of this, we also believe that the proposed methodology will properly discriminate among graphs with higher dimensional node attributes, and not only on those cases where these attributes are representing spatial positions of nodes. To evaluate this scenario and confirm our hypothesis, in Section 7.3 we study graphs whose node attributes are RGB values of segmented regions of object images, for which we reformulate the embedding strategy in terms of a domain-dependent selection of representatives.

Finally, let us mention the fact that for the continuously attributed graphs we have not proposed features for graphs with edge attributes. The discretization process makes it uncomfortable to define such kind of features. A possible approach to this situation could be also to discretize the edge labels and then perform similarly as we have done for the discrete attributed graphs where edges with different labels are differentiated in separated features. Although feasible, we understand these features would experience a similar behavior as the corresponding features for discrete attributed graphs. The structural information would be split into too many small pieces and will eventually lead to sparse and, thus weak, representations. We leave such a study for future research.

4.2 Selection of representatives

Selecting a proper set of node attribute representatives for the set of all graphs in a given dataset is a crucial issue in the proposed embedding methodology. In this section we explore four different approaches of classical cluster analysis in order to select representatives. Two of the considered methods are hard methods and two are soft (fuzzy) ones. For the rest of this section, let $\mathcal{P} \wedge \mathbb{R}^d$ be a set of m node labels from which we want to extract a set $\mathcal{W}$ of n representatives.

4.2.1 Spanning prototypes

The first representative (hard) selector considered in this work is an approach that tries to find points as much uniformly distributed as possible within the whole range of points at hand [125]. The algorithm starts by selecting the point that minimizes the sum of distances to all the other points (called the median vector) and then, it keeps adding points to the representative set by iteratively selecting the point which is furthest away   the one maximizing the sum of distances   to the already selected set of representatives. The algorithm stops when a predefined number of points is obtained.

The algorithm is capable to find a set of representative vectors $\mathcal{W}$ that *span* the whole range of points in $\mathcal{P}$ as uniformly as possible.

4.2.2 kMeans algorithm

The second hard selector for the set of representatives is the very well-known kMeans clustering algorithm [45]. Briefly, this algorithm starts by initializing the set of cluster centres $\mathcal{W}$ with some random points and then assign each point in $\mathcal{P}$ to its closest centre. The set $\mathcal{W}$ is updated as the mean of all points assigned to the same cluster. These steps run iteratively until there are no changes in the set $\mathcal{W}$.

A common problem one encounters using this algorithm is the uncertainty in the results due to the random initialization of the set $\mathcal{W}$. A possible solution is to repeat the experiment a certain number of times and finally average the results. In our case, though, in order to avoid such an uncertainty, the kMeans algorithm is always deterministically initialized using the spanning prototypes described in the previous section.

4.2.3 Fuzzy kMeans

The next approach for the selection of representatives is the Fuzzy kMeans algorithm [45]. Its main idea is to assign to every point $x \quad \mathcal{P}$ a degree of belongingness to each cluster center in $\mathcal{W}$, $p_i(x)$. This degree of belongingness is set to be inversely proportional to the distance between x and the cluster center. This leads to

$$p_i(x) = \alpha \cdot \left(\frac{1}{x - w_i \quad {}_2} \right)^s \tag{4.10}$$

where α is a constant assuring that $\sum_{i=1}^{n} p_i(x) = 1$ and s is a parameter that controls the amount of *fuzzyness* given to the assignment. The larger s is, the more weight is given to points close to the centres and thus the more similar is this method to its respective hard version. In our experiments we use $s = 2$ since it offers a fair compromise between the hard and the fuzzy versions of kMeans.

The algorithm is basically the same as kMeans, although the assignment from points to clusters is made by means of Eq. (4.10), and the update of the clusters at each iteration is done by a weighted mean of all points (weighted by their degree of belongingness to each specific cluster). To avoid uncertainty, we also initialize the algorithm using the spanning prototypes.

4.2.4 Mixture of Gaussians

For the last representative set selector, we make use of a Gaussian Mixture Model (GMM) or Mixtures of Gaussians [45]. It is an important probabilistic framework in which all the data are assumed to be generated by a model in which several probability distributions are involved. Technically, the set of points in $\mathcal{P}$ is assumed to be generated by

$$f(x) = \sum_{i=1}^{n} \pi_i \mathcal{N}(x \quad \mu_i \quad \Sigma_i) \tag{4.11}$$

which is a linear mixture of n Gaussian densities $\mathcal{N}(\cdot \;\; \mu_k \;\; \Sigma_k)$, where μ_k is the mean vector and Σ_k the covariance matrix. The parameters π_k are usually called mixing coefficients and can be understood as probability weights for the mixture components. The estimation of the involved parameters (mixing coefficients, means and covariances) is usually done by maximization of the log-likelihood of the mixture with respect to the parameters. In our experiments, this estimation has been carried out by means of the Expectation-Maximization (EM) algorithm [40, 61].

Note that in this scenario the set of representatives $\mathcal{W}$ is no longer a set of points but a set of probability (Gaussian) densities. We can thus assign the degree of membership of a point x to a certain representative $w_i = \mathcal{N}(\cdot \;\; \mu_i \;\; \Sigma_i)$ by the probability

$$p_i(x) = \beta \cdot \mathcal{N}(x \;\; \mu_i \;\; \Sigma_i) \tag{4.12}$$

where β will be again a constant making $\sum_{i=1}^{n} p_i(x) = 1$.

For the initialization of the parameters in the EM algorithm we use the kMeans results as described above. For a Gaussian $\mathcal{N}(\cdot \;\; \mu_i \;\; \Sigma_i)$, its mean μ_i is initialized as the i-th kMeans cluster centre, the mixing coefficient π_i is the amount of point mass assigned to this cluster and the covariance Σ_i is the covariance matrix of the data that were assigned to the i-th kMeans cluster.

4.2.5 Summary

We here briefly summarize the different configurations of the presented graph embedding methodology. In the hard case of the representative set construction we have two configurations corresponding to the two methods for selecting representatives:

- Spanning prototypes (hard assignment)
- kMeans (hard assignment)

For the *soft* version of the embedding we have two ways of constructing the representative set and thus to assign nodes to representatives , and also two ways of constructing the edge-based features B_{ij} . This leads to another four embedding configurations:

- Fuzzy kMeans + Soft *max* edge assignment
- Fuzzy kMeans + Soft *all* edge assignment
- Mixture of Gaussians + Soft *max* edge assignment
- Mixture of Gaussians + Soft *all* edge assignment

In the experimental part of this chapter (see Section 4.4), we will test these six embedding configurations on different datasets to get more insight into the strength of the proposed methodology.

These formulations of the proposed embedding methodologies are generic enough to deal with graphs whose node attributes are from $\mathbb{R}^d$. In any case, there might be some situations where it is preferable to put attention on the semantics of the node labels and try to select representatives so that this semantics is exploited. In particular, in the experimental part of this thesis (Chapter 7), we will work with graphs whose node attributes are RGB values of segmented regions of object images and thus the selection of representatives is going to be based on a color-dependent strategy.

4.3 The embedding space under different metrics

As in the case of discretely attributed graphs, we can potentially use several metrics to compare the vectors obtained after embedding (Section 3.3). Thus, in this section we analyse which is the vectorial metric that is capable to extract the best out of the different embedding methodologies that we have proposed. We use the same three vectorial distances as in the previous chapter. In particular, the L_1, L_2 and $\epsilon\ ^2$ distances,

$$d_{L_1}(x\ \ y) = \sum_{i=1}^{n} \ x_i - y_i \tag{4.13}$$

$$d_{L_2}(x\ \ y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{4.14}$$

$$d_{\omega^2}(x\ \ y) = \frac{1}{2}\sum_{i=1}^{n}\frac{(x_i - y_i)^2}{(x_i + y_i)} \tag{4.15}$$

Let us make here a brief comment about the relation of graph edit distance with the L_1 distances on the embedded continuously attributed graphs. In this case we have not perform analogous experiments to those in Section 3.2 but we can say that, although some correlation might still exists, the assignment of node to representatives will create quite some confusion and thus the relations will not be that clear. A deeper analysis of this scenario is left for future investigation.

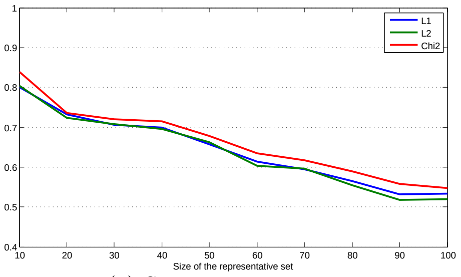
Back to the metrics, the criterion for their optimality will be again a kNN classifier on the validation sets of different databases. We pick a simple classifier like kNN for two reasons: the first one is that it really does not need a complex parameter tuning step (besides the value of the number of neighbours), and the second one is that this classifier already gives a good and reliable measure of how the vectors are class-wise distributed in the input space. In this section we use the Digits dataset described in Appendix A, Section A.2.1.

For the six configurations of the embedding methodology summarized in Section 4.2.5, we build up the vectorial representations of graphs for sets of representatives of sizes from 10 to 100 elements, in steps of 10. In Fig. 4.2 we show the results of the classifier as a function of the size of the set of representatives for the Digits dataset. Moreover, in Appendix C, the corresponding figures for the rest of the datasets described in Section A.2.1 are shown.

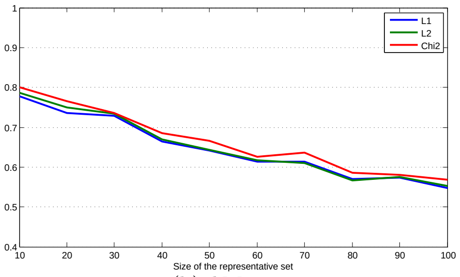
Although in the hard configurations the difference is not that clear, the $\epsilon\ ^2$ distance is usually the one that obtains a better result with respect to the other vectorial distances. Specially in the soft configurations with the *all* edge assignment, which are as deeply discussed in the next section the configurations that offer the best results (note the ranges of the figures are all the same). This behavior is generally observed for the other datasets we have considered (see figures in Appendix C), and thus, unless stated otherwise, we will be working with the $\epsilon\ ^2$ distance from now on.

4.4 The effect of the representative set

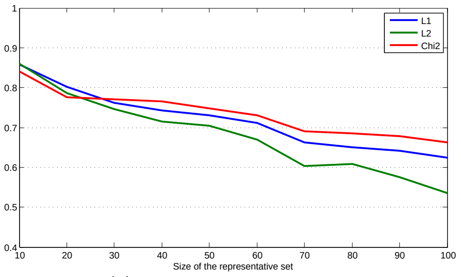
As it can already be seen in the figures describing the distance impact on the different embedding representations, the results clearly depend on both the way the sets of representatives are constructed and how many elements are considered in these sets. In this section, we validate both things.



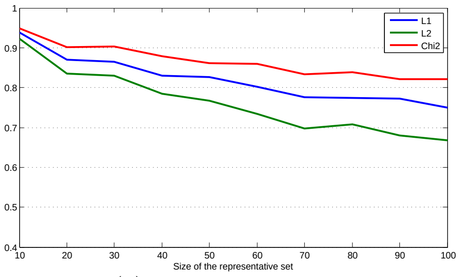
(a) Spanning prototypes



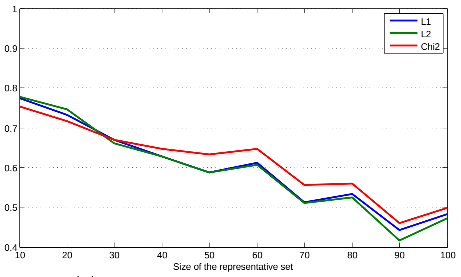
(b) *k*Means



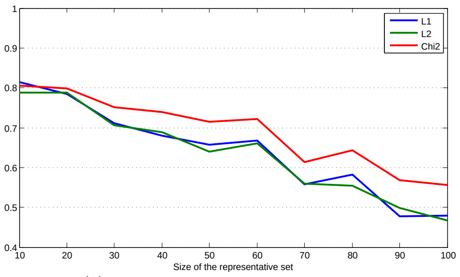
(c) Fuzzy *k*Means - *max*



(d) Fuzzy *k*Means - *all*



(e) Mixture of Gaussians - *max*



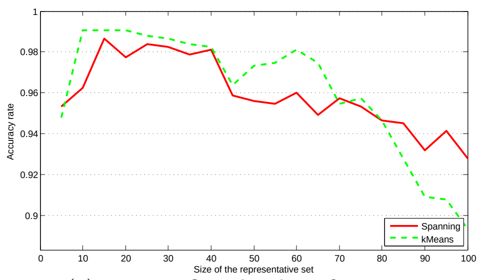
(f) Mixture of Gaussians - *all*

Figure 4.2: Study of the effect of distances for the Digits database using a *k*NN classifier.

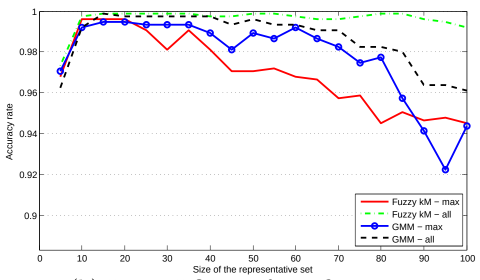
4.4.1 Experimental results

We reproduce the same experiment of the previous section but we compare the different embedding configurations between each other. In particular, we construct all vectors with the 6 different configurations of the embedding using a number of representatives that ranges from 5 to 100, in steps of 5. This assures that a rather large interval of essentially different representations of the graphs is explored. After the vectors are constructed, we classify them using a *k*NN classifier. Together with the *k*NN classifier, we use the ϵ^2 distance to measure graph dissimilarities since, as discussed above, is the one that has experimentally shown the best results. Figs. 4.3 and 4.4 show the results on the validation sets for all the datasets described in Section A.2.1.

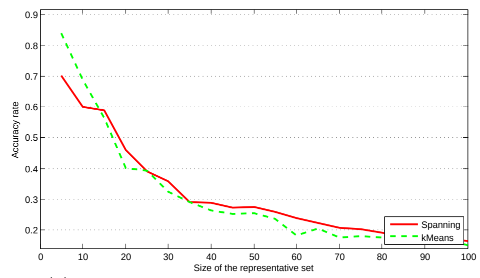
Let us first discuss the case of the Letter databases. There is a clear difference in the results between the low distortion case and the medium and high ones. The more distorted



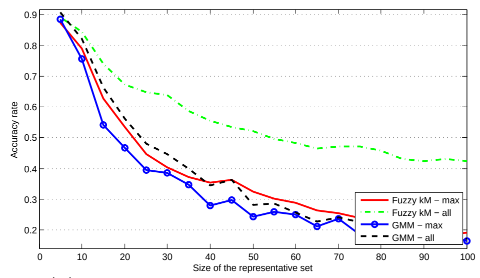
(a) Letter LOW; hard configurations



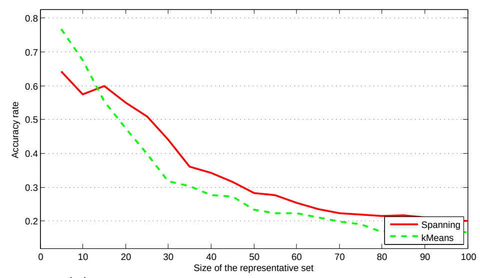
(b) Letter LOW; soft configurations



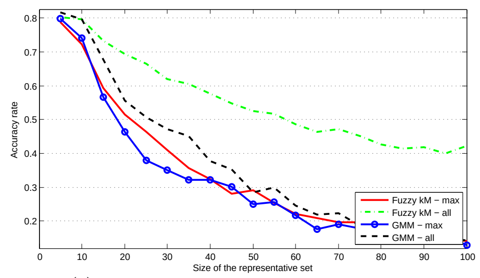
(c) Letter MEDIUM; hard configurations



(d) Letter MEDIUM; soft configurations



(e) Letter HIGH; hard configurations

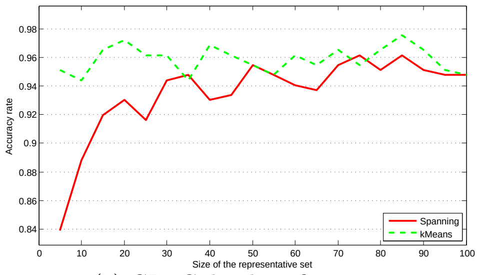


(f) Letter HIGH; soft configurations

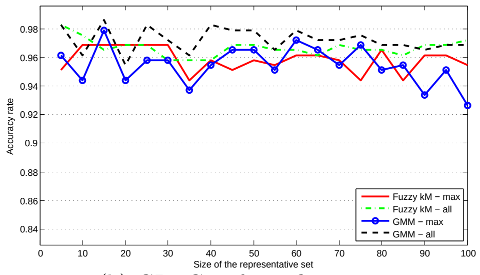
Figure 4.3: Validation results for the different databases of letters. Accuracy rates of a kNN classifier in conjunction with a ϵ^2 distance on the validation set. The horizontal axis shows different choices of the size of the representative set. The hard configurations of the proposed embedding are depicted in the figures on the left. Soft configurations are shown on the right column of figures.

the graphs are, the lower the results are. We discuss these cases separately.

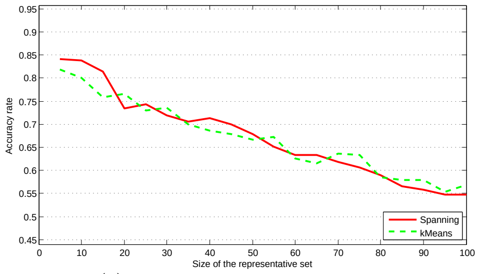
With respect to the low distorted database, we can see that the kMeans configuration adapts more properly to the node distribution than the spanning prototypes method, and thus the accuracy rates are higher (Fig. 4.3(a)). Nevertheless, the recognition rate tends to decrease as the size of the representative set is increasing, which is explained by the fact that the more representatives are considered, the more sparse the resulting vectors become, making the classifier not able any more to distinguish among the different classes. The soft versions (Fig. 4.3(b)) perform generally better than the hard ones and the fuzzification makes the results more stable as the size of the representative set increases. It is also interesting to



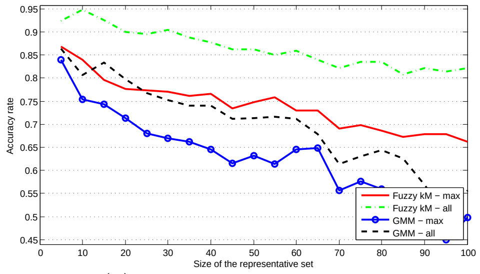
(a) GREC; hard configurations



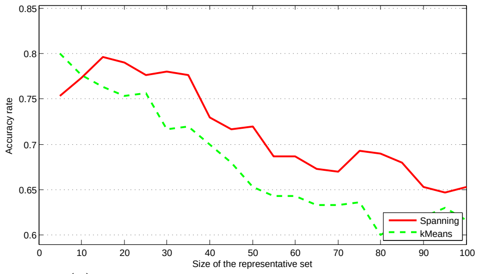
(b) GREC; soft configurations



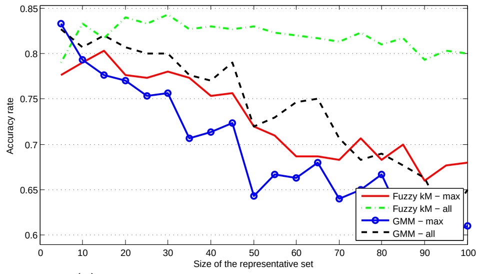
(c) Digits; hard configurations



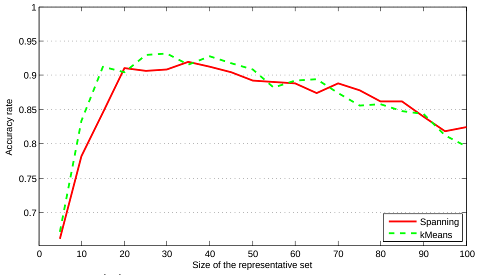
(d) Digits; soft configurations



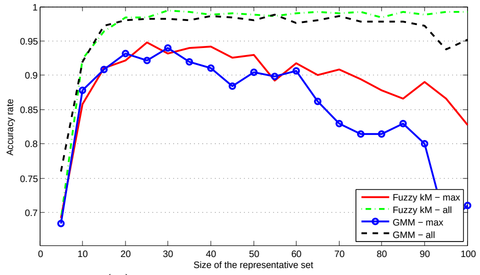
(e) Fingerprints; hard configurations



(f) Fingerprints; soft configurations



(g) COIL; hard configurations



(h) COIL; soft configurations

Figure 4.4: Validation results for the GREC, Digits, Fingerprints and COIL databases. Accuracy rates of a kNN classifier in conjunction with a ϵ^2 distance on the validation set. The horizontal axis shows different choices of the size of the representative set. The hard configurations of the proposed embedding are depicted in the figures on the left. Soft configurations are shown on the right column of figures.

note that the *all* configurations are more capable than the *max* ones.

In cases with medium and high distortion (Figs. 4.3(c)-4.3(f)), the input graphs are so badly distorted that there is no actual way to properly reflect the discriminative features of every class in the vectorial representation. Since the nodes of the graph are not in their expected location, the representatives they are assigned to are not the ones they should be, which results in highly distorted vectorial features. This fact even leads to accuracy rates as low as 20% for large representative sets, while for small ones the results are still acceptable but definitively not good. This behaviour is evident for the hard and the soft versions of our embedding procedure.

Regarding the GREC dataset for the hard configurations (Fig. 4.4(a)), the results of the kMeans selector are again better than the spanning prototypes ones, which makes sense since kMeans adapts in a more accurate manner to the inherent clusters of the label space. Here again, the soft versions (Fig. 4.4(b)) obtain better results than the hard configurations, supporting the idea of a better adaptation to the possible deformations in the represented objects. Concerning the differences between the soft assignments, both the fuzzy kMeans and the mixture of Gaussians methods obtain better results for the *all* approach than for the *max* one.

The results for the Digits and the Fingerprint datasets obey similar behaviours and thus can be discussed together. In these two datasets of graphs a peculiar phenomenon happens. Here, the spanning prototypes usually perform equally or even better than the kMeans configurations (Figs. 4.4(c) and 4.4(e)). The reason for this fact is the inherent distribution of the nodes in the label space. These are uniformly distributed among their range just like a regular grid in the space, preventing kMeans from properly discovering good representatives. In any case, small sets of representatives perform better since global relations in the nodes of the involved graphs are more accurate to describe their shape (and thus their class). Among the soft versions (Figs. 4.4(d) and 4.4(f)), the fuzzy kMeans with the *all* edge assignment stands out with respect to the other soft versions, since this configuration is capable to perform in a stable manner along the increasing sets of representatives.

Finally, the COIL database shows no relevant differences between the two hard versions of the proposed embedding (Fig. 4.4(g)) but a performance worth mentioning in the case of the *all* edge assignment configurations. This behaviour is explained by the nature of the database itself. The Harris salient point detector is quite unstable and, moreover, the images in the COIL database are turning around. By fuzzifing the assignment of nodes (fuzzy kMeans and GMM) and the assignment of edges (*all*), the proposed methodology is able to adapt to the changes that two similar images may show in their respective graph representations.

4.4.2 Discussion

After the previous experiments and discussions, we should recap the fact there is no clear *a priori* way to select a set of representatives of a certain size in the general case. On the contrary, such a parameter should be selected and validated for specific dataset under study. Nevertheless, different sizes of the representative sets may lead to semantically different vectorial representations of graphs, and such a situation should be explored and exploited. Chapter 6 could be a way to approach it.

In any case, we may say that the fuzzy kMeans version of the set of representatives together with the *all* edge assignment methodology is the configuration that provides a most robust vectorial representation of graphs. It clearly outperforms the other ones and is the one that presents a more stable behavior along the curves, this is, when varying the sizes of the sets of representatives. From now on, in the next two chapters of this thesis, this will be the only embedding approach that will be considered. Nevertheless, in Chapter 7, we will revisit these other configurations to check their performance on the test sets of the databases and compare them with other external reference systems.

Chapter 5

Feature Selection

The steps that define the embedding that we have defined in the previous chapters provide us with representations of graphs that might suffer from some problems. First, since the selection of representatives is an unsupervised task, we do not have any control on these elements. We might be selecting irrelevant points in the set of representatives for the task of graph representation. Moreover, the number of edge based features is quadratic in the size of the representative set, leading to high dimensional vectors for large sets of representatives. This may weaken efficient operations between the vectorial representations of graphs.

Another consequence of the quadratic number of edge based features could be some sparsity in the vectorial representations. The selection of representatives might come up with some elements that are barely represented in the graphs under consideration, and also, to representatives the relations between which are not present in the vectors. These situations would lead to too many zero-valued features that would impoverish the final representation. As a final concern, we could also wonder how much correlation is there between the node-based and edge-based features extracted from a specific element in the representative set. Correlation between features is not desired and we ought to tackle this scenario. This chapter covers these situations by different feature selection approaches.

5.1 Feature selection

Feature selection algorithms try to select a proper set of features such that the performance of a certain learning algorithm is improved [64]. A broad taxonomy of these methods is based on two families, namely, ranking methodologies and variance-based algorithms. The first group, usually called *search strategies*, are those methods that assign a weight to every feature in its original form and search for the most relevant ones in terms of these weights. Search strategies can be split into *forward* selection and *backward* elimination. The former starts with an empty set and iteratively adds important features, while the later keeps eliminating useless features from the set of all features. Also *floating* search strategies have been proposed that allow to variably add relevant and remove useless features [126]. The second category of feature selection methods is formed by those methods that initially transform the original features in such a way that the variance within components is kept and then rank the resulting features by means of variance measurements coming from the transformation itself.

In order to detect the most relevant features from the proposed embedding representations of graphs, we will use different kinds of feature selection methods. From the search strategies family, we select three algorithms that are well established and that have proved their good performance on different scenarios. The first one is based on the ability to discriminate among classes in terms of relative distances between feature values, the second one on entropy measurements and the third one is based on the SVM classifier. From the variance-based methods, we use PCA and its non-linear generalization Kernel PCA.

5.1.1 Ranking methods

Ranking methods are based on a ranking map that gives to every feature at hand a certain value that is eventually used to rank it with respect to the others. Based on different ranking strategies we have different ranking methods.

Relief

The Relief algorithm is a classical ranking method that is based on the ability of features to discriminate between different classes [91]. For every instance of a given feature, the closest value among elements of the same class (*Near Hit*) and the closest value among elements of other classes (*Near Miss*) are found. Then a weight is given to every feature in terms of the distances of every sample to the Near Hit and the Near Miss. This is, given the set $\mathcal{S}$ of m samples of feature i, we compute the rank value as

$$_i = \frac{1}{m}\sum_{x\in\mathcal{S}} x - Z_x^- \ - \ x - Z_x^+ \tag{5.1}$$

where Z_x^+ and Z_x^- are the near hit and the near miss of the sample x, respectively. It is clear that a good feature should give low values to the distances between each sample and its near hit and high values to the distances to the near miss. Thus, a good feature should have a high ranking value $_i$.

Mutual Information

Mutual information is a measure of dependency between random variables. Let X and Y be two random variables. Their mutual information $I(X\ Y)$ is defined by

$$I(X\ Y) = \int_Y \int_X p(x\ y) \log\left(\frac{p(x\ y)}{p(x)p(y)}\right) dxdy \tag{5.2}$$

where $p(x\ y)$, $p(x)$ and $p(y)$ are the joint and the marginal probability density functions, respectively. Being the features and the classes the random variables, one is capable to measure how dependent is a feature for a given class. Those feature with higher relevance to the existing classes are weighted higher and thus selected for further use.

Our data needs for a discretization of the feature values so that integrals can be reduced to sums. To discretize features, we make use of the multi-interval discretization of continuous-valued attributes algorithm described in [49]. This particular approach is iteratively based on class information entropies between sets of binary partitions of the whole range of feature values. Formally, features are sorted and all possible binary cuts of the range of values are evaluated in terms of the class entropy that these cuts generate. The cut that provides the minimum class information entropy is chosen and the process is repeated for the subsequent subsets of feature values.

Once discretized, if we consider $X^{(i)}$ the set of discrete values that the feature $\mathbf{x}_i$ can take, the mutual information between $\mathbf{x}_i$ and the set of class labels Ω reduces to

$$I(\mathbf{x}_i \ \Omega) = \sum_{\in\Omega} \sum_{x_i \in X^{(i)}} p(x_i \quad) \log \left(\frac{p(x_i \quad)}{p(x_i)p(\ )} \right) \tag{5.3}$$

where both the joint and marginal density functions can be estimated by counting the instances in the training set. Finally, features are ranked based on their mutual information in a forward selection fashion.

SVM based ranking

The last ranking method we will use in our experimental evaluation was originally proposed in [63] and is based on the support vector machine classifier (SVM). An SVM classifier seeks for a hyperplane $f(x) = \ w \ x \ + b$, where $w \quad \mathbb{R}^n$ and $b \quad \mathbb{R}$, that best separates the involved classes. The components of the vector w can be used as feature rankings since they weight how much each of the components (features) influences the final decision boundary. The idea is thus to consider those features with high values in the vector w as relevant features.

5.1.2 PCA-based methods

The other category of feature selection methods considered in this work does not rank the original features but, instead, a transformation of those.

Principal Component Analysis

Given a set of N feature vectors $x_1 \quad\quad x_N \quad \mathbb{R}^n$, principal component analysis (PCA) finds a linear transformation of the data $y_i = Ax_i \quad \mathbb{R}^m$ so that linear correlation among the new features is reduced and the new $m \leq n$ features capture most of the variance. Such transformation is obtained by an orthogonal mapping where each column of the matrix A is an eigenvector of the covariance matrix of the centered original data. These eigenvectors $v_1 \quad\quad v_n$ are called principal components and are ordered from greater to smaller variance. By taking $m \leq n$ principal components, the dimensions are reduced and most of the variance is being kept.

Kernel PCA

Kernel principal component analysis (kPCA) is a non-linear generalization of PCA by means of the kernel trick [146]. kPCA finds linear behaviors of the data in the implicit space of the kernel function, which in general correspond to non-linear properties of the input patterns. The projection of $\nu(x)$ onto a (non-linear) principal component u_p of the input feature space is given by

$$u_p \cdot \nu(x) = \sum_{j=1}^{N} \beta_j^p \kappa(x_j \ x) \tag{5.4}$$

where $\beta^p = (\beta_1^p \quad\quad \beta_N^p) \quad \mathbb{R}^N$ is the p-th leading eigenvector of the kernel matrix $K = (\kappa(x_s \ x_t))_{1 \leq s \ t \leq N}$. The final transformation is given by $y_i = (u_1 \cdot \nu(x_i) \quad\quad u_n \cdot \nu(x_i))$. Exactly as in PCA, by keeping $m \leq n$ principal components one captures most of the variance in $\mathcal{H}$.

Besides standard PCA, in the experimental part of this chapter (Section 5.2), we have used two other well-known and used kernel functions, namely, the radial basis function -or Gaussian kernel- and the ϵ^{2} kernel:

$$\kappa_{\mathrm{rbf}}(x\ \ y) = \exp(-\gamma \cdot \quad x - y \quad ^{2}) \quad \gamma > 0 \tag{5.5}$$

$$\kappa_{\omega^2}(x\ \ y) = \exp(-\gamma \cdot d_{\omega^2}(x\ \ y)) \quad \gamma > 0 \tag{5.6}$$

where $\cdot$ stands for the L_2-norm and $d_{\omega^2}(\cdot\ \ \cdot)$ is the ϵ^{2} distance (Eq. (4.15)).

5.2 Experimental evaluation

Given the seven continuous attributed graph datasets described in Section A.2.1, we build up the embedding representations by the use of fuzzy kMeans for the set of representatives construction and the *all* edge assignment setting. We construct the vectorial representations for different sizes of the set of representatives.

As previously discussed, the choice of the representative elements is of crucial importance because the semantics of the representation will depend on them. Nevertheless, we have no presumptive manner to select them beforehand and thus we assume this step as one to be validated. Again thus, and for each dataset, we have built sets of representative elements of sizes ranging from 5 elements up to 100, in steps of 5, leading to 20 different vectorial representations for each dataset of graphs. We apply the described feature selection algorithms to all of them.

5.2.1 Ranking methods

For each of these representations based on a different size of the set of representatives, we have to select the subset of features that best solves a specific task. In particular, as we did in most validation stages so far, we are interested in solving a classification problem by means of a kNN classifier. Given one of these representations, based on a set of representatives of size m, the number of resulting features is $M \sim \mathcal{O}(m^2)$

$$x_1\ \ x_2 \qquad x_M \tag{5.7}$$

and according to a ranking criteria, they are sorted from greater to lower relevance

$$x_{(1)}\ \ x_{(2)} \qquad x_{(M)} \tag{5.8}$$

Once the features are ranked, we can construct a structure of nested features: from the most important one, we iteratively add the rest of them in decreasing order of importance, obtaining several subsets of features,

$$x_{(1)} \quad \wedge \quad x_{(1)}\ \ x_{(2)} \quad \wedge \qquad x_{(1)}\ \ x_{(2)} \qquad x_{(M)} \tag{5.9}$$

These nested features are the candidates for the optimal subset of features that we seek for each representation. If we try out all these subset candidates for all the different vectorial representations that we have created, the computational complexity of the validation stage increases dramatically. To avoid this situation, we do not use all candidates but instead we use just some of them. In particular, we use those subsets that correspond to the most relevant feature (first subset) and that of all features (last subset), plus the subsets containing $\frac{1}{16}$, $\frac{1}{8}$, $\frac{1}{4}$, $\frac{3}{8}$, $\frac{1}{2}$, $\frac{5}{8}$, and $\frac{3}{4}$ of the most relevant features.

For the Relief and Mutual Information cases, we compute the nested feature subsets after ranking each feature. This can be done this way since the ranking itself is not a complex

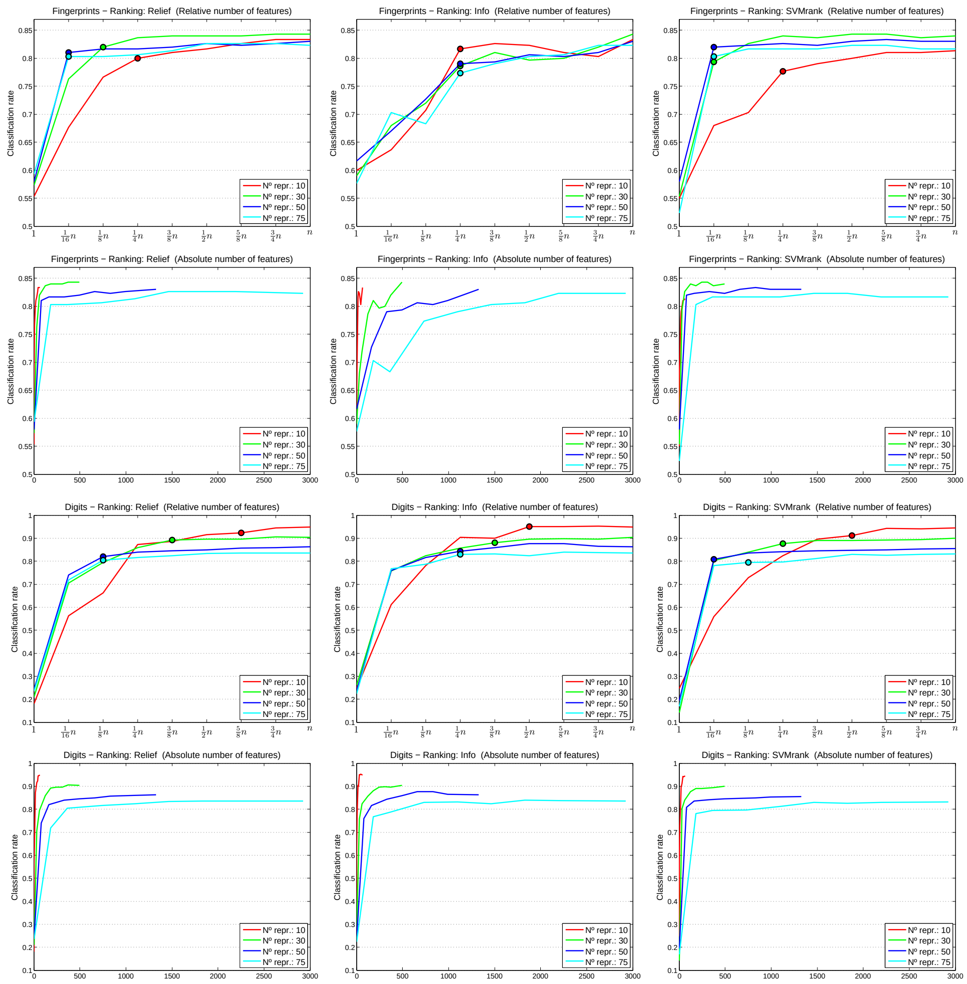


Figure 5.1: Validation results for some configurations on the Fingerprint and Digits databases. Accuracy rates of a kNN classifier in conjunction with a ϵ^2 distance on the validation set. First and third rows show the behavior when keeping a relative number of components. Dots on the curves show the best configuration. Second and fourth rows plot the same curves in absolute terms of the selected features.

task. For the SVM ranking case, we proceed differently. Instead of ranking all features and then building up the nested structure, we directly build such a structure in its reduced version. We initially train an SVM classifier with all features and then remove all of them except for $\frac{3}{4}$ of the most relevant features. With the remaining ones, we proceed analogously and, after training, remove all except for $\frac{5}{8}$ of the most relevant features. This procedure is repeated until, finally, we end up with the most relevant feature.

In Fig. 5.1, we show classification results on the validation sets of the Fingerprints and the Digits datasets (kNN classifier with the ϵ^2 distance). These are two representative examples of the general behavior observed. We plot, for each ranking method, different curves that correspond to different choices of the size of the representative set (10, 30, 50 and 75 representatives). And we plot these curves in two different ways: in relative and in absolute terms regarding the size of the feature subset that is being used for classification. Moreover, on each relative curve and using a Z-test of statistical significance with a confidence level of $\alpha = 0\ 05$, we draw a dot at the best configuration, where by *best* we understand the one based on the least number of features from all those configurations that are statistically at the same level of significance than the maximum accuracy rate obtained.

The main behavior we can observe in Fig. 5.1 is the fact that, in general, those configurations that are constructed with larger sets of representatives need a relatively smaller number of components in order to reach the proper subset of features. This can be seen on the relative curves since the dots corresponding to the larger sets of representatives can be found before those of the smaller ones. In the absolute curves this behaviour can also be noticed by the fact that curves tend to flatten rapidly when the representative set becomes larger. This situation also suggests that large sets of representatives introduce noisy and redundant features to a higher degree than smaller sets. It is clear that most of the elements in the set of representatives will tend to be not edge-linked in the graphs as the size of the representative set increases. However, as it happens in the Fingerprint dataset, a larger representative set might obtain better results than a smaller one.

In Table 5.1 we show a deeper analysis into the actual features the ranking methods are considering as relevant. In particular, we put attention on whether the ranking methods keep the node-based features U_i and on how much they influence the final subsets in the nested structures. For each choice of the size of the representative set, we show the configuration that has obtained the best classification results of the kNN classifier on the validation set.

Several statistics are shown. First, the size of the set of representatives (*rss*) which is, actually, the number of U_i features before feature selection. The resulting dimensionality of the vectors after mapping the graphs under the described embedding methodology, this is, the original number of features (*onf*), all U_i and B_{ij} features. The next column of the table (*onbf*, original node-based features) tells which is the percentage of the node-based features over all features in the original representation (first column over the second column).

We display the number of significant features (*nsf*), this is, the size of the optimal subset of features that leads to the best classification performance given a set of representatives. From these sets, we are interested in the proportion of features that originally come from U_i, namely, the significant node-based features (*snbf*) and also the proportion of node-based features that are kept in the final optimal subset (*nbfk*), this is, the actual number of U_i features that the ranking algorithm has selected. We finally show the classification rate (*cr*) of each specific configuration in %.

A first observation we make is how much all feature ranking methods reduce the original number of features. By comparing the *nof* and *nsf* columns, regardless of the database and the ranking methodology we work with, we see that the number of features is, in general, drastically reduced, resulting in a situation in which further learning algorithms are computationally more feasible than when using all the original features.

Another interesting observation is the fact that node-based features are more present in the reduced version this is, in the optimal subsets of features than in the original sets (see *onbf* versus *snbf*). Indeed, it only happens in a very few number of cases that the percentage of features coming from node probabilities in the original vector representations is higher than in the reduced versions. This is indicating the importance of these U_i features. Nevertheless, B_{ij} features do introduce important additional information in the embedded representation as long as several of these features are kept in the reduced versions. We also observe that the SVM ranking methodology tends to keep a higher proportion of the features than the other methods.

Table 5.1: Feature statistics for the different ranking methods under different embedding configurations.

				Ranking method											
Dataset	Original configuration			Info				Relief				SVMrank			
	rss	*nof*	*onbf*	*nsf*	*snbf*	*nbfk*	*cr*	*nsf*	*snbf*	*nbfk*	*cr*	*nsf*	*snbf*	*nbfk*	*cr*
Letter LOW	10	65	15.4	24	29.2	70.0	99.2	65	15.4	100.0	99.7	24	33.3	80.0	99.3
	20	230	8.7	86	14.0	60.0	99.3	57	10.5	30.0	99.3	143	11.9	85.0	99.5
	30	495	6.1	123	8.1	33.3	99.2	61	6.6	13.3	99.2	61	44.3	90.0	99.9
	50	1325	3.8	496	5.0	50.0	99.3	496	7.3	72.0	99.7	165	27.9	92.0	99.6
	75	2925	2.6	731	4.8	46.7	99.3	365	7.9	38.7	98.9	182	34.6	84.0	98.9
	100	5150	1.9	1287	3.9	50.0	98.7	321	6.2	20.0	98.8	321	26.5	85.0	98.9
Letter MED	10	65	15.4	48	12.5	60.0	83.5	65	15.4	100.0	84.4	48	16.7	80.0	80.1
	20	230	8.7	86	7.0	30.0	66.5	143	7.7	55.0	63.2	143	13.3	95.0	63.2
	30	495	6.1	309	7.4	76.7	59.3	309	5.2	53.3	60.5	247	11.7	96.7	60.4
	50	1325	3.8	82	11.0	18.0	47.7	331	2.7	18.0	50.1	662	7.4	98.0	48.7
	75	2925	2.6	182	7.7	18.7	44.8	182	2.2	5.3	42.5	1096	6.7	97.3	43.3
	100	5150	1.9	321	7.5	24.0	39.6	321	3.1	10.0	44.0	1931	5.0	96.0	38.9
Letter HIGH	10	65	15.4	32	21.9	70.0	76.4	40	17.5	70.0	77.6	24	29.2	70.0	75.5
	20	230	8.7	86	10.5	45.0	69.5	86	14.0	60.0	65.2	57	28.1	80.0	67.2
	30	495	6.1	123	12.2	50.0	61.2	123	5.7	23.3	60.1	61	39.3	80.0	60.7
	50	1325	3.8	165	6.1	20.0	53.1	82	3.7	6.0	49.9	82	46.3	76.0	51.7
	75	2925	2.6	182	7.1	17.3	49.6	182	1.6	4.0	50.8	182	34.1	82.7	50.3
	100	5150	1.9	321	5.9	19.0	48.8	321	1.9	6.0	42.4	321	26.8	86.0	43.2
Digits	10	65	15.4	32	25.0	80.0	95.0	40	22.5	90.0	92.4	32	21.9	70.0	91.2
	20	230	8.7	115	11.3	65.0	88.4	86	18.6	80.0	88.2	57	29.8	85.0	87.4
	30	495	6.1	185	9.2	56.7	88.0	185	13.0	80.0	89.2	123	18.7	76.7	87.6
	50	1325	3.8	331	7.3	48.0	84.4	165	12.7	42.0	82.0	82	48.8	80.0	80.8
	75	2925	2.6	731	5.7	56.0	83.0	365	7.9	38.7	80.4	365	19.2	93.3	79.4
	100	5150	1.9	1287	4.3	55.0	82.0	1287	5.9	76.0	79.8	643	14.9	96.0	76.8
Fingerprints	10	65	15.4	16	31.3	50.0	81.7	16	43.8	70.0	80.0	16	31.3	50.0	77.7
	20	230	8.7	86	11.6	50.0	81.0	28	42.9	60.0	81.3	57	29.8	85.0	82.0
	30	495	6.1	123	10.6	43.3	78.7	61	34.4	70.0	82.0	30	60.0	60.0	79.3
	50	1325	3.8	331	7.3	48.0	79.0	82	28.0	46.0	81.0	82	46.3	76.0	82.0
	75	2925	2.6	731	4.4	42.7	77.3	182	24.7	60.0	80.3	182	35.2	85.3	80.3
	100	5150	1.9	643	5.1	33.0	77.3	321	27.4	88.0	79.3	321	27.7	89.0	79.7
GREC	10	65	15.4	8	12.5	10.0	94.8	16	31.3	50.0	96.9	16	43.8	70.0	97.9
	20	230	8.7	28	14.3	20.0	95.8	57	24.6	70.0	96.9	28	57.1	80.0	96.5
	30	495	6.1	30	10.0	10.0	94.4	30	26.7	26.7	93.7	30	73.3	73.3	96.9
	50	1325	3.8	165	7.3	24.0	97.2	82	22.0	36.0	95.8	82	50.0	82.0	95.8
	75	2925	2.6	182	9.3	22.7	94.1	182	19.8	48.0	96.2	182	36.3	88.0	95.5
	100	5150	1.9	643	4.7	30.0	95.5	321	14.3	46.0	96.9	321	28.0	90.0	96.2
COIL	10	65	15.4	40	17.5	70.0	91.0	24	20.8	50.0	89.4	40	17.5	70.0	90.6
	20	230	8.7	172	9.9	85.0	96.8	86	18.6	80.0	97.0	57	26.3	75.0	96.2
	30	495	6.1	309	8.1	83.3	98.6	123	13.0	53.3	98.0	123	18.7	76.7	98.4
	50	1325	3.8	496	7.3	72.0	98.4	331	7.6	50.0	98.8	165	21.2	70.0	98.0
	75	2925	2.6	1096	5.1	74.7	98.2	365	6.0	29.3	97.8	365	17.0	82.7	98.2
	100	5150	1.9	1931	3.8	73.0	98.4	1287	3.0	39.0	98.6	321	24.0	77.0	98.2

rss: representative set size. *onf*: original number of features. *onbf*: original node-based features (%).
nsf: number of significant features. *snbf*: significant node-based features (%). *nbfk*: node-based features kept (%).
cr: classification rate on the validation set (%).

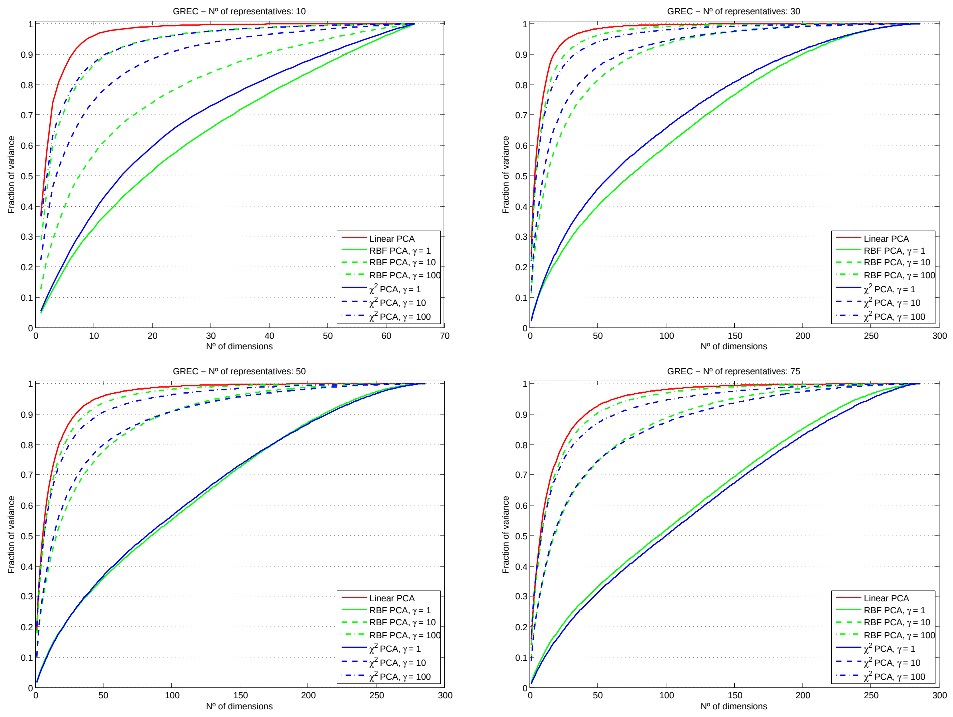


Figure 5.2: Fractions of variance for different choices of the representative set on the GREC dataset.

Finally, let us mention the fact that the accuracy of the classifier is not necessarily a monotonic function with regard to the representative set. In some cases, it turns out that small or large representative sets are not performing as well as intermediate sizes. In any case, these functions tend to stabilize, suggesting that the ranking methods are capable of reducing the noisy and redundant features that we are producing with our embedding methodology when we increase the size of the set of representatives. We do observe, though, that there is some correlation between the size of the graphs and the accuracy rates obtained. The datasets that have a small average number of nodes tend to obtain better results using small sets of representatives (see Letters datasets) and those with larger average number of nodes achieve better results when using larger sets of representatives (see GREC and COIL cases).

5.2.2 PCA-based methods

For the PCA-based methods we have also built the same vectorial representations based on the 20 different sets of representatives. We have, however, adopted another validation strategy. Although we also find a ranking on the transformed features, we make use of the variance that each component is preserving and we threshold these values, keeping a certain amount of them as relevant.

In particular, we initially apply the PCA and Kernel PCA transformations and keep all features. In Fig. 5.2 we show the variance for different sets of representatives in the GREC dataset. We depict the fraction of variance curves for PCA, and for different values of the γ parameter in the Kernel PCA approach with the two mentioned kernel functions.

We clearly see how PCA is capable of easily keeping most of the variance with just a small number of features. It is much faster than any of the kernel PCA approaches in all cases. The same behavior is observed in all the other datasets we work with. In any case, this does not necessarily mean that PCA reduced features outperform the kernel PCA ones since the performance will depend on the transformation rather than the precise number of features that the method is keeping. It is also worth noticing that higher values of the γ parameter for the kernel functions will produce transformations that maintain the same rate of variance with less dimensions than lower values of it. Nevertheless, this is again not synonymous to the fact that these higher values will produce better transformations of features with regard to the classification performances.

The optimal subset of the transformed features is obtained by different cut-off points that we do on the variance values that each component is preserving. Particularly, we make cuts on the fraction of variance at the following points: 0.9, 0.925, 0.95, 0.975, 0.99 and 0.999. Each of these cut-off points determines a particular number of features that are being considered as potential candidates for the optimal subset of features in the final representation.

Again, the accuracy of a classifier can be regarded as a function of both the size of the representative set and the amount of variance that is being kept. In Table 5.2 we show some statistics of different configurations for all datasets. Using a representative set of a certain size (*rss* in the table), with its respective number of original features (*onf*), we apply all the cut-off points mentioned above. For each of them, we apply a kNN classifier together with the Euclidean distance (features are transformed making them no longer histogram-based and thus the ϵ^2 distance is not used). For each of these representations, we report the best classification rate (*cr*), the cut-off point on the fraction of variance that has produced this performance (*fov*) and the corresponding number of features in the reduced version of the embedded graphs (*ndrv*).

A first and important comment we should make here is that almost all configurations that we have considered (the ones we show and the ones we do not show) already reach the best performance by using the lowest threshold that we have considered for the variance cut-off points. This means that all further cut-off points define sets of features that do not actually improve the performance of this lowest cut, and thus, most of the redundancy is removed from the vectorial representations. It also suggests the use of lower cut-off points. Yet, we have experimentally seen that these lower points lead to too few features and too low classification rates.

Related to this finding is the fact that the final number of dimensions is drastically reduced with respect to the original ones. This fact is even more prominent when compared to the size of the optimal subsets that were obtained using the ranking methods. Thus, PCA-based methods reduce to a higher degree the dimensionality of the embedded representations of graphs than the ranking methodologies.

On the other hand, we encounter that such a reduction is not necessarily related to the performance of the considered classifier. In general, when comparing both tables, we observe that the ranking methods usually outperform methods based on PCA or Kernel PCA. In other words, transforming the features does not seem to make the final configuration stronger. We should of course check whether this is a problem of the general methodology used or just the fact that other kernel functions should be applied together with other distance measures in the kNN algorithm. However, we understand that such a deeper study is out of scope with regard to the original objectives of this work and we leave it for future work.

Table 5.2: Feature statistics for the different PCA-based methods under different embedding configurations.

			Reduction method											
Dataset	Original configuration		Linear PCA			*rbf* PCA				χ^2 PCA			
	rss	*onf*	*fov*	*ndrv*	*cr*	*fov*	*ndrv*	γ	*cr*	*fov*	*ndrv*	γ	*cr*
Letter LOW	10	65	0.9	12	97.87	0.9	28	2.00	98.80	0.9	27	2.00	99.47
	20	230	0.9	19	96.80	0.9	112	2.00	99.60	0.9	125	2.00	99.87
	30	495	0.9	30	93.33	0.9	310	1.00	98.80	0.9	192	4.00	99.87
	50	1325	0.9	54	91.73	0.9	532	0.50	95.47	0.9	414	2.00	99.60
	75	2925	0.9	82	90.00	0.9	575	0.50	92.40	0.9	380	4.00	99.47
	100	5150	0.9	108	87.87	0.9	589	0.50	91.20	0.975	702	0.50	98.80
Letter MED	10	65	0.9	21	52.27	0.9	52	0.50	58.27	0.9	43	2.00	74.00
	20	230	0.9	48	38.93	0.9	88	10.00	39.73	0.9	168	2.00	60.93
	30	495	0.9	71	37.07	0.9	157	10.00	37.33	0.9	423	1.00	65.47
	50	1325	0.9	113	31.73	0.9	224	10.00	37.20	0.9	622	1.00	65.47
	75	2925	0.9	132	32.40	0.9	246	10.00	35.73	0.925	659	1.00	66.67
	100	5150	0.9	157	30.00	0.9	229	16.00	36.67	0.9	598	2.00	65.47
Letter HIGH	10	65	0.9	26	56.40	0.9	44	2.00	60.40	0.9	49	1.00	74.67
	20	230	0.9	58	51.47	0.9	99	10.00	55.47	0.9	174	2.00	71.87
	30	495	0.9	87	50.27	0.9	173	10.00	52.67	0.9	394	2.00	70.80
	50	1325	0.9	135	43.33	0.9	263	8.00	49.33	0.9	472	5.00	72.80
	75	2925	0.9	167	42.13	0.9	271	10.00	45.33	0.9	458	8.00	72.40
	100	5150	0.9	192	38.53	0.9	258	16.00	44.40	0.9	524	5.00	72.80
Digits	10	65	0.9	11	83.40	0.9	44	4.00	86.00	0.9	50	1.00	89.00
	20	230	0.9	26	81.20	0.9	143	5.00	83.60	0.9	177	2.00	89.00
	30	495	0.9	39	76.20	0.9	153	16.00	78.80	0.9	349	4.00	87.60
	50	1325	0.9	69	74.20	0.9	441	8.00	76.20	0.999	997	1.00	91.00
	75	2925	0.9	106	69.80	0.9	503	8.00	72.60	0.9	864	2.00	89.20
	100	5150	0.9	140	68.60	0.9	491	10.00	72.00	0.975	965	2.00	90.00
Fingerprints	10	65	0.9	7	80.00	0.9	8	500.00	80.33	0.9	10	100.00	82.67
	20	230	0.9	13	79.67	0.9	17	100.00	79.67	0.9	70	8.00	78.33
	30	495	0.9	20	77.00	0.9	133	5.00	78.67	0.9	66	16.00	81.33
	50	1325	0.9	34	80.67	0.9	35	500.00	80.67	0.9	56	64.00	81.00
	75	2925	0.9	52	80.67	0.9	58	100.00	82.67	0.9	71	100.00	80.33
	100	5150	0.9	66	80.67	0.9	87	32.00	81.33	0.9	113	32.00	80.33
GREC	10	65	0.9	6	84.27	0.9	32	16.00	92.31	0.9	43	2.00	95.10
	20	230	0.9	12	87.06	0.9	100	5.00	93.36	0.9	82	4.00	97.20
	30	495	0.9	17	86.36	0.9	128	4.00	90.91	0.9	154	2.00	96.50
	50	1325	0.9	29	90.56	0.9	144	4.00	94.06	0.9	184	2.00	96.85
	75	2925	0.9	41	93.71	0.9	53	64.00	95.10	0.9	154	5.00	97.20
	100	5150	0.9	56	91.61	0.9	72	50.00	92.31	0.9	186	4.00	97.90
COIL	10	65	0.9	7	52.60	0.9	25	500.00	60.80	0.9	37	16.00	75.20
	20	230	0.9	15	63.80	0.9	24	1000.00	66.00	0.9	180	4.00	80.00
	30	495	0.9	25	65.20	0.9	35	1000.00	67.40	0.9	395	5.00	85.20
	50	1325	0.9	50	65.60	0.9	74	1000.00	65.60	0.9	1126	5.00	87.80
	75	2925	0.9	112	72.00	0.95	2115	2.00	79.80	0.975	2245	2.00	97.40
	100	5150	0.9	191	73.40	0.95	2117	2.00	77.60	0.99	2312	2.00	96.60

rss: representative set size. *onf*: original number of features. *fov*: fraction of variance.
ndrv: number of dimensions in the reduced version. γ: weighting parameter in kernel PCA methods.
cr: classification rate on the validation set (%).

In any case, the results of Table 5.2 suggest again that there is no clear *a priori* way to define which is the number of elements in the set of representatives and that this step should always be validated since it depends on the dataset under study.

5.3 Discussion

Let us summarize the main points tackled in this chapter. We have faced the dimensionality problem that the vectorial representation we proposed for graph embedding exhibits due to its quadratic dependency on the number of elements in the set of representatives. Moreover, some correlation and redundancy on the features is also present and feature selection algorithms have been applied to deal with these problems.

We have observed that the major gain of applying such algorithms is on those cases where a larger set of representatives is used. Clearly, these configurations of the embedding add more noise to the features and redundancy or irrelevance of some of them will be there in a higher degree than in those cases where less representatives are used. Nevertheless, ranking methodologies have revealed themselves as more appropriate for the task of feature selection in the proposed feature vectors of graphs than PCA-based methods. Transforming the features is not convenient since many information is lost in the transition. Although with PCA-based methods the dimensionality is reduced in a higher degree than with the ranking ones, this translates into a drop of the performance of the embedding methods for the task of classification. A careful investigation on which are the features that ranking methods rank as more discriminative tells us that node-based features are certainly more important than the edge-based ones, but still these latter ones are relevant for the task of graph characterization since they convey most of the structural information of the graphs, and thus it is important to consider them in this framework.

Chapter 6

Ensembles of Classifiers

One of the main conclusions we can draw from the previous chapters is that there is no *a priori* manner to define the magnitude or the number of features we build for our graph representations. For instance, in the case of discretely attributed graphs, besides what the experimental validations tell us, we cannot say beforehand whether it is more convenient to use short or long path based-features. Also, for the continuously attributed graphs, we really have no clue when it comes to selecting a small or a large set of representatives, and we have seen that such a parameter is dependant on the dataset under study.

We do acknowledge, however, the plurality and diversity that these parameters eventually offer us in terms of the vectorial representations of graphs. It is clear that short path features will regard local information of discrete attributed graphs while longer ones will account for a more global information of the structures. Regarding continuous attributes, a similar argument can be made. Small sets of representatives will retain the global topology of graphs since every representative element will attract more nodes to it and the corresponding edge relations. On the other hand, large sets of representatives will distribute the structural information into more bins of the vectors by assigning a fewer number of nodes to each representative element and thus better describing local information of the graphs.

In this chapter, instead of assuming a best vectorial representation out of all the ones that are constructed, we make use of the diversity all of them offer when considered together, and combine them in a multiple classifier system in order to boost the performance of the classification of graphs.

6.1 Multiple Classifiers on Graphs

A multiple classifier system tries to combine several base classifiers in such a way that the resulting classification performance improves the accuracy rates of the underlying individual classifiers [97]. A common way to build base classifiers for further combination is by randomly selecting different subsets of features and training classifiers on those subsets [72]. By feature subset selection one is usually able to obtain classifiers with enough diversity, in terms of their discriminative power. Such procedures can be rather easily implemented and have been widely studied for statistical feature vectors. However, when it comes to graph based representations, the construction of single base classifiers has not a straightforward solution.

Based on the idea of feature subset selection, just a few works aiming at constructing multiple classifier systems for graph based representations have been proposed. While for feature vectors the idea is to select a subset of features, for graph representations the way to proceed is to construct base classifiers by selecting subgraphs of the training graphs. For instance, in [140] the authors are able to define different classifiers by randomly removing nodes and their incident edges from the graphs. This methodology has the problem that the need arises of comparing several instances of labelled graphs, which makes the combination of multiple classifiers a highly complex task. In [101], a more efficient way is proposed by decomposing a set of labelled graphs into several unlabelled subgraphs based on their labelling information. All these subgraphs are compared one by one with respect to different labels which create different base classifiers. Finally, in [102], in order to build several base classifiers, the authors claim for the beneficial use of full graphs instead of just subgraphs. They transform labelled full graphs into different unlabelled graphs by removing information from the nodes and re-weighting the edges based on the linked nodes information. Although the graphs are altered, the topology is preserved and several base classifiers can be constructed.

The main drawback of all the approaches described above is that they necessarily need to work in the graph domain when comparing graphs and, therefore, the base classifiers to be used are restricted to be of the kNN type. In this work we adopt another approach to construct several base classifiers for graph based representations. We use all the embedding representations that we have at hand based on different order edge-based features or on different representative set sizes and build up a base classifier for each of them. Then, we combine the base classifiers under any of the common statistical combination frameworks (see Section 6.2). The idea is somewhat similar to the one in [131] where the authors transform a graph into a feature vector by computing edit distances to a predefined set of prototypes. By using different sets of prototypes, the authors create different populations of vectors leading to several base classifiers that can be combined.

6.2 Multiple Classifier Methods

In this section we intend to give a brief description of the existing methodologies for combination of multiple classifiers. We start by categorizing the set of classifiers in terms of their output values and then explain how every class of classifier can be combined.

In [182], the authors originally put together various classifiers under the following taxonomy, based on the output information they are able to supply. Given $\mathbf{x}$ $\mathcal{P}$ a pattern to be classified, let L_j $_{1\leq j\leq N}$ be N different classifiers solving an n-class classification problem and let $_i$ $_{1\leq i\leq n}$ be the labels of the n different classes. There are three levels of classifiers:

1. **The abstract level**: The output of the classifiers is a unique label corresponding to the class that has been recognized, this is, $L_j(\mathbf{x}) = \ _{ji}$.
2. **The rank level**: At this level, the classifier outputs are a ranked list of all the labels $L_j(\mathbf{x}) = [\ _{ji_1}\ \ _{ji_2}\ \ \ _{ji_n}]$, sorted from the most to the least plausible labels.
3. **The measurement level**: The last level of classifiers is the one whose classifiers output an n-dimensional vector $L_j(\mathbf{x}) = [p_{j1}(\mathbf{x})\ p_{j2}(\mathbf{x})\ \ \ p_{jn}(\mathbf{x})]$, where the component $p_{ji}(\mathbf{x})$ is defining the degree of confidence that the pattern to be classified belongs to the class i, under the classifier L_j.

The problem is now how to combine these outputs from a set of different classifiers in order to obtain a final unique label. Such combination will obviously depend on whether the classifiers belong to the first, the second or the third level.

- **Voting**: For classifiers in the abstract level, voting is a commonly used technique [97]. The L_j output $_{ji}$ is considered as one vote for the class i. Then, the final decision is taken as the class with the maximum number of votes.

- **Borda Count**: In the rank level, the problem is re-ranking the labels for the final decision. From the decision of the classifier L_j, Borda Count gives to each class the value of its position. Then, for all classifiers, the minimum of the sum of these positions is taken as the final decision label.
- **Bayesian Combinations**: When working in the measurement level, [93] provides a really elegant understanding of the situation. Based on the Bayes decision rule, different assumptions on the probability dependencies lead to different combination strategies such as the maximum, the minimum, the mean or the product of all confidence values for a specific class. In all cases, the class with the maximum value after each of these strategies is taken as the final decision.

There is also another way of combining classifiers in the measurement level. The Bayesian Combination methodologies for the measurement level of classifiers assumes the independence of all classifiers. This might not be a proper assumption. To investigate this fact, this assumption is eliminated in [129] and two new methodologies are proposed based on the independence of the validation variables. In any case, we shall not work with these non-Bayesian strategies. In particular, in this work, we use the voting combination strategy for the first level of classifiers, the Borda Count for the second, and the product and the mean rules in the Bayesian combination framework.

6.3 Classifier Selection

In order to build a diverse set of base classifiers, for the discretely attributed case of graphs, we will use the different representations that we may extract based on different path lengths of the edge-based features. For the continuous case, the base classifiers will be learnt from the different representations that we can extract out of different sets of representatives based on their size. An important issue that deserves our attention is thus the question of how to build the final ensemble from the available classifiers. Among the options that are available in the literature, we have addressed this problem by a forward selection strategy since it provides a simple and successful framework for the selection of classifiers [97].

From the set of N classifiers that we have available, we will construct N ensembles, considering an increasing number of classifiers (from 1 to N) in each ensemble. The first ensemble will be constituted by one classifier, the second by two classifiers, etc., until the last ensemble that is going to be the ensemble of all classifiers that we have trained. These ensembles are iteratively build by, first, taking the best single classifier as the first ensemble, and then adding to the k-th ensemble the classifier that best fits the previous ensemble, in terms of the accuracy of the combination on a validation set. As a result, the final ensemble that is applied to the test set is the one with the highest accuracy rate. Algorithm 1 summarizes the steps of the procedure.

6.4 Experimental evaluation

In this section we experimentally investigate the ensembles of classifiers for discretely and continuously attributed graphs. In the former case, we construct different classifiers based on the length of paths that are used for the edge-based features. In the latter, based on the sizes of the sets of representatives.

For each of the vectorial representations we learn a model for classification. In particular, we use a Support Vector Machine [147] in conjunction with a ϵ^2 kernel (Eq. (5.6)). We have

Algorithm 1 Classifier Selection

```
Input:    Set of all single classifiers: $L = \quad L_1 \qquad L_N$
Output:   Combination with highest accuracy rate: $C^m$

1: Initialize $N$ empty classifier combinations: $C = \quad C_1 \qquad C_N$
2: Set the best individual classifier as the first combination: $C_1 = \underset{L_i \in L}{\text{argmax}}\, acc(L_i)$
3: $k := 1$
4: **while** $k < N$ **do**
5:     From the remaining classifiers, take the one that best complements the previous combination: $L^m = \underset{L_i \in L \setminus C_k}{\text{argmax}}\, acc(C_k \quad L_i)$
6:     Add it to the previous combination: $C_{k+1} = C_k \quad L^m$
7:     $k = k + 1$
8: **end while**
9: **return** The combination of highest performance: $C^m = \underset{C_i \in C}{\text{argmax}}\, acc(C_i)$
```

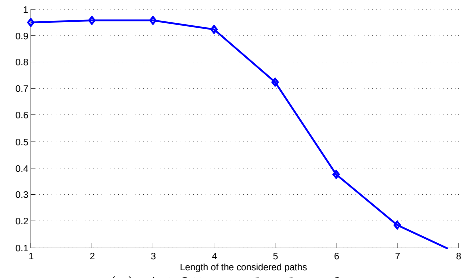
used the implementation described in [29], which provides a way to extract outputs on the three different levels that have been described in Section 6.2.

6.4.1 Ensembles for discrete attributed graphs

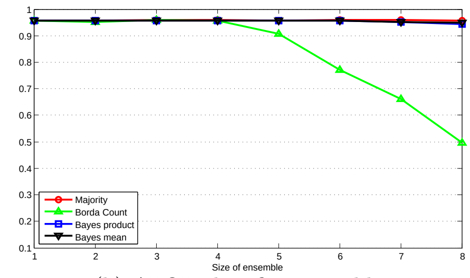
Given the four datasets used in Chapter 3, we build 8 vectorial representations using the edge-based features described in Section 3.4. In particular, we consider path lengths from 1 (this is, the original embedding representation) to 8. For each of these representations, we train the SVM classifier and validate the meta-parameters on the validation sets. In the left column of Fig. 6.1 we show the best result of each of these configurations on the validation sets of each dataset.

The SVM classifiers for the object datasets show a trend where, for small values of the path length, the accuracy is maintained and eventually drops significantly. Such a decay might be explained by the fact that larger lengths of the paths introduce much more noise that finally impoverishes the embedding representations. Although the shape of the curves is the same for the AIDS dataset, here again   as in other situations   , the corresponding results are stable and very saturated almost classifying correctly all elements in the validation set. However, they decrease a bit after $k = 5$. For the MUTAG dataset, interestingly, we observe a zigzag pattern of the curve. On odd values of the path lengths the performance of the classifiers is better than on even values, overall, with the tendency to decrease. An explanation for this situation is that even lengths for the paths tend to trace back, walking along the same edges with higher frequencies, thus incorporating more noise to the features and eventually to the vectorial representations.

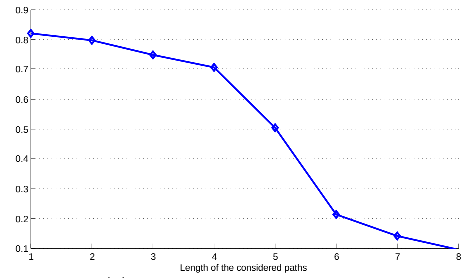
In comparison to the kNN results (see Fig. 3.9, curves for the $\epsilon\ ^2$ distance), we observe that for the object datasets, the SVM classifiers obtain better results for those features that are based on short paths than for those based on longer paths. For the molecules datasets, this behavior is not observed. For the MUTAG dataset, we can see that the SVM results are always better than the corresponding kNN ones. Finally, let us highlight the fact that in the ALOI and the MUTAG datasets results for path lengths of $k = 2$ or $k = 3$ outperform those of $k = 1$. These behaviors are not observed in the kNN curves which means that the SVM classifier is abler to exploit the potential of such features. Nevertheless, these results



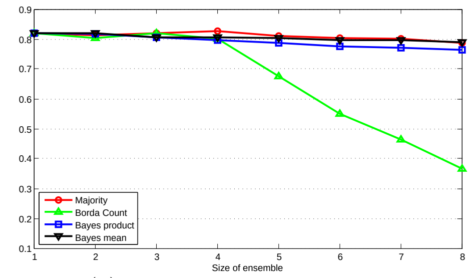
(a) ALOI, single classifiers



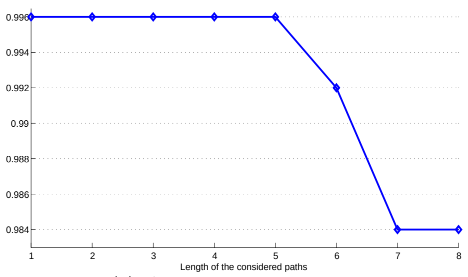
(b) ALOI, classifier ensembles



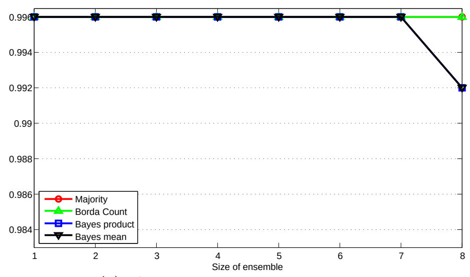
(c) ODBK, single classifiers



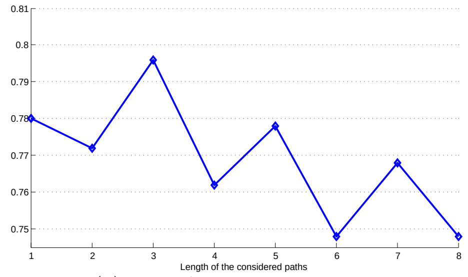
(d) ODBK, classifier ensembles



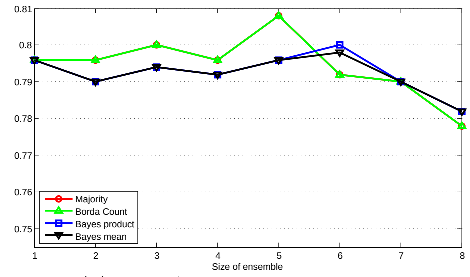
(e) AIDS, single classifiers



(f) AIDS, classifier ensembles



(g) MUTAG, single classifiers



(h) MUTAG, classifier ensembles

Figure 6.1: ϵ^2 SVMs. Left column: accuracy as a function of the length of the paths considered for the edge-based features. Right column: ensembles of the classifiers.

Table 6.1: Ensemble distributions for the discrete datasets.

	Combination strategy			
Dataset	Voting	Borda Count	Bayes product	Bayes mean
ALOI	2,3,6	2,1,4	2	2
ODBK	1,2,5,7	1	1	1
AIDS	1	1	1	1
MUTAG	3,1,2,5,8	3,1,2,5,8	3,5,4,2,8,1	3,5,4,2,8,1

justify that we keep investigating on these higher order features that are capable of extract better characteristics of graphs than the single binary relations we have originally proposed.

Concerning the ensembles we have built out of the different base classifiers, we show the corresponding performances on the right columns of Fig. 6.1. The figures show the performance of each ensemble as a function of the number of base classifiers it contains. For those datasets where the single classifiers tend to perform stably (ALOI, ODBK and AIDS) the combination of base classifiers do not really obtain any improvements besides the fact that they are capable to keep the same recognition rates for most of the ensembles. On the other hand, the dataset where there is more variability on the results obtained by the base classifiers (MUTAG) does take profit of the combination of classifiers and such a combination is capable to improve the best single base classifier.

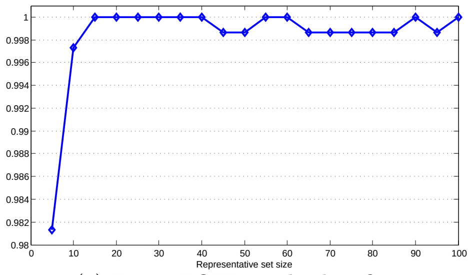
For a more proper insight on what we just said, in Table 6.1 we show, for each dataset and each combination strategy, the ensemble of classifiers that has led to the best performance on the corresponding validation set. In particular, we show the numbers of the length of paths with which we have built the base classifiers that are eventually in the best ensemble.

Those cases where only one base classifier is shown are the ones where no ensemble could provide an improvement over the best base classifier. If two ensembles give tied results, we assume a better performance for the one formed by less base classifiers. In the case of the AIDS dataset as happened many times so far , no improvements are obtained since the best base classifier (the original vector representation considering 1-length paths) is already saturated at the top and further accuracy gains are complicated. Regarding the MUTAG dataset, we can see what we have discussed concerning the performance curves of the ensembles. Five or six base classifiers are combined together for the final ensemble and they lead to higher accuracy rates. The interesting thing to see here is that the *Voting* and the *Borda Count* strategies start considering short length paths (1 and 2) after the best base classifier (3) while the Bayes strategies select longer path features (5 and 4).

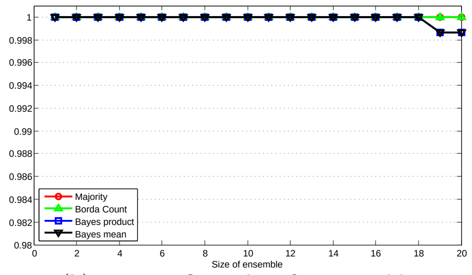
With respect to the object datasets the results are half way between the two extremes that the molecule datasets exhibit. The ODBK dataset can be improved by the *voting* strategy using base classifiers of short (1 and 2) and long (5 and 7) length path. Nevertheless, the improvements are not significant and for the other combination schemes the best base classifier is still more optimal than any ensemble. Also for the ALOI dataset, the Bayes strategies do not consider other classifiers but the best base one and the voting and Borda count combinations select diverse in terms of path length base classifiers but the improvements gained are not significant.

6.4.2 Ensembles for continuous attributed graphs

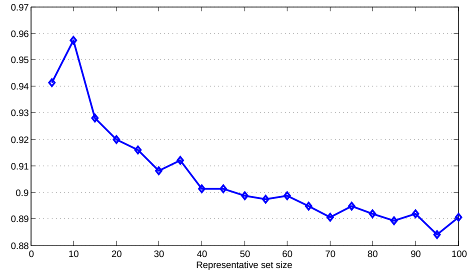
Given the seven datasets of continuous attributed graphs described in Section A.2.1 and used in Chapters 4 and 5, we build different vectorial representations for each of them in order to



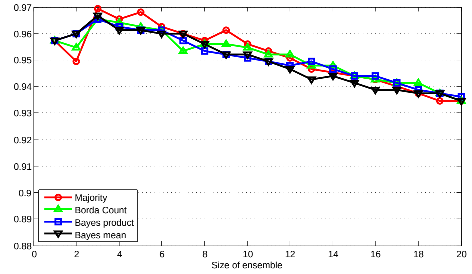
(a) Letter LOW, single classifiers



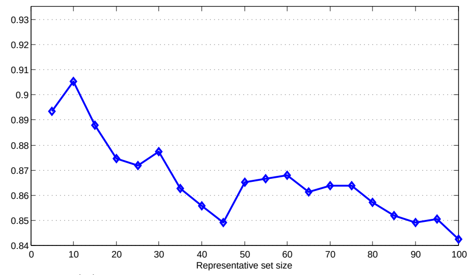
(b) Letter LOW, classifier ensembles



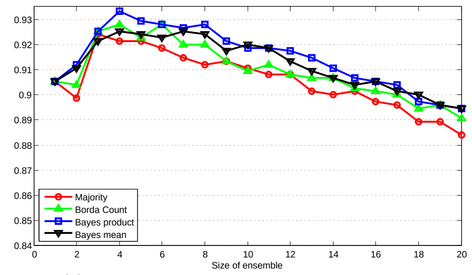
(c) Letter MED, single classifiers



(d) Letter MED, classifier ensembles



(e) Letter HIGH, single classifiers

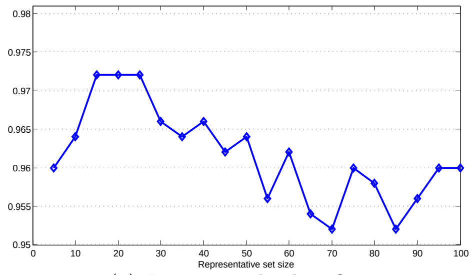


(f) Letter HIGH, classifier ensembles

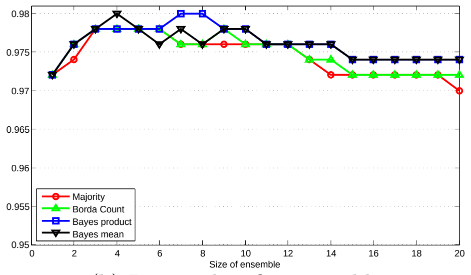
Figure 6.2: ϵ^{2} SVMs. Left column: accuracy as a function of the size of the representative set. Right column: ensembles of the classifiers.

build different base classifiers. In particular, for each graph dataset, we use again the fuzzy kMeans representative constructor and the *all* edge assignment methodology to construct 20 different representations based on 20 different sizes of the set of representatives. With these vectorial representations we learn a ϵ^{2} SVM model and the classification performance is shown for all of them on the left columns of Figs. 6.2 and 6.3.

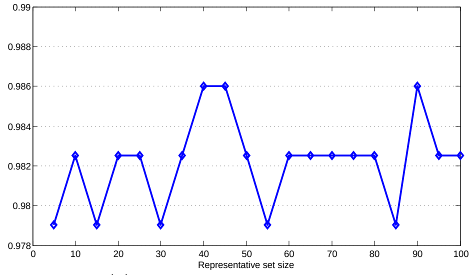
A general conclusion that can be extracted from these graphs is the fact that, as happened in the discrete attributed case, the SVM classifier has more potential than the kNN one in order to extract the best out of the features we propose. If we compare these results with those for the kNN classifier (Figs. 4.3 and 4.4, fuzzy kMeans with *all* edge assignment) we can see that the performance for all datasets has increased. For instance, the Letter MEDIUM and HIGH datasets have gone from low accuracy rates at around 0 4 for large sets of representatives up to performances of around 0 85.



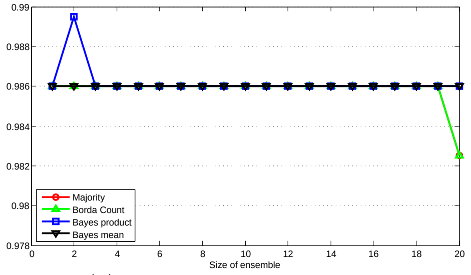
(a) Digits, single classifiers



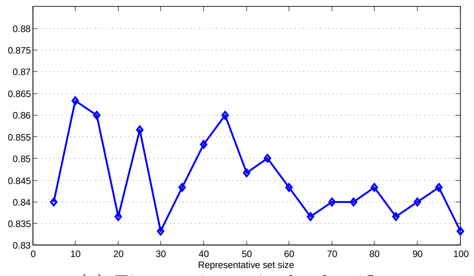
(b) Digits, classifier ensembles



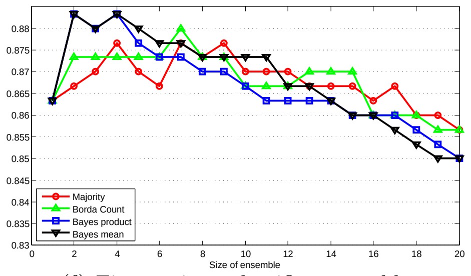
(c) GREC, single classifiers



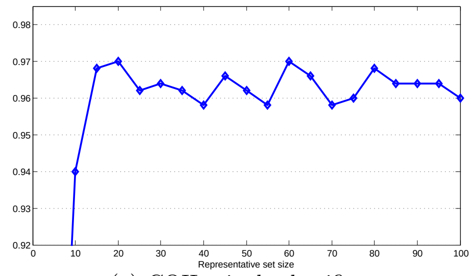
(d) GREC, classifier ensembles



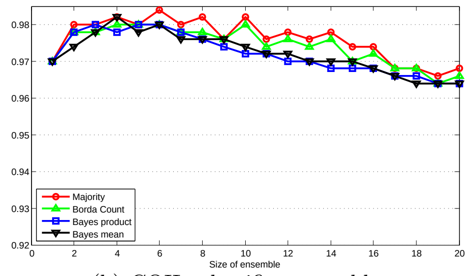
(e) Fingerprints, single classifiers



(f) Fingerprints, classifier ensembles



(g) COIL, single classifiers



(h) COIL, classifier ensembles

Figure 6.3: ϵ^2 SVMs. Left column: accuracy as a function of the size of the representative set. Right column: ensembles of the classifiers.

Table 6.2: Ensemble distributions for the continuous datasets.

	Combination strategy			
Dataset	Voting	Borda Count	Bayes product	Bayes mean
Letter LOW	{15}	{15}	{15}	{15}
Letter MED	{10,5,90}	{10,5,90}	{10,5,90}	{10,5,90}
Letter HIGH	{10,5,55}	{10,5,25,50}	{10,5,60,25}	{10,5,20,85}
GREC	{40}	{40}	{40,65}	{40}
Digits	{15,20,30}	{15,30,20}	{15,20,45,30,5,10,35}	{15,20,5,35}
Fingerprints	{10,45,80,55}	{10,45,80,55,15,50,25}	{10,45}	{10,45}
COIL	{20,60,45,5,15,100}	{20,25,45,15}	{20,45,25}	{20,25,5,45}

In any case, the main conclusions remain the same. This is, there is no size for the set of representatives that can be chosen *a priori* in a general setting and such a parameter should be validated for each dataset. We do observe though a major improvement when combining the base classifiers we have just discussed. The result curves are shown on the right columns of Figs. 6.2 and 6.3.

With respect to the Letter datasets we see that the low distorted version is perfectly classified. Indeed, the best base classifier already obtained a perfect classification score. The other two Letter datasets gain quite a lot on performance while maintaining acceptable rates for large ensembles. In any case, the best results and this is a general behavior are obtained by ensembles of a few base classifiers (see discussion regarding the next table). The very same effect is also observe on the other datasets. The diversity of the classifiers provides enough information to boost the performance of the base classifiers by combining some of them. Only in the GREC dataset we can observe that no major improvement is achieved although some stability is observed.

More insight can be extracted by checking Table 6.2, where we show the resulting best ensembles for each of the datasets and combination strategy. In particular we identify each base classifier by the size of the corresponding set of representatives that it is built from. Besides the already mentioned cases of the Letter LOW and the GREC datasets the former is perfectly classified while the later collapses at some point we can see in the table that the considered combination strategies are able to acknowledge the diversity of the base classifiers so they can improve them. It is also worth noting that the sizes of the corresponding representative sets are usually from different ranges, which supports the initial idea that such differences in the sizes would provide different meaningful information that would eventually be combined to obtain richer representations for the graphs.

6.5 Discussion

In this chapter we have proposed a way of exploiting both local and global information of graphs in terms of the embedding features. In particular, for the discretely attributed graphs, the different representations based on different path lengths for the edge-based features offer both local relations (short paths) and global information (long paths). These informations for the continuously attributed graphs are based on the size of the sets of representatives, where large ones offer local information about the topology of graphs and small ones global relations.

As we already pointed out before, the higher order edge-based features need a refinement of their computation in order to avoid adding too much noise. This way, an eventual situation where diverse classifiers are available can occur and thus we can take more profit out of them by combining them in a multiple classifier system. On the contrary, the ensembles of classifiers for the continuous case has been shown to add quite some more information with respect to the base configurations, boosting their performance and suggesting the suitability of this methodology.

As a last comment, interestingly, both in the discrete and continuous cases, for those selection strategies that offer a diverse ensemble   3 or more base classifiers   the classifiers that appear in the ensembles are usually the same. This means that there are some of the representations that are potentially more discriminant than others and thus all selection strategies are able to detect them.

Chapter 7

Experimental Evaluation

In this chapter we assess the performance of all the embedding configurations defined along this thesis in comparison with classic and state-of-the-art graph matching methodologies. In particular, classification of all datasets described in Chapter **??** is done by means of two classifiers such as kNN and SVM. In the graph domain, the comparison is made with respect to graph edit distance. In particular, a kNN classifier is used with the resulting distances and an SVM one in conjunction with a kernel function directly computed from the edit distance values. In the vector domain, we select as a reference system another embedding methodology also based on the edit distance that has reported sound and consistent results. For this other embedding, we also compare our results with a kNN and an SVM classifiers. We also experimentally assess the computational complexity of our methodology with respect to the reference systems. Finally, we perform graph clustering experiments within the ICPR Graph Embedding Contest framework, for which we redefine the embedding features in terms of a domain-dependent scenario.

7.1 Discrete Attributed Graphs Classification

In this section we compare the embedding methodologies for discrete attributed graphs described in Chapter 3 with two reference systems. First, we use a kNN classifier together with the graph edit distance and compare the results with those of the same classifier on the propose vectorial representations of graphs using a ϵ^2 distance. Also, we will see the performance of the statistics based embedding under an SVM classifier with a ϵ^2 kernel in comparison with a graph kernel also using SVMs.

First, we review the technical details of the reference systems used in this section. We then present an experimental evaluation of the computational runtime of both the reference systems and our methodologies. Finally, we present and discuss the results obtained.

7.1.1 Reference Systems

Graph Edit Distance

In Section 3.2.1, the edit cost function was formulated in such a way that node and edge costs where assumed to have a unitary value. Then a weighting parameter was introduced in order to calibrate the effect of both types of operations. Here use a more refined version of such edit costs functions. In particular, instead of a cost 1, we give to node deletions and insertions a cost of ω_n (respectively ω_e for edges) and twice this cost to nodes substitutions when the corresponding labels are different. On top of that, we also introduce a parameter $\rho \quad [0 \; 1]$ to balance the amount of weight given to either node costs or edge costs. All in all, the edit cost function used here can be summarized as follows:

- Edit costs for nodes:

$$c(u \quad \epsilon) = c(\epsilon \quad v) = (1-\rho)\cdot\omega_n$$
$$c(u \quad v) = \begin{cases} 0 & \text{if } \mu(u) = \mu(v) \\ 2\cdot(1-\rho)\cdot\omega_n & \text{otherwise} \end{cases}$$

- Edit costs for edges:

$$c(e_1 \quad \epsilon) = c(\epsilon \quad e_2) = \rho\cdot\omega_e$$
$$c(e_1 \quad e_2) = \begin{cases} 0 & \text{if } \nu(e_1) = \nu(e_2) \\ 2\cdot\rho\cdot\omega_e & \text{otherwise} \end{cases}$$

The validation of the triplet $(\omega_n \; \omega_e \; \rho)$ is done using the validation sets of the datasets under study. Several values for the parameters have been tried using a grid structure. Those triplets that provided the best results on the validation sets are the ones that are eventually used for the test sets.

The dissimilarity kernel

In order to move from the graph domain where almost only kNN-based algorithms are applicable to a more complex setting we make use of a graph kernel. This graph kernel will allow us to use a kernel machine such as the SVM. In [122], graph edit distance is used to define several graph kernels. The idea is to convert a dissimilarity measure into a similarity measure by a monotonically decreasing transformation.

Definition 7.1 (Similarity Kernel) *Given two graphs $g_1 = (V_1 \; E_1 \; \mu_1 \; \nu_1)$ and $g_2 = (V_2 \; E_2 \; \mu_2 \; \nu_2)$, the similarity kernel based on the edit distance is defined as*

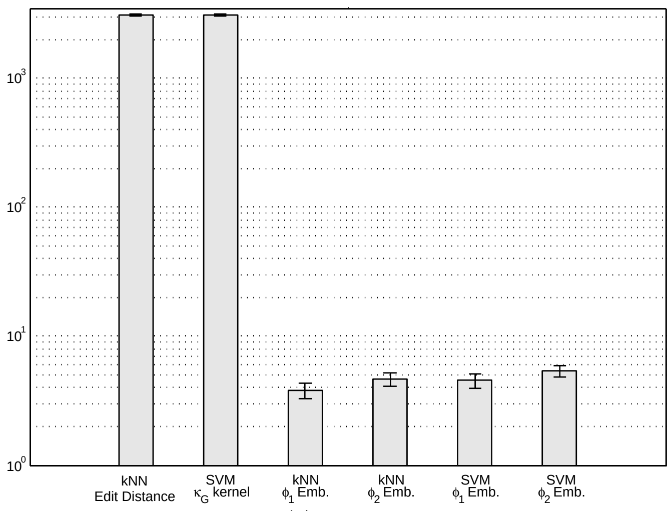
$$\kappa_G(g_1 \; g_2) = \exp\left(-\gamma\cdot d(g_1 \; g_2)\right) \tag{7.1}$$

where $d(g_1 \; g_2)$ is the edit distance between graphs g_1 and g_2, and γ is a positive real-valued parameter, $\gamma > 0$.

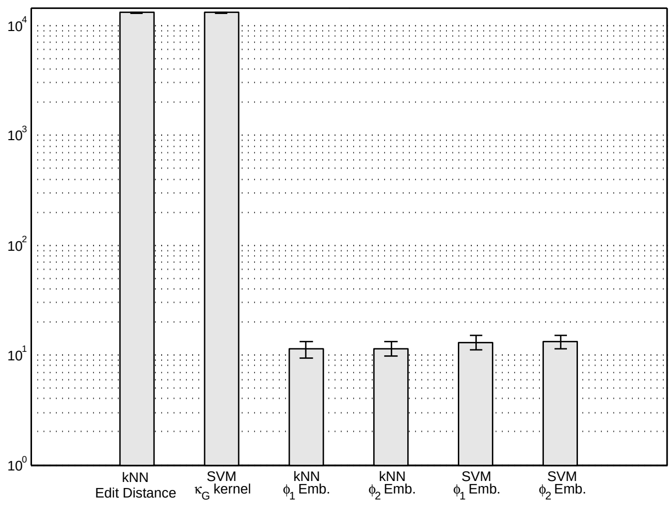
Under this kernel, very dissimilar graphs (large edit distance) are given a low value, close to zero, while similar graphs (low edit distance) will produce kernel values close to one. This kernel function is, however, not positive definite since graph edit distance is not generally a proper metric. Nevertheless, as pointed out in [122] and [65], there is evidence that proves the utility of indefinite kernels in conjunction with kernel machines under certain specific conditions. In any case, among all possible kernels we could have used as reference systems, this kernel function has been selected due to its simplicity and to the fact that no extra expensive computations have to be done with respect to the edit distances.

7.1.2 Computational Time Analysis

Before analysing thoroughly the specific results, we want to put special emphasis on the computational time that all of the involved systems need at the testing phase. We do not consider the time required for applying the corresponding classifiers as it is the same in all



(a) ODBK



(b) MUTAG

Figure 7.1: Average times in milliseconds (vertical axis, logarithmic scale) of the test stage of both the reference and the proposed systems. The length of the ticks on the top of the bars show the variance around the mean values.

representations. We just consider the time required for the distance and kernel computations and the construction of the embedding representations of the graphs. For a test graph to be evaluated under the first reference system, we need to compute the graph edit distance to all graphs in the training set. Using these distances, kNN can be applied. For SVM using kernels on graphs, in addition to the computation of edit distances, we have to exponentiate them to obtain the kernel values (Eq. (7.1)). Regarding the embedding methods, the first

Table 7.1: Discrete attributed graph datasets. Classification results on the test sets. Accuracy rates given in %.

	Reference Systems		Embedding Systems			
Dataset	kNN Edit Distance	SVM κ_G kernel	kNN φ_1 Emb.	kNN φ_2 Emb.	SVM φ_1 Emb.	SVM φ_2 Emb.
ALOI	85.8	91.0 ①	92.2 ①	90.0 ①	94.8 ①②	94.2 ①
ODBK	70.0	80.0 ①	76.6	73.6	80.0 ①	81.3 ①
AIDS	94.9	97.0 ①	98.4 ①②	98.4 ①②	98.5 ①②	99.2 ①②
MUTAG	66.9	68.6	73.6 ①②	74.9 ①②	75.1 ①②	76.4 ①②

①/② Statistically significant improvement over the first/second reference system (Z-test, $\alpha = 0.05$).
❶/❷ Statistically significant deterioration over the first/second reference system (Z-test, $\alpha = 0.05$).

step is always to construct the vector representation. Afterwards, in order to apply the kNN classifier, the ϵ^2 distances of a test element to all training elements have to be computed. In the case of the SVM classifiers, there is a final step in which kernel values are computed (Eq. (5.6)). In Fig. 7.1, we show results on two datasets, ODBK and MUTAG. Those are the two sets with the smallest and the largest average number of nodes per graph, respectively. We can see that the relative behavior of the graph-based versus the embedded vector representations is basically the same independently of the dataset and the size of the graphs.

One can observe that the proposed embedding systems are several orders of magnitude faster when compared to the reference systems. Even in the most favorable case for graph edit distance computation ODBK, which has the smallest average number of nodes per graph , the proposed embedding systems are about 10^3 times faster than the reference systems. In all other datasets, we observe a similar behavior. Apart from that, validating the parameters of the reference systems is indeed a hard task. For instance, using a Java implementation, to obtain the optimal values of the graph edit distance for the two object datasets took around 10 days on a computer cluster using 24 CPUs of 2.4GHz each. On the other hand, in the proposed embedding methodology, one has almost no parameters to tune at the training stage besides those related to the kNN and the SVM classifiers, which took in the worse case just a few minutes under a MATLAB implementation.

7.1.3 Results

Next, we consider the recognition performance of all systems. In Table 7.1 the recognition rates of all classifiers on all datasets are shown. First, we notice that SVM results always outperform the kNN ones, in all databases and in both the $_1$ and $_2$ representations (Eqs. (3.10) and (3.11)). This result is most likely due to the strength of the SVM classifier.

Another interesting observation already made in the validation of the methodology in Chapter 3 is that, for the object datasets, the performance of the embedding representation in which edge labels are not taken into account ($_1$) usually outperforms that of the representation in which this particular information is taken into account ($_2$). Theoretically, one would expect it the other way around since more information is being included in the feature vector representation when edge labels are taken into account. However, by considering such features, the structural information of the graphs falls apart, and the relevant information is divided into many isolated pieces that eventually may lead to a lower performance. This behaviour may also be explained by the larger dimensionality of the $_2$ representation with respect to $_1$ because larger vectors are more prone to overfitting. This situation is beneficial for the proposed methodology in the sense that the resulting feature vectors of the embedding systems have less components (less features are taken into account) and thus the complexity of the classifiers that are eventually used is drastically reduced. With respect to the molecule datasets, this behavior is not observed due to the fact that molecule graphs are less populated with edges than object graphs. Therefore, the described effect of dividing the structural information into many pieces does not occur.

Regarding the comparison of the proposed embedding methodology with the reference systems, we see how the proposed graph embedding procedures improve the recognition rates on all datasets. For the kNN classifiers, results using either of the two configurations of the embedded vectors are always better than their counterpart in the graph domain. For the SVM classifiers, the worse case is a tied result of the reference system with the $_1$ representation in the ODBK dataset. Indeed, there is no statistically significant deterioration in any of the four datasets with respect to the reference systems, and actually three of all the four improvements are statistically significant.

Even in case no significant improvement over any of the reference systems is achieved, the proposed approach still seems to be a better option since the computational time is significantly lower than that of the reference systems. Computing the suboptimal solution of the edit distance [133] between two input graphs is a procedure that requires a cubic number of operations in the number of nodes of the involved graphs. In order to classify any graph in the test set, this computational procedure has to be done against all elements in the training set, and this certainly requires much more operations than just arranging the graphs in the vectorial form that is proposed in this thesis and then computing the distances among them. Besides this, the number of parameters to validate is higher in both reference systems and requires quite some time for training.

Ensembles of higher-order edge-based representations

The last experiment with discrete attributed graphs is the one concerning the higher-order edge-based features defined in Section 3.4 and the corresponding ensembles of classifiers. In particular, we compare the original $_1$ embedding representation which is the particular case of considering paths of length $k = 1$ for the edge-based features to the configuration for a certain path length k that provides the best result on the validation sets (Section 3.4.2). For those cases, we compute the corresponding features for the graphs in the test sets and apply an SVM classifier.

After that, we build the ensembles of classifiers that provided the best results on the validation sets. The base classifiers were reported in Table 6.1. All these results are now shown in Table 7.2. Note that we do not report the result of the best path length (second column) whenever this parameter coincides with $k = 1$. This is the case where higher-order edge-based features do not contribute to the original representation in terms of the performance. We do not show either the results of the ensembles of classifiers when the optimal ensemble is composed of only one base classifier.

We discuss the results of each dataset separately. With respect to the ALOI dataset, we observe that the original representation gives better results than that of the best path length. This is not an incoherence since the concept of *best* is based on the classification performance on the validation sets, and in this case, the performance of the edge-based features for a certain $k > 1$ provided better results than those for $k = 1$. The same thing happens regarding the ensembles of classifiers, where results are lower than the original representation. Although results are not statistically significantly deteriorated, in this particular case, the outcome of the ensemble methodology suggests not to go further on this track.

The same thing happens for the ODBK and AIDS datasets. No improvements are obtained and a non statistically significant deterioration is obtained with respect to the

Table 7.2: Performance of the ensembles of classifiers for the discrete attributed graph datasets. Single classifier results are shown only if $k = 1$. Multiple classifier results are shown only if the resulting ensemble is compounded of more than 1 base classifier.

	Single Classifiers		Multiple Classifiers			
Dataset	$\varphi_1(k=1)$	Best k	Voting	Borda Count	Bayes product	Bayes mean
ALOI	94.8	93.2	92.4	93.6	-	-
ODBK	80.0	-	78.6	-	-	-
AIDS	98.5	-	-	-	-	-
MUTAG	75.1	76.1	78.1 ✓	78.1 ✓	77.1	77.3

✓ Statistically significant improvement over the φ_1 representation (Z-test using $\alpha = 0.05$).
✗ Statistically significant deterioration over the φ_1 representation (Z-test using $\alpha = 0.05$).

results of the original representation. On the other hand, the MUTAG dataset has experienced a statistically significant improvement with respect to the 1-length edge-based features after combining several classifiers. This is due to the diversity the different base classifiers have shown (see Fig. 6.1(g)) and the corresponding discussion) which allow, not only the improvement of the best k-length path based-features with respect to the original representation, but also of the ensembles of the classifiers.

7.1.4 Discussion

We have proposed embedding formulations of discrete attributed graphs based on counting appearances of node labels and appearances of specific edges between them. We have shown that they have a very efficient computational complexity with respect to well-known graph matching methodologies. Also with respect to these reference systems, we have seen how by means of the proposed embedding features we can obtained statistically significant improvements in several datasets of graphs of rather different nature. In summary, the simplicity of the embedding representations, both conceptually and computationally, and their good performance on classification problems when compared to classical and modern graph matching approaches make them an attractive new tool for graph-based pattern recognition problems.

Regarding the edge-based features and their formulation in terms of paths of length k between nodes with certain labels, there is still large room for improvement. As already discussed before, this is just an idea in an embryonic stage which might lead in the future to more robust features able to cope with global descriptions of the graph structures. Possible ways to seek such improvements would have to go through those research directions in where paths do not walk through the same edges twice. The fact that this is now not considered introduces noise to our representations and thus impoverishes the classification performances. Nevertheless, the statistically significant improvement in one of the datasets encourages us to keep investigating on this line.

7.2 Continuous Attributed Graphs Classification

A comparison of the proposed embedding methodologies for continuously attributed graphs with state-of-the-art graph matching algorithms is described in this section. In particular, as we did for the discretely attributed case, we first use the graph edit distance together with a kNN classifier in order to see which are the benefits and disadvantages of the embedding representations in the graph domain. We then transit to a more complex learning framework such as SVMs by comparing our approaches with another embedding configuration. We will finally discuss the effect of performing feature selection on the vectorial representations of graphs and combining different classifiers.

7.2.1 Reference Systems

In contrast to the discrete attributed case, the reference systems used in this section are not implemented by ourselves. Instead, we have used the work of [134] and the related publications of the same authors [24 26]. The datasets used are exactly the same and we have followed the exact same evaluation protocols.

Graph Edit Distance

Graph edit distance is computed, as it has been done along this thesis, using the suboptimal approach of [133]. The edit cost functions involved for each dataset are defined with respect to the underlying labelling information. In general, the deletion and insertion of nodes and edges have a fixed cost (ω_n and ω_e respectively) weighted by a parameter α, and the substitution of nodes and edges is proportional to a suitable distance measure between the corresponding labels. Formally,

- Edit costs for nodes:

$$\begin{aligned} c(u \quad \epsilon) = c(\epsilon \quad v) &= \alpha \cdot \omega_n \\ c(u \quad v) &= \alpha \cdot d_n(\mu(u) \ \mu(v)) \end{aligned}$$

- Edit costs for edges:

$$\begin{aligned} c(e_1 \quad \epsilon) = c(\epsilon \quad e_2) &= (1-\alpha) \cdot \omega_e \\ c(e_1 \quad e_2) &= (1-\alpha) \cdot d_e(\nu(e_1) \ \nu(e_2)) \end{aligned}$$

where d_n and d_e are the corresponding distance measures between node and edge labels, respectively (see [134] for details concerning each of the datasets). The triplets $(\omega_n \ \omega_e \ \alpha)$, together with the k parameter in the kNN classifier, are validated on the validation sets and those sets of values that provide the best results on these subsets are eventually used in the test sets.

The Dissimilarity Embedding

Also from [134], we use an embedding methodology based on the dissimilarity representation formalism [124,125]. A graph is represented as a vector whose components are edit distances to a predefined set of prototypes. Formally,

Definition 7.2 (Dissimilarity Embedding) *Let $\mathcal{P} = \ p_1 \qquad p_n$ be a set of graph prototypes. The dissimilarity embedding of a graph g is defined as*

$$\nu_n^{\mathcal{P}}(g) = (d(g \ p_1) \qquad d(g \ p_n)) \tag{7.2}$$

where $d(g \ p_i)$ is the edit distance between the graph g and the prototype p_i.

We will not explain here how the set of prototypes is selected nor how many prototypes were used, but will only report the best results of this approach using a Support Vector Machine on the set of vectors defined by Eq. (7.2). It is worth mentioning, though, that this system has been selected for reference since it has interesting generalization properties and it can handle arbitrary graphs as long as only a dissimilarity measure is required. Besides, it has been proven to obtain remarkable results in graph classification and clustering problems. On top of these things, we use this reference system since, for it, results have been reported on the same datasets used in this work and using the exact same validation settings, so that there was no need to reproduce the experiments again.

7.2.2 Results

In order to make the comparison we use the following setup for our embedding proposals. For each dataset and each of the six embedding configurations summarized in Section 4.2.5, we construct 20 representations based on 20 different sizes of the set of representatives. In particular, we use sizes ranging from 5 up to 100 representatives, in steps of 5. Each one of these 20 representations is built for the validation sets and a *k*NN classifier together with the ϵ^2 distance is applied. For the best set of representatives, this is, for the representative set size providing the best result in such validation sets, we construct the corresponding vectors for the test elements. Using these test vectors we first apply a *k*NN classifier also with the ϵ^2 distance and then we perform SVM classification. The SVM model is learnt on the corresponding validation vectors with a ϵ^2 kernel. The results of the *k*NN classifier in comparison with the graph edit distance ones are shown in Table 7.3. Those for the SVM classifier in comparison to the reference embedding can be found in Table 7.4.

Regarding the reference system in the graph domain, there is at least, in four of the seven datasets, one of the six proposed configurations that is tied or even statistically better than a *k*NN classifier using the graph edit distance. These results suggest that the proposed methodology is able to keep those distances among graphs reflected in the vectorial representations. A tied result compared to the edit distance classifier is indeed a success since the computation of the edit distance is more costly than the methodology we propose in this work. The results that are statistically significantly lower than those obtained with the first reference system can all be explained by the nature of the graphs where these results are gotten from. In particular, we get lower results than the reference system in all configurations of the proposed embedding approach for the medium and high distorted letters and the Digits databases. The graphs in these sets represent objects with an inherent distorted structure, making the assignment of nodes to representatives a confusing step for the proposed approach, and thus obtaining low classification rates. In other words, in those cases where graphs are heavily distorted, the proposed embedding methodology cannot retrieve the inherent topological model that is sought.

With respect to the reference embedding and the SVM classifier, the results of the proposed embedding approach are again adequate. The soft versions of the embedding methodology together with the *all* assignment of edges give rise to accuracy rates that are comparable to that of the embedding reference system. Only in the case of the high distorted letters and the Digits dataset -which are also highly distorted in nature- the results are statistically worse (the medium distorted case of letters is now statistically tied with the reference system). The explanation for this fact is just as the one given for the first reference system. In the other databases, we obtain a tied result (or better in the GREC and COIL cases) which should be considered as a success due to the computational complexity of both methods. Such complexity for the reference system is governed by the computation of the edit distance between each graph and the set of prototype graphs. As reported in [133], the edit distance approximation has complexity of $\mathcal{O}(n^3)$, where n is the size of the involved graphs (number of nodes), plus the fact that such computation has to be performed for every prototype graph. In our case, the complexity of the graph description by a feature vector has to be performed by visiting the n nodes and the m edges of a graph, thus obtaining a complexity of $\mathcal{O}(n+m)$, plus the fact that each node label has to be compared with the set of representatives. We will not here report test evaluation runtime as it has been done for the discrete case (Section 7.1.2), but the behavior is analogous to the one described there.

Despite the good results obtained for the soft versions of the proposed embedding, together with the *all* edge assignment, it should be noted that the other four representations are not generally capable to get comparable results to the second reference system. Only by fuzzifying the node (Fuzzy kMeans and GMM) and the edge assignments (*all* method, Eq. (4.9)), we are capable to compete with the chosen reference system. Nevertheless, these remarkable results, together with its low computational complexity, foster the consideration of the described embedding methodology of graphs into the set of algorithms that are able to bridge the gap between the structural representation of objects by means of graphs and the statistical pattern recognition field.

Table 7.3: *k*NN results on the test sets of the continuous attributed graph datasets for the different embedding configurations. The best result for each dataset is shown bold face.

	Reference system	Embedding configurations					
Dataset	*k*NN - Graph Edit Distance	Spanning Hard	*k*Means Hard	Fuzzy *k*Means Soft *max*	Fuzzy *k*Means Soft *all*	GMM Soft *max*	GMM Soft *all*
Letter LOW	99.3	98.1 ✗	99.1	99.5	**99.7**	99.2	99.5
Letter MED	94.4	68.0 ✗	84.3 ✗	85.6 ✗	88.4 ✗	87.6 ✗	**90.7** ✗
Letter HIGH	89.1	60.9 ✗	78.0 ✗	79.7 ✗	**80.8** ✗	80.3 ✗	79.9 ✗
GREC	95.5	93.9	95.6	96.9	96.6	98.5 ✓	**98.7** ✓
Digits	97.4	81.7 ✗	82.7 ✗	86.7 ✗	**93.1** ✗	81.5 ✗	84.0 ✗
Fingerprints	79.1	77.7	79.8	76.1 ✗	80.6	**82.1** ✓	80.6
COIL	93.3	89.1 ✗	91.8	93.2	96.6 ✓	91.7	**98.3** ✓

✓ Statistically significant improvement over the reference system (Z-test using $\alpha = 0.05$).
✗ Statistically significant deterioration over the reference system (Z-test using $\alpha = 0.05$).

Table 7.4: SVM results on the test sets of the continuous attributed graph datasets for the different embedding configurations. The best result for each dataset is shown bold face.

	Reference system	Embedding configurations					
Dataset	SVM - Dissimilarity Embedding	Spanning Hard	*k*Means Hard	Fuzzy *k*Means Soft *max*	Fuzzy *k*Means Soft *all*	GMM Soft *max*	GMM Soft *all*
Letter low	99.3	99.0	99.2	99.6	**99.8**	99.4	99.7
Letter medium	94.9	75.4 ✗	88.4 ✗	88.1 ✗	92.8	90.0 ✗	**93.0**
Letter high	92.9	67.2 ✗	82.0 ✗	85.0 ✗	87.7 ✗	84.6 ✗	**87.8** ✗
GREC	95.1	97.7 ✓	97.9 ✓	97.9 ✓	98.1 ✓	**99.2** ✓	99.0 ✓
Digits	98.7	88.9 ✗	89.3 ✗	91.3 ✗	**97.1** ✗	87.0 ✗	91.4 ✗
Fingerprint	83.1	79.7 ✗	81.5	80.4 ✗	81.5	**82.0**	81.8
COIL	96.8	92.1 ✗	93.1 ✗	92.9 ✗	97.3	93.5 ✗	**98.1** ✓

✓ Statistically significant improvement over the reference system (Z-test using $\alpha = 0.05$).
✗ Statistically significant deterioration over the reference system (Z-test using $\alpha = 0.05$).

Feature Selection

In Chapter 5 we have assessed how the embedding representations behave under feature selection methods. We now here evaluate the same strategies on the test sets of continuously attributed graphs. Recall that, once all the embedding configurations have been evaluated, we keep on working only with the one build by the Fuzzy kMeans representative set constructor and the *all* edge assignment method. For this embedding configuration we also need to validate the size of the set of representatives as well as the portion of feature we end up keeping for the final vector representation.

From the same 20 different representations in the previous section based on 20 different sizes of the set of representatives we validate both this size and the corresponding number of features we keep. We do it by using a kNN classifier with the ϵ^2 distance in the case of the ranking methods and the same classifier with the Euclidean distance in the case of the PCA-based algorithms. We assume the best representation to be the one that has provided results at the same level of statistical significance (Z-test, $\alpha = 0\ 05$) as the maximum number achieved by any of the representations (see Section 5.2 for more validation details). Using this representation, we build up the test elements by, first, constructing the whole vectorial representation and, second, reducing the dimensions to the optimal rate. The test evaluation is done using a ϵ^2-kNN classifier and a ϵ^2-SVM classifier for the ranking methods and a Euclidean kNN classifier and a linear SVM classifier for the transformation-based feature selection methods. The former results are shown in Table 7.5 while the latter in Table 7.6.

With regard to the kNN results, we observe the same phenomenon as in the previous section, in which those datasets with an inherent distorted nature are properly classified under any of the proposed configurations. As we discussed above, the embedding methodology is not capable to extract a robust representation for distorted graphs and the feature selection methods used in this work do not either solve such inconvenient. Nevertheless, for the rest of the datasets, we still get statistically significant improvements with respect to the graph edit distance, which suggests that the feature selection step keeps considering those features that are able to emulate and improve the graph similarities that edit distance is providing.

Concerning a comparison between ranking and PCA-based methods, it is clear that those methodologies that transform the original features obtain worse results than those that just rank them. In general, results are lower for the PCA-based methods, but the best results in each dataset are certainly obtained by the ranking methodology and mostly all statistically significant improvements are also reported by this methods. In any case, among the PCA methodologies the ϵ^2 version is the most stable: less significant deteriorations, more significant improvements, more times ranking the best of the three considered.

The SVM results are similar to the ones just described, although in comparison to the reference system, we impoverish the performance. Despite there is a statistically significant tie in the medium distorted case of the Letters dataset, results are still bad with respect to the reference embedding for those datasets of distorted nature. In this case, also the Fingerprint dataset is drastically affected. The explanation is strictly related to that of the same situation in the kNN case, and it is due to the broader flexibility of graph edit distance to structural deformations compared to the proposed methodologies. This correlation between the classifiers is reinforcing the idea that the features we propose might not be a good option for these cases. Anyhow, some statistically relevant results are still obtained, and for those cases, the SVM classifier can properly exploit the dimensionality reduced versions of our embedding methodology, which is also a great success due to the low complexity of the embedding configurations we have proposed.

Table 7.5: kNN results on the test sets of the continuous attributed graph datasets after feature selection. The best result for each dataset is shown bold face.

	Reference System	Feature Ranking			PCA-based methods		
Dataset	kNN - Graph Edit Distance	Relief	Info	SVMrank	Linear PCA	*rbf* PCA	χ^2 PCA
Letter LOW	99.3	99.1	99.5	**100** ✓	96.7	98.9	98.9
Letter MED	94.4	**88.4** ✗	85.9 ✗	86.4 ✗	72.1 ✗	76.5 ✗	80.5 ✗
Letter HIGH	89.1	80.8 ✗	**80.9** ✗	70.1 ✗	67.9 ✗	69.9 ✗	69.2 ✗
GREC	95.5	96.4	**98.7** ✓	95.5	93.6	94.9	96.6
Digits	97.4	89.5 ✗	**92.5** ✗	89.8 ✗	82.3 ✗	86.3 ✗	71.5 ✗
Fingerprints	79.1	77.6	77.7	78.8	79.5	**80.5**	77.3
COIL	93.3	97.0 ✓	**97.6** ✓	96.8 ✓	71.1 ✗	79.6 ✗	96.3 ✓

✓ Statistically significant improvement over the reference system (Z-test using $\alpha = 0.05$).
✗ Statistically significant deterioration over the reference system (Z-test using $\alpha = 0.05$).

Table 7.6: SVM results on the test sets of the continuous attributed graph datasets after feature selection. The best result for each dataset is shown bold face.

	Reference System	Feature Ranking			PCA-based methods		
Dataset	SVM - Dissimilarity embedding	Relief	Info	SVMrank	Linear PCA	*rbf* PCA	χ^2 PCA
Letter LOW	99.3	99.6	**99.9**	99.7	98.0 ✗	99.3	99.7
Letter MED	94.9	**92.8**	88.8 ✗	90.5 ✗	85.5 ✗	80.9 ✗	85.5 ✗
Letter HIGH	92.9	87.7 ✗	**88.4** ✗	82.1 ✗	81.2 ✗	78.4 ✗	78.5 ✗
GREC	95.1	97.0	97.9 ✓	96.8	95.5	96.0	**98.3** ✓
Digits	98.7	94.8 ✗	**96.0** ✗	94.1 ✗	87.3 ✗	90.1 ✗	67.5 ✗
Fingerprints	83.1	79.1 ✗	80.1 ✗	81.5 ✗	**80.3** ✗	79.6 ✗	79.9 ✗
COIL	96.8	97.0	97.2	**97.4**	86.5 ✗	59.7 ✗	82.0 ✗

✓ Statistically significant improvement over the reference system (Z-test using $\alpha = 0.05$).
✗ Statistically significant deterioration over the reference system (Z-test using $\alpha = 0.05$).

Table 7.7: Performance of the ensembles of classifiers for the continuously attributed graph datasets. Results are shown only if the resulting ensemble is compounded of more than 1 base classifier. The best result for each dataset is shown bold face.

	Single Classifier	Multiple Classifiers			
Dataset	Best repr. set	Voting	Borda Count	Bayes product	Bayes mean
Letter LOW	99.7	-	-	-	-
Letter MED	93.8	94.1	94.5	**95.3**	95.0
Letter HIGH	86.8	89.6	89.4	91.2 ✓	**91.7** ✓
GREC	**98.2**	-	-	**98.2**	-
Digits	96.3	97.1	97.1	**97.8** ✓	97.5 ✓
Fingerprints	82.5	82.2	**83.5**	83.1	83.1
COIL	95.6	**97.1**	96.6	96.6	96.7

✓ Statistically significant improvement over the reference system (Z-test using $\alpha = 0.05$).
✗ Statistically significant deterioration over the reference system (Z-test using $\alpha = 0.05$).

Ensembles of Classifiers

In this section we present and discuss the improvement obtained after combining classifiers based on different vectorial representations of graphs. We here again only use the embedding representation based on the Fuzzy kMeans representative set constructor and the *all* edge assignment method. From the diversity obtained using different sizes for the set of representatives, we build a multiple classifier system. In particular, we train ϵ^2 SVMs on the validation sets as described in Section 6.4.2. The optimal ensembles obtained (see Table 6.2) are constructed for the test sets of the databases. We show the results of such configurations in Table 7.7. We also show the results for the best classifier that provided the best result. Whenever the optimal ensemble is only composed of a single base classifier, we do not report the result since it is the same as the best classifier.

First of all, note that the results of the best classifiers do not coincide with those of the corresponding column in Table 7.4. This makes perfect sense since the results shown here are the ones obtained by validating the size of the representative set using an SVM classifier. In the other case, such a step is done using a kNN classifier. Both cases are not necessary the same.

Results of the ensembles of classifiers are generally satisfactory. In most of the cases we obtain an improvement over the representation on the best set of representatives. Specially relevant is the fact that the statistically significant improvements are obtained on those datasets where we have been reporting really bad results in the previous sections. These were the cases with high distorted nature. It seems that by combining diverse base classifiers we are able to boost the performance of the final recognition system. Also worth-mentioning, such improvements are obtained by the two bayesian combination strategies, although the other considered ones are capable of outperforming the best single classifier.

7.2.3 Discussion

In this section we have shown and discussed a set of results that let us say that the proposed embedding methodology for continuously attributed graphs is worth being considered in the broad set of graph matching methodologies available nowadays. First, we have demonstrated to be competitive with two very well known and established algorithms that solve the problem of graph comparison. Not only because they classification performance is higher in many case but also because of the computational complexity of our proposal. We base our embedding methodology in a set of features that regard statistics of labelling information and thus they can be extracted at a low cost.

Nevertheless, we have also discussed some cases where the proposed methodology seems not to be the best option. By assigning node labels to representatives we aim at discovering an inherent topological model of graph categories, undoing possible deformations to such models. In this scenario, it makes sense that heavily distorted graphs do not really fit into our proposal, and graph edit distance, although its much higher complexity, is more capable to detect graph dissimilarities and to properly categorize graphs.

Apart from showing the proper performance of the raw versions of the embedding, we have also investigated how they behave under feature selection methods and how several configurations of the embedding can be combined in order to improve the eventual performance of the classifiers. In particular, we have observed that reducing the dimensionality of the vectorial representations is beneficial for those case where the original formulations already worked properly, since we still get improvement with respect to the reference systems but by using lower dimensional vectors. On the contrary, for those case where the original representations failed due to the distorted nature of graphs, feature selection did not solve the problem. In any case, we do observe an improvement in all cases by means of combining several diverse classifiers.

7.3 Graph Clustering under Domain-specific Embedding: color RAGs

All along this thesis, for the case of continuously attributed graphs, we have not put any assumptions on the nature of the node labelling space, besides the fact that it should be a subset of $\mathbb{R}^d$. We have argued, though, that the proposed methodology should perform well for those graphs whose node labels are $(x\ y)$ coordinate positions, and all the experiments carried on so far have been for graphs of this kind. In this section we want to go one step further and check for the suitability of the proposed methodology for graphs whose node labels are not points on the plane but non-spatial coordinates such as RGB color values. This situations needs for a revision of how the set of representatives is selected, and this is also covered in this section. Moreover, we will evaluate the methodology within the ICPR Graph Embedding Contest, where a clustering measure is the performance evaluation for the embedding configurations. More details are given below.

7.3.1 Motivation

Color region adjacency graphs (RAGs) are a quite common tool for representing object images. Object images are segmented into meaningful regions and each of these regions constitutes a node of a graph labelled with the corresponding region color. Nodes are linked by edges in terms of the neighboring relations of the image segments. We want to apply our methodology to these type of graphs and see how well we can perform.

The first observation to be made is concerning the set of representative elements. Up to this point, the way these elements have been selected assumed an Euclidean space and so clustering approaches served us for our purposes. In the present case, we do not really work on such a vector space since color spaces have a more complex structure. Let us illustrate the process with an example. From a given dataset of color RAGs (see Section A.2.2), in Fig. 7.2(a), we plot a random sample of node labels, this is, a set of RGB values. It can be observed that the point cloud distribution is not uniform nor lineal in the RGB space. Colors are distributed along *ridges* of similar chromaticity from dark to light intensities. In this scenario, it seems natural to select representatives for the node labels as these ridges appear in the RGB space. In other words, instead of considering spherical clusters as representatives, these very same ridges should be the representatives. In Fig. 7.2(b) we show the result of applying the kMeans algorithm for $k = 10$ to the set of points in Fig. 7.2(a). The clusters obtained do not regard the natural structure of the RGB space and thus they seem not to be the best choice.

We propose here to use a color-based approach for selecting the node label representatives. In particular, we put our attention in the color naming theory field to see how this can be done. The aim of color naming scientists is to come up with ways to categorize the whole palette of colors into a finite set of color names. In this work, we use the approach proposed in [9], where each point in the RGB space is automatically assigned to each one of the eleven basic color categories in the color naming theory with a certain degree probability of belongingness. The eleven categories are

Black, Blue, Brown, Green, Grey, Orange, Pink, Purple, Red, White, Yellow

and each RGB point is distributed among these eleven bins by means of a parametric model that has been learnt from psychophysical experiments.

To illustrate that this domain specific approach approaches our desired solution of extracting the existing ridges in the RGB space, in Fig. 7.2(c) we plot the same cloud of points in the RGB space after applying the color naming algorithm. First, each point is assigned to the eleven categories with a degree of probability and then we keep the color with maximum probability as the corresponding color. In the figure, we can appreciate that the inherent structure of the cloud of points is much more exploited than in the kMeans case and, as we will show in the experimental part, this will in general favour our methodology in a graph clustering scenario.

7.3.2 Color Naming based Embedding

Given a color RAG $g = (V \ E \ \mu)$, each node $v \quad V$ is assigned to the color naming categories using the parametric model mentioned above. Each node is thus represented by eleven probability values

$$v - \quad (p_1(v) \qquad p_{11}(v)) \tag{7.3}$$

Note the analogy of this assignment with the one of Eq. (4.5). We can thus define the node-based and edge-based features in the same way for this case,

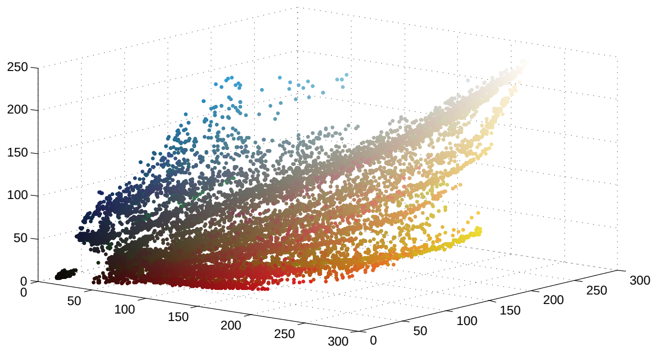
$$U'_i = \sum_{v \in V} p_i(v) \tag{7.4}$$

$$B'_{ij} = \sum_{(u \ v) \in E} p_i(u)p_j(v) + p_j(u)p_i(v) \tag{7.5}$$

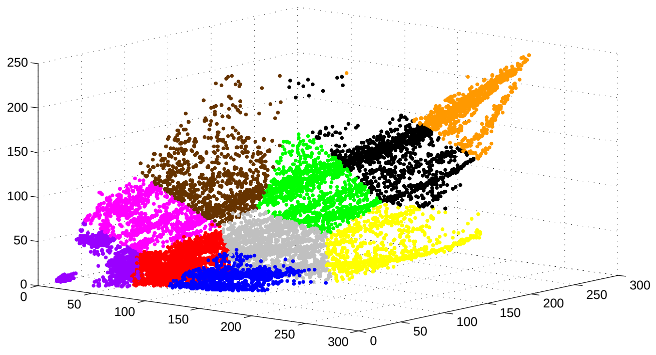
and again define the final embedding representation by the concatenation of all these features together into a unique vectorial form. This embedding configuration will be known as *Soft Color*.

For the sake of completeness, we also want to work with a *hard* configuration of the embedding, as it was defined in Section 4.1.1. To do so, we assign each node to the color that has produced a maximum probability after the color naming assignment. Embedding features are then defined analogously to those for the general case, Eqs. (4.2) and (4.3). This configuration will be referred to as *Hard Color*.

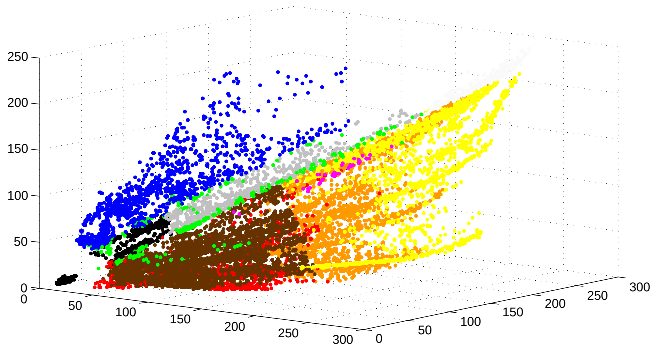
To be able to compare the hypothetical improvements of these configurations over the generic embedding approaches defined in Chapter 4, we will also work with two of them as they were defined there. In particular, we use the kMeans clustering for the construction of



(a) Original Color



(b) *k*Means Clusters



(c) Color Naming

Figure 7.2: Distributions of the graphs nodes in the RGB space. (a) Original color of each node. (b) *k*Means clusters for $k = 10$. (c) Color naming distribution.

the set of representatives (Section 4.2.2) and the corresponding embedding features (this one will be referred to as *Hard kM*) and the fuzzy *k*Means algorithm (Section 4.2.3) together with the *all* edge assignment for the construction of the edge-based features (this configuration is going to be called *Soft kM*).

Summarizing, for the color region adjacency graphs, we have defined four embedding configurations. Two of them are generic ones in the sense that they build a set of representatives using generic clustering algorithms

- *Hard kM*,
- *Soft kM*,

and two domain-specific approaches that try to capture the inherent RGB structure of the labelling space

- *Hard Color*,
- *Soft Color*.

7.3.3 Experimental validation on the ICPR Graph Embedding Contest

To check the performance of these color-based embedding configurations we make use of the ICPR Contest datasets (Section A.2.2) together with its evaluation framework [55]. This contest was organized in order to provide a framework for direct comparison between embedding methodologies for the purpose of graph clustering. Three object image datasets were chosen and converted into graphs, divided into a training and a test set. The participants also received a code with which they could assess their own methodologies in terms of a clustering measure. Object images were first segmented into different regions and a region adjacency graph was constructed. Each node representing a region was attributed with the corresponding relative size and the average RGB color components, while edges remained unattributed. As we have been doing up to now for the case of continuously attributed graphs, we neglect edge attributes.

Given a set of vectors $f_i \;_{i \in I}$ representing a set of graphs $g_i \;_{i \in I}$ of different categories and a distance measure $d(\cdot \;\cdot)$ between two of these vectorial representations, the *C index* is computed [75]. This clustering measure is defined as follows. Let S_w be the set of distances $d_{ij} = d(f_i \;f_j)$ such that g_i and g_j belong to the same class. Let M be the cardinality of S_w. Let also $S_{\min}$ and $S_{\max}$ be the sets of the M shortest distances and the M largest distances among all possible values of d_{ij}, respectively. The *C index* is defined as

$$C = \frac{sum(S_w) - sum(S_{\min})}{sum(S_{\max}) - sum(S_{\min})} \tag{7.6}$$

The smaller is C the better is the separation of the classes in the set of graphs. The index lies in the interval $[0 \;1]$, in which the ideal case $C = 0$ is that where all inter-class distances are smaller than all intra-class distances. Regarding the distance measure used in the contest, explicit formulation of vector space embeddings are compared using the Euclidean distance (Eq. 4.14).

Explicit Embeddings

Regarding the generic configurations of the embedding for the color RAGs we show the clustering performance as a function of the size of the set of representatives. In particular, we have tried sizes from only 2 representatives up to 100. On top of that, and as we have

Table 7.8: Generic *vs* color-based representations. Clustering results on the train sets by the use of different distance measures. The geometric mean of the results on the different datasets is also shown. The best results are shown bold face.

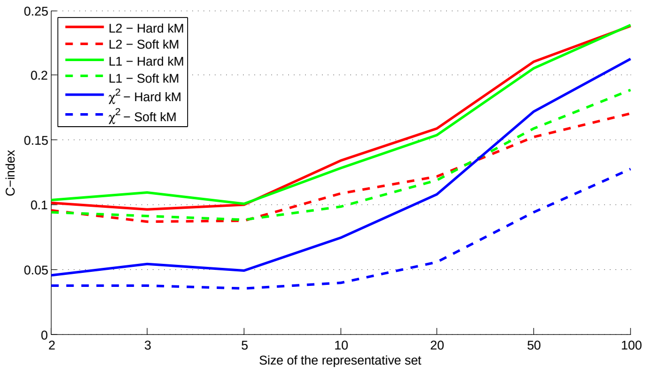
Distance	Representation	ALOI	COIL	ODBK	Geo. Mean
L_2	*Hard kM*	0.096	0.093	0.043	0.073
	Soft kM	0.086	0.093	0.043	0.070
	Hard Color	0.102	0.076	0.049	0.073
	Soft Color	0.088	0.066	0.050	0.066
L_1	*Hard kM*	0.100	0.079	0.051	0.074
	Soft kM	0.088	0.088	0.048	0.072
	Hard Color	0.102	0.071	0.056	0.074
	Soft Color	0.092	0.054	0.054	0.065
χ^2	*Hard kM*	0.045	0.045	0.024	0.037
	Soft kM	**0.035**	0.046	**0.022**	**0.033**
	Hard Color	0.057	0.044	0.033	0.044
	Soft Color	0.043	**0.029**	0.029	**0.033**

been doing along this work, we try not also the Euclidean distance to measure the similarity between embedded representations but also we use the L_1 and ϵ^2 distances. Results are shown in Fig. 7.3.

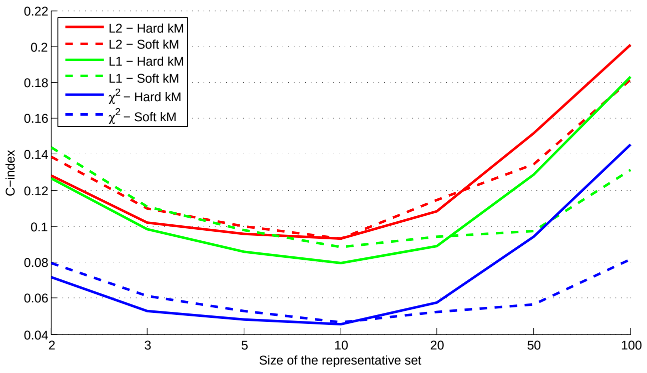
The conclusions are again similar to those for the generic formulations of the embedding. In general, the soft configurations tend to outperform their corresponding hard versions, and this is due to their ability to adapt to possible deformations in the object models. In particular, in this case, the intensity variability of every color is responsible of such perturbations. It also happens the that ϵ^2 distance is, from those that have been tried, the one that provides the best results. The corresponding curves are almost always below the other distances ones. Finally, it is not clear to tell in the general case which is the optimum number of elements in the set of representatives, although it seems that numbers below 10 are a reasonable choice.

Next, we discuss a comparison between the generic and the color versions. We use the same distances to measure the clustering performance, and for the generic representations, we report the results of the configurations based on the size of the set of representatives that give the best clustering results on the training sets. For each dataset and each embedding version we report the clustering results. We also show the geometric mean of the results on the three datasets as a measure for ranking configurations (this is actually the way it is done in the contest report [55]). Results are shown in Table 7.8.

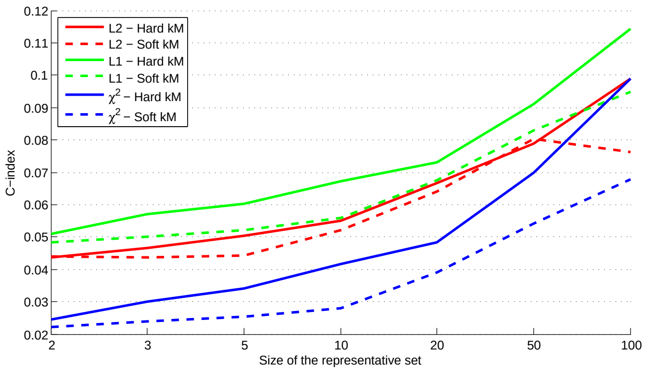
From these results we can argue again that the ϵ^2 distance is again the metric that best exploits the features we are proposing. Indeed, the best results for all datasets are always configurations of the embedding under the ϵ^2 distance. It is also worth pointing out the fact that soft configurations (*Soft kM* and *Soft Color*) always perform better (in terms of the geometric means) than their corresponding hard versions. Finally, we observe that the *Soft kM* configuration is competitive with the *Soft Color* one. It beats the color-based approach in two of the three cases, and the geometric means are equivalent. This tells that the generic approach is still a very good approach for solving this problem. Nonetheless, the color-based one does not depend on the validation stage of the number of representative elements and can be directly computed without having to tune a parameter of this nature.



(a) ALOI



(b) COIL



(c) ODBK

Figure 7.3: Clustering performance of the generic versions of the embedding as a function of the size of the set of representatives. Results for different vectorial measures.

Table 7.9: Generic *vs* color-based representations. Clustering results on the train sets by the use of different kernel functions. The geometric mean of the results on the different datasets is also shown.

Kernel	Representation	ALOI	COIL	ODBK	Geo. Mean
κ_{L_2}	*Hard kM*	0.167	0	0.129	0
	Soft kM	0.150	1.64e-08	2.4e-10	8.40e-07
	Hard Color	0.167	7.20e-12	0.127	5.35e-05
	Soft Color	0.152	8.70e-12	0.129	5.56e-05
κ_{L_1}	*Hard kM*	0.176	0	0.142	0
	Soft kM	0.154	0.041	4.40e-11	6.57e-05
	Hard Color	0.173	6.17e-12	0.137	5.27e-05
	Soft Color	0.160	2.55e-12	0.134	3.80e-05
κ_{χ^2}	*Hard kM*	0.104	9.29e-10	0.097	2.11e-04
	Soft kM	0.080	6.58e-10	1.91e-12	2.33e-08
	Hard Color	0.113	2.41e-05	0.097	6.44e-03
	Soft Color	0.091	6.21e-09	0.089	3.71e-04

Implicit Embeddings

The contest organization also allowed implicit expressions of graph embeddings, this is, graph kernels. In order to compute the distance value between two graphs they would use the following expression

$$d_{ij} = \sqrt{\kappa(g_i\ g_i) + \kappa(g_j\ g_j) - 2\kappa(g_i\ g_j)} \tag{7.7}$$

where $\kappa(g_i\ g_j)$ is the kernel function between graphs g_i and g_j. Straightforward operations and the use of the kernel trick show that this equation is regarding the Euclidean distance in the implicit Hilbert space associated to the kernel function.

In order to be able to compare our methodology with the contest participants that provided implicit embedding representations (such a comparison will be make in forthcoming sections), we define kernel functions between graphs by means of the kernel functions between the explicit vector representations we propose. Formally, given graphs g_1 and g_2, we define κ_d as the kernel

$$\kappa_d(g_1\ g_2) = \exp\left(-\gamma \cdot d(\nu(g_1)\ \nu(g_2))\right) \tag{7.8}$$

where $\nu(g)$ is an embedding formulation of the graph g, $d(\cdot\ \cdot)$ is any vectorial distance at hand and $\gamma > 0$. In particular, we have used the same three distances we have been using so far, L_1, L_2 and ϵ^2. By computing the distances using Eq. (7.7), we report the results obtained on the training sets in Table 7.9. Different values of γ have been tried. We omit the validation plots for these cases.

As it can be seen, the values reported in the table justify the use of these kernel functions since the clustering performances are of almost perfect results in many cases. Results are of orders of magnitude that little that it is hard to really explain a general behavior of these data. Nevertheless, the conclusions that can be extracted are similar to those from the previous table.

7.3.4 Contest Participants

In the next section we will report results obtained on the test sets of the contest datasets. We here explain briefly the embedding approaches of the contest participants. In particular, four approaches were submitted. Three of them were explicit embeddings of graphs, one was an implicit formulation, this is, a graph kernel.

Table 7.10: Clustering results on the test sets. C-index under the Euclidean distance. Comparison with the contest participants. The best results are shown bold face.

Embedding	ALOI	COIL	ODBK	Geo. Mean
Osmanlıoğlu et al.	0.088	**0.067**	0.105	0.085
Jouili and Tabbone	0.136	0.199	0.138	0.155
Riesen and Bunke	**0.048**	0.128	0.132	0.093
Luqman et al.	0.379	0.377	0.355	0.370
Hard kM	0.088	0.160	0.067	0.098
Soft kM	0.073	0.136	0.058	0.083
Hard Color	0.067	0.143	0.061	0.083
Soft Color	0.056	0.121	**0.051**	**0.070**

Jouili and Tabbone method starts by constructing a dissimilarity matrix between all pairs of graphs. Such dissimilarities are based on node signatures and an optimal assignment between them using the Hungarian method [83]. The dissimilarity matrix is latter converted into a positive semi-definite matrix, the eigenvalues of which are used as features for the final embedding [84]. The second embedding methodology was proposed by Luqmann *et al.* The feature vector extracted for each graph encodes meaningful information of nodes and edges by the use of fuzzy intervals. For instance, histograms of node degrees or node attributes [111]. The third explicit embedding proposal way proposed by Riesen and Bunke and it is exactly the one we have used for comparison in the previous sections in which a graph is represented by a feature vector the components of which are edit distances to a set of prototypes [134]. Finally, the fourth embedding approach is an implicit one. Proposed by Osmanlıoğlu *et al.*, it maps each node of each graph to a vector space by means of the caterpillar decomposition, and computes a kernel value between two given graphs in terms of a point set matching algorithm based on the Earth Mover s distance [39].

7.3.5 Results

In the previous sections, we have validated the four embedding configurations proposed and summarized in Section 7.3. Recall we have two generic versions in which a set of representatives is obtained by either kMeans or Fuzzy kMeans. They are referred to as *Hard kM* and *Soft kM* respectively. We also have to color-based versions, in which we use a color naming methodology to assign each RGB node label to either a color name (*Hard Color*) or a set of probability values for each of the color names (*Soft Color*).

In this section, we show a comparison between these configurations and the contest particpants. For those cases that a certain parameter needs to be validated   the number of representative elements, a meta-parameter of a kernel function, etc.   , we have used the corresponding training sets and discussed in the previous sections.

Distance-based

The contest participants were evaluated under the Euclidean distance for the explicit methodologies or using the corresponding analogous of the same metric in the implicit vector space (Eq. 7.7) for the kernel-based methodologies. For the sake of fairness, in Table 7.10, we first show a comparison of the results reported in the contest communication [55] and the results of our approaches under the Euclidean metric.

Concerning the results of our methodologies, as expected, the *Soft* approaches obtain better results than the hard ones, and the color-based versions improve the generic ones.

Table 7.11: Clustering results on the test sets. Comparison of the C-index under the use of different distances. The best results are shown bold face.

		Results with distance			
Distance	Representation	ALOI	COIL	ODBK	Geo. Mean
L_2	*Hard kM*	0.088	0.160	0.067	0.098
	Soft kM	0.073	0.136	0.058	0.083
	Hard Color	0.067	0.143	0.061	0.083
	Soft Color	0.056	0.121	0.051	0.070
L_1	*Hard kM*	0.082	0.156	0.067	0.095
	Soft kM	0.064	0.130	0.063	0.080
	Hard Color	0.071	0.129	0.071	0.086
	Soft Color	0.060	0.110	0.061	0.073
χ^2	*Hard kM*	0.037	0.098	0.037	0.051
	Soft kM	**0.031**	0.066	**0.033**	**0.040**
	Hard Color	0.045	0.086	0.045	0.055
	Soft Color	0.032	**0.064**	0.037	0.042

Compared to the participants methods, the proposed embedding approach ranks second on two databases and first on the third one. This leads to the best geometric mean among all tested methods. Moreover, let us mention again the high efficiency of our approach which arises from the fact that we base our embedding method on very simple features with a fast computation.

As it has been proved along this thesis, the Euclidean distance is not the best metric for the analysis of the vectors we propose. We refine our results by computing the C-index for clustering under the L_1 and ϵ^2 distances. Results of these experiments are shown in Table 7.11. The ϵ^2 distance is again providing the best results, ranking best on all datasets, even when compared to the contest participants (we, however, want to point out that a direct comparison to the results obtained by the contest participants would not be fair since we do not know how their algorithms would perform under other metrics). Interestingly, the ϵ^2 distance extracts the best out of the *Soft kM* versions since it outperforms the *Soft Color* one in two of the three datasets, which does not happen when using the two other metrics.

Kernel-based

Finally, in order to relate our methodology to those that provide an implicit embedding of graphs we compute kernel values between embedded graphs as in Eq. (7.7). Results are shown on Table 7.12.

Although the results for the ALOI database worsen when using the kernel values for all versions of the embedding, the most significant point to highlight from this table is that we obtain almost perfect separation indexes for the COIL dataset under all embedding configurations and kernels and also for the ODBK under the *Soft kM* one. This makes the geometric means to drastically decrease and demonstrates the embedding methodology we propose in this work to be a strong approach for graph clustering.

7.3.6 Discussion

We have presented a way to adapt the generic configurations of the proposed embedding to a specific case of graphs whose node labels regard RGB color values of object images. In particular, this adaptation has been made through a color naming based selection of the set of representatives. The experimental evaluation has been done, this time, in terms

Table 7.12: Clustering results on the test sets. Comparison of the C-index under the use of different kernel functions.

		Results with kernel			
Kernel	Representation	ALOI	COIL	ODBK	Geo. Mean
κ_{L_2}	*Hard kM*	0.148	6.25e-10	0.165	2.48e-04
	Soft kM	0.134	3.83e-08	1.24e-09	1.85e-06
	Hard Color	0.130	7.71e-12	0.141	5.20e-05
	Soft Color	0.115	2.05e-11	0.128	6.70e-05
κ_{L_1}	*Hard kM*	0.158	6.25e-10	0.167	2.54e-04
	Soft kM	0.134	4.13e-07	1.21e-12	4.06e-07
	Hard Color	0.138	7.71e-12	0.154	5.47e-05
	Soft Color	0.122	2.05e-11	0.140	7.04e-05
κ_{χ^2}	*Hard kM*	0.117	1.88e-07	0.120	1.40e-03
	Soft kM	0.088	3.10e-08	8.67e-10	1.36e-06
	Hard Color	0.106	3.60e-03	0.114	0.035
	Soft Color	0.083	9.04e-07	0.097	1.94e-03

of a clustering measure. Again, we have seen that the soft versions of the embedding both the generic and the color ones always perform better than the corresponding hard configurations, which is a common phenomenon along this work. Also, the soft color version is at the same performance level as the generic soft version using fuzzy kMeans, with the advantage that there is no need for a validation of the number of elements in the set of representatives. This suggests that it is indeed a good choice for clustering color region adjacency graphs.

Finally, with regard to the comparison to the ICPR Contest participants, we have shown superior performance with respect to all of them, confirming again that the embedding proposal we have been explaining along this thesis is worth considering in the repository of tools for the analysis and solutions of structural pattern recognition problems.

Chapter 8

Conclusions

Pattern recognition can be divided into two main branches based on how patterns are formally presented to the machine. On the one hand, statistical pattern recognition represents patterns as a set of numerical measurements arranged in the form of a feature vector. The vast and flexible geometrical apparatus of the vector spaces allows the design and efficiency of several ways to process data but feature vectors might not be a good representational paradigm in those cases where relations between parts of the patterns clearly need to be described. Structural pattern recognition, on the other hand, makes use of graphs in order to present patterns to the machine. Graphs allow to encode complex information of patterns in terms of binary relations inside the underlying represented objects. Moreover, they are not restricted to a predefined complexity and can adapt the representation to the patterns under study, in contrast to feature vectors, where always the same number of features have to be computed for all patterns regardless of their complexity.

Although graphs provide a fancy solution for representing patterns in a given problem, they suffer from a high computational complexity problem and just a few ways to treat and analyse graphs are there available. The optimal situation under this scenario would thus be to be able to represent patterns using relational structures such as graphs and analyse such structures using any of the data processing algorithms that the statistical pattern recognition field is capable to come up with. Bridging this gap between the statistical and the structural pattern recognition domains has been studied for the research community for some time. Two main investigations are there that provide elegant solutions to this problem, namely, graph kernels and graph embeddings. Graph kernels aim at defining positive definite symmetric functions between instances of graphs so the set of learning algorithms called kernel machines becomes applicable to the graph domain by means of the kernel trick. Graph embeddings, for their part, seek for those graph features that best describe the structural information of graphs and arrange them as a feature vector. In other words, graph embeddings map each graph into an appropriate and convenient feature space where as much of the graph information is preserved. By this way, any data processing algorithm originally developed for statistical feature vectors now becomes also applicable to the graph domain.

In this thesis we have tackled the problem of transiting from the structural to the statistical pattern recognition worlds from a graph embedding point of view. The main contributions of this work and how the proposal we have made can be extended in the future is what this chapter is devoted to.

8.1 Summary of the Contributions

In this thesis, we have proposed an efficient solution for embedding graphs into vector spaces for the discrete and the continuous attributed cases. In particular, the embedding map is based on counting frequencies of unary and binary relations of the node labels of the graphs.

Discrete attributed graphs

After an introductory chapter, where the problem to be solved has been stated, and after a revision of the graph matching methodologies we can find in the vast structural pattern recognition literature, the discrete attributed case has been faced in the third chapter of this thesis. We have proposed to extract features from graphs that regard the number of nodes with a given label and the number of edges between nodes with two given labels. We distinguish the edge attributed case by considering different features those cases of edges that have different labels although they link nodes with the same label.

First of all, we have evaluated how this methodology emulates the graph edit distance when the L_1 distance is considered to compare our vector representations. It turns out that the embedding proposal together with this metric is, for the node-based features, computing how many nodes of a certain label are there in one graph that are not in the other. This is what the graph edit distance does when assuming edit costs that penalize substitutions of nodes with different labels by forcing to delete and insert nodes from one graph to the other. Besides the fact that our proposal is much more efficient, we have come to the conclusion that the graph features we suggest are highly correlated with the edit distance whenever more importance to the node operations are given. When this situation occurs, our embedding methodology could thus be used as an efficient alternative for the analysis of graph distributions. On top of that, classification results of the embedding vectors outperform those of the graph edit distance for all considered datasets.

Although the L_1 metric provides a measure of how related the embedding methodology is with the edit distance, we have also analysed other vectorial distances to check for the best one in terms of classification performances. We have evaluated the proposed graph features under the Euclidean distance and also under a histogram-based metrics such as the ϵ^2 distance. It is not by chance that this last distance is the one that generally performs best for the datasets that have been used in this work. Indeed, the vectorial representation of graphs that we propose are histograms of frequencies of some specific substructures in the graphs, namely, nodes and edges with certain labelling configurations. The ϵ^2 distance has thus experimentally placed itself as the best choice concerning the metrics of the embedding space.

Chapter 3 finished by giving another interpretation of the features we propose. In particular, edge features are understood as the number of walks of length 1 between two specific labels. In that line, and in order to improve the vectorial representations of graphs, we generalize the features by proposing to use not only walks of length 1 but walks of any length. By such generalization we aim at providing a broader diversity, since several vectorial representations are built for each graph in terms of different walk lengths. This way, we introduce more global features for the graph representations than just the local features that the 1-length walks provide. We experimentally observe that only in very few cases, longer walks provide better results than shorter ones. Even though some more insight has to be given to this idea, the diversity of the representations is still there and it is exploited in a later chapter.

Continuous attributed graphs

The embedding representation for discrete attributed graphs has been extended to the case of continuous attributed graphs. It turns out that counting frequencies of continuous labels is unfeasible since it might happen that two graphs do not even share any label in common. To overcome this problem we have proposed to discretize the continuous node labels by selecting a set of node label representatives and assigning nodes to elements in this set. The underlying idea is to assume that a graph model exists for every category of continuous attributed graphs and by assigning nodes to representatives we aim at undoing the possible deformations that each graph has suffered with respect to such a model. Once the discretization has been done, the embedding methodology is defined analogously to the discrete case, this is, counting how many nodes has been assigned to each representative and how many edges are there between nodes that have been assigned to two given representatives.

In particular, we propose essentially two versions of the assignment step, namely, a hard version where nodes are assigned to just one representative and a soft version where such assignment is done in a fuzzy fashion, this is, probabilities of belongingness to each representatives are computed. By fuzzyfing the assignment of nodes to representatives we expect to deal with noisy situations in which node labels have no clear representative in the set of representatives. In other words, it might happen that, for a given node label, there is no clear rule telling to which element in the set of representatives the node should be assigned to. On top of these things, the edge-based features should also be redefined when a set of probabilities is given to every node. In particular, we adopt two strategies. The first one is by considering those representatives that turned out to be the ones with highest probability and count the corresponding edge as an appearance of the relation between these two representatives. The second strategy is rather more robust since it keeps considering all probabilities and thus all possible counts for the relations between any pair of representatives. As expected, it turns out that the soft assignment provides better results than the hard ones because it solves the problem of representing node labels by elements in the representative set rather more firmly than the hard way, where only a single representative is assumed for each node label. Also, the fuzzy version of edge-based features performs generally better than that where a relation between only two representatives is assumed.

Regarding how to select the set of representatives, we have proposed to use several algorithms from the data clustering domain. In particular, we use two approaches for the case of hard assignment spanning prototypes and kMeans and two for the soft assignment fuzzy kMeans and mixtures of gaussians . A later experimental evaluation tells that the fuzzy kMeans is generally more stable than any other methodology and thus it becomes our option in further developments of the embedding approach. Moreover, as it was done in the discrete case, the embedding space is evaluated under different metrics in order to check for the one that best suits the proposed features. Results advice to keep using a histogram-based metric such as the ϵ^2 distance, as they did in the discrete case.

It is clear that the graph features we have proposed not only depend on how the set of representatives is selected but also on how many. In fact, different number of elements in the set of representatives have different interpretations on what the eventual embedding features are describing. For instance, a small number of representatives would make each of them to attract several nodes of the graphs and thus the edge-based features will be regarding for more global relations among the structure of graphs. On the contrary, a large number of elements will make representatives to attract less nodes of each graph, and thus accounting for a more local description of the graphs. There is no clear way to select beforehand the number of elements in the set of representatives since we do not really know whether a given database would respond better to a local or a global embedding representation. We thus need to validate such a parameter for each dataset and, at the end, we observe how this number depends on each of the datasets we have considered.

Feature selection and classifier ensembles

Chapters 5 and 6 of this thesis are devoted to study two important properties that the proposed embedding methodology present, namely, the high dimensionality and feature correlation on one side and the diversity of the different representations we might build up for each graph on the other.

The vectorial representation of graphs that we have proposed in this work has a quadratic dimensionality with respect to the number of elements in the set of representatives we select. Not only this, but it may also happen that some of the representatives we end up selecting are not relevant at all in the final vectorial configuration of the embedding, plus some of the features can present correlation among them, which may turn into noisy and redundant representations of graphs. Such situations have been tackled using feature selection algorithms, which generally try to get the subset of features from all the available ones that best solve a particular task. We have used two different strategies for feature selection. The first one is based on ranking methodologies, that assign to each feature a ranking value regarding its discriminative power. Once the ranks are associated to each features, one keeps the first important ones and so the task that is wanted to be solved can be evaluated with lower dimensional vectors. The second strategy we have used is based on PCA methods, where a transformation of the features is sought such that most of the variance is kept. Once the features are transformed into a lower dimensional space, the same task can also be solved using the new transformed features.

We have applied the mentioned feature selection strategies to different configurations of the embedding methodology based on different sizes of the set of representatives. The gain we have obtained is much more relevant in those cases where we originally used a large set of representatives. Clearly, this is because a larger set of representatives includes more superfluous elements which leads to noisier representations. In any case, we have observed that the ranking methodologies are more adequate for the embedding proposal of this work than those that look for a transformation of the features. Moreover, a deep evaluation of the ranking methods tells how the node-based features have more weight than the edge-based ones in the final reduced versions. This is not discarding, though, that edge-based features need to be there since they keep adding structural information to the embedding representations that help on the performance in classification problems. PCA-based methods are able to reduce redundancy of our features by removing several components of the vectors, but this generally leads to poor classification performances when compared to the ranking methods. All in all, we have extracted some insight about the features we are proposing by means of feature selection methods that let us work with lower dimensional vectorial representations of graphs.

With respect to how to take profit of the diversity of the different representations we may construct for each graph, we have adopted a multiple classifier system point of view. Ensembles of classifiers aim at boosting the performance of different (base) classifiers by combining the outputs of all of them. In particular, for the discrete attributed graphs, we have used different representations with different walk lengths, and for the continuous attributed case, we use different configurations based on different sizes of the set of representatives. In both cases, the different representations can be seen as regarding both local and global structural information of graphs.

The results confirm that the discrete attributed case, where walks of different length are considered, need to be redefined in such a way that more robust representations are obtained. Dealing with noisy features due to tottering effects and obtaining more diverse representations will eventually result in more discriminative embedding formulations. On the contrary, the continuous attributed case do offer some diversity in the representations that lead to improvements of the combination of classifiers with respect to the base configurations. In most of the evaluated datasets, the optimal set of base classifiers is composed of embedding vectors that are based on rather different representative set sizes. This supports the idea that both local and global information of graphs is important to be taken into account.

Results and comparisons

This thesis finished with a vast experimental evaluation of the different proposed methodologies in comparison to other graph matching techniques. The discrete attributed case is compared with graph edit distance in the graph domain and with a graph kernel based on the edit distance in the vector domain, where a support vector machine is applicable. We have put quite some emphasis on an empirical comparison of the computational times of all methods ours and the reference ones and concluded that, while we still obtain tied or even statistically better results than the reference systems, our methodology is much more efficient than graph edit distance and its associated kernel. Ensembles of different classifiers only reported improvements whenever we detected a clear diversity in the original embedding configurations.

The continuous attributed case has been also compared with the graph edit distance. In this case, though, instead of a graph kernel we have used another embedding methodology based on the dissimilarity representation and the edit distance that has been reported as quite a robust approach for graph embedding. Our embedding fails in those cases where a severe deformation is present in the underlying objects that graphs represent, these are, distorted letters and handwritten digits. This is due to the fact that the assignment from node labels to representatives is not as meaningful as desired and not able to undo the inherent deformations that exist in the graphs with respect to its category model. In any case, the other considered cases have generally a similar result to the reference method while again here keep the computational time within reasonable limits, just the opposite as the other techniques do. Feature selection methodologies usually impoverish the original representations since some information is lost in the process, but still, the situation is only worse than the reference systems whenever we have objects with inherent deformations. Interestingly, the diversity of the different base classifiers results into an improvement on this datasets where we originally have more problems.

Finally, we have proposed to adapt the generic embedding methodologies to a specific case of color region adjacency graphs. We have used a color naming strategy in order to select more meaningful elements in the set of representatives. Clustering results using the proposed methodologies have reveal themselves as very competitive with other embedding methodologies within the ICPR Graph Embedding contest.

8.2 Future Perspective

New ideas are there on the horizon that time constraints did not allow us to evaluate. First of all, as it has been said plenty of times along this work, those features for discrete attributed graphs that account for walks of arbitrary lengths could be drastically improved. Particularly, the research line should look for ways to, first, efficiently compute walks of arbitrary length since the longer the walks the more the adjacency matrix should be exponentiated and, second, to remove those meaningless walks that, for instance, keep travelling back and forth along the same edges. Especially interesting would be to relate this embedding methodology with the so-called *label pairs* graph kernel [57, 58], a predecessor idea of the random walk kernel where some of this ideas are taken under consideration. A transition of these ideas to continuous attributed graphs would be straightforward and also definitively worth considering.

There is also a very important situation that should be tackled and that would close the circle with respect to the kind of graphs that the proposed methodology can handle. It is that of continuous attributed graphs with labelled edges. This case has not been considered in this thesis, and whenever edge labels were there in a given dataset of graphs we just neglected them. A possible way to adapt the embedding proposal to this type of graphs could be of course the one that is directly related to the way it has been done for the discrete case. This is, discretize the edge labels and count as different features those edges that, although they link nodes with the same (assigned) representatives, have different labels.

It will also be interesting to perform the same analysis we did about the correlation with edit distance in the discretely attributed case for the continuous one. Although the relation is not that clear in this case, we feel this analysis will reveal a similar behavior than that for the discrete case and thus we will be able to give quite more strength to the proposed methodology.

Related to this, another idea is the following. Consider two graphs. A necessary condition for one graph to be a subgraph of the other is that all of the features that we have described for the first one are component-wise smaller than the corresponding ones of the second graph. This is a necessary but not sufficient condition and thus it will not be useful for checking subgraph isomorphism, but it would be of great help to efficiently crop potential solutions in a retrieval problem where a large database of graphs is evaluated. In particular, retrieval scenarios are those that aim to recover similar objects to a given query element from a large indexed dataset. Efficient comparisons by means of the proposed features in a subgraph retrieval problem would easily discard unrelevant elements from the dataset.

Regarding the validation of the number of elements in the set of representatives, we could perhaps use smarter ideas than just try several values in a large range. In particular, we may evaluate a clustering measure in the labelling space and see whether there is any correlation between those sets of representatives that obtain good clustering measures of the set of labels and the corresponding performance of the eventual embedding representations. If that is so, the range of the number of representatives where we seek the optimal one could be reduced, and thus the validation of the set of representatives could be done faster and more efficiently.

Also on the experimental side, we could enrich this thesis if we apply feature selection algorithms on the discrete attributed case, as we did in the continuous version. The main reason for not doing so is that the labelling sets for the discrete graphs were always small enough to not worry about the dimensionality of the resulting vectors. Anyway, a similar study on which type of features a ranking method would reveal as the most relevant ones would definitively be interesting.

The diversity of different base classifiers in the ensembles of multiple classifiers has been sought and obtained by using different configurations of the embedding based on different sizes of the sets of representatives. We have no clue whether this is the optimal way to approach this situation or whether other ways to look for diversity in base classifiers would give more discriminative representations. We may of course try to use several sets of representatives of the same size but construct them in a different manner. For instance, we could try to randomly select several sets of representatives of a certain size and they will all be regarding different information from the graphs that may lead to an enrichment of the final ensemble of the base classifiers. Also, we may combine different of the representations that have been proposed, for instance, hard and soft versions, and see how they complement each other in a final classification setting.

Finally, regarding the validation of most of our methodologies, we have used a simple classifier such as the kNN on the validation sets and eventually used an SVM for the test stage. This procedure is clearly suboptimal but it has been done to keep the validation stages within reasonable amounts of time. Maybe also the comparison with other graph matching, and particularly, graph embedding methodologies could have been carried out. We have used the ones described above since they provide a very general understanding of the problem. Graph edit distance can deal with any type of graphs and thus the embedding methodology we have used as the baseline does too. On top of these things, results on the same datasets and with the exact same validation settings as the ones we have used are reported for these methodologies and thus the comparison has been much more easier.

Appendix A

Graph Data

Along this work we have used several datasets of graphs. In this chapter we give a detailed description of the nature of all of them. In particular, we describe the patterns each of them is representing and, when necessary, the graph extraction process. The set of graph collections is divided between those graphs with discrete attributes and those with continuous labels.

A.1 Discrete Attributed Graphs

Concerning graphs with discrete labels, we have considered 4 different datasets. These collections are split into two main categories regarding the nature of patterns they represent. The first group of graphs represent objects in images and the second one represent molecule structures.

A.1.1 Object Image Datasets

We construct graphs with discrete attributes from two publicly available large image databases. The databases represent different objects under a rotating viewpoint. In particular, we used the Amsterdam Library of Object Images (ALOI) [60], and the Object Databank by Carnegie-Mellon University (ODBK) [161].

The ALOI dataset is a generalization of the COIL dataset [118]. Images of 1000 objects are acquired by changing the illumination and the view pose of the objects several times, leading to a total amount of 110,250 images. In the top row of Fig. A.1, some examples can be found. To keep the computational time of the tested algorithms within reasonable limits, we only use 50 of the 1000 objects in this datasets. For the 72 different images of an object at 5 degrees of rotation, we keep 24 images (one at each 15 degrees of rotation) for training and, from the remaining ones, we randomly select 5 for validation and 10 for testing. The training set thus consists of 1200, the validation set of 250, and the test set of 500 images.

The second collection, the ODBK dataset, is rather different from ALOI. It is a collection of 209 different object models that have been rendered with photo-realistic quality using 14 different viewpoints. The bottom row of Fig. A.1 shows some examples. We select 100 of the 209 objects. Out of the 14 viewing points of each object model, 12 are views arising from 30 degrees rotation and 2 are the top and the bottom views. We only keep the 12 rotated viewpoints, 6 of which are used for training (one at each 60 degrees of rotation) and the remaining six are randomly put either in the validation or in the test set. We eventually have a training set of 600 images, and a validation and test sets of 300 images each.

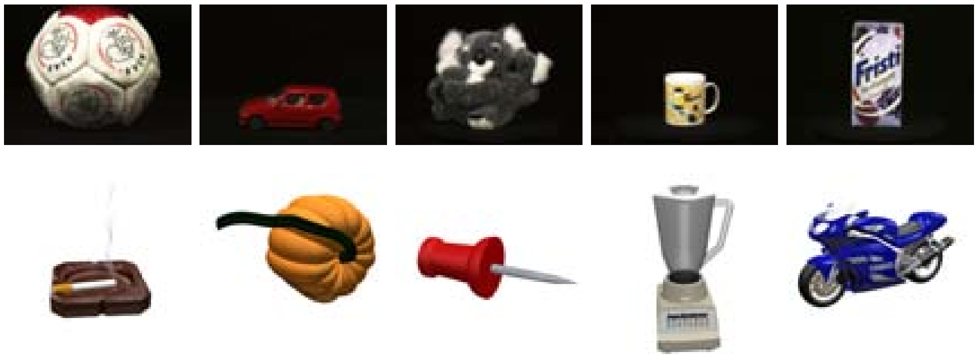


Figure A.1: Examples of images in the object datasets.

The graph extraction process is the same for the two datasets and it is illustrated in Fig. A.2. Given the original image (Fig. A.2(a)), we first segment it (Fig. A.2(b)) using the graph-based image segmentation approachdescribed in [50]. We crop the meaningless regions by convoluting the image with a mask (Fig. A.2(c)) that distinguishes the object from the background. The mask is constructed by first thresholding the grey-level image to remove the background and then closing possible holes in the object by mathematical morphology operations. We finally obtain a segmented version of the original object (Fig. A.2(d)).

From the segmented object images we extract discrete attributed graphs in the following way. Each region in the image is assigned to one of the eleven colors of the color naming theory. In [9] the authors proposed a model by which an RGB value is assigned to one of the eleven basic colorsWe label each region in our segmented objected by the color naming output of this model when providing the mean RGB value of all pixels in the region. By doing so, our node labelling alphabet will be

$$\begin{aligned} L_V = \quad & Black\ \ Blue\ \ Brown\ \ Green\ \ Grey\ \ Orange \\ & Pink\ \ Purple\ \ Red\ \ White\ \ Yellow \end{aligned} \tag{A.1}$$

Regarding the edges of the graphs, we link all adjacent regions, this is, we put an edge between every two regions whose borders have neighbouring pixels. In order to label these edges, we compute the length in pixels of the common border between two adjacent regions. Such length is normalized by the sum of the lengths of all common borders in the image and further discretized in three bins regarding the amount of pixels they share. The edge of

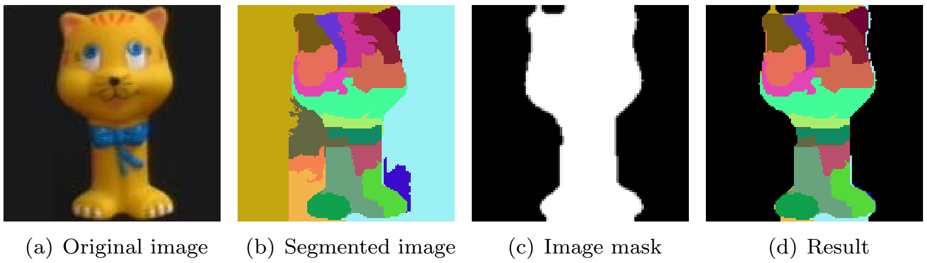


Figure A.2: Object segmentation for graph extraction.

Table A.1: Characteristics of the discrete object image datasets. Size of the training (tr), validation (va) and test (te) sets, the number of classes for each dataset (#classes), the average number of nodes and edges ($\pm V / \pm E$), the maximum number of nodes and edges (max V /max E) and the size of the node (L_V) and the edge (L_E) label alphabets.

Dataset	Size (tr, va, te)	#classes	$\pm\lvert V\rvert$	max$\lvert V\rvert$	$\pm\lvert E\rvert$	max$\lvert E\rvert$	$\lvert L_V\rvert$	$\lvert L_E\rvert$
ALOI	1200, 250, 500	50	22.9	78	49.6	204	11	3
ODBK	600, 300, 300	100	15.2	62	30.8	172	11	3

those regions that share a short border will be labelled by *Short*, those falling in the second bin by *Medium* and those in the third by *Long*. The edge alphabet is thus

$$L_E = \quad Short \;\; Medium \;\; Long \tag{A.2}$$

Finally, some other interesting characteristics of these datasets, such as the average and maximum number of nodes and edges, are shown in Table A.1.

A.1.2 Molecule Datasets

The second group of discrete attributed datasets is formed by two molecule datasets. Molecule compounds are certain biological structures that are easily represented by graphs. Atoms in the molecules are represented by nodes whose labels are the corresponding atomic elements. Edges represent the covalent bonds and they are labelled with the corresponding bond type.

The first dataset of molecules is the AIDS database from the IAM Graph Database Repository [132]. Graphs are constructed from the AIDS Antiviral Screen Database of Active Compounds [2]. They consist of molecules that are either positive or negative against HIV activity. In Fig. A.3 an example of a molecule of each class is shown. For training and validation purposes, 150 molecules are used in each set, while 1500 are used for testing. In this dataset, 21 different atomic elements happen to be the label of a node in the training set. Thus the size of the node label alphabet is 21. For the edges, only three labels are there: single, double, and triple bonds.

The second dataset of molecules is also taken from the IAM Graph Database Repository [132]. The Mutagenicity dataset (MUTAG) is composed of molecules that are divided into two classes, *mutagen* and *non-mutagen*. The mutagenicity of a molecule is a biological property that restricts its potential to become a commercial drug [90]. In this dataset, 1500

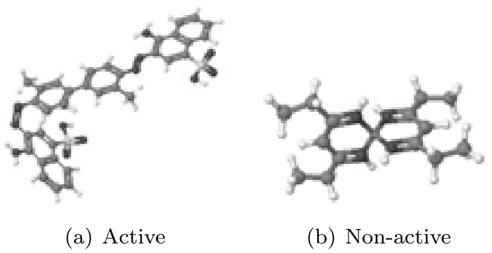

(a) Active (b) Non-active

Figure A.3: Examples of molecules from the AIDS dataset.

Table A.2: Characteristics of the molecule datasets. Size of the training (tr), validation (va) and test (te) sets, the number of classes for each dataset (#classes), the average number of nodes and edges ($\pm\ V\ /\pm\ E$), the maximum number of nodes and edges (max V /max E) and the size of the node (L_V) and the edge (L_E) label alphabets.

Dataset	Size (tr, va, te)	#classes	$\pm\|V\|$	max$\|V\|$	$\pm\|E\|$	max$\|E\|$	$\|L_V\|$	$\|L_E\|$
AIDS	250, 250, 1500	2	15.7	95	16.2	103	21	3
MUTAG	1500, 500, 2337	2	30.3	417	30.8	112	13	3

molecules are used for training, 500 for validation and 2337 for testing. In this case, 13 different atomic elements constitute the node label alphabet, while again there are three different types of atomic bonds represented by the edges labels.

As a summary, in Table A.2 we show the main characteristics of each of these datasets.

A.2 Continuous Attributed Graphs

Regarding the graph collections of continuous attributed graphs, we have used two important sets of databases. The first one, the IAM graph database repository is an important and widely used set of collections for graph-based algorithms benchmarking. The second one is the ICPR 2010 Graph Embedding Contest datasets which is rather new and also provides a practical framework for algorithm comparison.

A.2.1 IAM Database Repository

The IAM Graph Database Repository was created at the University of Bern [132]. The original purpose was to provide a set of standarized graph collections for benchmarking graph-based pattern recognition and machine learning algorithms. It has been clearly accomplished since many works use these datasets for comparing the algorithms proposed. The IAM Database covers a wide range of pattern recognition applications due to the nature of the representations that can be found, wrapping topics from document analysis problems (letters and symbols) to biometric identification (fingerprints). The IAM databases we have used in this work are described in the following.

Letter Databases

The first three datasets of graphs are the *Letter Databases*, which represent synthetic distorted letter drawings. Starting from a manually constructed prototype of every of the 15 Roman alphabet letters that consist of straight lines only (*A, E, F, H, I, K, L, M, N, T, V, W, X, Y* and *Z*), different degrees of distortion are applied: low, medium and high. For each of the classes, 150 instances are acquired under every degree of distortion, leading to three different datasets with 2250 graphs each. The datasets are split into a train, a validation and a test sets of 750 instances each. Concerning the graph representation of the letters, each ending point of a line is represented by a node of the graph and labelled with its (x y) coordinates. Unlabelled edges represent the existing lines in the letters by linking the corresponding nodes. An example of the A prototype and three instances of each of the different distortions are shown in Fig. A.4.

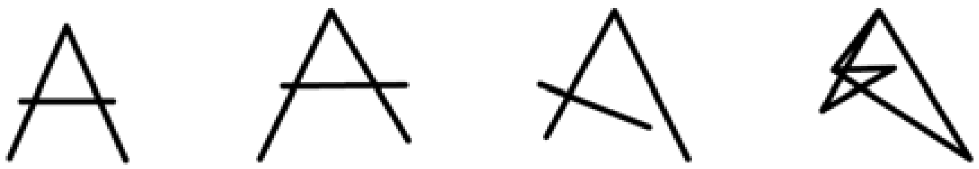


Figure A.4: Synthetic prototype of the letter *A* (left). One instance of the same class for each of the distortion levels (low, medium, high).

GREC Database

The *GREC Database* [43] represents architectural and electronic symbols under different levels of noise. In Fig. A.5, we show an example of each distortion level for different drawings. Depending on the level of noise, different morphological operations are applied to the symbols until lines of one pixel width are obtained. Intersections and corners of such lines constitute the set of nodes, which are labelled with their position on the 2-dimensional plane. Edges are traced between such interest points and labelled as *line* or *arc*. From the original dataset described in [43], 22 classes are considered, containing 50 instances of each distorted symbol. Around one quarter of them are used for training purposes, another quarter for validation and the rest for testing.

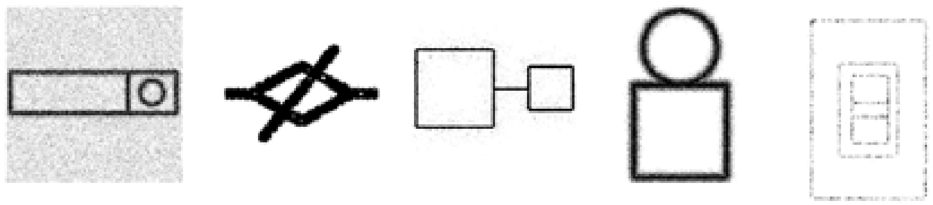


Figure A.5: An example of each of the distortion levels in different symbols of the GREC dataset.

Digits Database

The *Digits database* represents a set of handwritten digits. In [3] digits were originally acquired by recording the pen position at constant steps of time. Based on the sequences of $(x\ y)$ coordinates, graphs are constructed by adding nodes in regular intervals between the starting and ending points of a line. Nodes are labelled with their $(x\ y)$ position and connected by undirected edges. The orientation of each edge with respect to the horizontal direction is its label. In Fig. A.6 one instance of each class is depicted. From the original 10992 handwritten digits, 350 are considered for each class: 100 for training, 50 for validation and 200 for testing purposes.

Fingerprint Database

The *Fingerprint Database* consists of graphs that are obtained from a subset of the NIST-4 fingerprint image database [178] by means of particular image processing operations [119].

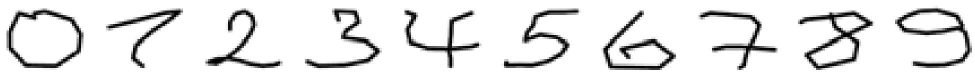


Figure A.6: Examples of each of the classes of the Digits dataset.

Ending points and bifurcations of the skeleton of the processed images constitute the $(x\ y)$-attributed nodes of the graphs, plus some additional nodes that are inserted between these points. All points connected through a ridge in the image skeleton are connected with an unlabelled edge. From the original set of images [178], 500 are considered for a training set, 300 for a validation set and 2000 for a test set. Fingerprints are distributed among four different classes: *arch*, *left*, *right* and *whorl* from the Galton-Henry classification system. The *arch* class has twice the number of element than the other classes since it also includes the so-called *tented arch* class. An instance of each of the four classes in the Fingerprint dataset is shown in Fig. A.7.

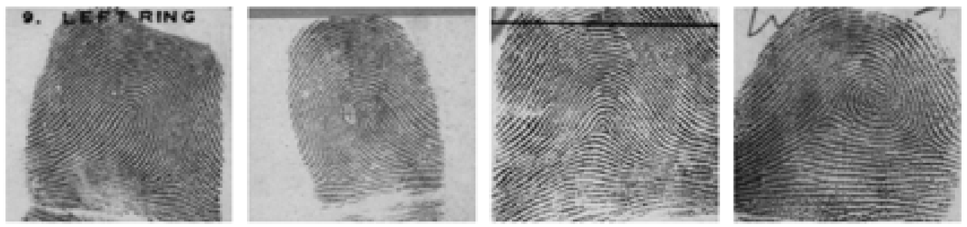


Figure A.7: Fingerprints Dataset: one example of each class. From left to right: *left*, *right*, *arch* and *whorl*.

COIL Database

The last IAM Dataset we have used in this work is also a set of graphs extracted from the COIL-100 database [118]. In this case, though, graphs are extracted by considering salient points in the images using the Harris corner detection algorithm [69], labelling these points with their corresponding $(x\ y)$ coordinates on the plane, and linking points using a Delaunay triangulation. Nevertheless, the distribution of the training, validation and test sets is exactly as the one described in Section A.1.1. In the first row of Fig. A.1, color images of five instances of the COIL dataset are shown. In Fig. A.8 there is a grey-level picture of each of the 100 objects.

Summary and discussion

The IAM Graph Database Repository includes several collections of graphs describing patterns of different nature. In this section we have just described those that are represented in terms of locations of interest points, this is, those sets whose graph nodes are $(x\ y)$ coordinates in the two-dimensional plane. Some of these datasets include also edge labels, which, unless indicated, are neglected by the algorithms proposed in this work.

Moreover, each of the datasets is split into a training set, a validation set and a test set. In Table A.3, we summarize the main properties of them, in particular we show the size

Figure A.8: The 100 COIL objects.

of the corresponding training, validation and test sets, the number of classes and also the average and maximum number of nodes and edges.

Table A.3: Characteristics of the IAM repository datasets. Size of the training (tr), validation (va) and test (te) sets, the number of classes (#Cls), the average number of nodes and edges (An/Ae) and the maximum number of nodes and edges (Mn/Me)

Dataset	Size tr, va, te	#Cls	An/Ae	Mn/Me
Letter low	750, 750, 750	15	4.7/3.1	8/6
Letter medium	750, 750, 750	15	4.7/3.2	9/7
Letter high	750, 750, 750	15	4.7/4.5	9/9
GREC	286, 286, 528	22	11.5/12.2	25/30
Digits	1000, 500, 2000	10	8.9/7.9	17/16
Fingerprints	500, 300, 2000	4	5.4/4.4	26/25
COIL	2400, 500, 1000	100	21.5/54.2	77/222

A.2.2 ICPR 2010 Graph Embedding Contest

The ICPR 2010 Graph Embedding Contest organizers prepared three datasets of graphs with continuous attributes for nodes [55]. In this case, though, unlike the IAM graphs, nodes are not attributed with location coordinates but with RGB values of segmented regions of object images and relative size of such regions.

In particular, the object image collections that were used are exactly the same as those described in Section A.1.1. Namely, the Columbia Object Image Library (COIL) [118], the Amsterdam Library of Object Images (ALOI) [60], and the Object Databank by Carnegie-Mellon University (ODBK) [161].

Given an object image, it is initially smoothed using a Gaussian filter and then segmented with a pyramidal segmentation algorithm. A region adjacency graph is constructed: regions become nodes labelled with the mean RGB value of their pixels and the corresponding relative size. Edges remain unattributed.

Being these datasets part of a contest and due to the will of making algorithms as much class-independent as possible, the distribution of training/test sets is different from the one described in the previous section regarding the IAM repository. In particular, there is only a training and a test set for each object image collection (no validation set is available). The evaluation protocols that are used for these datasets are detailed in this work whenever they are used (Section 7.3.5). On top of this, the classes present in the training set do not intersect with those of the test set. Particularly, the train set for the ALOI and COIL datasets contains 25 classes and the test set contains 25 other classes. The ODBK is distributed with 104 classes for training and 104 for testing. We show these number and other important information regarding these datasets in Table A.4.

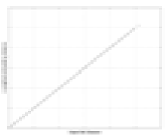
Table A.4: Characteristics of the ICPR 2010 Graph Embedding Contest datasets. Size of the sets, the number of classes (#Cls), the average number of nodes and edges (An/Ae) and the maximum number of nodes and edges (Mn/Me)

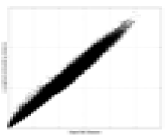
Dataset	Size	#Cls	An / Ae	Mn / Me
ALOI train	1800	25	29.24 / 28.37	103 / 112
ALOI test	1800	25	18.37 / 17.25	134 / 156
COIL train	1800	25	33.31 / 32.30	107 / 97
COIL test	1800	25	34.88 / 32.33	100 / 92
ODBK train	1248	104	65.18 / 62.45	636 / 598
ODBK test	1248	104	56.91 / 54.37	528 / 512

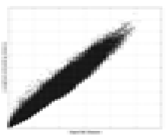
Again here, and unless indicated, edge labels have been neglected for all these datasets and are not used by any of the algorithms proposed in this work. Moreover, the node attribute regarding the relative size of the corresponding color regions is not either considered.

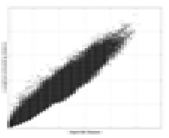
Appendix B

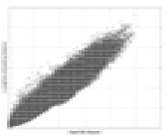
Scatter Plots of the Correlation between Graph Edit Distance and the L_1 distance for the Embedding Features

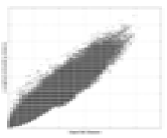

(a) $\rho = 0, \alpha = 0$

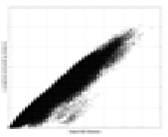

(b) $\rho = 0, \alpha = 0.2$

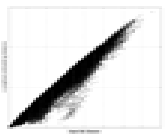

(c) $\rho = 0, \alpha = 0.4$

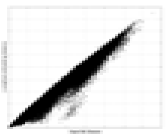

(d) $\rho = 0, \alpha = 0.6$

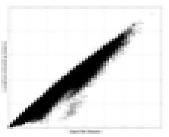

(e) $\rho = 0, \alpha = 0.8$

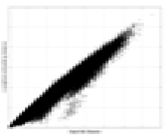

(f) $\rho = 0, \alpha = 1$

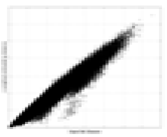

(g) $\rho = 0.2, \alpha = 0$

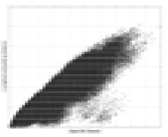

(h) $\rho = 0.2, \alpha = 0.2$

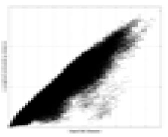

(i) $\rho = 0.2, \alpha = 0.4$

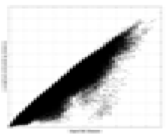

(j) $\rho = 0.2, \alpha = 0.6$

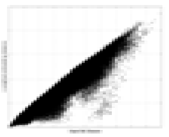

(k) $\rho = 0.2, \alpha = 0.8$

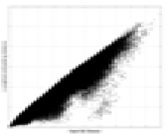

(l) $\rho = 0.2, \alpha = 1$


(m) $\rho = 0.4, \alpha = 0$


(n) $\rho = 0.4, \alpha = 0.2$


(o) $\rho = 0.4, \alpha = 0.4$


(p) $\rho = 0.4, \alpha = 0.6$


(q) $\rho = 0.4, \alpha = 0.8$

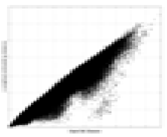
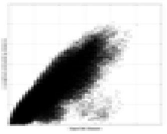
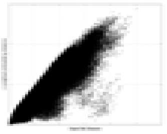
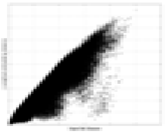
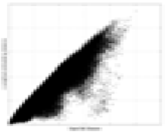

(r) $\rho = 0.4, \alpha = 1$

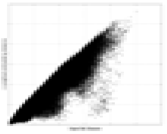
Figure B.1: ALOI Dataset: Scatter plots of the Graph Edit Distance versus the L_1 distance on the Embedding Features. Results for $\rho = 0\ \ 0\ 2\ \ 0\ 4$.

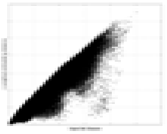

(a) $\rho = 0.6, \alpha = 0$

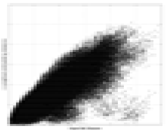

(b) $\rho = 0.6, \alpha = 0.2$

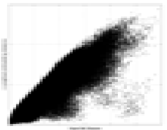

(c) $\rho = 0.6, \alpha = 0.4$

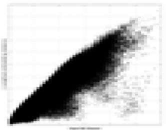

(d) $\rho = 0.6, \alpha = 0.6$

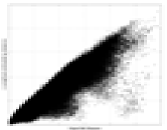

(e) $\rho = 0.6, \alpha = 0.8$

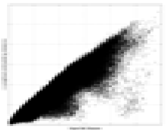

(f) $\rho = 0.6, \alpha = 1$

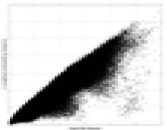

(g) $\rho = 0.8, \alpha = 0$

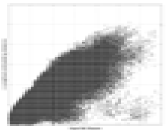

(h) $\rho = 0.8, \alpha = 0.2$

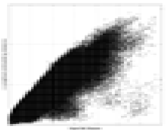

(i) $\rho = 0.8, \alpha = 0.4$

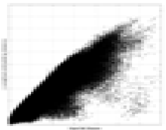

(j) $\rho = 0.8, \alpha = 0.6$

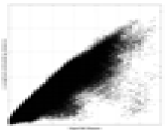

(k) $\rho = 0.8, \alpha = 0.8$

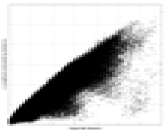

(l) $\rho = 0.8, \alpha = 1$


(m) $\rho = 1, \alpha = 0$


(n) $\rho = 1, \alpha = 0.2$


(o) $\rho = 1, \alpha = 0.4$


(p) $\rho = 1, \alpha = 0.6$


(q) $\rho = 1, \alpha = 0.8$

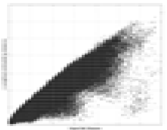
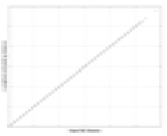
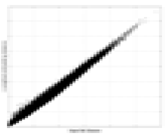
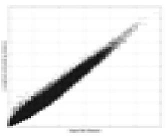
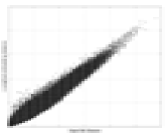

(r) $\rho = 1, \alpha = 1$

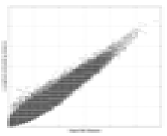
Figure B.2: ALOI Dataset: Scatter plots of the Graph Edit Distance versus the L_1 distance on the Embedding Features. Results for $\rho = 0\ 6\ \ 0\ 8\ \ 1$.

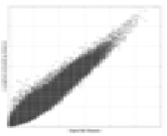

(a) $\rho = 0, \alpha = 0$

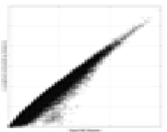

(b) $\rho = 0, \alpha = 0.2$

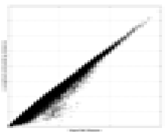

(c) $\rho = 0, \alpha = 0.4$

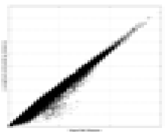

(d) $\rho = 0, \alpha = 0.6$

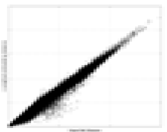

(e) $\rho = 0, \alpha = 0.8$

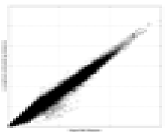

(f) $\rho = 0, \alpha = 1$

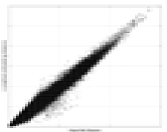

(g) $\rho = 0.2, \alpha = 0$

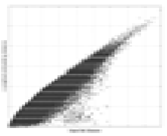

(h) $\rho = 0.2, \alpha = 0.2$

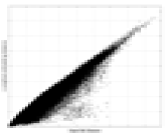

(i) $\rho = 0.2, \alpha = 0.4$

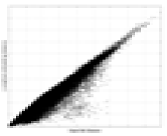

(j) $\rho = 0.2, \alpha = 0.6$

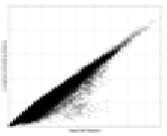

(k) $\rho = 0.2, \alpha = 0.8$

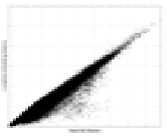

(l) $\rho = 0.2, \alpha = 1$


(m) $\rho = 0.4, \alpha = 0$


(n) $\rho = 0.4, \alpha = 0.2$


(o) $\rho = 0.4, \alpha = 0.4$


(p) $\rho = 0.4, \alpha = 0.6$


(q) $\rho = 0.4, \alpha = 0.8$

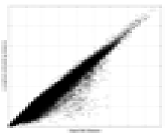
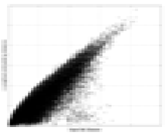
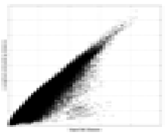
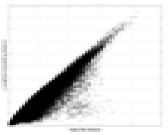
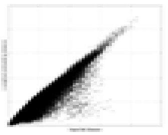

(r) $\rho = 0.4, \alpha = 1$

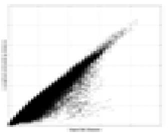
Figure B.3: ODBK Dataset: Scatter plots of the Graph Edit Distance versus the L_1 distance on the Embedding Features. Results for $\rho = 0\ \ 0\ 2\ \ 0\ 4$.

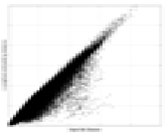

(a) $\rho = 0.6, \alpha = 0$

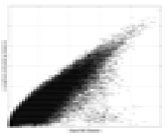

(b) $\rho = 0.6, \alpha = 0.2$

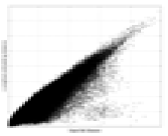

(c) $\rho = 0.6, \alpha = 0.4$

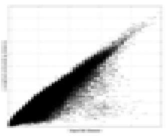

(d) $\rho = 0.6, \alpha = 0.6$

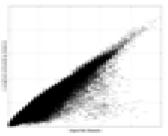

(e) $\rho = 0.6, \alpha = 0.8$

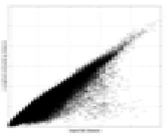

(f) $\rho = 0.6, \alpha = 1$

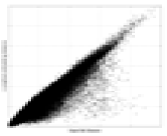

(g) $\rho = 0.8, \alpha = 0$

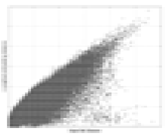

(h) $\rho = 0.8, \alpha = 0.2$

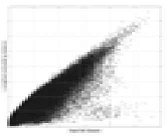

(i) $\rho = 0.8, \alpha = 0.4$

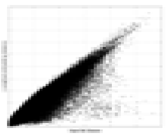

(j) $\rho = 0.8, \alpha = 0.6$

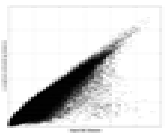

(k) $\rho = 0.8, \alpha = 0.8$

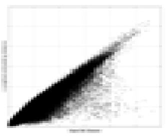

(l) $\rho = 0.8, \alpha = 1$


(m) $\rho = 1, \alpha = 0$


(n) $\rho = 1, \alpha = 0.2$


(o) $\rho = 1, \alpha = 0.4$


(p) $\rho = 1, \alpha = 0.6$


(q) $\rho = 1, \alpha = 0.8$

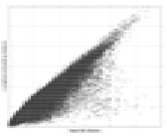
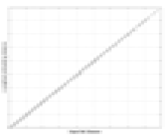
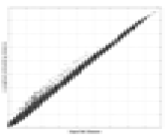
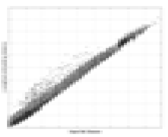
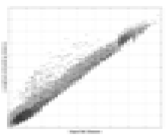

(r) $\rho = 1, \alpha = 1$

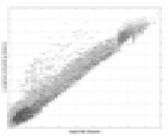
Figure B.4: ODBK Dataset: Scatter plots of the Graph Edit Distance versus the L_1 distance on the Embedding Features. Results for $\rho = 0\ 6\ \ 0\ 8\ \ 1$.

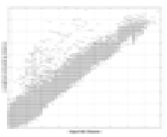

(a) $\rho = 0, \alpha = 0$

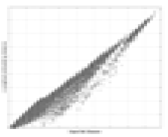

(b) $\rho = 0, \alpha = 0.2$

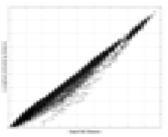

(c) $\rho = 0, \alpha = 0.4$

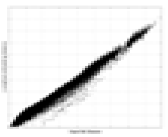

(d) $\rho = 0, \alpha = 0.6$

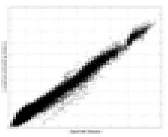

(e) $\rho = 0, \alpha = 0.8$

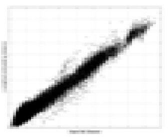

(f) $\rho = 0, \alpha = 1$

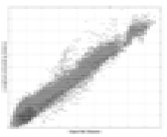

(g) $\rho = 0.2, \alpha = 0$

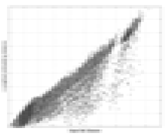

(h) $\rho = 0.2, \alpha = 0.2$

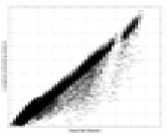

(i) $\rho = 0.2, \alpha = 0.4$

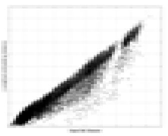

(j) $\rho = 0.2, \alpha = 0.6$

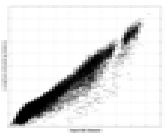

(k) $\rho = 0.2, \alpha = 0.8$

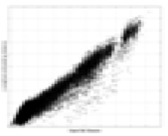

(l) $\rho = 0.2, \alpha = 1$


(m) $\rho = 0.4, \alpha = 0$


(n) $\rho = 0.4, \alpha = 0.2$


(o) $\rho = 0.4, \alpha = 0.4$


(p) $\rho = 0.4, \alpha = 0.6$


(q) $\rho = 0.4, \alpha = 0.8$

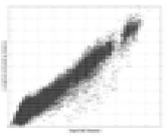
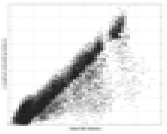
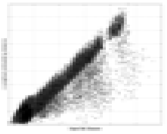
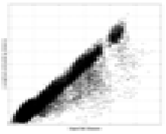
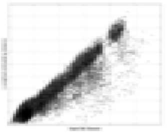

(r) $\rho = 0.4, \alpha = 1$

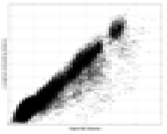
Figure B.5: AIDS Dataset: Scatter plots of the Graph Edit Distance versus the L_1 distance on the Embedding Features. Results for $\rho = 0 \ \ 0\ 2\ \ 0\ 4$.

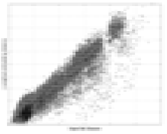

(a) $\rho = 0.6, \alpha = 0$

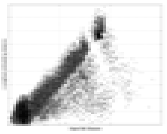

(b) $\rho = 0.6, \alpha = 0.2$

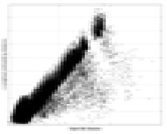

(c) $\rho = 0.6, \alpha = 0.4$

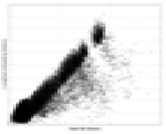

(d) $\rho = 0.6, \alpha = 0.6$

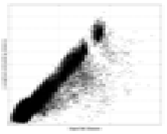

(e) $\rho = 0.6, \alpha = 0.8$

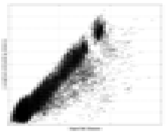

(f) $\rho = 0.6, \alpha = 1$

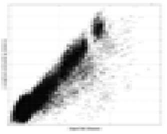

(g) $\rho = 0.8, \alpha = 0$

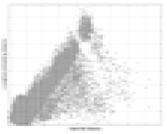

(h) $\rho = 0.8, \alpha = 0.2$

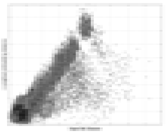

(i) $\rho = 0.8, \alpha = 0.4$

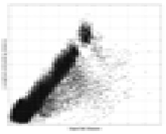

(j) $\rho = 0.8, \alpha = 0.6$

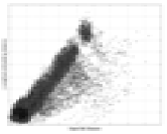

(k) $\rho = 0.8, \alpha = 0.8$

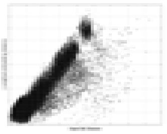

(l) $\rho = 0.8, \alpha = 1$


(m) $\rho = 1, \alpha = 0$


(n) $\rho = 1, \alpha = 0.2$


(o) $\rho = 1, \alpha = 0.4$


(p) $\rho = 1, \alpha = 0.6$


(q) $\rho = 1, \alpha = 0.8$

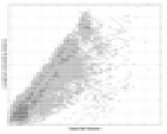
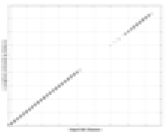
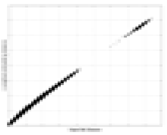
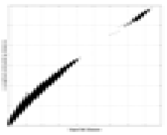
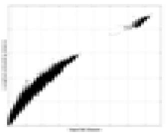

(r) $\rho = 1, \alpha = 1$

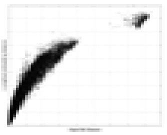
Figure B.6: AIDS Dataset: Scatter plots of the Graph Edit Distance versus the L_1 distance on the Embedding Features. Results for $\rho = 0\ 6\ \ 0\ 8\ \ 1$.

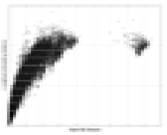

(a) $\rho = 0, \alpha = 0$

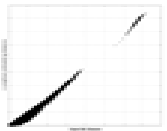

(b) $\rho = 0, \alpha = 0.2$

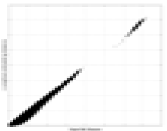

(c) $\rho = 0, \alpha = 0.4$

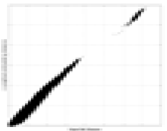

(d) $\rho = 0, \alpha = 0.6$

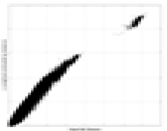

(e) $\rho = 0, \alpha = 0.8$

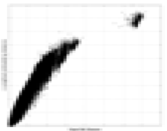

(f) $\rho = 0, \alpha = 1$

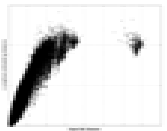

(g) $\rho = 0.2, \alpha = 0$

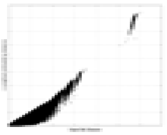

(h) $\rho = 0.2, \alpha = 0.2$

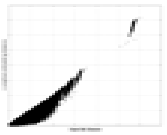

(i) $\rho = 0.2, \alpha = 0.4$

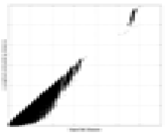

(j) $\rho = 0.2, \alpha = 0.6$

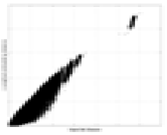

(k) $\rho = 0.2, \alpha = 0.8$

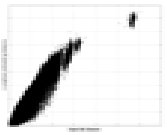

(l) $\rho = 0.2, \alpha = 1$


(m) $\rho = 0.4, \alpha = 0$


(n) $\rho = 0.4, \alpha = 0.2$


(o) $\rho = 0.4, \alpha = 0.4$


(p) $\rho = 0.4, \alpha = 0.6$


(q) $\rho = 0.4, \alpha = 0.8$

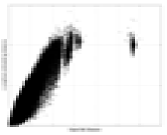
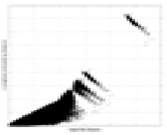
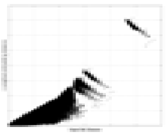
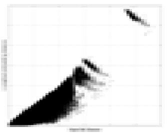
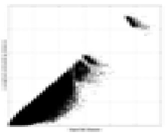

(r) $\rho = 0.4, \alpha = 1$

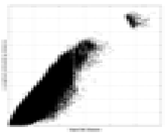
Figure B.7: MUTAG Dataset: Scatter plots of the Graph Edit Distance versus the L_1 distance on the Embedding Features. Results for $\rho = 0\ \ 0\ 2\ \ 0\ 4$.

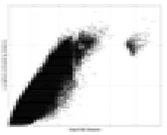

(a) $\rho = 0.6, \alpha = 0$

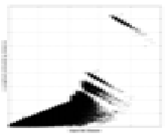

(b) $\rho = 0.6, \alpha = 0.2$

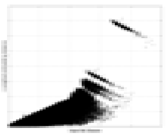

(c) $\rho = 0.6, \alpha = 0.4$

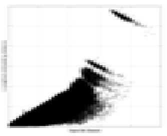

(d) $\rho = 0.6, \alpha = 0.6$

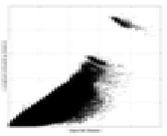

(e) $\rho = 0.6, \alpha = 0.8$

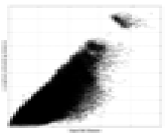

(f) $\rho = 0.6, \alpha = 1$

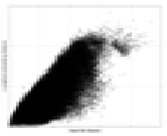

(g) $\rho = 0.8, \alpha = 0$

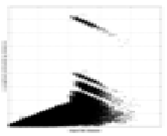

(h) $\rho = 0.8, \alpha = 0.2$

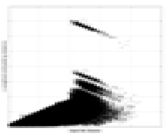

(i) $\rho = 0.8, \alpha = 0.4$

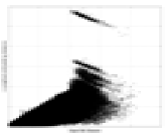

(j) $\rho = 0.8, \alpha = 0.6$

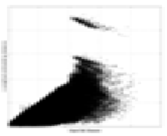

(k) $\rho = 0.8, \alpha = 0.8$

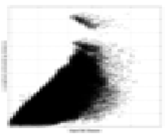

(l) $\rho = 0.8, \alpha = 1$


(m) $\rho = 1, \alpha = 0$


(n) $\rho = 1, \alpha = 0.2$


(o) $\rho = 1, \alpha = 0.4$


(p) $\rho = 1, \alpha = 0.6$


(q) $\rho = 1, \alpha = 0.8$

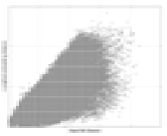
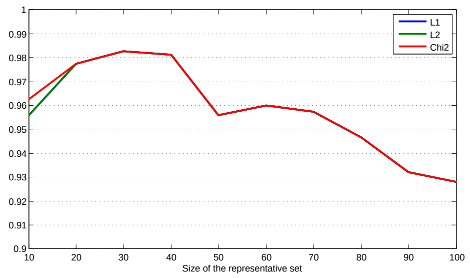

(r) $\rho = 1, \alpha = 1$

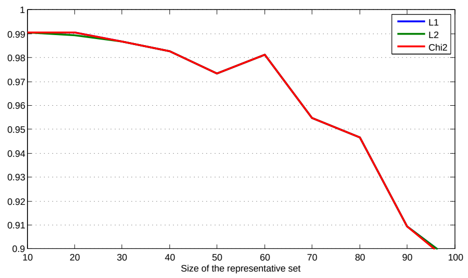
Figure B.8: MUTAG Dataset: Scatter plots of the Graph Edit Distance versus the L_1 distance on the Embedding Features. Results for $\rho = 0\ 6\ \ 0\ 8\ \ 1$.

Appendix C

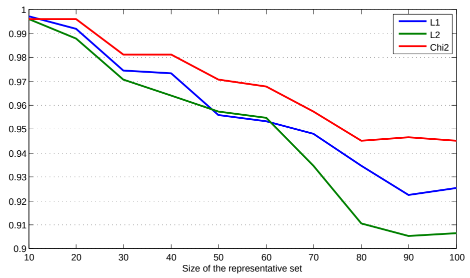
Study of Different Distances for the Continuous Attributed Graphs



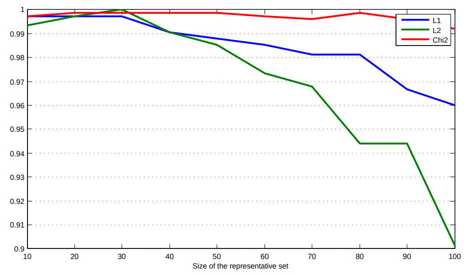
(a) Spanning prototypes



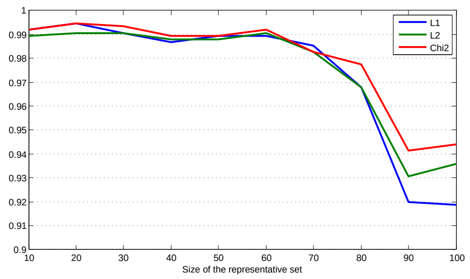
(b) *k*Means



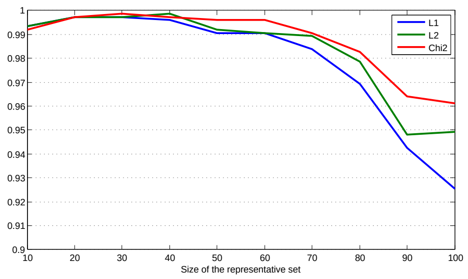
(c) Fuzzy *k*Means - *max*



(d) Fuzzy *k*Means - *all*

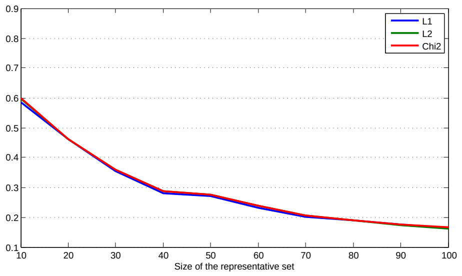


(e) Mixture of Gaussians - *max*

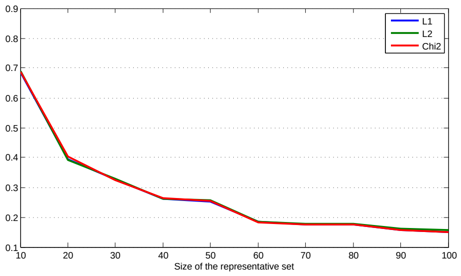


(f) Mixture of Gaussians - *all*

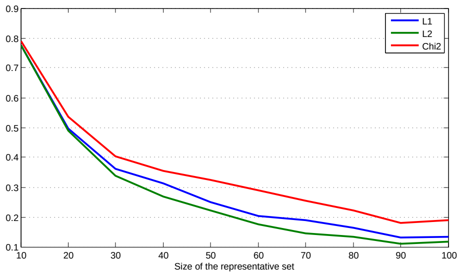
Figure C.1: Study of the effect of distances for the Letter LOW database using a *k*NN classifier.



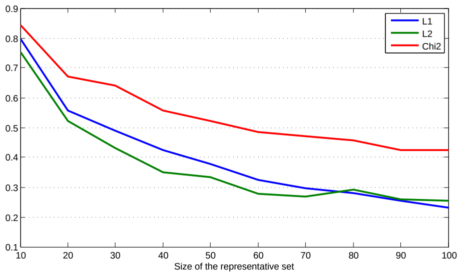
(a) Spanning prototypes



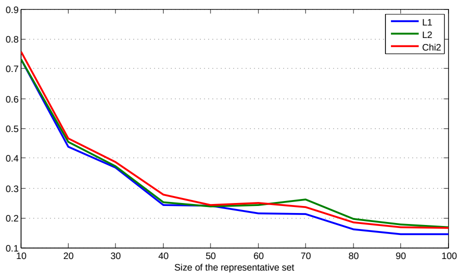
(b) *k*Means



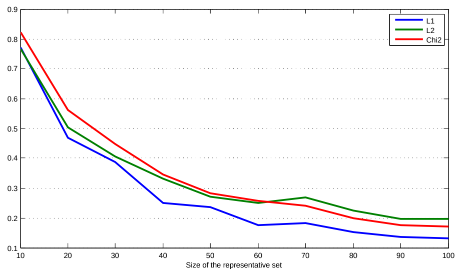
(c) Fuzzy *k*Means - *max*



(d) Fuzzy *k*Means - *all*

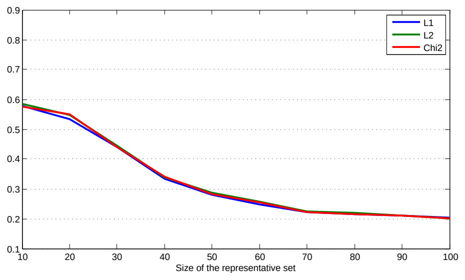


(e) Mixture of Gaussians - *max*

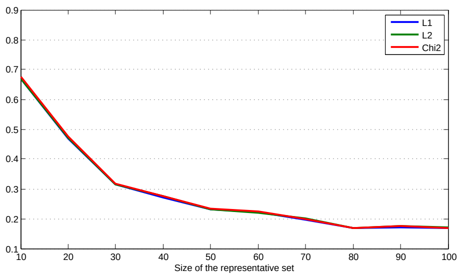


(f) Mixture of Gaussians - *all*

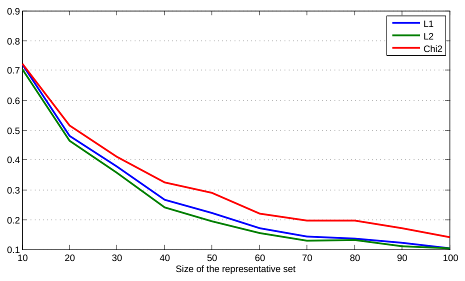
Figure C.2: Study of the effect of distances for the Letter MEDIUM database using a *k*NN classifier.



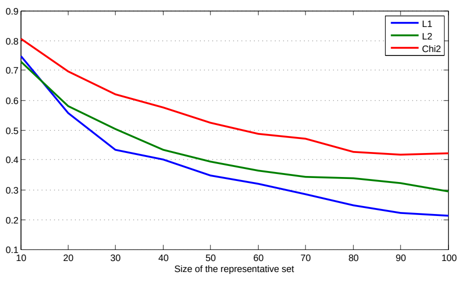
(a) Spanning prototypes



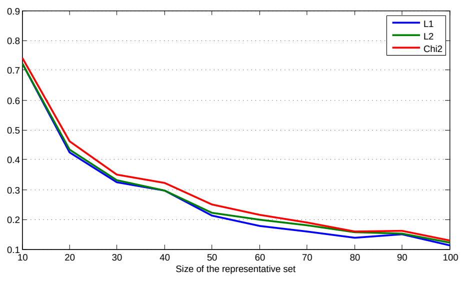
(b) *k*Means



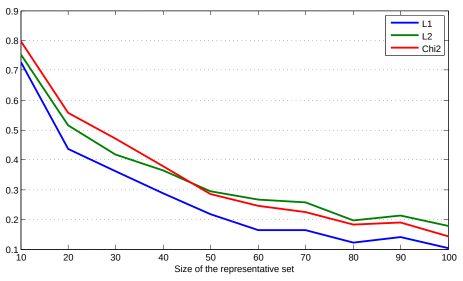
(c) Fuzzy *k*Means - *max*



(d) Fuzzy *k*Means - *all*

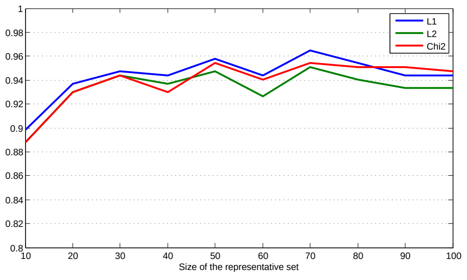


(e) Mixture of Gaussians - *max*

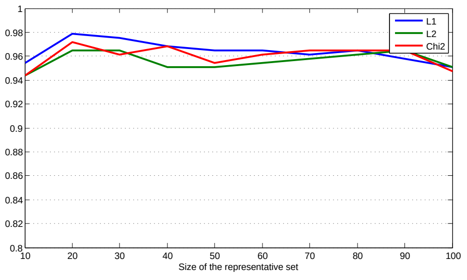


(f) Mixture of Gaussians - *all*

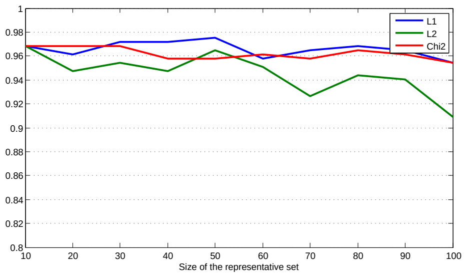
Figure C.3: Study of the effect of distances for the Letter HIGH database using a *k*NN classifier.



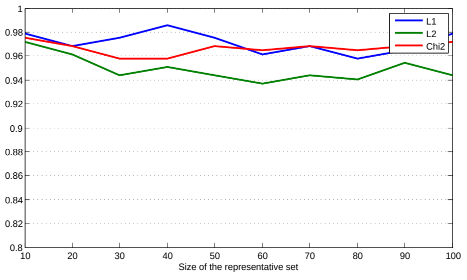
(a) Spanning prototypes



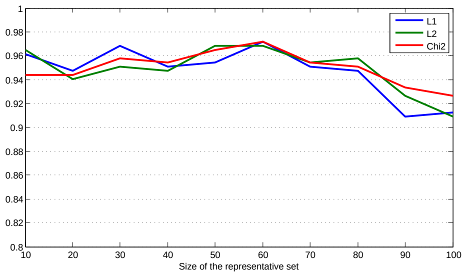
(b) *k*Means



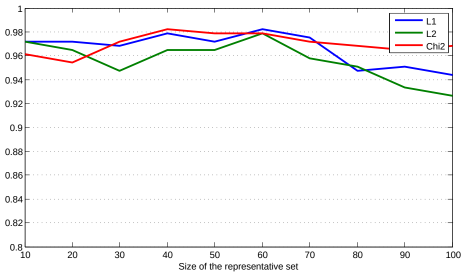
(c) Fuzzy *k*Means - *max*



(d) Fuzzy *k*Means - *all*

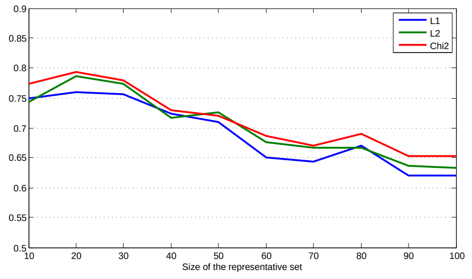


(e) Mixture of Gaussians - *max*

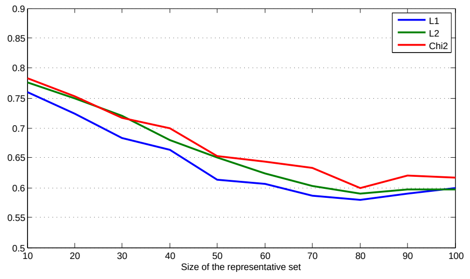


(f) Mixture of Gaussians - *all*

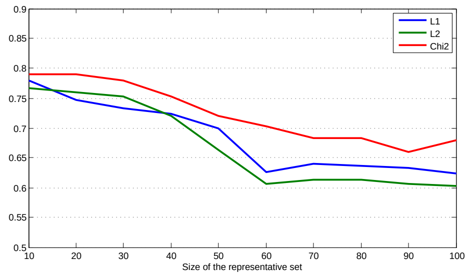
Figure C.4: Study of the effect of distances for the GREC database using a *k*NN classifier.



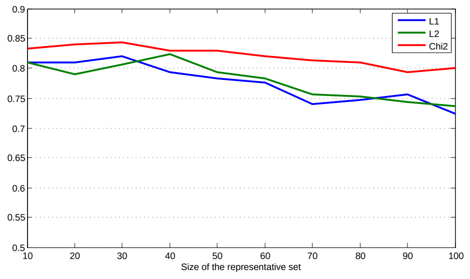
(a) Spanning prototypes



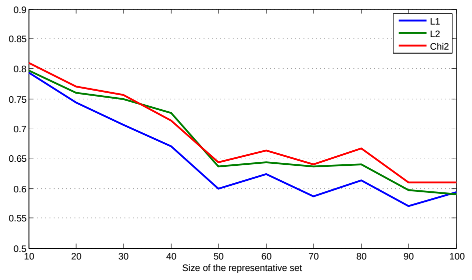
(b) kMeans



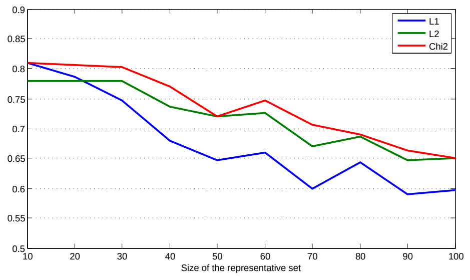
(c) Fuzzy kMeans - *max*



(d) Fuzzy kMeans - *all*

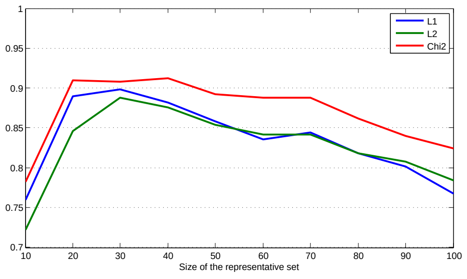


(e) Mixture of Gaussians - *max*

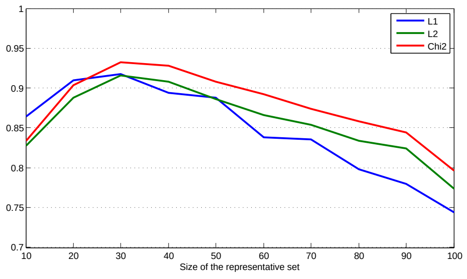


(f) Mixture of Gaussians - *all*

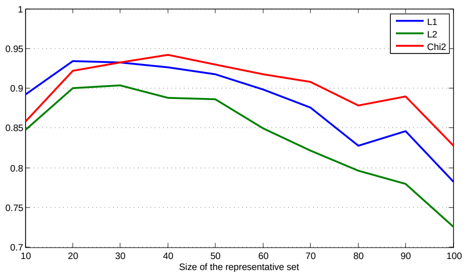
Figure C.5: Study of the effect of distances for the Fingerprints database using a kNN classifier.



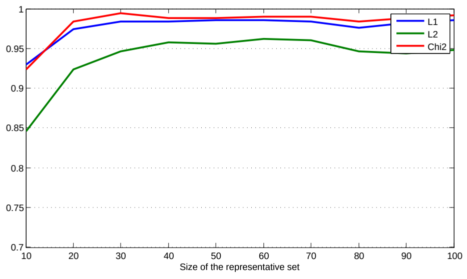
(a) Spanning prototypes



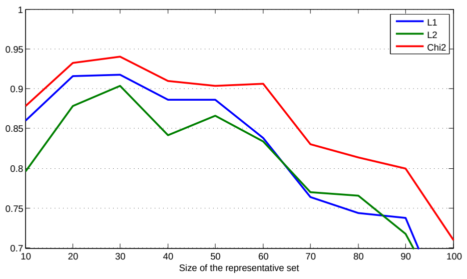
(b) *k*Means



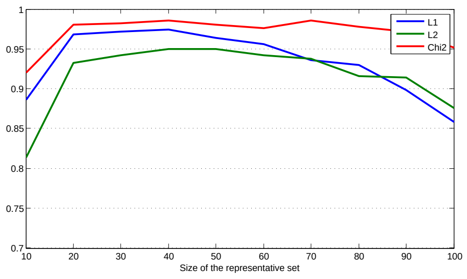
(c) Fuzzy *k*Means - *max*



(d) Fuzzy *k*Means - *all*



(e) Mixture of Gaussians - *max*



(f) Mixture of Gaussians - *all*

Figure C.6: Study of the effect of distances for the COIL database using a *k*NN classifier.

List of Publications

This dissertation has led to the following communications:

Journal Papers

- ***Graph embedding in vector spaces by node attribute statistics***. Jaume Gibert, Ernest Valveny and Horst Bunke.
  Pattern Recognition, 45(9):3072-3083, 2012.

- ***Feature selection on node statistics based embedding of graphs***. Jaume Gibert, Ernest Valveny and Horst Bunke.
  Pattern Recognition Letters, 2012. doi: 10.1016/j.patrec.2012.03.017.

- ***Embedding of graphs with discrete attributes via label frequencies***. Jaume Gibert, Ernest Valveny and Horst Bunke.
  Submitted to Internation Journal of Pattern Recognition and Artificial Intelligence.

Conference Contributions

- ***Vocabulary selection for graph of words embedding***. Jaume Gibert, Ernest Valveny and Horst Bunke.
  In: J. Vitrià , J.M. Sanches, M.Hernández (Eds.), Proceedings of the 5th Iberian Conference on Pattern Recognition and Image Analysis, Lecture Notes in Computer Science, vol.6669, Springer, 2011, pp. 216-223.

- ***Dimensionality reduction for graph of words embedding***. Jaume Gibert, Ernest Valveny and Horst Bunke.
  In: X. Jiang, M. Ferrer., A. Torsello (Eds.), Proceedings of the 8th IAPR TC-15 International Workshop on Graph-Based Representations for Pattern Recognition, Lecture Notes in Computer Science, vol.6658, Springer, 2011, pp. 22-31.

- ***Multiple classifiers for graph of words embedding***. Jaume Gibert, Ernest Valveny, Oriol Ramos Terrades and Horst Bunke.
  In: C. Sansone, J. Kittler., F. Roli (Eds.), Proceedings of the 10th International Workshop on Multiple Classifier Systems, Lecture Notes in Computer Science, vol.6713, Springer, 2011, pp. 36-45.

- ***Graph of words embedding for molecular structure-activity relationship analysis***. Jaume Gibert, Ernest Valveny and Horst Bunke.
  In: I. Bloch, R.M. CesarJr.(Eds.), Proceedings of the 15th Iberoamerican Congress on Pattern Recognition, Lecture Notes in Computer Science, vol.6419, Springer, 2010, pp. 30-37.

- ***Graph embedding based on nodes attributes representatives and a graph of words representation***. Jaume Gibert and Ernest Valveny.
  In: E.R. Hancock, R.C. Wilson., T. Windeatt, I. Ulusoy, F. Escolano (Eds.), Proceedings of the Joint IAPR International Workshops on Structural and Syntactic Pattern Recognition and Statistical Pattern Recognition, Lecture Notes in Computer Science, vol.6218, Springer, 2010, pp. 223-232.

- ***A kernel based approach to document retrieval***. Albert Gordo, Jaume Gibert, Ernest Valveny and Marçal Rusinol.
  In: Proceeding of the 9th IAPR International Workshop on Document Analysis Systems, ACM, 2010, pp. 377-384.

Local Workshops

- ***Advances on graph embedding by statistics on node representatives***. Jaume Gibert and Ernest Valveny.
  In: F. Diego, D. Gerónimo, J. Vázquez-Corral (Eds.), Proceedings of the 6th CVC Workshop on the Progress of Research & Development, 2011, pp. 86-89.

- ***Classifiers Ensembles of Reduced Graph of Words Embedded Graphs***. Jaume Gibert and Ernest Valveny.
  In: A. Fornés, A. Hernández-Sabaté, D. Ponsa, M. Rusinol (Eds.), Proceedings of the 5th CVC Workshop on the Progress of Research & Development, 2010, pp. 139-142.

- ***Graph-based Representations for Object Recogntion***. Jaume Gibert and Ernest Valveny.
  In: X. Baró, S. Escalera, M. Ferrer (Eds.), Proceedings of the 4th CVC Workshop on the Progress of Research & Development, 2009, pp. 185-190.

Bibliography

[1] *English Language Dictionary, Collins Birmingham University International Language Database.* William Collins Sons & Co. Ltd, 1987.

[2] Development therapeutics program DTP, AIDS antiviral screen. *http://dtp.nci.nih.gov/docs/aids/aids data.html*, 2004.

[3] E. Alpaydin and F. Alimoglu. Pen-based recognition of handwritten digits. *Department of Computer Engineering, Bogazici University*, 1998.

[4] R. Ambauen, S. Fischer, and H. Bunke. Graph edit distance with node splitting and merging, and its application to diatom identification. In E. Hancock and M. Vento, editors, *Proceedings of the 4th International Workshop on Graph Based Representations in Pattern Recognition*, volume 2726 of *Lecture Notes in Computer Science*, pages 259 264. Springer, 2003.

[5] N. Arica and F.T. Yarman-Vural. Optical character recognition for cursive handwritting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(6):801 813, 2002.

[6] S. Auwatanamongkol. Inexact graph matching using a genetic algorithm for image recognition. *Pattern recognition letters*, 28(12):1428 1437, 2007.

[7] F. Aziz, R. Wilson, and E. Hancock. Graph characterization via backtrackless paths. In M. Pelillo and E. Hancock, editors, *Proceedings of the first International Workshop on Similarity-Based Pattern Recognition*, volume 7005 of *Lecture Notes in Computer Science*, pages 149 162. Springer, 2011.

[8] M. Barthelemy. Spatial networks. *Physics review*, 499(1-3):1 101, 2011.

[9] R. Benavente, M. Vanrell, and R. Baldrich. Parametric fuzzy sets for automatic color naming. *J. Optical Society of America A*, 25(10):2582 2593, 2008.

[10] S. Berretti, A. Del Bimbo, and E. Vicario. The computational aspect of retrieval by spatial arrangement. In *Proceedings of the 15th International Conference on Pattern Recognition*, volume 1, pages 1047 1051, 2000.

[11] S. Berretti, A. Del Bimbo, and E. Vicario. A look-ahead strategy for graph matching in retrieval by spatial arrangement. In *IEEE International Conference on Multimedia and Expo*, volume 3, pages 1721 1724, 2000.

[12] S. Berretti, A. Del Bimbo, and E. Vicario. Efficient matching and indexing of graph models in content-based retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(10):1089 1105, 2001.

[13] R. Bertolami, S. Uchida, M. Zimmermann, and H. Bunke. Non-uniform slant correction for handwritten text line recognition. In *Proceedings of the 9th International Conference on Document Analysis and Recognition*, volume 1, pages 18 22, 2007.

[14] C. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2008.

[15] M. Boeres, C. Ribeiro, and I. Bloch. A randomized heuristic for scene recognition by graph matching. *Experimental and Efficient Algorithms*, pages 100 113, 2004.

[16] K. Borgwardt. *Graph Kernels.* PhD thesis, Ludwig-Maximilians-University Munich, 2007.

[17] K. Borgwardt, C. Ong, S. Schonauer, S. Vishwanathan, A. Smola, and H.P. Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(1):47 56, 2005.

[18] K.M. Borgwardt and H.P. Kriegel. Shortest-path kernels on graphs. In *IEEE International Conference on Data Mining*, pages 74 81. IEEE, 2005.

[19] R.D. Brown and Y.C. Martin. Use of structure-activity data to compare structure-based clustering methods and descriptors for use in compound selection. *Journal of Chemical Information and Computer Science*, 36(3):572 584, 1996.

[20] H. Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters*, 18(8):689 694, 1997.

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

[22] H. Bunke, P.J. Dickinson, M. Kraetzel, and W.D. Wallis. *Applied Graph Theory in Computer Vision and Pattern Recognition*, volume 24 of *Progress in Compter Science and Applied Logics (*PCS*).* Birkhauser, 2007.

[23] H. Bunke, P. Foggia, C. Guidobaldi, C. Sansone, and M. Vento. A comparison of algorithms for maximum common subgraph on randomly connected graphs. In T. Caelli, A. Amin, R. Duin, M. Kamel, and D. de Ridder, editors, *Proceedings of the International Workshops on Structural, Syntactic, and Statistical Pattern Recognition*, volume 2396 of *Lecture Notes in Computer Science*, pages 85 106. Springer, 2002.

[24] H. Bunke and K. Riesen. Improving vector space embedding of graphs through feature selection algorithms. *Pattern Recognition*, 44(9):1928 1940, 2011.

[25] H. Bunke and K. Riesen. Recent advances in graph-based pattern recognition with applications in document analysis. *Pattern Recognition*, 44(5):1057 1067, 2011.

[26] H. Bunke and K. Riesen. Towards the unification of structural and statistical pattern recognition. *Pattern Recognition Letters*, 33(7):811 825, 2012.

[27] H. Bunke and K. Shearer. A graph distance metric based on the maximal common subgraph. *Pattern recognition letters*, 19(3):255 259, 1998.

[28] T. Caelli and S. Kosinov. Inexact graph matching using eigen-subspace projection clustering. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(3):329 354, 2004.

[29] C.C. Chang and C.J. Lin. Libsvm: A library for support vector machines. *http://www.csie.ntu.edu.tw/∼cjlin/libsvm*, 2001.

[30] W.J. Christmas, J. Kittler, and M. Petrou. Structural matching in computer vision using probabilistic relaxation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):749 764, 1995.

[31] F.R.K. Chung. *Spectral graph theory.* Number 92. Amer Mathematical Society, 1997.

[32] 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(3):265 298, 2004.

[33] D. Cook and L. Holder, editors. *Mining Graph Data.* Wiley-Interscience, 2007.

[34] L.P. Cordella, P. Foggia, C. Sansone, and M. Vento. Fast graph matching for detecting CAD image components. In *Proceedings of the 15th International Conference on Pattern Recognition*, volume 2, pages 1038 1041, 2000.

[35] L.P. Cordella, P. Foggia, C. Sansone, and M. Vento. An improved algorithm for matching large graphs. In *3rd IAPR-TC15 Workshop on graph-based representations in pattern recognition*, pages 149 159, 2001.

[36] L.P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub) graph isomorphism algorithm for matching large graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(10):1367 1372, 2004.

[37] T. Cox and M. Cox. *Multidimensionality scaling.* Chapman and Hall, 1994.

[38] A.D.J. Cross, R.C. Wilson, and E.R. Hancock. Inexact graph matching using genetic search. *Pattern Recognition*, 30(6):953 970, 1997.

[39] M.F. Demirci, Y. Osmanlıoğlu, A. Shokoufandeh, and S. Dickinson. Efficient many-to-many feature matching under the l_1 norm. *Computer Vision and Image Understanding*, 115(7):976 983, 2011.

[40] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood for incomplete data via the em algorithm. *Journal of the Royal Statistical Society B*, 39(1):1 38, 1977.

[41] P.J. Dickinson, H. Bunke, A. Dadej, and M. Kraetzl. Matching graphs with unique node labels. *Pattern Analysis and Applications*, 7(3):243 254, 2004.

[42] P.J. Dickinson, M. Kraetzl, H. Bunke, M. Neuhaus, and A. Dadej. Similarity measures for hierarchical representations of graphs with unique node labels. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(3):425 442, 2004.

[43] P. Dosch and E. Valveny. Report on the second symbol recognition contest. In W. Liu and J .Lladós, editors, *Graphics Recognition. Ten Years Review and Future Perspectives. Proceedings of the 6th International Workshop on Graphics Recognition*, volume 3926 of *Lecture Notes in Computer Science*, pages 381 397. Springer, 2005.

[44] O. Duchenne, A. Joulin, and J. Ponce. A graph-matching kernel for object categorization. In *IEEE International Conference on Computer Vision*, pages 1792 1799, 2011.

[45] R. Duda, P. Hart, and D. Stork. *Pattern Classification.* Wiley-Interscience, 2000.

[46] A.C.M. Dumay, R.J. van der Geest, J.J. Gerbrands, E. Jansen, and J.H.C. Reiber. Consistent inexact graph matching applied to labelling coronary segments in arteriograms. In *Proceedings of 11th IAPR International Conference on Pattern Recognition*, volume 3, pages 439 442, 1992.

[47] M.A. Eshera and K.S. Fu. A graph distance measure for image analysis. *IEEE Transactions on Systems, Man, and Cybernetics (Part B)*, 14(3):398 408, 1984.

[48] E. Estrada. *The structure of complex networks. Theory and Applications.* Oxford University Press, 2011.

[49] U. Fayyad and K. Irani. Multi-interval discretization of continuous valued attributes for classification learning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1022 1027. Morgan Kaufmann, 1993.

[50] P.F. Felzenszwalb and D.P. Huttenlocher. Efficient graph-based image segmentation. *International Journal in Computer Vision*, 59(2):167 181, 2004.

[51] J. Feng, M. Laumy, and M. Dhome. Inexact matching using neural networks. *Machine Intelligence and Pattern Recognition*, 16:177 177, 1994.

[52] M.L. Fernández and G. Valiente. A graph distance metric combining maximum common subgraph and minimum common supergraph. *Pattern Recognition Letters*, 22(6-7):753 758, 2001.

[53] A.M. Finch, R.C. Wilson, and E.R. Hancock. Symbolic graph matching with the em algorithm. *Pattern Recognition*, 31(11):1777 1790, 1998.

[54] M.A. Fischler and R.A. Elschlager. The representation and matching of pictorial structures. *IEEE Transactions on Computers*, 100(1):67 92, 1973.

[55] P. Foggia and M. Vento. Graph embedding for pattern recognition. In D. Unay, Z. Çataltepe, and S. Aksoy, editors, *Proceedings of the 20th International Conference on Pattern Recognition*, volume 6388 of *Lecture Notes in Computer Science*, pages 75 82. Springer, 2010.

[56] P. Frasconi, M. Gori, and A. Sperduti. A general framework for adaptive processing of data structures. *IEEE Transactions on Neural Networks*, 9(5):768 786, 1998.

[57] T. Gartner. *Kernels for Structured Data*. World Scientific, 2008.

[58] T. Gartner, P. Flach, and S. Wrobel. On graph kernels: hardness results and efficient alternatives. In B. Scholkopf and M. Warmuth, editors, *Proceedings of the 16th Annual Conference on Learning Theory*, pages 129 143, 2003.

[59] B. Gauzère, L. Brun, and D. Villemin. Two new graph kernels and applications to chemoinformatics. In X. Jiang, M. Ferrer, and A. Torello, editors, *IAPR International Workshop on Graph-Based Representations in Pattern Recognition*, volume 6658 of *Lecture Notes in Computer Science*, pages 112 121. Springer, 2011.

[60] J.M. Geusebroek, G.J. Burghouts, and A.W.M. Smeulders. The amsterdam library of object images. *International Journal in Computer Vision*, 61(1):103 112, 2005.

[61] T. Krishnan G.J. McLachlan. *The EM algorithm and its extensions*. Wiley, 1997.

[62] M. Gori, M. Maggini, and L. Sarti. Exact and approximate graph matching using random walks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7):1100 1111, 2005.

[63] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1-3):389 422, 2002.

[64] I. Gyon, S. Gunn, and M. Nikravesh, editors. *Feature Extraction, Foundations and Applications*. Springer, 2006.

[65] B. Haasdonk. Feature space interpretation of SVMs with indefinite kernels. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(4):482 492, 2005.

[66] E.R. Hancock and J. Kittler. Discrete relaxation. *Pattern Recognition*, 23(7):711 733, 1990.

[67] E.R. Hancock and R.C. Wilson. Pattern analysis with graphs: Parallel work at Bern and York. *Pattern Recognition Letters*, 33(7):833 841, 2012.

[68] Z. Harchaoui and F. Bach. Image classification with segmentation graph kernels. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 147 154, 2005.

[69] C. Harris and M. Stephens. A combined corner and edge detector. In *Proceedings of the 4th Alvey Vision Conference*, pages 147 151, 1988.

[70] D. Haussler. Convolution kernels on discrete structures. Technical report, University of California, Santa Cruz, 1999.

[71] E. Hjelmasa and B.K. Lowb. Face detection: A survey. *Computer Vision and Image Understanding*, 83(3):236 274, 2001.

[72] T.K. Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832 844, 1998.

[73] T. Horváth. Cyclic pattern kernels revisited. *Advances in knowledge discovery and data mining*, pages 139 140, 2005.

[74] T. Horváth, T. Gartner, and S. Wrobel. Cyclic pattern kernels for predictive graph mining. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 158 167. ACM, 2004.

[75] L. Hubert and J. Schultz. Quadratic assignment as a general data-analysis strategy. *British Journal of Mathematical and Statistical Psychology*, 29:190 241, 1976.

[76] B. Huet and E.R. Hancock. Shape recognition from large image libraries by inexact graph matching. *Pattern Recognition Letters*, 20(11):1259 1269, 1999.

[77] J.J. Hull, G. Krishnan, P. Palumbo, and S.N. Srihari. Optical character recognition techniques in mail sorting: A review of algorithms. Technical report, Department of Computer Science, University of Buffalo, State University of New York, 1984.

[78] A. Hyvarinen, J. Karhunen, and E. Oja. *Independent component analysis*, volume 26. Wiley-interscience, 2001.

[79] A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *Proceedings of the 4th PKDD*, pages 13 32, 2000.

[80] A.K. Jain, S. Prabhakar, and L. hong. A multichannel approach to fingerprint classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(4):348 359, 1999.

[81] B.J. Jain and F. Wysotzki. Solving inexact graph isomorphism problems using neural networks. *Neurocomputing*, 63:45 67, 2005.

[82] I.T. Jolliffe. *Principal component analysis*. Springer, 1986.

[83] S. Jouili and S. Tabbone. Graph matching based on node signatures. In A. Torsello, F. Escolano, and L. Brun, editors, *Proceedings of the 7th IAPR TC-15 International Workshop on Graph-Based Representations in Pattern Recognition*, volume 5534 of *Lecture Notes in Computer Science*, pages 154 163. Springer, 2009.

[84] S. Jouili and S. Tabbone. Graph embedding using constant shift embedding. In D. Unay, Z. Çataltepe, and S. Aksoy, editors, *Proceeding of the 20th International Conference on Pattern Recognition*, volume 6388 of *Lecture Notes in Computer Science*, pages 83 92. Springer, 2010.

[85] A. Juan and E. Vidal. On the use of bernoulli mixture models for text classification. *Pattern Recognition*, 35(12):2705 2710, 2002.

[86] D. Justice and A. Hero. A binary linear programming formulation of the graph edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(8):1200 1214, 2006.

[87] A. Kandel and H. Bunke, editors. *Applied Graph Theory in Computer Vision and Pattern Recognition*, volume 52 of *Studies in Computational Science*. Springer, 2007.

[88] J. Kandola, J. Shawe-Taylor, and N. Cristianini. Learning semantic similarity. *Advances in Neural Information Processing Systems*, 15:657 664, 2002.

[89] H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labelled graphs. In *Proceedings of the 20th International Conference on Machine Learning*, pages 321 328, 2003.

[90] J. Kazius, R. McGuire, and R. Bursi. Derivation and validation of toxicophores for mutagenicity prediction. *J. Med. Chemistry*, 45(1):312 320, 2005.

[91] K. Kira and L.A. Rendell. The feature selection problem: Traditional methods and a new algorithm. In *Proceeding of the 9th International Conference on Artificial Intelligence*, pages 129 134, 1992.

[92] J. Kittler and E.R. Hancock. Combining evidence in probabilistic relaxation. *International Journal of Pattern Recognition and Artificial Intelligence*, 3(1):29 51, 1989.

[93] J. Kittler, M. Hatef, R.P.W. Duin, and J. Matas. On combining classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(3):226 239, 1998.

[94] R. Kondor and J. Lafferty. Diffusion kernels on graphs and other discrete input spaces. In *Proceedings of 19th International Conference on Machine Learning*, pages 315 322, 2002.

[95] S. Kosinov and T. Caelli. Inexact multisubgraph matching using graph eigenspace and clustering models. In T. Caelli, A. Amin, R. P. W. Duin, M. S. Kamel, and D. de Ridder, editors, *Joint IAPR International Workshops on Structural, Syntactic, and Statistical Pattern Recognition*, volume 2396 of *Lecture Notes in Computer Science*, pages 133 142. Springer, 2002.

[96] S. Kramer and L. De Raedt. Feature construction with version spaces for biochemical application. In *Proceedings of the 18th International Conference on Machine Learning*, pages 258 265, 2001.

[97] L.I. Kuncheva. *Combining Pattern Classifiers: Methods and Algorithms.* John Wiley, 2004.

[98] J. Lafferty and G. Lebanon. Information diffusion kernels. In *Advances in Neural Information Processing Systems*, volume 15, pages 375 382. MIT Press, 2003.

[99] J. Lafferty and G. Lebanon. Diffusion kernels on statistical manifolds. *Journal of Machine Learning Research*, 6:129 163, 2005.

[100] J. Larrosa and G. Valiente. Constraint satisfaction algorithms for graph pattern matching. *Mathematical Structures in Computer Science*, 12(4):403 422, 2002.

[101] W.J. Lee and R.P.W. Duin. A labelled graph based multiple classifier system. In *Proceedings of the 8th International Workshop on Multiple Classifier Systems*, volume 5519 of *Lecture Notes in Computer Science*, pages 201 210. Springer, 2009.

[102] W.J. Lee, R.P.W. Duin, and H. Bunke. Selecting structural base classifiers for graph-based multiple classifier systems. In *Proceedings of the 9th International Workshop on Multiple Classifier Systems*, volume 5997 of *Lecture Notes in Computer Science*, pages 155 164. Springer, 2010.

[103] V.I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet Physics Doklady*, volume 10, pages 707 710, 1966.

[104] G. Levi. A note on the derivation of maximal common subgraphs of two directed or undirected graphs. *Calcolo*, 9(4):341 352, 1973.

[105] M. Liwicki and H. Bunke. *Recognition of Whiteboard Notes – Online, Offline and Combination*. World Scientific, 2008.

[106] J. Lladós and G. Sánchez. Graph matching versus graph parsing in graphics recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(3):455 475, 2004.

[107] Y. Lu and C.L. Tan. Combination of multiple classifiers using probabilistic dictionary and its application to postcode recognition. *Pattern Recognition*, 35(12):2823 2832, 2000.

[108] B. Luo and E.R. Hancock. Structural graph matching using the em algorithm and singular value decomposition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(10):1120 1136, 2001.

[109] B. Luo, R. Wilson, and E. Hancock. Spectral feature vectors for graph clustering. In T. Caelli, A. Amin, R. P. W. Duin, M. S. Kamel, and D. de Ridder, editors, *Joint IAPR International Workshops on Structural, Syntactic, and Statistical Pattern Recognition*, volume 2396 of *Lecture Notes in Computer Science*, pages 423 454. Springer, 2002.

[110] B. Luo, R.C. Wilson, and E.R. Hancock. Spectral embedding of graphs. *Pattern Recognition*, 36(10):2213 2230, 2003.

[111] M.M. Luqman, J. Lladós, J.-Y. Ramel, and T. Brouard. A fuzzy-interval based approach for explicit graph embedding. In D. Unay, Z. Çataltepe, and S. Aksoy, editors, *Proceeding of the 20th International Conference on Pattern Recognition*, volume 6388 of *Lecture Notes in Computer Science*, pages 93 98. Springer, 2010.

[112] P. Mahé, N. Ueda, and T. Akutsu. Graph kernels for molecular structure-activity relationship analysis with support vector machines. *Journal of Chemical Information and Modeling*, 45(4):939 951, 2005.

[113] P. Mahé, N. Ueda, T. Akutsu, J.L. Perret, and J.P. Vert. Extensions of marginalized graph kernels. In C.E. Brodley, editor, *Proceedings of the twenty-first international conference on Machine learning*, volume 69. ACM, 2004.

[114] J.J. McGregor. Backtrack search algorithms and the maximal common subgraph problem. *Software: Practice and Experience*, 12(1):23 34, 1982.

[115] B.D. McKay. Practical graph isomorphism. *Congressus Numerantum*, 30:45 87, 1981.

[116] J. Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32 38, 1957.

[117] R. Myers, R.C. Wison, and E.R. Hancock. Bayesian graph edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(6):628 635, 2000.

[118] S. Nene, S. Nayar, and H. Murase. Columbia object image library: Coil-100. Technical report, Department of Computer Science, Columbia University, 1996.

[119] M. Neuhaus and H. Bunke. A graph matching based approach to fingerprint classification using directional variance. In T. Kanade, A.K. Jain, and N.K. Ratha, editors, *Proceedings of the 5th International Conference in Audio- and Video-Based Biometric Person Authentication*, volume 3546 of *Lecture Notes in Computer Science*, pages 191 200. Springer, 2005.

[120] M. Neuhaus and H. Bunke. Self-organizing maps for learning the edit costs in graph matching. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 35(3):503 514, 2005.

[121] M. Neuhaus and H. Bunke. Automatic learning of cost functions for graph edit distance. *Information Sciences*, 177(1):239 247, 2007.

[122] M. Neuhaus and H. Bunke. *Bridging the Gap between Graph Edit Distance and Kernel Machines*. World Scientific, 2007.

[123] M. Neuhaus, K. Riesen, and H. Bunke. Fast suboptimal algorithms for the computation of graph edit distance. *Structural, Syntactic, and Statistical Pattern Recognition*, pages 163 172, 2006.

[124] E. Pekalska and R. Duin. *The Dissimilarity Representation for Pattern Recognition: Foundations and Applications*. World Scientific, 2005.

[125] E. Pekalska, R. Duin, and P. Paclick. Prototype selection for dissimilarity-based classifiers. *Pattern Recognition*, 39(2):189 208, 2006.

[126] P. Pudil, J. Novovicova, and J. Kittler. Floating search methods in feature-selection. *Pattern Recognition Letters*, 15(11):1119 1125, 1994.

[127] L. Ralaivola, S.J. Swamidass, H. Saigo, and P. Baldi. Graph kernels for chemical informatics. *Neural Networks*, 18(8):1093 1110, 2005.

[128] J. Ramon and T. Gartner. Expressivity versus efficiency of graph kernels. In *First International Workshop on Mining Graphs, Trees and Sequences*, pages 65 74, 2003.

[129] O. Ramos, E. Valveny, and S. Tabbone. Optimal classifier fusion in a non-bayesian probabilistic framework. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(9):1630 1644, 2009.

[130] P. Ren, R. Wilson, and E. Hancock. Graph characterization via ihara coefficients. *IEEE Transactions on Neural Networks*, 22(2):233 245, 2011.

[131] K. Riesen and H. Bunke. Classifier ensembles for vector space embedding of graphs. In *Proceedings of the 7th International Workshop on Multiple Classifier Systems*, volume 4472 of *Lecture Notes in Computer Science*, pages 220 230. Springer, 2007.

[132] K. Riesen and H. Bunke. IAM graph database repository for graph based pattern recognition and machine learning. In N. da Vitoria Lobo et al., editor, *SSPR&SPR*, volume 5342 of *Lecture Notes in Computer Science*, pages 287 297. Springer-Verlag, 2008.

[133] K. Riesen and H. Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image and Vision Computing*, 27(4):950 959, 2009.

[134] K. Riesen and H. Bunke. *Graph Classification and Clustering Based on Vector Space Embedding*, volume 77 of *Machine Perception and Artificial Intelligence*. World Scientific, 2010.

[135] A. Robles-Kelly and E.R. Hancock. String edit distance, random walks and graph matching. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(3):315 327, 2004.

[136] A. Robles-Kelly and E.R. Hancock. Graph edit distance from spectral seriation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(3):365 378, 2005.

[137] A. Robles-Kelly and E.R. Hancock. A riemannian approach to graph embedding. *Pattern Recognition*, 40(3):1042 1056, 2007.

[138] J. Rocha and T. Pavlidis. A shape analysis model with applications to a character recognition system. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(4):393 404, 1994.

[139] A. Sanfeliu and K.S. Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics (Part B)*, 13(3):353 363, 1983.

[140] A. Schenker, H. Bunke, M. Last, and A. Kandel. Building graph-based classifier ensembles by random node selection. In *Proceedings of the 5th International Workshop on Multiple Classifier Systems*, volume 3077 of *Lecture Notes in Computer Science*, pages 214 222. Springer, 2004.

[141] A. Schenker, H. Bunke, M. Last, and A. Kandel. Classification of web documents using graph matching. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(3):475 496, 2004.

[142] A. Schenker, H. Bunke, M. Last, and A. Kandel. *Graph-Theoretic Techinques for Web Content Mining*. World Scientific, 2005.

[143] A. Schlapbach and H. Bunke. A writer identification and verification system using HMM based recognizers. *Pattern Analysis and Applications*, 10(1):33 43, 2007.

[144] A. Schlapbach and H. Bunke. Offline writer identification and verification using gaussian mixture models. In S. Marinai and H. Fujisawa, editors, *Machine Learning in Document Analysis and Recognition*, volume 90. Springer, 2008.

[145] D.C. Schmidt and L.E. Druffel. A fast backtracking algorithm to test directed graphs for isomorphism using distance matrices. *Journal of* ACM, 23(3):433 445, 1976.

[146] B. Scholkopf, A. Smola, and K.R. Muller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10:1299 1319, 1998.

[147] B. Scholkopf and A.J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* MIT Press, 2002.

[148] F. Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1 47, 2002.

[149] S.M. Selkow. The tree-to-tree correction problem. *Information Processing Letters*, 6(6):184 186, 1977.

[150] F. Serratosa, R. Alquézar, and A. Sanfeliu. Function-described graphs: a fast algorithm to compute a sub-optimal matching measure. In *Proceedings of the 2nd IAPR-TC15 Workshop on graph-based representations in pattern recognition*, pages 71 77, 1999.

[151] J. Shawe and N. Cristianini. *Kernel methods for pattern analysis.* Cambridge University Press, 2004.

[152] N. Shervashidze, SVN Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt. Efficient graphlet kernels for large graph comparison. In *Proceedings of the International Workshop on Artificial Intelligence and Statistics. Society for Artificial Intelligence and Statistics*, 2009.

[153] A. Shokoufandeh, D. Macrini, S. Dickinson, K. Siddiqi, and S.W. Zucker. Indexing hierarchical structures using graph spectra. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7):1125 1140, 2005.

[154] N. Sidere, P. Heroux, and J.Y. Ramel. A vectorial representation for the indexation of structural informations. In F. Roli J.T.-Y. Kwok M. Georgiopoulos G.C.. Anagnostopoulos M. Loog N. da Vitoria Lobo, T. Kasparis, editor, *Joint IAPR International Workshops on Structural, Syntactic, and Statistical Pattern Recognition*, volume 5342 of *Lecture Notes in Computer Science*, pages 45 54. Springer, 2008.

[155] N. Sidere, P. Héroux, and J.Y. Ramel. Vector representation of graphs: application to the classification of symbols and letters. In *10th International Conference on Document Analysis and Recognition*, pages 681 685. IEEE, 2009.

[156] M. Singh, A. Chatterjee, and S. Chaudhury. Matching structural shape descriptions using genetic algorithms. *Pattern Recognition*, 30(9):1451 1462, 1997.

[157] A. Smola and R. Kondor. Kernels and regularization on graphs. *Learning theory and kernel machines*, pages 144 158, 2003.

[158] S. Sorlin and C. Solnon. Reactive tabu search for measuring graph similarity. In L. Brun and M. Vento, editors, *Proceedings of the 5th IAPR International Workshop on Graph-Based Representations in Pattern Recognition*, volume 3434 of *Lecture Notes in Computer Science*, pages 172 182. Springer, 2005.

[159] A. Sperduti and A. Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714 735, 1997.

[160] P.N. Suganthan. Structural pattern recognition using genetic algorithms. *Pattern Recognition*, 35(9):1883 1893, 2002.

[161] M.J. Tarr. The object databank.
http://www.cnbc.cmu.edu/tarrlab/stimuli/objects/index.html.

[162] J.B. Tenenbaum, V. De Silva, and J.C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290:2319 2323, 2000.

[163] W.H. Tsai and K.S. Fu. Error-correcting isomorphisms of attributed relational graphs for pattern analysis. *Systems, Man and Cybernetics, IEEE Transactions on*, 9(12):757 768, 1979.

[164] J.R. Ullman. An algorithm for subgraph isomorphism. *Journal of* ACM, 23(1):31 42, 1976.

[165] S. Umeyama. An eigendecomposition approach to weighted graph matching problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(5):695 703, 1988.

[166] J. van Gemert, J. Geusebroek, C. Veenman, and A. Smeulders. Kernel codebooks for scene categorization. In D. Forsyth, P. Torr, and A. Zisserman, editors, *Proceedings of the 10th European Conference in Computer Vision*, volume 5304 of *Lecture Notes in Computer Science*, pages 696 709. Springer, 2008.

[167] M.A. Van Wyk and J. Clark. An algorithm for approximate least-squares attributed graph matching. *Problems in Applied Mathematics and Computational Intelligence*, pages 67 72, 2000.

[168] M.A. Van Wyk, T.S. Durrani, and B.J. Van Wyk. A rkhs interpolator-based graph matching algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):988 995, 2002.

[169] J.P. Vert and M. Kanehisa. Graph-driven features extraction from microarray data using diffusion kernels and kernel cca. *Advances in Neural Information Processing Systems*, 15:1425 1432, 2002.

[170] A. Vinciarelli. Noisy text categorization. In *Proceedings of the 17th International Conference on Pattern Recognition*, volume 2, pages 554 557, 2004.

[171] P. Viola and M.J. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137 154, 2004.

[172] S. Vishwanathan, N.N. Schraudolph, R. Kondor, and K. Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11:1201 1242, 2010.

[173] R.A. Wagner and M.J. Fischer. The string-to-string correction problem. *Journal of the* ACM, 21(1):168 173, 1974.

[174] W.D. Wallis, P. Shoubridge, M. Kraetz, and D. Ray. Graph distances using graph union. *Pattern Recognition Letters*, 22(6):701 704, 2001.

[175] Y.K. Wang, K.C. Fan, and J.T. Horng. Genetic-based search for error-correcting graph isomorphism. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 27(4):588 597, 1997.

[176] C. Watkins. Kernels from matching operations. Technical report, Royal Holloway College, 1999.

[177] C. Watkins. Dynamic alignment kernels. In A. Smola, P. Bartlett, B. Scholkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 39 50. MIT Press, 2000.

[178] C. Watson and C. Wilson. Nist special database 4, fingerprint database. *National Institute of Standards and Technology*, 1992.

[179] R. Wilson, E. Hancock, and B. Luo. Pattern vectors from algebraic graph theory. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7):1112 1124, 2005.

[180] R.C. Wilson and E.R. Hancock. Structural matching by discrete relaxation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(6):634 648, 1997.

[181] B. Xiao, A. Torsello, and E.R. Hancock. Isotree: Tree clustering via metric embedding. *Neurocomputing*, 71(10):2029 2036, 2008.

[182] L. Xu, A. Krzyzak, and C.Y. Suen. Methods of combining multiple classifiers and their applications to handwriting recognition. *IEEE Transactions on Systems, Man and Cybernetics*, 22(3):418 425, 1992.

[183] N. Yager and A. Amin. Fingerprint classification: A review. *Pattern Analysis and Applications*, 7(1):77 93, 2004.

[184] J. Zhang, M. Marszalek, S. Lazebnik, and C. Schmid. Local features and kernels for classification of texture and object categories: A comprehensive study. *International Journal of Computer Vision*, 77(2):213 238, 2007.

[185] M. Zimmermann, J.-C. Chappelier, and H. Bunke. Offline grammar-based recognition of handwritten sentences. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(5):818 821, 2006.