



**Universitat
Autònoma
de Barcelona**

**Theory and Algorithms on the Median
Graph. Application to Graph-based
Classification and Clustering**

A dissertation submitted by **Miquel Ferrer
Sumsi** at Universitat Autònoma de Barcelona to
fulfil the degree of **Doctor en Informàtica**.

Bellaterra, June 2008

Director: **Dr. Ernest Valveny Llobet**
Universitat Autònoma de Barcelona
Dep. Ciències de la Computació & Centre de Visió per Computador
Co-director: **Dr. Francesc Serratosa Casanelles**
Universitat Rovira i Virgili
Dep. Informàtica i Matemàtiques



This document was typeset by the author using L^AT_EX 2_ε.

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

Copyright © 2008 by Miquel Ferrer Sumsi. All rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopy, recording, or any information storage and retrieval system, without permission in writing from the author.

ISBN: 978-84-935251-7-0

Printed by Ediciones Gráficas Rey, S.L.

This thesis is dedicated to my wife and my future child

Agraïments

Considero plausible el fet que l'elaboració d'una tesi doctoral és un camí que cada persona el viu a la seva manera, des de les seves pròpies circumstàncies. Així doncs, jo també l'he viscuda des d'una òptica molt personal. Darrere d'aquest treball hi ha no només moltes hores d'esforç davant de llibres, articles, l'ordinador, etc. Hi ha també una petita història personal que fa més de 15 anys que va començar, fins i tot, molt temps abans que ni jo mateix sabés que algun dia trepitjaria la universitat. En els fulls que segueixen no la hi podreu trobar aquesta història, almenys d'una forma explícita (això ho haurem de deixar per un altre moment), però hi és. Probablement tot el que segueix ara no s'hagués escrit si la meva història personal hagués estat una altra.

Tanmateix, tot aquest treball no hagués estat possible sense l'ajut de molta gent que durant aquests darrers anys han estat, d'una manera o d'una altra, sempre al meu costat. A totes elles el meu més sincer agraïment.

En primer lloc vull donar les gràcies als meus directors de tesi, l'Ernest Valveny i el Francesc Serratosa. He de dir sobre ells que la seva incondicional confiança en la meva persona i la seva gran paciència i professionalitat, han estat del tot claus per tirar endavant aquesta empresa. Sense la seva inestimable ajuda, de ben segur que tot el que segueix no s'hagués escrit mai. Moltes gràcies a tots dos.

Vull agrair també al CVC i a tota la seva gent els bons moments que m'heu donat durant aquests darrers tres anys. Sense vosaltres, hagués estat tot molt més complicat. Especialment vull agrair a la gent del DAG: el Josep Lladós, la Gemma, l'Alícia, el Marçal, l'Agnès, el Joan, el Josep R., en Partha i l'Oriol, per la confiança i col·laboració rebudes durant aquest treball. A tots vosaltres, gràcies. Vull agrair també els bons moments que m'han donat la gent del despatx: la Debora, l'Aura, el Jaume, el David i el Fernando. Gràcies pels vostres consells, han estat de gran ajuda per caminar per aquest sender que és la tesi. Voldria agrair també al Departament de Ciències de la Computació i a tot el seu personal, amb el Xavi Roca al capdavant, l'inestimable suport que m'han donat sempre. Finalment, ha estat per a mi una extraordinària experiència poder treballar amb la gent de Suïssa, sota la direcció del Prof. Dr. Horst Bunke, i comptar amb la gran ajuda d'en Kaspar Riesen i compartir bons moments amb el Roman Bertolami, l'Andreas Schlapbach, el Vivian Kilchherr, el Marcus Liwicki i la Susanne Thüler.

Vull donar també les gràcies a la meva família. Als meus pares (Pere i Montse) i també als meus tres germans (Toni, Pol i Albert). Sempre us he tingut al meu costat, en els bons moments i també en els ... no tan bons. La constància, les ganes de

treballar i l'afany de superació us les dec a vosaltres. Gràcies de tot cor.

Finalment, m'agradaria dedicar aquest treball a dues persones molt especials per a mi. La primera és la Mireia. La Mireia, que amb la força i la tenacitat que caracteritzen a la gent de muntanya i l'alegria i bon fer de la gent del vi, m'ha donat les suficients forces durant tots aquests anys que fa que estem junts per arribar on sóc ara. No canviïs mai. La segona el fill o filla que estem esperant. Encara no et conec, però ja em tens el cor robat. Per tu el camí començarà en breu, per nosaltres dos continuarà, ara al teu costat. Segur que serà una experiència extraordinària. I qui sap fill meu, potser algun dia la teva història personal et portarà a escriure línies com aquestes.

Terrassa, 7 d'abril de 2008.

Resum

Donat un conjunt d'objectes, el concepte genèric de *mediana* està definit com l'objecte amb la suma de distàncies a tot el conjunt, més petita. Sovint, aquest concepte és usat per a obtenir el representant del conjunt.

En el reconeixement estructural de patrons, els grafs han estat usats normalment per a representar objectes complexos. En el domini dels grafs, el concepte de *mediana* és conegut com *median graph*. Potencialment, té les mateixes aplicacions que el concepte de *mediana* per poder ser usat com a representant d'un conjunt de grafs.

Tot i la seva simple definició i les potencials aplicacions, s'ha demostrat que el seu càlcul és una tasca extremadament complexa. Tots els algorismes existents només han estat capaços de treballar amb conjunts petits de grafs, i per tant, la seva aplicació ha estat limitada en molts casos a usar dades sintètiques sense significat real. Així, tot i el seu potencial, ha restat com un concepte eminentment teòric.

L'objectiu principal d'aquesta tesi doctoral és el d'investigar a fons la teoria i l'algorísmica relacionada amb el concepte de *median graph*, amb l'objectiu final d'extendre la seva aplicabilitat i lliurar tot el seu potencial al món de les aplicacions reals. Per això, presentem nous resultats teòrics i també nous algorismes per al seu càlcul. Des d'un punt de vista teòric aquesta tesi fa dues aportacions fonamentals. Per una banda, s'introdueix el nou concepte d'*spectral median graph*. Per altra banda es mostra que certes de les propietats teòriques del *median graph* poden ser millorades sota determinades condicions. Més enllà de les aportacions teòriques, proposem cinc noves alternatives per al seu càlcul. La primera d'elles és una conseqüència directa del concepte d'*spectral median graph*. Després, basats en les millores de les propietats teòriques, presentem dues alternatives més per a la seva obtenció. Finalment, s'introdueix una nova tècnica per al càlcul del median basat en el mapeig de grafs en espais de vectors, i es proposen dos nous algorismes més.

L'avaluació experimental dels mètodes proposats utilitzant una base de dades semi-artificial (símbols gràfics) i dues amb dades reals (mol·lècules i pàgines web), mostra que aquests mètodes són molt més eficients que els existents. A més, per primera vegada, hem demostrat que el *median graph* pot ser un bon representant d'un conjunt d'objectes utilitzant grans quantitats de dades. Hem dut a terme experiments de classificació i *clustering* que validen aquesta hipòtesi i permeten preveure una pròspera aplicació del *median graph* a un bon nombre d'algorismes d'aprenentatge.

Abstract

Given a set of objects, the generic concept of *median* is defined as the object with the smallest sum of distances to all the objects in the set. It has been often used as a good alternative to obtain a representative of the set.

In structural pattern recognition, graphs are normally used to represent structured objects. In the graph domain, the concept analogous to the median is known as the median graph. By extension, it has the same potential applications as the generic median in order to be used as the representative of a set of graphs.

Despite its simple definition and potential applications, its computation has been shown as an extremely complex task. All the existing algorithms can only deal with small sets of graphs, and its application has been constrained in most cases to the use of synthetic data with no real meaning. Thus, it has mainly remained in the box of the theoretical concepts.

The main objective of this work is to further investigate both the theory and the algorithmic underlying the concept of the median graph with the final objective to extend its applicability and bring all its potential to the world of real applications. To this end, new theory and new algorithms for its computation are reported. From a theoretical point of view, this thesis makes two main contributions. On one hand, the new concept of spectral median graph. On the other hand, we show that some of the existing theoretical properties of the median graph can be improved under some specific conditions. In addition to these theoretical contributions, we propose five new ways to compute the median graph. One of them is a direct consequence of the spectral median graph concept. In addition, we provide two new algorithms based on the new theoretical properties. Finally, we present a novel technique for the median graph computation based on graph embedding into vector spaces. With this technique two more new algorithms are presented.

The experimental evaluation of the proposed methods on one semi-artificial and two real-world datasets, representing graphical symbols, molecules and webpages, shows that these methods are much more efficient than the existing ones. In addition, we have been able to prove for the first time that the median graph can be a good representative of a class in large datasets. We have performed some classification and clustering experiments that validate this hypothesis and permit to foresee a successful application of the median graph to a variety of machine learning algorithms.

Contents

Agraiments	i
Resum	iii
Abstract	v
1 Introduction	1
1.1 Motivation	1
1.2 Median Graph	3
1.3 Aims and Objectives of this Thesis	4
1.4 Organization	6
2 Graph Matching	9
2.1 Graph and Subgraph	9
2.2 Exact Graph Matching	11
2.2.1 Graph Similarity Measures Based on the <i>mcs</i> and the <i>MCS</i>	14
2.3 Error-Tolerant Graph Matching	16
2.3.1 Graph Edit Distance	18
2.4 Discussion	19
3 Median Graph	21
3.1 Median Graph	21
3.2 Theoretical Properties of the Median Graph	22
3.2.1 Bounds on the Size of the Median Graph	23
3.2.2 Bounds on the SOD of the Median Graph	23
3.3 Median Graph Computation	24
3.3.1 Computational Complexity	24
3.3.2 Algorithms for the Set Median Graph Computation	24
3.3.3 Algorithms for the Generalized Median Graph Computation	25
3.4 Discussion	29
4 Spectral Median Graph	31
4.1 Graph Matching using Spectral Techniques	32
4.1.1 Matrices of Graphs	32
4.1.2 Spectral Decomposition of Matrices	33

4.1.3	Graph Matching using the Spectral Graph Theory	35
4.1.4	Weighted Graph Isomorphism by means of Spectral Techniques	36
4.2	Spectral Median Graph	40
4.2.1	Synthesis of the Generalized Median Eigenmode	41
4.2.2	Experimental Results	44
4.3	Discussion	45
5	Median Graph under a Particular Cost Function	49
5.1	A Particular Cost Function	50
5.2	New Theoretical Properties on the Median Graph	51
5.2.1	Maximum Common Subgraph and Minimum Common Supergraph of a Set of Graphs	51
5.2.2	New Bounds on the Size of the Median Graph	52
5.2.3	Reducing the Upper Bound of the SOD of the Median Graph	53
5.3	New Search Space	54
5.3.1	Influence of the Cost Function on the Search Space	54
5.3.2	The New Search Space	55
5.4	New Exact Algorithm	59
5.4.1	Prediction of the SOD	59
5.4.2	The New Exact Algorithm	61
5.4.3	Experimental Setup	62
5.5	New Genetic Algorithm	69
5.5.1	Basics on Genetic Search	69
5.5.2	Our Approach	69
5.5.3	Experimental Setup	72
5.6	Discussion	74
6	Median Graph by Means of Graph Embedding in Vector Spaces	77
6.1	Graph Embedding	78
6.2	Median Graph via Graph Embedding in Vector Spaces: An Overview	78
6.2.1	Graph Embedding in a Vector Space	80
6.3	Exact Median Graph via Embedding	81
6.3.1	Graph Embedding in Vector Spaces	81
6.3.2	Median Vector Computation	83
6.3.3	Back to the Graph Domain	83
6.3.4	Experimental Setup	86
6.4	Approximate Median Graph via Embedding	86
6.4.1	Graph Embedding in a Vector Space	88
6.4.2	Median Vector Computation	88
6.4.3	Back to the Graph Domain	89
6.4.4	Discussion on the Approximations	90
6.4.5	Assessment of the Median Quality	93
6.5	Discussion	96
7	Application of the Median Graph	99
7.1	Classification Experiments	99

7.1.1	Graph-based classification	99
7.1.2	Experimental Setup	100
7.1.3	Results	101
7.1.4	Discussion on Graph-based Classification	114
7.2	Clustering	115
7.2.1	Graph-Based Clustering	115
7.2.2	Clustering Performance Measures	116
7.2.3	Experimental Setup	117
7.2.4	Results	118
7.3	Discussion	119
8	Conclusions	123
8.1	Conclusions	123
8.1.1	Theory	123
8.1.2	Algorithms	124
8.1.3	Applications	125
8.2	Discussion	126
8.3	Future Work	127
A	Databases	133
A.1	Letter Database	133
A.1.1	Variation Letter-1	133
A.2	GREC 2005 Database	134
A.2.1	Variation GREC-1	136
A.2.2	Variation GREC-2	140
A.3	Molecule Database	140
A.4	Webpage Database	142
	Bibliography	145

List of Tables

5.1	Detail of the cost function	50
5.2	Sum of nodes in the set S function of the class and the number of nodes in S	63
5.3	Percentage of time and SOD computations required for the new exact algorithms with respect to the Multimatch algorithm.	65
5.4	Configuration parameters for the genetic algorithm.	71
5.5	Statistics for median graph grouped by class.	73
5.6	Statistics for median graph grouped by the number of graphs used to compute the median.	73
6.1	Number of Graphs in S for each database.	94
7.1	Summary of the tested methods and their characteristics.	101
7.2	Configuration parameters for the experiment 1.	102
7.3	Configuration parameters for the experiment 2.	103
7.4	Configuration parameters for the experiment 3.	105
7.5	Configuration parameters for the experiment 4 using the Molecule dataset.	106
7.6	Configuration parameters for the experiment 4 using the Webpage dataset.	108
7.7	Configuration parameters for the experiment 4 using the GREC-2 dataset.	111
7.8	Number of classes and number of elements per class for each database.	117
7.9	Minimum, average and maximum values of the Rand index for different datasets.	118
7.10	Minimum, average and maximum values of the Dunn index for different datasets.	120
7.11	Dunn Index for the groundtruth for each dataset.	120
8.1	Number of graphs in each class.	127
A.1	Entire Letter-1 database.	135
A.2	Some characteristics of Letter-1 dataset.	135
A.3	GREC 2005 Database (1)	137
A.4	GREC 2005 Database (2)	138
A.5	GREC 2005 Database (3)	139

A.6	Some characteristics of GREC-1 dataset.	140
A.7	Some characteristics of GREC-2 dataset.	141
A.8	Some characteristics of Molecule dataset.	141
A.9	Number of graphs in each class.	142
A.10	Some characteristics of Webpage dataset.	143

List of Figures

2.1	Original model graph g (a), an induced subgraph of g (b) and a non-induced subgraph of g (c).	11
2.2	Two graphs g_1 and g_2 (a), a possible $mcs(g_1, g_2)$ (b) and a possible $MCS(g_1, g_2)$ (c).	14
2.3	A possible edit path between two graphs g_1 and g_2	19
3.1	Two graphs, g_1 and g_2	26
3.2	Graphic example of the Multimatch structure.	26
3.3	Chromosome representation of a possible candidate median g (a), a graphical representation of the mapping the chromosome codifies (b).	28
4.1	Example of a weighted graph and its corresponding adjacency and Laplacian matrices.	33
4.2	Example of an unweighted graph and its corresponding adjacency and Laplacian matrices.	34
4.3	Example of two weighted undirected graphs.	37
4.4	Node correspondence given by Umeyama's method.	39
4.5	Example of the synthesis of the generalized spectral median graph.	43
4.6	SOD comparison using the genetic approach of [53].	45
4.7	SOD comparison using the generalized median graph approach.	46
5.1	Two graphs g_1 (a) and g_2 (b), a possible $mcs(g_1, g_2)$ (c) and a possible edit path between g_1 and g_2 (d).	55
5.2	Detail of the rhombus search space.	57
5.3	Reduction in the search space due to the new bounds.	58
5.4	Computation time (a) and number of SOD computations (b) as a function of the total number of nodes of the graphs in S	64
5.5	Computation time for Active (a) and Inactive (b) compounds as a function of the size of $MCS(S)$	67
5.6	SOD of computed median for Active (a) and Inactive (b) compounds as a function of the size of $MCS(S)$	68
5.7	A supposed $g_M(S)$ (a), an induced subgraph g of $g_M(S)$ (b) and the chromosome representing g (c).	70

5.8	SOD comparison for the Set Median (SM) and the Approximate Genetic (AG) algorithm, function for the different classes.	74
5.9	SOD comparison function of the number of graphs in S	75
6.1	Overview of the general embedding procedure for median graph computation.	79
6.2	Detail of the first step (Graph embedding).	81
6.3	Review of rhombus search space.	82
6.4	Example of a median vector and its interpretation.	83
6.5	Interpretation of the median vector.	84
6.6	Explored part of the search space (grey part).	85
6.7	Computation time (a) and number of SOD computations (b) as a function of the total number of nodes of the graphs in S	87
6.8	Example of the weighted mean of a pair of graphs	89
6.9	Illustration of the triangulation procedure.	91
6.10	Illustration of the linear interpolation procedure.	92
6.11	SOD evolution on the Molecule dataset	94
6.12	SOD evolution on the Webpage dataset	95
6.13	SOD evolution on the GREC dataset	95
7.1	Classification accuracy for 1NN, EE, AELI and AET methods using the Molecule dataset.	103
7.2	Classification accuracy for 1NN, AS, AELI and AET methods using the GREC-1 dataset.	104
7.3	Classification accuracy for 1NN, AG, AELI and AET methods using the Webpage dataset.	105
7.4	Classification accuracy for 1NN, SM, AELI and AET methods using the Molecule dataset and for different number of graphs to compute the median graph.	107
7.5	Classification accuracy for 1NN, SM, AELI and AET methods using the Webpage dataset and for different number of graphs to compute the median graph.	108
7.6	Appearance frequency of the correct class using 3 classes.	109
7.7	Appearance frequency of the correct class using 5 classes.	110
7.8	Multiclass classification accuracy for the Webpage database with an increasing number of graphs classes taken into account (number of graphs to compute the median = 10).	111
7.9	Classification accuracy for 1NN, SM, AELI and AET methods using the GREC-2 dataset and for different number of graphs to compute the median graph.	112
7.10	Appearance frequency of the correct class using 5 elements.	113
7.11	Appearance frequency of the correct class using 10 elements.	113
7.12	Multiclass classification accuracy for the GREC database with an increasing number of classes taken into account (number of graphs to compute the median = 15).	114

A.1	Original prototypes for the Letter dataset.	134
A.2	Graph-based representations of the original prototypes.	134
A.3	Graph-based representation of a GREC symbol.	136
A.4	Example of distorted symbols.	140
A.5	Examples of GREC symbols with some distortions. (a) Architectural symbol (b) Electrical symbol.	141
A.6	Example of some active compounds (a) and some inactive compounds (b)	142
A.7	Webgraph example.	143

Chapter 1

Introduction

1.1 Motivation

One of the basic objectives in pattern recognition is to develop systems for the analysis or classification of objects [35, 40]. In principle, these objects or patterns can be of any kind. For instance they can include images taken from a digital camera, electronic signals captured with a transducer or words provided to an OCR to name a few. A first issue to be addressed in any pattern recognition system is how to represent these objects. Feature vectors are one of the most common and widely used data representations. That is, for each object, a set of relevant properties, or features, are computed and arranged in a vector form. Then, a classifier can be trained to recognize the unknown objects. The main advantage of this representation is that a large number of algorithms for pattern analysis and classification become immediately available [35]. This is mainly due to the fact that vectors are simple structures with good mathematical properties that can be readily manipulated algebraically.

However, some disadvantages arise from the simple structure of feature vectors. Regardless of the complexity of the object, feature vectors have always the same length and structure (a simple list of pre-determined components). Then, for the representation of complex objects where the relations between their parts become important for their analysis or classification, graphs appear as an appealing alternative. One of the main advantages of graphs over feature vectors is that graphs can explicitly model the relations between the different parts of the object, whereas feature vectors are only able to describe the object as an aggregation of numerical properties. In addition, graphs permit to associate any kind of label (not only numbers) to both edges and nodes, improving in this way the spectrum of properties of the object that can be represented. Furthermore, the dimensionality of graphs, that is, the number of nodes and edges, can be different for every object even for objects of the same class. Thus, the more complex an object is, the larger the number of nodes and edges can be. Recently, an extensive work comparing the representational power of the feature vectors and graphs under the context of web content mining has been presented in [89]. Experimental results consistently show an improvement in the accuracy of the graph-based approaches over the comparable vector-based methods. In addition, in

some cases the experiments also showed an improvement in execution time over the vector model.

Actually, graphs have been used to solve computer vision problems for decades in many applications. Some examples include recognition of graphical symbols [62, 63], character recognition [65, 86], shape analysis [22, 80], 3D-object recognition [96, 117] and video and image database indexing [97].

However, in spite of the strong mathematical foundation underlying graphs and their high power of representation, working with graphs is harder and more challenging than working with feature vectors.

On the one hand, the computational complexity of the algorithms related to graphs is usually high. For instance, the simple task of comparing two graphs, which is commonly referred as graph matching, becomes exponential in the size of graphs, specially in the case of optimal algorithms. For some applications, this may be unacceptable or even intractable when the size of the graphs becomes large. Conversely, approximate algorithms have only polynomial time complexity, but do not guarantee to find the optimal solution. A number of similarity measures on graphs and related computational procedures have been proposed in this context [11, 13, 72, 87, 109]. An extensive survey about graph matching in pattern recognition can be found in [26].

On the other hand, some basic operations that are used in many pattern recognition methods and that might appear quite simple in the vector domain, such as the sum or the mean, turn very difficult or even impossible in the graph domain. Therefore, graphs are typically used in the context of nearest-neighbor classification where we only need the definition of a similarity measure between two graphs. That is, an unknown input pattern is compared (matched) with a number of prototypes stored in the database. The unknown input is then assigned to the same class as the most similar prototype.

To overcome this limitation, some attempts to fusion both domains, i.e. vector and graph domains, have been presented in the literature [79, 83]. The basic idea underlying these methods is to try to take advantage of the best of each domain. That is, the high representational power of graphs and the simplicity of the vectors. The aim is to extend the existing machine learning algorithms, typically applied to feature vectors, to the graph domain.

In some of these machine learning algorithms a representative of a set is often needed. For instance, in the classical k -means clustering algorithm, a representative of each cluster is computed and used at the next iteration to reorganize the clusters. In classification tasks, a representative of a class could be useful to reduce the number of comparisons needed to assign the unknown input pattern to its closest class. While the computation of a representative of a set is a relatively simple task in the vector domain (it can be computed by means of the mean or the median vector), it is not clear how to obtain a representative of a set of graphs.

In the literature we can distinguish two different methodologies for that, depending on whether they keep probabilistic information in the structure that represents the cluster of graphs or not.

In the first type of methods, the models belonging to the class, which are usually called Random Graphs (RG), are described, in the most general case, through a joint probability space of random variables ranging over graph vertices and arcs. They

are the union of the graphs in the cluster, according to a synthesis process, together with its associated probability distribution. In this manner, a structural pattern can be explicitly represented in the form of a graph and an ensemble of instances of the pattern can be considered as a set of outcomes of the RG. Two important probabilistic methods are First-Order Random Graphs (FORGs) [116] and Function-Described Graphs (FDGs) [95, 96]. Another similar approach has been presented by Sengupta et al. in [91] and can be regarded as similar to the FORG approach. Finally, Second-Order Random Graphs (SORGs) which can be seen as a generalization of both of them have been introduced in [94].

Four different non-probabilistic methods exist. The self-organizing map (SOM) is a useful method to cluster sets of objects. It consists of a layer of units (neurons), that adapt themselves to a population of input patterns. SOM was first presented by Kohonen with the limitation that patterns had to be represented in terms of feature vectors only. Afterwards, the same authors presented an extension of this method to strings [48] and then Günter and Bunke proposed in [43] a generalization of the clustering method applied to graphs. Then, Seong et al. [92], developed a hierarchical model that summarizes and organizes the input instances incrementally and is built up with a succession of graphs. In the approach of Cordella et al. [29], the set of graphs is represented by the maximally general prototype that can be seen as the union of the graphs. Finally, Jiang et al. [53], introduced the concepts of set and generalized median graph to define the representative of a set of graphs.

Unlike the above outlined methods, most of them defined using heuristic procedures, the concept of median graph is sustained by a strong theoretical basis. This is the main reason of selecting the median graph as the topic of study in this work. In the next section we introduce the definition and the main properties of the median graph.

1.2 Median Graph

Given a set of graphs, the median graph has been defined as the graph which has the smallest sum of distances (SOD) to all the graphs in the set [53]. This simple definition hides a powerful concept to describe the representative of a set of graphs. The unique restriction imposed by such definition is that a similarity measure between graphs needs to be defined. In fact, the definition of median graph has an analogous definition in the concept of the median vector, or in the more general case in the concept of median of a set of objects, which is known as one of the most important ways to obtain a representative of a set. This simple but powerful definition makes the concept of median graph very attractive.

In addition, this strong theoretical definition implicitly carries the possibility of defining some interesting theoretical properties. As a matter of fact, two properties concerning the size and the limits of the SOD of the median graph were already introduced in the initial work on median graphs. Such theoretical properties are useful to better understand the underlying concept and can be crucial to make practical advances.

In addition median graph has strong potential applications. For instance, it could

be used in graph clustering to represent the center of each cluster. Furthermore, it is also of interest in the context of object prototype learning. In this case the objective is to infer a representative model out of a collection of noisy samples of the same object. Finally, the representatives obtained with the median graph could also be used in classification tasks. Even in a more general scheme, the median graph may be potentially used in any application where a representative of a set of graphs is needed.

However, the computation of the median graph is exponential both in the number of input graphs (that makes the search space for the median graph computation very large) and their size (due to the complexity of computing the distance between two graphs). A number of algorithms have been reported in the past to compute the median graph. The only exact algorithm proposed up to now is based on an A^* algorithm using a data structure called multimatch [76]. As the computational cost of this algorithm is very high, a set of approximate algorithms have also been presented in the past based on different approaches such as genetic search [53, 76] and a greedy-based algorithm [46].

Up to now the median graph has been successfully applied to obtain prototypes of graphical symbols [51], to obtain median words for OCR tasks [53] and to perform content-based image clustering [46]. Nevertheless, all these applications have been performed either with synthetic data, or with very small graphs, sometimes represented only with nodes. Thus, despite its potential application the use of the median graph has been constrained to very limited scenarios and few real applicability.

1.3 Aims and Objectives of this Thesis

Main Objective of this Work

The main objective of this work is to further investigate both the theory and the algorithmic underlying the concept of the median graph in order to extend its applicability to real problems.

To this end, the problem will be tackled from different points of view and this main objective can be detailed into the following points:

1. New Theoretical Properties about the Median Graph

The first issue to be addressed is the study of the underlying theory related to the median graph. With this study we aim to derive new theoretical properties about the median graph. The main objectives are basically twofold and they can be summarized as follows:

- **Median Graph Understanding:** An exhaustive knowledge about the median graph in terms of its underlying theory is crucial to exploit all the possibilities that this concept offers. With these new theoretical properties we will be able to better understand the behavior of the median graph and consequently we will be able to use all of its potential in posterior stages.

- **Algorithm Enhancement:** The second objective is to take advantage of these new properties to enhance the algorithmic part. That is, either the new properties will be directly applied to the existing algorithms or they will serve to investigate new and unknown techniques for the median graph computation.

2. New Algorithms for the Median Graph Computation

The second main objective of this work is to present new and more efficient algorithms for the median graph computation. To this end the first task is to implement all the previous existing algorithms in order to better understand from a practical point of view how the median graph can be obtained. This bird view of the theory and the algorithmic related to the median graph will serve to present new exact and approximate algorithms:

- **Exact Algorithms:** Despite their inherent complexity, we think that the development of exact algorithms is still an important task for two basic reasons. The former is that exact strategies for the median graph computation may help to better understand the way in which the median can be obtained. The latter is that although exact algorithms have limited applicability they can still be used as a baseline for the results provided by the approximate algorithms. They can also be helpful to derive good approximate algorithms in posterior stages.
- **Approximate Algorithms:** The second objective is to propose new approximate algorithms for the median graph computation, faster and more accurate than the existing ones and applicable to a wide spectrum of problems.

With these new algorithms the aim is to give the median graph the opportunity to leave the box of the theoretical concepts and to become a practical concept useful in machine learning and pattern recognition applications.

3. Extend the Applicability of the Median Graph to Real Data

As it has been noted in Section 1.2, the median graph has limited applicability. The last objective of this thesis is to be able to apply the concept of median graph to real problems. For real problems we understand widely used algorithms in machine learning (such as k -means clustering or kNN classification algorithms) using real data, that is, unconstrained graphs in terms of size and attribute nature.

- **Graph-based Classification:** In many classification algorithms it is often desirable to obtain a representative of a set in order to represent a class of objects using a collection of noisy samples of the same class. Therefore, we aim to obtain either exact or good approximate median graphs as the representative of a given class and then use these representatives in classification tasks. The objective is to use real data in the experiments in order to apply the median graph to real-world applications.
- **Graph-based Clustering:** The second objective is to be able to use the median graph as a cluster representative in a graph-based k -means clustering algorithm using real data. This point is a bit more challenging than the previous

one in the sense that the cluster center (the median) is computed using elements of different classes (as opposed to classification where the median is obtained using elements of the same class). This fact means that the algorithms must be able to obtain the median not only of a set of similar graphs but also of a set with some outliers. It must therefore be robust to this kind of variability.

1.4 Organization

The rest of this thesis is organized in seven chapters and one appendix.

- In Chapter 2 we will introduce the basic concepts and the notation that we will use in the rest of this work. First, we start by defining the basic concepts of graph and subgraph, which are the atomic entities that we will need to develop the rest of the work. Then an introduction to graph matching is presented. We divide this part into two subparts: the exact graph matching and the error-tolerant or inexact graph matching. In the exact graph matching we emphasize the concepts of the maximum common subgraph and the minimum common supergraph. These two concepts will be specially important in Chapter 5 and in some part of Chapter 6. We also give some similarity measures based on these concepts that will be crucial in those chapters. Finally we provide a brief introduction on error-tolerant graph matching. Into this introduction we give a special relevance to the weighted graph isomorphism problem, that will be the basis for Chapter 4. The chapter finishes describing one of the most used techniques in error-tolerant graph matching: the graph edit distance. This concept will be very important in Chapter 6.
- Chapter 3 is devoted to explain in detail the concept of median graph. Firstly we give a formal introduction of this concept, and we present the theoretical properties developed up to now. After that, we provide an extensive overview about its computation. We first describe the intrinsic computational complexity underlying the median graph. After that, we provide an explanation of all the existing exact and approximate methods to compute the median graph, emphasizing their computational complexity and their applications. The chapter concludes with a discussion about the potential applications of the median graph, and also pointing out the limitations of the existing methods.
- In Chapter 4 we provide the first new algorithm for the median graph computation. This algorithm is based on the spectral graph theory and is only able to work with weighted graphs. The chapter starts with a brief overview of what the spectral graph theory is and how it can be used to perform graph matching tasks. After that, we propose the new concepts of set and generalized spectral median graph and we provide an incremental algorithm for its computation. The chapter ends with a comparison of the algorithm with one of the previous existing methods for the median graph computation.
- Chapter 5 is devoted to the study of the median graph under a particular cost function. Firstly, we provide two enhancements on the existing theoretical prop-

erties that will help to reduce the search space for the median graph computation. After that, we will show that under this particular cost function and using the new theoretical properties, the search space can be drastically reduced. In a practical way, we will propose two different strategies to explore the reduced search space. First, we will provide a new and more efficient exact algorithm for the median graph computation. Some preliminary experiments will show that this new method clearly outperforms the traditional exact algorithm and, in some cases, even the previous approximate algorithm based on genetic search, in terms of computation time. After that, and taking again advantage of the new theoretical results, we will present a new approximate algorithm based on genetic search. The experiments show that this method is able to compute the median graph with small sets of graphs but with large sizes. In both algorithms we have performed experiments on real data.

- In Chapter 6 a novel technique for the median graph computation based on graph embedding in vector spaces will be presented. The main idea will be to translate each graph into a vector in a n -dimensional space and then, to compute the median of the set in this vector space instead of doing that directly in the graph domain. The median graph is then recovered from this median vector. Using this technique we will present one exact and one approximate algorithm. We will see that the exact computation of the median graph will be extended to limited real data experiments, where never before an exact algorithm could be applied. With the approximate algorithm we will be able to compute the median graph using real data and with a large number of graphs and without any restriction on the type of graphs or the cost function.
- Chapter 7 is devoted to give an extensive experimental framework to evaluate the proposed methods. This chapter contains the major practical contribution of this work: the extension of the median graph to two important machine learning algorithms such as the k -nearest-neighbor classification and the k -means clustering algorithm using both synthetic and real data. The chapter is divided into two different parts. In the first part, the classification problem is addressed. The experiments are performed using all the proposed methods and they are compared with the classic graph-based kNN classifier. Depending on the method, the experiments are performed either on synthetic or real data. In the second part we propose for the first time the application of the median graph to the well-known k -means clustering algorithm using real data, where traditionally the set median has been used as a representative of each cluster. We perform several experiments on both synthetic and real massive data using the approximate method presented in Chapter 6. The results show that the obtained medians are able, in general, to provide better results than the set median.
- Finally in Chapter 8 we give some concluding remarks about this work, and we provide some possible future lines and strategies to continue investigating on median graphs. From a theoretical point of view, we point out possible new theoretical properties of the median graph more powerful than the existing ones.

We also propose possible options to improve some of the proposed algorithms. Finally, the option of using the median graph in more complex graph-based algorithms is introduced.

- Along this work different databases have been used to perform the experiments. All these datasets are explained in Appendix A. For every database we explain the kind of data it contains, the graph-based representation of this data and some characteristics about the database such as the number of elements, their size, etc. In addition, if any distortion has been introduced in the original elements, it is also explained.

Chapter 2

Graph Matching

Graphs are a powerful tool for object representation. In the introductory chapter, we have pointed out their advantages over feature vectors when objects are structured and their parts and the relations between them become important. This chapter is devoted to the formal definition of the concept of graph and some related concepts, with special emphasis to graph matching. Graph matching is the specific process of evaluating the structural similarity of two graphs. It can be split into two categories, namely exact and error-tolerant graph matching. In exact graph matching, the basic objective is to decide whether two graphs or parts of them are identical in terms of their structure and labels. Conversely, error-tolerant graph matching aims to give a measure of the similarity between two graphs.

The outline of this chapter is as follows. Firstly, in Section 2.1, the basic definitions of *graph* and *subgraph* are formally introduced. After that, in Section 2.2, the concept of *exact graph matching* and their classical paradigms, such as *graph isomorphism* and *subgraph isomorphism* are presented. We have also included in this section the definitions of *maximum common subgraph* and *minimum common supergraph* of two graphs. These two concepts can be seen as the bridge between the exact and the error-tolerant graph matching, and they will become an important part of this work in Chapter 5. Finally, in Section 2.3, a brief introduction to *error-tolerant graph matching* is given, and the different algorithms and techniques are explained, but we make special emphasis, in Section 2.3.1, on the *graph edit distance*. The graph edit distance has been shown as one of the most flexible and powerful techniques for the computation of graph similarity. This concept will become crucial in Chapter 6. For this reason we have given it a special treatment in this chapter.

2.1 Graph and Subgraph

There are different ways to define a graph depending mainly on how the labels of nodes and edges are defined. The definition of graph given below is sufficiently general to include the most important classes of graphs.

Definition 2.1 (*Graph*) Given L , a finite alphabet of labels for nodes and edges, a

graph g is defined by the four-tuple $g = (V, E, \mu, \nu)$ where,

- V is a finite set of nodes
- $E \subseteq V \times V$ is the set of edges
- μ is the node labeling function ($\mu : V \rightarrow L$)
- ν is the edge labeling function ($\nu : E \rightarrow L$)

The number of nodes of a graph g is denoted by $|g|$.

The set V of nodes can be seen as the set of node identifiers. Edges are defined as pairs of nodes (u, v) where $u, v \in V$. Usually, edges are defined in a directed way in the sense that $u \in V$ is the source node and $v \in V$ is the target node. An edge may connect a node $u \in V$ with itself. Nevertheless, in pattern recognition such kind of edges are often ignored. A graph is called *undirected* if for each edge $(u, v) \in E$ there is an edge $(v, u) \in E$ such that $\nu(u, v) = \nu(v, u)$. That is, if for each edge starting at u and ending at v there is always an edge starting at v and ending at u with the same label.

Without loss of generalization this definition can be seen as the definition of *labeled graph*. Notice that there is not any restriction about the nature of the labels of nodes and edges. That is, the label alphabet is not constrained at all. In most cases, L is defined as a vector space (i.e. $L = \mathbb{R}^n$) or simply as a set of discrete labels (i.e. $L = \{\alpha, \beta, \gamma, \dots\}$). The set of labels L can also include the *null* label (often represented as ε). If all the nodes and edges are labeled with the same *null* label, the graph is considered as *unlabeled*. A *weighted graph* is a special type of labeled graph in which each node is labeled with the null label and each edge (u, v) is labeled with a real number or *weight* w_{uv} , usually belonging, but not restricted, to the interval $[0, 1]$. An *unweighted graph*, can be seen as a particular instance of a *weighted graph* where $w_{uv} = 1 \forall (u, v) \in E$.

In the more general case, vertices and edges may contain more complex information. That is, they can contain information of different nature at the same time. For instance, a complex attribute for a node representing a region of an image could be composed of the histogram of the region, a description of the shape of the region and symbolic information relating this region with its adjacent regions. In this case graphs are called *attributed graphs* or simply AG. Notice that labeled, weighted and unweighted graphs are particular instances of attributed graphs, where the attributes are simple labels or numbers.

In this thesis, if the contrary is not specified, we will assume that graphs are fully connected, i.e., $E = V \times V$. Consequently, the set of *edges* is implicitly given. Such assumption is only for notational convenience, and it does not impose any restriction in the generality of the results. In the case where no real edge exists between two given nodes, we will label this edge with the special null label ε .

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 , denoted by $g_1 \subseteq g_2$ if,

- $V_1 \subseteq V_2$

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

From Definition 2.2 it follows that, given a graph $g = (V, E, \mu, \nu)$, a subset $V' \subseteq V$ of its vertices uniquely defines a subgraph, called the subgraph *induced* by V' . That is, an induced subgraph of g can be obtained by removing some of its nodes ($V - V'$) and all their adjacent edges. However, if the second condition of the Definition 2.2 is replaced by $E_1 \subseteq E_2$ then the resulting subgraph is called *non-induced*. In this case, a non-induced subgraph of g is obtained by removing some of its nodes ($V - V'$) and all their adjacent edges plus some additional edges. An example of a graph g , and an induced and a non-induced subgraph of g is given in the Figure 2.1. Of course, given a graph g , an induced subgraph of g is also a non-induced subgraph of g .

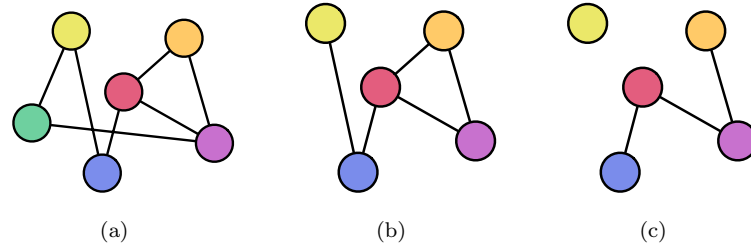


Figure 2.1: Original model graph g (a), an induced subgraph of g (b) and a non-induced subgraph of g (c).

2.2 Exact Graph Matching

The operation of comparing two graphs is commonly referred as *graph matching*. The aim of exact graph matching is to determine whether two graphs, or parts of two graphs, are identical in terms of their structure and labels. The equality of two graphs can be tested by means of a bijective function, called *graph isomorphism*, defined 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 bijective mapping $f : V_1 \rightarrow V_2$ such that,

- $\mu_1(u) = \mu_2(f(u))$ for all nodes $u \in V_1$
- for each edge $e_1 = (u, v) \in E_1$, there exists an edge $e_2 = (f(u), f(v)) \in E_2$ such that $\nu_1(e_1) = \nu_2(e_2)$
- for each edge $e_2 = (u, v) \in E_2$, there exists an edge $e_1 = (f^{-1}(u), f^{-1}(v)) \in E_1$ such that $\nu_1(e_1) = \nu_2(e_2)$

It is clear from this definition that isomorphic graphs are identical in terms of structure and labels. To check whether two graphs are isomorphic or not, we have to find a function mapping every node of the first graph to a node of the second graph in such a way that the edge structure of both nodes is preserved and the labels for the nodes and the edges are consistent. The graph isomorphism is an equivalence relation on graphs, since satisfies the conditions of reflexivity, symmetry and transitivity.

Related to graph isomorphism, the concept of subgraph isomorphism permits to check whether a part (subgraph) of one graph is identical to another graph. A subgraph isomorphism exists between two given graphs g_1 and g_2 if there is a graph isomorphism between the smaller graph and a subgraph of the larger graph. In other words, if the smaller graph is contained in the larger graph. Formally speaking, subgraph isomorphism is defined as follows:

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 $f : V_1 \rightarrow V_2$ is called a subgraph isomorphism from g_1 to g_2 if there exists a subgraph $g \subseteq g_2$, such that f is a graph isomorphism between g_1 and g .

Most of the algorithms for *graph isomorphism* and *subgraph isomorphism* are based on some form of tree search with backtracking. The main idea is to iteratively expand a partial match (initially empty) by adding new pairs of nodes satisfying the constraints imposed by the matching method with respect to the previously matched pairs of nodes. These methods usually apply some heuristic conditions to prune unfruitful search paths as early as possible. Eventually, either the algorithm finds a complete match or reaches a point where the partial match cannot be further expanded because of the matching constraints. In this last case, the algorithm backtracks until it finds a partial match for which another alternative expansion is possible. The algorithm halts when all the possible mappings that satisfy the constraints have already been visited.

One of the most important algorithms based on this approach is described in [107]. It addresses both the graph and subgraph isomorphism problems. To early prune unfruitful paths the author proposes a refinement procedure that drops pairs of nodes that are inconsistent with the partial match being explored. Then, the branches of this partial match leading to these incompatible matches are not expanded. A similar strategy is used in [90]. In addition, they include a preprocessing step that creates an initial partition of the nodes of the graph based on a distance matrix to reduce the search space. A more recent approach are the VF [28] and the VF2 [64] algorithms. In this work, the authors define a heuristic based on the analysis of the nodes adjacent to the nodes of the partial mapping. This procedure is fast to compute and lead in many cases to improve the approach of [107].

Probably, the most important approach that is not based on tree search is described in [71]. It addresses the problem of graph isomorphism and is based on the group theory. First of all, they construct the automorphism group of each input graph. After that, a canonical labelling is derived and the isomorphism is checked by simply verifying the equality of the canonical forms.

It is still an open question whether the graph isomorphism problem belongs to the NP class or not. Polynomial algorithms have been developed for special kind of graphs

such as *bounded valence graphs* [66] (i.e. graphs where the maximum number of edges adjacent to a node is bounded by a constant); *planar graphs* [47] (i.e. graphs that can be drawn in a plane without graph edges crossing) and *trees* [80] (i.e. graphs with no cycles). But no polynomial algorithms are known for the general case. Conversely, the subgraph isomorphism problem is proven to be NP-complete.

Matching graphs by means of graph and subgraph isomorphism is limited in the sense that an exact correspondence must exist between two graphs or between a graph and a part of another graph. But, let us consider the situation of the Figure 2.2(a). It is clear that the two graphs are similar, since most of their nodes and edges are identical. But it is also clear that none of them is related to the other one by (sub)graph isomorphism. Therefore, under the (sub)graph isomorphism paradigm, they will be considered as different graphs.

Then, in order to overcome the drawbacks of the (sub)graph isomorphism and to establish a measure of partial similarity between any two graphs, the concept of the largest common part of two graphs is introduced.

Definition 2.5 (*Maximum Common Subgraph (mcs)*) 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 g is called a *common subgraph (cs)* of g_1 and g_2 if there exists a subgraph isomorphism from g to g_1 and from g to g_2 . A common subgraph of g_1 and g_2 is called *maximum common subgraph (mcs)* if there exists no other common subgraph of g_1 and g_2 with more nodes than g .

The notion of the maximum common subgraph of two graphs can be seen as the intersection between them. Intuitively, it is the largest part of them that is identical in terms of structure and labels. It is clear that the more similar two graphs are the larger their maximum common subgraph is.

The *mcs* computation has been widely investigated. There are two major approaches in the literature. In [59, 70, 110] a backtracking search is used. A different strategy is proposed in [5, 37]. They cast the *mcs* problem by reducing it to the problem of finding the *maximum clique* (i.e. a completely connected subgraph) of a suitably constructed *association graph* [9]. A comparison of some of these methods on large databases of graphs is given in [14, 27].

It is well known that the *mcs* and the *maximum clique* problems are NP-complete. Therefore, some approximate algorithms have been developed. A survey of these approximate approaches and the study of their computational complexity is given in [49]. Nevertheless, in [57] it is shown that when graphs have unique node labels, the computation of the *mcs* can be done in polynomial time. This important result has been exploited in [89] to perform data mining on Web pages based on their content. Other applications of the *mcs* include comparison of molecular structures [81, 105] and matching of 3-D graph structures [104].

Dual to the definition of the maximum common subgraph (interpreted as the intersection) there is the concept of the minimum common supergraph.

Definition 2.6 (*Minimum Common Supergraph (MCS)*) 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 g is called a *common supergraph (CS)* of g_1 and g_2 if there exists a subgraph isomorphism from g_1 to g and from

g_2 to g . A common supergraph of g_1 and g_2 is called *minimum common supergraph (MCS)* if there exists no other common supergraph of g_1 and g_2 with less nodes than g .

The minimum common supergraph of two graphs can be seen as the graph with the minimum required structure so that both graphs are contained in it as subgraphs. In [17], it is demonstrated that the computation of the minimum common supergraph can be reduced to the computation of the maximum common subgraph. This result has not only relevant theoretical implications, but also practical implications. From this result it can be deduced that any algorithm for the *mcs* computation, such as all the algorithms described before, can also be used to compute the *MCS*.

An example of the maximum common subgraph and the minimum common supergraph of the two graphs of Figure 2.2(a) is given in the Figure 2.2(b) and 2.2(c) respectively.

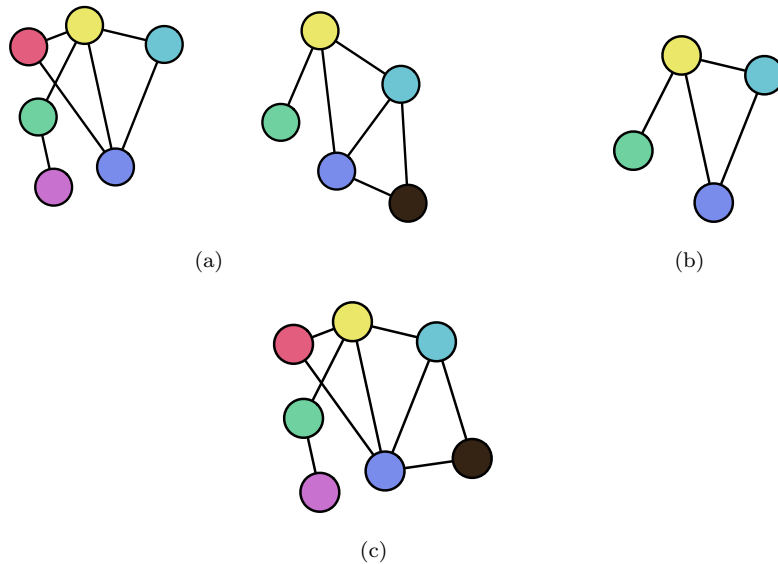


Figure 2.2: Two graphs g_1 and g_2 (a), a possible $mcs(g_1, g_2)$ (b) and a possible $MCS(g_1, g_2)$ (c).

Notice that, in general, neither $mcs(g_1, g_2)$ nor $MCS(g_1, g_2)$ are uniquely defined for two given graphs g_1 and g_2 .

2.2.1 Graph Similarity Measures Based on the *mcs* and the *MCS*

The concepts of maximum common subgraph and minimum common supergraph presented before, can be used to measure the similarity of two graphs. The basic idea is the intuitive thought that the larger the common part of two graphs, the higher their similarity. In the following, several distance measures between graphs based on the

concepts of the *mcs* and the *MCS* will be presented.

For instance, in [10] the distance metric between two graphs is defined in the following way:

$$d_1(g_1, g_2) = |g_1| + |g_2| - 2|mcs(g_1, g_2)| \quad (2.1)$$

It is clear in this definition that, if two graphs are similar, they will have a *mcs*(g_1, g_2) similar to both of them. Then the term $|mcs(g_1, g_2)|$ will be close to $|g_1|$ and $|g_2|$ and therefore the distance close to 0. Conversely if the graphs are dissimilar, the term $|mcs(g_1, g_2)|$ will tend to 0 and the distance will be large.

Another distance based on the *mcs* [20] is:

$$d_2(g_1, g_2) = 1 - \frac{|mcs(g_1, g_2)|}{\max(|g_1|, |g_2|)} \quad (2.2)$$

In this case, if the two graphs are very similar, then their maximum common subgraph will obviously be almost as large as one of the two graphs. Then, the value of the fraction will therefore tend to 1 and the distance will be close to 0. For two dissimilar graphs, their maximum common subgraph will be small, and the ratio will then be close to 0 and the distance close to 1. Clearly, the distance metric d_2 is bounded between 0 and 1, and the more similar two graphs the lower d_2 .

A similar distance is defined in [109], but the union graph is used as normalization factor instead of the size of the largest graph,

$$d_3(g_1, g_2) = 1 - \frac{|mcs(g_1, g_2)|}{|g_1| + |g_2| - |mcs(g_1, g_2)|} \quad (2.3)$$

By "graph union" we mean that the denominator represents the size of the union graph in the set theory point of view. The behavior of such distance is similar to that of d_2 . The use of the graph union is motivated by the fact that changes in the size of the smallest graph that keep the *mcs*(g_1, g_2) constant are not taken into account in d_2 whereas this distance d_3 does take this variations into account. This measure was also demonstrated to be a metric and gives a distance in the interval $[0, 1]$.

Another approach based on the difference between the minimum common supergraph and the maximum common subgraph is given in [39]. In this approach, the distance is given by,

$$d_4(g_1, g_2) = |MCS(g_1, g_2)| - |mcs(g_1, g_2)| \quad (2.4)$$

The basic idea is that for similar graphs, the sizes of the maximum common subgraph and the minimum common supergraph will be similar and the resulting distance will also be small. On the other hand, if the graphs are dissimilar, the two terms will be significantly different, resulting in larger distance values. It is important to notice that d_4 is also a metric.

All these similarity measures admit a certain amount of error-tolerance. That is, with the use of these similarity measures, two graphs do not need to be directly related in terms of (sub)graph isomorphism to be successfully matched. Nevertheless, it is still imperative for these graphs to share isomorphic parts to be considered similar. This means that the two graphs must be identical to a large extent in terms of structure and labels to produce small similarity measures. In practice, node and edge labels which are used to describe the properties of the underlying object (represented by the graph), are usually of continuous nature. In this situation, at least two problems come at hand. First, it is not sufficient to evaluate whether two labels are identical or not, but also to evaluate their similarity. In addition, using all the distances given so far, two labels will be considered different regardless of the difference between them. This may lead to consider two graphs with similar, but not identical labels, completely dissimilar even if they have identical structure. It is therefore clear the need of a more sophisticated method to measure the dissimilarity between graphs which takes into account such limitations. This leads to define the *inexact* or *error-tolerant graph matching*.

2.3 Error-Tolerant Graph Matching

The methods for exact graph matching outlined above have a strong definition and are sustained by a solid mathematical foundation. Nevertheless, their stringent conditions make them applicable only to a very small range of real world problems. In the real world, when objects are encoded using graph-based representations some degree of distortion may be introduced due to multiple reasons. For instance, some form of noise in the acquisition process, non-deterministic elements in some of the processing steps, etc. Hence, graph representations of the same object may differ somewhat, and then two identical objects may not have an exact match when transformed into their graph-based representations. Therefore, it is necessary to introduce some degree of error tolerance into the matching process, so that it can, at a certain extent, take into account these structural differences between the models. For this reason, a number of error-tolerant or inexact graph matching methods have been proposed. They deal with a more general graph matching problem than the (sub)graph isomorphism or the *mcs* and *MCS* problems.

The key idea of error-tolerant graph matching is to measure the similarity between two given graphs instead of simply giving the answer of whether they are identical or not. Usually, in these algorithms the matching between two different nodes that do not preserve the edge compatibility is not forbidden. But a penalty cost is introduced to measure the difference. The task of the algorithm is to find the mapping that minimizes the matching cost. For instance, an error-tolerant graph matching algorithm is expected to discover not only that the graphs in the Figure 2.2(a) share some parts of their structure but also that they are rather similar despite the difference in their structure.

As in the case of exact graph matching, techniques based on tree search can also

be used for inexact graph matching. Differently from exact graph matching where only identical nodes are matched, in this case, the search is usually handled by the partial matching cost obtained so far and an heuristic function to estimate the cost of matching the remaining nodes. This heuristic information can be used to prune unfruitful paths in a *depth-first* search approach or to determine the order in which the search tree has to be traversed in an A* algorithm. For instance, the use of an A* strategy is proposed in [36] to obtain an optimal algorithm to compute the graph distance. In [6, 7, 8] an A*-based approach is used to tackle the matching between two graphs as a bipartite matching problem. A different approach is used in [93], where they present an inexact graph matching method that also exploits some form of contextual information, defining a distance between Function Described Graphs (FDG). Finally, a parallel branch and bound algorithm has been presented in [2] to compute a graph distance between two graphs with the same number of nodes.

Another class of approaches for inexact graph matching are those based on genetic algorithms [3, 30, 100, 103, 111]. The main advantage of genetic algorithms is that they are able to cope with huge search spaces. In genetic algorithms, the possible solutions are encoded as chromosomes. These chromosomes are generated randomly following some operators inspired in the evolution theory, such as mutation and crossover. To determine how good a solution (chromosome) is, a fitness function is defined. The search space is explored with the combination of the fitness function and the randomness of the biologically inspired operators. But the algorithm tends to favor the well-performing candidates. The two main drawbacks of the genetic algorithms are that they are non-deterministic algorithms and that the final output is really dependent on the initialization phase. On the other hand, they are able to cope with difficult optimization problems and they have been widely used to solve NP-complete problems [54]. For instance, in [100] the matching between two given graphs is encoded in a vector form as a set of node-to-node correspondences. Then an underlying distortion model (similar to the idea of cost function presented in the next section) is used to evaluate the quality of the match, by simply accumulating individual costs.

Spectral methods [21, 69, 84, 99, 108, 114, 118] have also been used for inexact graph matching. The basic idea of these methods is to represent graphs by the eigendecomposition of their adjacency or Laplacian matrix. Among the pioneering works of the spectral graph theory applied to graph matching problems there is [108]. This work proposes an algorithm for the Weighted Graph Isomorphism Problem, where the objective is to match a subset of nodes of the first graph with a subset of nodes of the second graph, usually by means of a matching matrix M . One of the major limitations of this approach is that the graphs must have the same number of nodes. A more recent paper [118], proposes a solution to the same problem by combining the use of eigenvalues and eigenvectors with continuous optimization techniques. In [23] some spectral features are used to cluster sets of nodes that are likely to be matched in the optimal correspondence. This method does not suffer from the limitation that all the graphs must have the same number of nodes, like in [108]. Another approach combining the clustering and the spectral graph theory has been presented in [58]. In this approach the nodes are embedded into a so-called graph eigenspace using the eigenvectors of the adjacency matrix. Then a clustering algorithm is used to find

nodes of the two graphs that are to be put in correspondence. A work that is partly related to spectral techniques has been proposed in [98, 99]. Here, the authors assign a *Topological Signature Vector* or TSV to each non-terminal node of a Directed Acyclic Graph (DAG). The TSV associated to a node is based on the sum of the eigenvalues of its descendant DAGs. It can be seen as a signature to represent the shape and can be used both for indexing graphs in a database and for graph matching.

A radically different approach is to cast the inexact graph matching problem as a nonlinear optimization problem. The first family of methods based on this approach is *relaxation labelling* [24, 113]. The main idea is to formulate the graph matching problem as a labeling problem. That is, each node of one of the graphs is to be assigned to one label out of all the possible labels. This label determines the corresponding node of the other graph. To model the compatibility of the node labeling, a Gaussian probability distribution is used. Then, the labelling is iteratively refined until a sufficiently accurate labelling is found. In another approach described in [68], the nodes of the input graph are seen as the observed data while the nodes of the model graph act as hidden random variables. The iterative matching process is handled by the Expectation-Maximization algorithm [33]. Finally, in another classical approach [41], the problem of Weighted Graph Matching is tackled using a technique called graduated non-convexity that permits to avoid poor local optima.

In the last paragraphs we have described some of the most relevant algorithms for error-tolerant graph matching. Besides these algorithms, there exists a lot of other approaches related to the inexact graph matching problem. An excellent review of algorithms for both exact and error-tolerant graph matching can be found in [26].

However, one of the most widely used methods for error-tolerant graph matching is the *graph edit distance*. For this reason, we devote the next section to explain in detail this concept.

2.3.1 Graph Edit Distance

The basic idea behind the graph edit distance is to define a dissimilarity measure between two graphs by the minimum amount of distortion required to transform one graph into the other [13, 38, 87, 106]. To this end, a number of distortion or edit operations e , consisting of the insertion, deletion and substitution of both nodes and edges must be defined. Then, for every pair of graphs (g_1 and g_2), there exists a sequence of edit operations, or edit path $p(g_1, g_2) = (e_1, \dots, e_k)$ (where each e_i denotes an edit operation) that transforms one graph into the other. In Figure 2.3 an example of an edit path between two given graphs g_1 and g_2 is given. In this example, the edit path consists of one edge deletion, one node substitution, one node insertion and two edge insertions.

In general, several edit paths may exist between two given graphs. This set of edit paths is denoted by $\wp(g_1, g_2)$. To quantitatively evaluate which is the best edit path, edit cost functions are introduced. The basic idea is to assign a penalty cost c to each edit operation according to the amount of distortion that it introduces in the transformation. The edit distance between two graphs g_1 and g_2 , denoted by $d(g_1, g_2)$, is defined by the minimum cost edit path that transforms one graph into the other. Formally speaking, the graph edit distance is defined by,

Definition 2.7 (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 graph edit distance between g_1 and g_2 is defined by:

$$d(g_1, g_2) = \min_{(e_1, \dots, e_k) \in \wp(g_1, g_2)} \sum_{i=1}^k c(e_i) \quad (2.5)$$

where $\wp(g_1, g_2)$ denotes the set of edit paths that transforms g_1 into g_2 and $c(e)$ denotes the cost of an edit operation e .

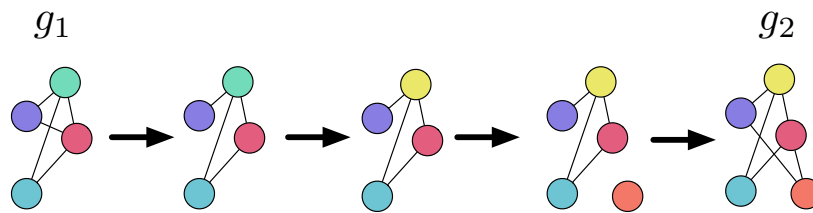


Figure 2.3: A possible edit path between two graphs g_1 and g_2 .

Optimal and approximate algorithms for the graph edit distance computation have been presented so far. Optimal algorithms are usually based on combinatorial search procedures that explore all the possible mappings of nodes and edges of one graph to the nodes and edges of the second graph [13, 87]. The major drawback of such approach is its computational complexity, which is exponential in the number of nodes of the involved graphs. Consequently, its application is restricted to graphs of rather small size in practice. Then, a number of suboptimal methods have been proposed to make the graph edit distance less computationally demanding. Some of these methods are based on local instead of global or optimal optimization [77]. A linear programming method to compute the graph edit distance with unlabeled edges is presented in [56]. Such a method can be used to obtain lower and upper edit distance bounds in polynomial time. In [78] they propose simple variants of the standard method [13] to derive two suboptimal algorithms, that make the computation substantially faster. Finally, in [82], a new efficient algorithm is presented based on a fast suboptimal bipartite optimization procedure.

2.4 Discussion

In this chapter we have introduced the basic terminology and concepts that will be used later in this work. Firstly, we have introduced the concepts of *graph* and *subgraph*. Then, the notion of *graph matching* has been presented, discussing both exact and inexact versions. Regarding exact graph matching, *graph isomorphism* and *subgraph isomorphism* have been defined, as well as the key concepts of *maximum common subgraph* and *minimum common supergraph* and related distance measures. In the context of inexact graph matching a brief review of different techniques has

been presented, and we have focused on the *graph edit distance*, one of the most popular strategies for the graph similarity computation.

In this thesis, we will use several of these concepts and techniques. The basic concepts of graph, subgraph, *mcs*, *MCS* and graph matching will be used throughout all this work. In addition, the Umeyama's spectral-based method [108] for the WGMP problem will become a basic pillar for our work later in Chapter 4. The distance d_2 and the related concepts of *mcs* and *MCS* will form the basis of Chapter 5. Finally, graph edit distance will be extensively used in Chapter 6 to develop the last part of this work. To this end we will basically use the methods introduced in [82] and [78].

Chapter 3

Median Graph

In Chapter 1 we have briefly introduced the concept of *median graph* as one of the possible alternatives to obtain a representative of a set of graphs. This chapter is devoted to present in detail the concept of median graph, its theoretical properties, the existing algorithms for its computation and the applications where median graphs have been used up to now. This extensive introduction to the median graph will serve as the basis for all the work developed in this thesis.

The chapter is structured in four sections. In Section 3.1, the concept of median graph is formally introduced. We will see that there are two different definitions of the median graph: the *set median* and the *generalized median*. After that, we will focus on the generalized median graph. In Section 3.2 we will review some basic theoretical properties of the median graph. Then, in Section 3.3, the existing algorithms for the computation of the median graph will be reviewed, pointing out for each of them their main properties and applications. The chapter concludes with a brief review of the strengths and weaknesses of the median graph and reinforcing the objectives of this thesis.

3.1 Median Graph

Given a set of graphs, the concept of median graph has been presented as a useful tool to obtain a representative of the set. The formal definition of the median graph is given in the following.

Definition 3.1 (*Set and Generalized Median Graph*) Let U be the set of all graphs that can be constructed using labels from L . Given $S = \{g_1, g_2, \dots, g_n\} \subseteq U$, the set median graph \hat{g} and the generalized median graph \bar{g} of S are defined by:

$$\hat{g} = \arg \min_{g \in S} \sum_{g_i \in S} d(g, g_i) \quad (3.1)$$

and

$$\bar{g} = \arg \min_{g \in U} \sum_{g_i \in S} d(g, g_i) \quad (3.2)$$

respectively.

The basic idea behind the concept of median graph is the minimization of the sum of distances (SOD) to all the graphs in the set S . Nevertheless, note that we can distinguish between the *set median graph* and the *generalized median graph*. There is a subtle but really important difference between these definitions: the search space where the median is searched for. While the set median graph is just one graph of the sample set S , the generalized median graph can be any graph of the whole universe U of all the possible graphs that can be constructed using the set of labels L . Thus, \bar{g} is not usually a member of S . This difference makes the computational complexity of both medians drastically different. While the number of pairwise distance computations to find the set median is intuitively upper-bounded by $\frac{1}{2}n(n-1)$, for the generalized median graph, this number becomes exponential with respect to the sum of nodes of the graphs in S . We will go back to the computational complexity of both types of median graph in Section 3.3.

In spite of its lower computational cost, the set median graph is usually not the best representative of a set of graphs, as we will show later in the next chapters. However, it is often a good starting point towards the search of the generalized median graph. On the other hand, the generalized median graph is a more general concept than the set median graph, and it is usually a better representative of a set of graphs. In fact, from a generic point of view, the notion of *median* is accepted as one of the choices to obtain a representative of a set. Thus, in this context, it is clear that the generalized median graph should be a better representative than the set median graph. Consequently, much more effort has been spent to the computation of the generalized median graph than to the computation of the set median graph. Usually, the generalized median graph is considered as the unique option when looking for a prototype of a set of graphs. Notice that, in general, more than one generalized median graph can be found in U for a given set S . However, this last condition is usually not a drawback in practice, since any of such graphs may be used as the representative of a given set. Unless explicitly specified, for the rest of this thesis the concept of generalized median graph will be simply referred as median graph.

It is clear then, that the computation of the median graph is a challenging task. A number of algorithms have been developed up to now for its computation. In Section 3.3 all of these existing algorithms will be explained. But, beyond the algorithmic solutions, there are two interesting theoretical properties related to the median graph that may help us in its computation. Such theoretical properties are introduced in the next section.

3.2 Theoretical Properties of the Median Graph

In spite of its solid definition, there is few work about theoretical properties about the median graph. Two important theoretical properties have been presented [53].

Such properties define the bounds on the size and the SOD of the median graph. As they will form the basis of some parts of this work in forthcoming chapters, we briefly include them here for the sake of completeness.

3.2.1 Bounds on the Size of the Median Graph

Given a set of graphs $S = \{g_1, g_2, \dots, g_n\}$ and \bar{g} the median graph of S , it is shown in [53] that the minimum and maximum number of nodes of \bar{g} must be within the following limits,

$$0 \leq |\bar{g}| \leq \sum_{i=1}^n |g_i| \quad (3.3)$$

That is, it states that the size of the median graph has to be greater or equal than 0 and less or equal than the sum of nodes of all the graphs in S . The proof of this property can be found in [53].

3.2.2 Bounds on the SOD of the Median Graph

The bounds on the SOD of the median graph are slightly more difficult to derive. For the upper bound, it is assumed in [53] that the empty graph g_e and the union graph g_u are meaningful candidates for the median graph. Then, the upper limit on the SOD of the median graph is:

$$SOD(\bar{g}) \leq \min\{SOD(g_e), SOD(g_u)\} \quad (3.4)$$

In the same paper it is noted that this upper bound is very coarse and may thus not be of any practical use. However, they derive a more useful lower bound if the graph distance $d(g_1, g_2)$ fulfils the properties of a metric. For the lower bound, it is assumed that the generalized median graph \bar{g} of n input graphs (n even) is subject to:

$$\begin{aligned} SOD(\bar{g}) &= [d(\bar{g}, g_1) + d(\bar{g}, g_2) + \dots + d(\bar{g}, g_n)] \\ &\geq d(g_1, g_2) + d(g_3, g_4) + \dots + d(g_{n-1}, g_n) \end{aligned} \quad (3.5)$$

Obviously, such a relationship remains true for any partition of S into $\frac{n}{2}$ pairs.

$$(g_{11}, g_{12}), (g_{13}, g_{14}), \dots, (g_{1n-1}, g_{1n}) \text{ where } \{g_{11}, g_{12}, \dots, g_{1n}\} \in S$$

Totally, there exists $\frac{n!}{2^{\frac{n}{2}}(\frac{n}{2})!}$ different partitions, and the lower bound for the SOD is:

$$\begin{aligned} SOD(\bar{g}) &\geq \max\{d(g_{11}, g_{12}) + d(g_{13}, g_{14}) + \dots + d(g_{1n-1}, g_{1n}) \\ &\quad / ((g_{11}, g_{12}), (g_{13}, g_{14}) \dots (g_{1n-1}, g_{1n})) \\ &\quad \text{is a partition of } S\} \end{aligned} \quad (3.6)$$

A similar reasoning can be done when n is odd. In this case there exists $\frac{n!}{2^{\frac{n-1}{2}}(\frac{n-1}{2})!}$ different partitions of $S - \{a_k\}$ where $a_k \in S$. In this case, the lower bound for the SOD is:

$$\begin{aligned} SOD(\bar{g}) \geq & \max\{d(g_{l1}, g_{l2}) + d(g_{l3}, g_{l4}) + \dots + d(g_{ln-2}, g_{n-1}) \quad (3.7) \\ & / ((g_{l1}, g_{l2}), (g_{l3}, g_{l4}) \dots (g_{ln-2}, g_{ln-1})) \\ & \text{is a partition of } S - \{a_k\}, a_k \in S \} \end{aligned}$$

For more details about such theoretical properties, the reader is again referred to [53].

3.3 Median Graph Computation

3.3.1 Computational Complexity

As it has been shown in the definitions of the set and the generalized median graph, a distance between a candidate median g and every graph $g_i \in S$ must be computed to find the solution. Since the computation of the graph edit distance is a well-known problem which is NP-complete in terms of complexity, the computation of both types of median graphs can only be done in exponential time. Conceptually, the computation of the set median graph of n input graphs involves the calculation of $\frac{1}{2}n(n-1)$ pairwise graph distances. Much more complex is the problem of determining the generalized median graph. In this case, the computation is exponential both in the number of graphs in S and their size (even in the special case of strings, the time required is exponential in the number of input strings [32]). This exponential dependency on the number of input graphs comes from the fact that for the generalized median graph, the search space is composed of the graphs comprising all the possible combinations among the different labels in the graphs in S . Note that in the set median graph computation only few states of this huge search space (those corresponding to the graphs in S) have to be taken into account.

3.3.2 Algorithms for the Set Median Graph Computation

The set median graph computation is a relatively simple task. Conceptually, given a set of graphs S , to find the set median graph it is necessary to compute the SOD of each graph in the set, which involves the calculation of $\frac{1}{2}n(n-1)$ pairwise graph distances, and choose the graph with minimum SOD. Nevertheless, such naive approach still needs for a large number of distance computations, and due to the high cost of the graph distance computation, it is often desired to obtain the set median in a more efficient way. In [55] the Fast Search Median Algorithm (FSMA) is presented. They assume a lower bound function b for the SOD is defined. The FSMA partitions the set S into the sets U (*Used*), A (*Alive*) and E (*Eliminated*). The set A is used to keep track of those patterns whose sums have not been calculated and are candidates for being a median (initially $A=P$). Patterns in this set are selected using a g -guided

strategy which chooses a pattern having minimum b first. Once a pattern s is selected, then $b(s)$ is computed and it is transferred to A to U . If $b(s)$ is smaller than those SOD previously computed, both the current median and its associated SOD are updated. Then, b is computed for all alive patterns and those whose lower bounds are not smaller than the current SOD are transferred from A to E . That is, they will no further be considered as candidates. The process ends when $A = \phi$. With this strategy we can avoid the full evaluation of some possible candidates.

3.3.3 Algorithms for the Generalized Median Graph Computation

In this section we will briefly explain the existing algorithms for the generalized median graph computation. They include one exact and two approximate algorithms. The exact algorithm is called Multimatch [76], and was first presented by Munger and Bunke in 1995. Nevertheless, this approach suffers from a high computational complexity, and its application is very limited as we will see later in this chapter. As a consequence, the use of suboptimal methods is the unique feasible option to extend the use of the median graph to more realistic sets of graphs. In this case we will only obtain approximate solutions for the generalized median graph but in a more reasonable time. These two approximate algorithms include a genetic based strategy [53, 76] introduced by Jiang, Bunke and Munger and one greedy-based algorithm presented by Hlaoui and Wang [46]. Both solutions generally apply some kind of heuristics in order to reduce both the cost of the graph distance computation and the size of the search space. In the following, these three initial approaches will be explained and some of their characteristics will be pointed out.

Multimatch Algorithm

This optimal algorithm, introduced in [76], is based on a combinatorial search exploiting the fact that the size of the candidate median must be in between 0 and the sum of nodes of all the graphs in the set S , $\sum |g_i|$ (see Equation 3.3 or [53]). For a particular size n between these limits, all the possible graphs with n nodes can be enumerated. This is the set of candidate graphs for the generalized median graph \bar{g} with size n . Then, for each candidate g , all the possible mappings of its nodes to the nodes of all the graphs g_i can be systematically enumerated, too. Thus, for a fixed n , the graph g with the smallest SOD to all the graphs in S can be found. Then, the median is the graph with the best SOD over all the possible sizes between 0 and $\sum |g_i|$.

To perform this search the algorithm implements an A^* -based search procedure handled by a structure called Multimatch. In such a structure, a simultaneous transformation (edit operations) from a candidate median graph to all the graphs in the set is encoded. In the following, an example will be given in order to clarify how this structure works.

Suppose a set S which is composed of 2 graphs g_1 and g_2 with 3 and 2 nodes respectively (see Figure 3.1). In this case, the size of the median must be in between 0 and 5. The following multimatch structure encodes a possible mapping of a candidate

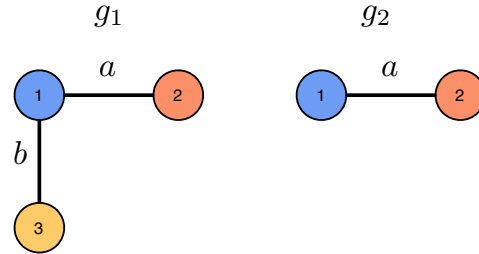


Figure 3.1: Two graphs, g_1 and g_2 .

median graph of size 2 to these graphs.

$$[(1 \mapsto (2, 1)), (2 \mapsto (1, \epsilon)), (\epsilon \mapsto (3, 2))]$$

Such a multimatch means that node 1 of the candidate median g is substituted by node 2 of g_1 and node 1 of g_2 . Similarly, the node 2 of g is substituted by node 1 of g_1 , and deleted in g_2 . Finally, we have an insertion of nodes 3 and 2 in g_1 and g_2 respectively. All these operations are graphically represented in Figure 3.2.

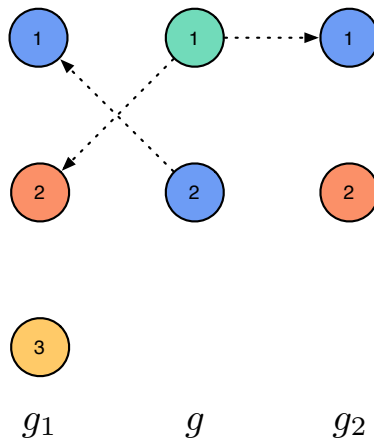


Figure 3.2: Graphic example of the Multimatch structure.

Such encoding clearly defines a transformation between nodes of the candidate median to the graphs in the set. The operations over the edges are implicitly given by such structure. The labels for each node/edge involved in a mapping are selected from the set of labels in such a way that the SOD of the encoded candidate median to all the graphs is minimized.

The algorithm starts by creating a list with all the multimatch structures encoding the possible mappings between the candidate median graphs with only one node and

all the graphs in the set. Then, the candidate median graph with the lowest cost is selected and expanded with one more node. This expansion generates a new set of multimatch structures taking into account all the remaining possibilities given the initial mapping. This new set is added to the initial list of multimatch structures and once again, the candidate with the lowest cost is taken from the list and expanded. This process is iteratively repeated until there are no more candidate medians in the list.

Unfortunately, this approach suffers from a high complexity. For a given set of graphs $S = \{g_1, g_2, \dots, g_k\}$ where $g_i = (V_i, E_i, \mu_i, \nu_i)$, the complexity of this algorithm in the worst case is $O(kN_L n_{max}^2 n_p^{n_{max}})$. Where k is the number of graphs in the set S , N_L is the size of the set of labels for the nodes and edges, $n_{max} = \max\{|g_i|\}$ ($i = 1 \dots k$) and $n_p = \prod_{i=1}^k \{|g_i|\}$. For this reason, this approach is unfeasible even for a small number of small graphs. The results presented in [19] show that for a set of two graphs with six nodes each, the time needed to compute the median graph grows up to 10^4 seconds (all the experiments were run on an IBM Thinnode Power 2 computer). Due to this large complexity, this method could only be applied to small graphs generated randomly. For a detailed explanation of this method, the reader is referred to [76] (in German).

Genetic-based Algorithm

A genetic algorithm has been used in [19, 52, 53, 76] to obtain approximate solutions for the median graph. In this approach, the chromosomes represent possible mappings from a candidate median to all the input graphs in a similar way to the multimatch approach. The fitness function of each chromosome is the SOD of the mapping codified by the chromosome.

Each chromosome is an array of integers representing the node assignment of the candidate graph g to every graph g_i in S . The length of the chromosome is equal to the sum of nodes of all input graphs. Each node of a graph g_i in the input set, has assigned a fixed position in the chromosome. The value in this position specifies the node of the candidate median assigned to that node (number 0 denotes a deletion).

Figure 3.3 shows an example of a possible codification. Suppose the set is the same as in the Figure 3.1. In this case we have two graphs with 3 and 2 nodes respectively. Thus, the chromosome has 5 positions. The first three positions correspond to the first graph while the two last positions correspond to the second graph. In this case, the candidate median g implicitly codified by the chromosome has a size of 2. When transforming g_1 into g , the first node of g_1 is substituted by the second node of g , while the third node of g_1 is substituted by the first node of g . The second node of g_1 is deleted. A similar reasoning can be done for the mapping between g_2 and g .

The candidate median g codified by a chromosome is partially complete because it only specifies the way in which the nodes of g are mapped to all the input graphs. The labeling for the nodes and edges is still missing. However, this missing information can be completed in low-polynomial time in such a way that the term $SOD(g)$ is minimized. The labeling for the nodes is carried out as follows. The cost of all the possible labels of the node is evaluated in terms of the edit operations. For each node the label with the lower cost is selected. The labeling for the edges can be done in a

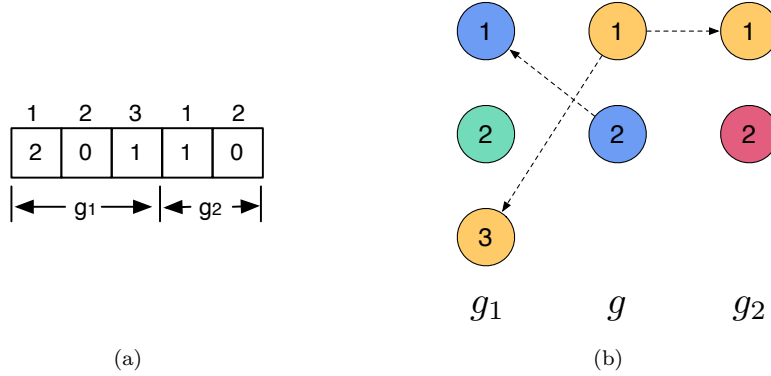


Figure 3.3: Chromosome representation of a possible candidate median g (a), a graphical representation of the mapping the chromosome codifies (b).

similar way. After this completion process, a chromosome fully specifies a candidate median together with its SOD.

The population of chromosomes evolves applying the classical genetic operators, until the maximum number of iterations is reached or until the population becomes homogeneous enough in terms of the variance of the SOD.

Applications of this algorithm include the computation of the median graph with randomly generated graphs [53], median words computation for OCR application [53], synthesis of character samples [51] and synthesis of graphical symbols [52]. Nevertheless, the sets used in these examples only include a small number of graphs with a small number of nodes each. These graphs were usually created synthetically and contain a small number of different labels in the nodes and the edges. It is important to notice that in some cases the graph-based representations were composed only of nodes, without any edge between them.

A complete description of this algorithm can be found in [53, 76].

Greedy-based Algorithm

In [45, 46], a greedy-based algorithm has been presented for the computation of the approximate median graph. The key idea of that algorithm is based on the observation that the edit distance between two graphs can be split into two parts: a node-to-node distance and an edge-to-edge distance (i.e. $d(g_1, g_2) = d_n(g_1, g_2) + d_e(g_1, g_2)$). Then, the term SOD is also decomposed into these 2 parts, which correspond to the SOD due to the nodes (SOD_n) and the SOD due to the edges (SOD_e).

$$SOD(g) = \sum d(g, g_i) = \sum d_n(g, g_i) + \sum d_e(g, g_i) = SOD_n(g) + SOD_e(g) \quad (3.8)$$

The search for the best configuration of nodes and edges is divided into two parts. First, the algorithm iteratively searches for the subset of nodes most likely to be part of the final solution (minimizing the term SOD_n), and then searches for the best subset of edges connecting the selected nodes (minimizing the term SOD_e).

Despite this initial heuristic, the algorithm still suffers from a large complexity because the size of the median graph is still unknown. For this reason, before the search step, the algorithm performs a preparation step in order to reduce the search space. This preparation step includes the determination of the size of the median graph and the reduction of the set of possible nodes used in the search step. The cardinality M of the median graph is determined using the following formula (based on some hypothesis on the edit operations which are partially validated through some experiments):

$$M = \frac{1}{n} \sum_{i=1}^n |g_i| \quad (3.9)$$

To reduce the number of nodes used in the search step, the k -means algorithm is applied to cluster the labels of the nodes (with $k = 2M$). Thus, in the search procedure only k nodes will be included, the closest to the k centers of the clusters.

In order to validate the usefulness of the proposed method, they first compare this method with the genetic-based approach explained before with randomly generated graphs. Then, they use the same kind of data to cluster similar graphs. Finally they apply the new algorithm to content-based image retrieval using a synthetic database. It is important to mention again that all of these experiments were done with synthetic data and graphs of a relatively small size (5-10 nodes per graph) and with a low number of labels. For more details about this method, the reader is referred to [45, 46].

3.4 Discussion

In this chapter we have presented a detailed introduction to the median graph concept. Its close definition to the generic concept of *median* gives the median graph the potential to be a good representative of a set of graphs. Therefore, the median graph could play an important role in any graph-based algorithm for machine learning where a representative is needed.

Despite its potential applications, the computation of the median graph has been shown to be a highly complex task. The existing algorithms are very limited in its application and are sometimes restricted to synthetic data with relatively small graphs. From a theoretical point of view, the properties mentioned in Section 3.2 can be used to bound the search space of the median, either by limiting the size of the candidate medians or by discarding some of these candidate medians based on the bounds on the SOD, for instance. Nevertheless, as mentioned in [53], these bounds are sometimes too coarse and are not very useful and they do not allow to reduce the complexity of the median graph computation in any of the existing algorithms.

For this reason, this thesis addresses the problem of the median graph computation. Along the next chapters we will propose new theoretical properties and new algorithms to optimize its computation with the main objective of giving the median graph the opportunity to leave the box of theoretical concepts and be applied to real problems.

Chapter 4

Spectral Median Graph

Let us recall from Chapter 2 that a weighted graph is a graph where the labels of edges are numerical values (or weights). These weights can be integers or real numbers, usually representing distances, costs or an affinity level between the nodes. Weighted graphs are useful in many applications. For example, let us think of a graph representing possible routes between cities, where the weight might be the distance along a given route, the time to get from one city to another or the cost of taking some action while traversing an edge. Another example could be the representation of graphical symbols. In this case nodes could represent either terminal or junction points in the symbol and edges the distance between two points in the symbol, for instance. Under these contexts, the median graph may represent the mean cost of an operation in the city example or the mean shape of the symbol. With this kind of potential applications at hand, it is therefore clear the usefulness of the median graph under the context of weighted graphs.

In this chapter, we will propose the first new strategy for the median graph computation, tackling the problem only under the particular class of weighted graphs. To this end, we will use the well-known discipline of spectral graph theory, that is the decomposition in eigenvalues and eigenvectors of the adjacency or Laplacian matrix of a graph. With the use of the spectral graph theory and the Umeyama's method for solving the Weighted Graph Isomorphism problem, we present in this chapter the novel concept of *Spectral Median Graph*, as well as an algorithm to compute it. With this new concept we aim to obtain good approximations of the median graph.

The chapter is organized as follows. In the next section, a brief introduction to the spectral graph theory and its applications to the graph matching problem is presented. We will specially emphasize in this section the Umeyama's method, one of the most well-known methods to solve the Weighted Graph Isomorphism problem under the optic of the spectral graph theory. Then, in Section 4.2, we introduce the novel concept of *Spectral Median Graph*, giving the definition of both set and generalized spectral median graph. We also provide an incremental algorithm to compute the generalized spectral median graph. After that, in Section 4.2.2 we present some experiments in order to evaluate the quality of the obtained median, and we compare the results with the genetic approach presented in [53]. Finally, the chapter concludes

with a brief discussion on the new theory and the results.

4.1 Graph Matching using Spectral Techniques

The foundations of spectral graph theory were laid in the fifties and sixties as a result of the work of a considerable number of mathematicians. Most of this early work was concerned with the relation between spectral and structural properties of graphs [31]. Such relations were obtained from the analysis of the spectra of specific matrices defined from the graph, including the adjacency matrix and the Laplacian matrix, among others. Over the last fifty years, a large number of results have been obtained describing the spectra of these specific matrices, and extracting information about the graph from this spectrum. These properties include, for instance, the number of nodes of the graph, the connectivity between them, the maximum degree on a regular graph, some geometric properties of the graph and others. More recently, generalized Laplacians, a whole family of matrices associated with a given graph, have been studied. In addition, impressive results characterizing some graph properties such as planarity, in terms of eigenvalues of generalized Laplacians, have been obtained [25].

Beyond these initial works and in a more practical sense, spectral graph theory has also been applied to solve the graph matching problem. With the use of spectral graph theory, approximate solutions for graph matching have been reported in polynomial time. In this work, we take advantage of such techniques to address the median graph computation using weighted graphs.

4.1.1 Matrices of Graphs

Basically, spectral graph theory makes use of two matrices, namely the *adjacency matrix* and the *Laplacian matrix* of a graph. Initial works in the field of spectral graph theory focused their attention on the adjacency matrix [31], but recently the Laplacian matrix has received an increasing interest [25, 75, 101]. In this section we will first briefly review how we can obtain these matrices from a weighted graph. Then, we will introduce the way in that these matrices can be spectrally decomposed to obtain the spectrum of a graph (list of eigenvalues) and the eigenvectors associated to this spectrum, that will permit to analyze some of its structural properties.

Let $g = (V, E, \mu, \nu)$ be a *weighted graph*. Its *adjacency matrix* denoted by A_g , with elements $a_{i,j}$ is defined as:

$$a_{i,j} = \begin{cases} w_{ij} & \text{if } (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$$

That is, the matrix (whose order is equal to the number of nodes in the graph) contains at each position (i, j) the weight of the edge linking the nodes i and j .

The *Laplacian matrix* of g denoted by L_g and whose elements are denoted by $l_{i,j}$ is given by:

$$l_{i,j} = \begin{cases} -w_{ij} & \text{if } (i,j) \in E \\ d_i(v) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

where $d_i(v)$ denotes the degree of a vertex v_i ¹.

Notice that for undirected graphs, the adjacency and the Laplacian matrices are symmetric. Figures 4.1 and 4.2 show an example of a weighted and an unweighted graph, g_w and g_u and their corresponding adjacency and Laplacian matrices respectively. In these figures, the numbers inside the nodes simply enumerate the nodes. They do not have the meaning of weights.

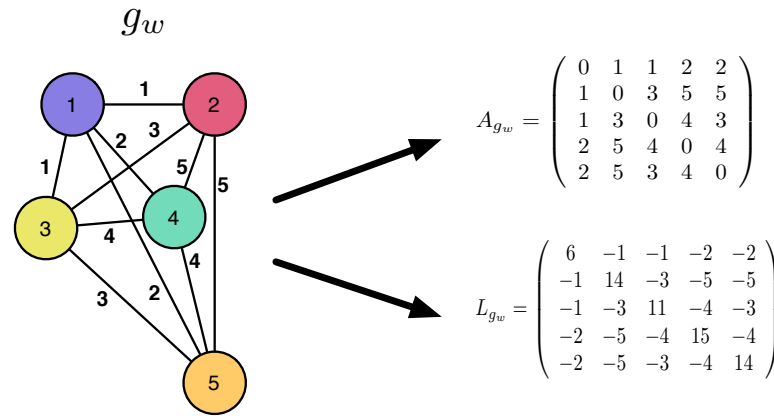


Figure 4.1: Example of a weighted graph and its corresponding adjacency and Laplacian matrices.

In the Spectral Graph Theory, such matrices are spectrally decomposed in order to obtain their eigenvalues (spectra of the graph) and their corresponding eigenvectors. For the sake of completeness, we will briefly recall such procedure in the next section².

4.1.2 Spectral Decomposition of Matrices

In the mathematical discipline of linear algebra [61], *spectral decomposition* (also called *eigendecomposition*) is the factorization of a matrix into a canonical form, whereby the matrix is represented in terms of its eigenvalues and eigenvectors.

Spectral Decomposition

Given an adjacency matrix \mathbf{A} of order n , we can calculate the eigenvalues $\lambda = (\lambda_1, \lambda_2, \dots, \lambda_n)$ by solving the equation,

¹The degree of a vertex in weighted and unweighted graphs is defined as the sum of the weights of edges at v , i.e. $d_i(v) = \sum_j w_{ij}$.

²For the rest of the work we assume that we only work with adjacency matrices of graphs.

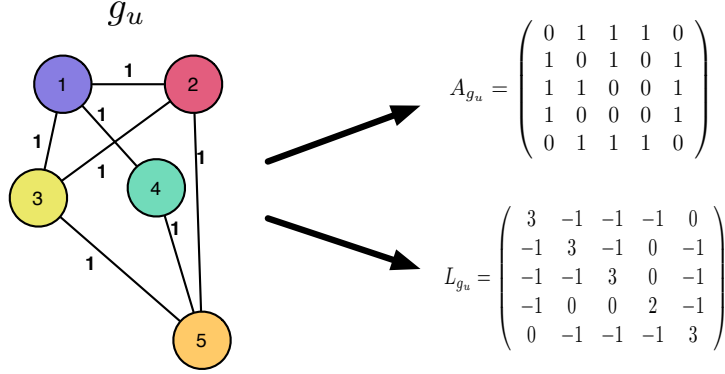


Figure 4.2: Example of an unweighted graph and its corresponding adjacency and Laplacian matrices.

$$p(\lambda) = \det(A - \lambda I) = |A - \lambda I| = 0 \quad (4.1)$$

which is known as the characteristic polynomial. The roots of $p(\lambda)$ correspond to the eigenvalues. Normally, the order of the eigenvalues is fixed according to the decreasing magnitude of their values, i.e. $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n$.

After that, we must solve, for every eigenvalue λ_i (with $i = 1 \dots n$), the system of equations,

$$(\mathbf{A} - \lambda_i \mathbf{I}) \vec{u}_i = 0 \quad (4.2)$$

to find the eigenvector \vec{u}_i associated to λ_i . After that, we can construct the modal matrix (also called eigenvector matrix) by placing all the eigenvectors as columns of the matrix in the following way,

$$\mathbf{U} = (\vec{u}_1 | \vec{u}_2 | \dots | \vec{u}_n) \quad (4.3)$$

We emphasize that if the initial matrix \mathbf{A} is not symmetric, the elements of the modal matrix are complex numbers whereas if the initial matrix is symmetric, they are real numbers. The original matrix \mathbf{A} can be recovered from its eigenvectors and eigenvalues by computing the following expression,

$$\mathbf{A} = \mathbf{U} \cdot \text{diag}(\lambda) \cdot \mathbf{U}^T \quad (4.4)$$

where $\text{diag}(\lambda)$ is a diagonal matrix with the eigenvalues at the main diagonal.

In practice, eigenvalues of large matrices are not computed using the characteristic polynomial. Computing the polynomial becomes intrinsically expensive, and exact (symbolic) roots of a high-degree polynomial can be difficult to compute and express: the Abel-Ruffini theorem [61] implies that the roots of high-degree (5 and above)

polynomials cannot in general be expressed by simply using n -th roots. Effective numerical algorithms for approximating roots of polynomials exist, but small errors in the eigenvalues can lead to large errors in the eigenvectors. Therefore, general algorithms to find eigenvectors and eigenvalues are iterative. The easiest method is the power method: a random vector \vec{v} is chosen and a sequence of unit vectors is computed as

$$\frac{A\vec{v}}{\|A\vec{v}\|}, \frac{A^2\vec{v}}{\|A^2\vec{v}\|}, \frac{A^3\vec{v}}{\|A^3\vec{v}\|}, \dots \quad (4.5)$$

This sequence will almost always converge to an eigenvector corresponding to the eigenvalue of greatest magnitude. This algorithm is simple, but not very useful by itself. However, it is the basis of some popular methods such as the QR factorization algorithm [42].

Once the eigenvalues are computed, the eigenvectors \vec{u}_i can be calculated by solving the Equation 4.2 using Gaussian elimination or any other method for solving matrix equations.

4.1.3 Graph Matching using the Spectral Graph Theory

Although spectral techniques can be used to derive some interesting properties of graphs as we have seen at the beginning of this section, from a practical point of view, they have been used in graph-based applications such as graph matching and shape description.

Graph matching techniques using spectral methods are based on the observation that eigenvalues and eigenvectors of the adjacency matrix are invariant to node permutations. In this sense, if two graphs are isomorphic, their adjacency matrices will have the same eigenvalues and eigenvectors. Since the eigendecomposition of a matrix is a well-known method that can be solved in polynomial time [42], it can be used to reduce the complexity of graph matching algorithms. Unfortunately, the reverse conversion is not true, i.e. having equal eigenvalues and eigenvectors does not guarantee that the two original graphs are isomorphic. In addition, this method suffers from the limitation that it is not able to exploit complex node or edge attributes such as feature vectors, probability density functions, etc. It can only take advantage of simple labels such as real numbers.

Among the pioneering works related to graph matching using spectral techniques we can find the paper of Umeyama [108], in which the *Weighted Graph Isomorphism Problem* (WGIP) is addressed. This method can only be applied to graphs with the same number of nodes, and it is only suitable to compare graphs which are nearly isomorphic. It has the advantage that it can be applied to both directed and undirected graphs and has a $O(n^3)$ computational complexity, where n is the size of the graphs. Since we will use this method in our work, it will be explained in detail in the next section. A more recent paper by Xu and King [118] proposes a different solution for the Weighted Graph Isomorphism Problem combining eigenvalues and eigenvectors with continuous optimization techniques. Although this technique provides slightly better results than the Umeyama's method, it is a bit more difficult to implement.

For this reason we have chosen the first method.

Also related to weighted graphs, in 2001, Carcassoni and Hancock [23] proposed a method based on spectral techniques to define clusters of nodes that are likely to be matched together in the optimal correspondence. Such method does not suffer from the limitation that the graphs must have the same number of nodes. Another idea that combines spectral approaches with clustering has been presented in [58]. In this work, the authors create a vector space called *graph eigenspace* using the eigenvectors of the adjacency matrices, and the nodes are projected onto points in this space. Then, a clustering algorithm is used to find nodes of the two graphs that are to be put into correspondence.

A work that is partly related to spectral techniques has been proposed in 2001 [98, 99]. Here, the authors assign a *Topological Signature Vector* or TSV to each non-terminal node of a Directed Acyclic Graph (DAG). The TSV associated to a node is based on the sum of the eigenvalues of its descendant DAGs. These TSV can be seen as signatures to represent shapes and are used later for both indexing graphs in a database and graph matching.

4.1.4 Weighted Graph Isomorphism by means of Spectral Techniques

One of the most well-known methods to perform inexact graph matching using spectral techniques was presented by S. Umeyama in 1988 [108]. This method is applied to the Weighted Graph Isomorphism Problem, which is a special case of error-correcting graph matching. We will adopt such approach in the rest of this chapter to compute an approximate solution to the distance between two graphs.

Umeyama's Method

The method proposed by Umeyama optimizes an objective function (the mapping between the nodes of both graphs) under the assumption that the graphs are isomorphic. Moreover, this method also obtains good approximate solutions if the graphs are nearly isomorphic. To this end, the author reformulates the objective function in terms of matrices using a permutation matrix. The domain of this objective function is extended to the set of orthogonal matrices (the permutation matrix is a kind of orthogonal matrix) to be able to use the eigendecomposition to solve the problem.

Given two graphs g_1 and g_2 , Umeyama's method works as follows. The first step is to compute the eigendecomposition of their adjacency matrices A_{g_1} and A_{g_2} respectively, as explained before. Thus, we obtain the spectrum of each graph λ_{g_1} and λ_{g_2} (ordered list of eigenvalues in decreasing order) and their modal matrices \mathbf{U}_{g_1} and \mathbf{U}_{g_2} (matrix representation of corresponding eigenvectors). Then, the absolute value of the modal matrices is computed, giving $\bar{\mathbf{U}}_{g_1}$ and $\bar{\mathbf{U}}_{g_2}$. The product $\bar{\mathbf{U}}_{g_2} \cdot \bar{\mathbf{U}}_{g_1}^T$ gives a matrix that can be interpreted as the similarity matrix between the eigenvectors of the graphs. That is, the value of the column i and row j in this matrix is the dot product (projection) of the i -th and j -th eigenvectors of g_1 and g_2 respectively. Finally, this matrix is processed using some assignment algorithm (in this case the Hungarian Method [60]) to obtain the permutation matrix \mathbf{P} , which represents the

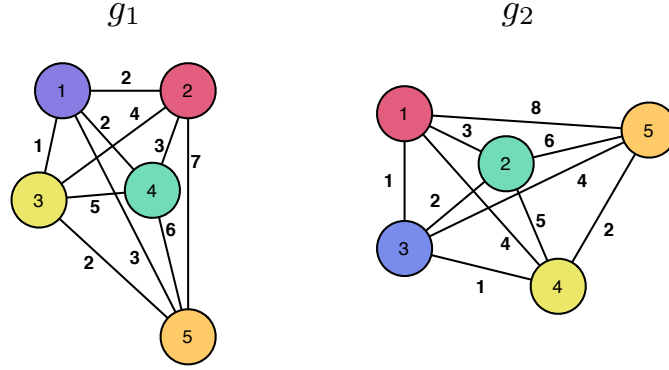


Figure 4.3: Example of two weighted undirected graphs.

correspondence between the nodes of g_1 and the nodes of g_2 , i.e.

$$P_{i,j} = \begin{cases} 1 & \text{if } f(v_i^1) = v_j^2 \\ 0 & \text{otherwise} \end{cases}$$

where v_i^1 and v_j^2 with $i, j = 1 \dots n$ are the nodes of g_1 and g_2 respectively. Using \mathbf{P} we are able to compute the distance $d(g_1, g_2)$ between g_1 and g_2 as:

$$d(g_1, g_2) = d(P) = \|PA_{g_1}P^T - A_{g_2}\|^2 \quad (4.6)$$

where $\|\cdot\|$ is the Euclidean matrix norm ($\|A\| = (\sum_{i=1}^n \sum_{j=1}^n |a_{ij}|^2)^{1/2}$).

Example

In this example we will use the graphs of Figure 4.3. As it can be seen in the figure, both graphs have the same number of nodes (5), and the same edge structure. Basically, there are two differences between them, which are: a) g_2 has been rotated 90° counterclockwise with respect to g_1 and b) some labels of g_2 differ in their values from the labels of the corresponding edges of g_1 . Even so, it is easy to see that the first node in g_1 corresponds to the third node in g_2 , the second node in g_1 corresponds to the first node in g_2 , the third node in g_1 corresponds to the fourth node in g_2 , the fourth node in g_1 corresponds to the second node in g_2 and the fifth node in g_1 corresponds to the fifth node in g_2 .

If we apply the Umeyama's method in order to find the correspondence between the nodes of g_1 and the nodes of g_2 , the first step is to calculate the adjacency matrices A_{g_1} and A_{g_2} , which are:

$$\mathbf{A}_{g_1} = \begin{pmatrix} 0 & 2 & 1 & 2 & 3 \\ 2 & 0 & 4 & 3 & 7 \\ 1 & 4 & 0 & 5 & 2 \\ 2 & 3 & 6 & 0 & 6 \\ 3 & 7 & 2 & 6 & 0 \end{pmatrix}$$

and

$$\mathbf{A}_{g_2} = \begin{pmatrix} 0 & 3 & 1 & 4 & 8 \\ 3 & 0 & 2 & 5 & 6 \\ 1 & 2 & 0 & 1 & 4 \\ 4 & 5 & 1 & 0 & 2 \\ 8 & 6 & 4 & 2 & 0 \end{pmatrix}$$

for g_1 and g_2 respectively. After that, if we compute the eigendecomposition of each adjacency matrix given above, we will obtain:

$$\lambda_{g_1} = (14.79, -0.346, -1.974, -3.656, -8.816)$$

and

$$\mathbf{U}_{g_1} = \begin{pmatrix} 0.268 & -0.720 & -0.604 & -0.204 & -0.041 \\ 0.496 & -0.064 & 0.499 & -0.494 & -0.505 \\ 0.390 & 0.597 & -0.408 & -0.452 & 0.344 \\ 0.487 & 0.250 & -0.269 & 0.648 & -0.453 \\ 0.539 & -0.239 & 0.380 & 0.297 & 0.646 \end{pmatrix}$$

for the first graph, and

$$\lambda_{g_2} = (14.79, -0.346, -1.974, -3.656, -8.816)$$

and

$$\mathbf{U}_{g_2} = \begin{pmatrix} -0.499 & -0.076 & 0.578 & -0.337 & 0.543 \\ -0.473 & -0.199 & -0.374 & 0.683 & 0.359 \\ -0.264 & 0 - 742 & -0.489 & -0.328 & -0.177 \\ -0.374 & -0.575 & -0.429 & -0.512 & -0.285 \\ -0.562 & 0.269 & 0.317 & 0.220 & -0.679 \end{pmatrix}$$

for the second graph.

Now we are able to compute the product $\bar{\mathbf{U}}_{g_1} \cdot \bar{\mathbf{U}}_{g_2}^T$ which, in this case, will be:

$$\bar{\mathbf{U}}_{g_2} \cdot \bar{\mathbf{U}}_{g_1}^T = \begin{pmatrix} 0.6308 & 0.6516 & 0.9761 & 0.8916 & 0.6103 \\ 0.9838 & 0.9542 & 0.6766 & 0.8353 & 0.9077 \\ 0.8174 & 0.8900 & 0.9567 & 0.9959 & 0.8442 \\ 0.8847 & 0.9881 & 0.7411 & 0.9045 & 0.8788 \\ 0.9603 & 0.8813 & 0.7200 & 0.8405 & 0.9938 \end{pmatrix}$$

Finally, if we apply the Hungarian Method to this matrix, we obtain the permutation matrix:

$$\mathbf{P} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

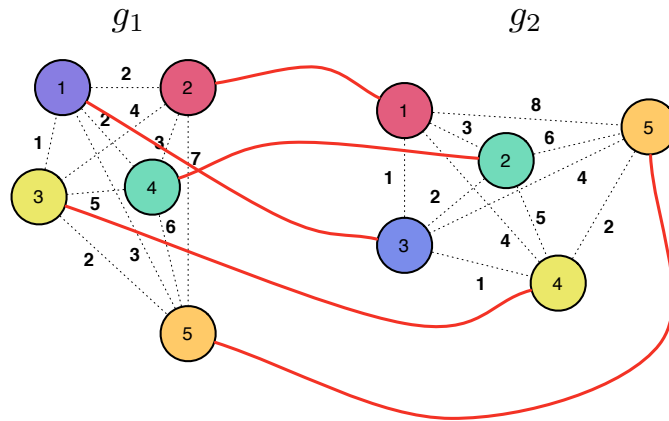


Figure 4.4: Node correspondence given by Umeyama's method.

Matrix \mathbf{P} indicates the mapping of the nodes of g_1 to the nodes of g_2 . In our case $f(1) = 3$, $f(2) = 1$, $f(3) = 4$, $f(4) = 2$ and $f(5) = 5$, which is the correspondence described at the beginning of this section. The result of such a correspondence can be seen in the Figure 4.4. Here, for clarity, the lines representing the original edges of both graphs have been picked, while the red continuous lines represent the relation (matching) between the nodes of g_1 and the nodes of g_2 .

Finally, we are able to compute the distance (cost) between g_1 and g_2 by computing Equation 4.6,

$$d(g_1, g_2) = d(P) = \|PA_{g_1}P^T - A_{g_2}\|^2 = 6$$

This method has the main advantage that good approximate solutions for the weighted graph matching problem can be found in polynomial time ($O(n^3)$, where n is the number of nodes in the graphs), if graphs are nearly isomorphic. However, it has two important restrictions. Firstly, this method can only be applied to graphs with the same number of nodes. As a consequence only the graph isomorphism problem can be addressed with this technique. In addition, it can only be applied to graphs with one numerical attribute in the nodes and/or edges. Consequently the possibility of associating more than one numerical value at the nodes and/or edges (a feature vector for instance) can not be directly exploited. These limitations must be taken

into account when this method is applied to the graph matching problem, since it can restrict the spectrum of the applications where it can be used. For further details on the Umeyama's method, see [108].

4.2 Spectral Median Graph

In this section, we will present the main contribution of this chapter, the *set* and the *generalized spectral median graph* (also called *set* and *generalized spectral median eigenmode* respectively). Combining the concepts of the generalized and the set median graph explained in Chapter 3, together with the spectral graph theory and, in particular, with the Umeyama's method to find a sub-optimal labeling between two weighted graphs, we define these two novel concepts using the modal matrices of graphs. In addition, we present an incremental algorithm in order to compute the median graph in linear time with respect to the number of graphs in the set. With the use of spectral graph matching the complexity of the matching process is reduced to polynomial time $O(n^3)$. Thus, we are able to compute good approximate solutions of the generalized median graph in a reasonable time.

Definition 4.1 (Set and Generalized Median Eigenmode) Let K be the set of all modal matrices of order p . Given a set of modal matrices $L = \{U_1, U_2, \dots, U_n\}$, we define the set median eigenmode \hat{U} and the generalized median eigenmode \bar{U} of L as:

$$\hat{U} = \arg \left(\max_{U \in L} \sum_{U_i \in L} \Gamma(U, U_i) \right) \quad (4.7)$$

and

$$\bar{U} = \arg \left(\max_{U \in K} \sum_{U_i \in L} \Gamma(U, U_i) \right) \quad (4.8)$$

where the function $\Gamma(A, B)$ is defined in the following way:

$$\Gamma(A, B) = \sum_{i=1}^p \vec{a}_i \cdot \vec{b}_i \quad (4.9)$$

Here $A = (\vec{a}_1, \vec{a}_2, \dots, \vec{a}_p)$ and $B = (\vec{b}_1, \vec{b}_2, \dots, \vec{b}_p)$ are two matrices and \vec{a}_i and \vec{b}_i denote the vectors formed by each column of these matrices. The term $\vec{a} \cdot \vec{b}$ denotes the inner product between two vectors.

That is, the function $\Gamma(A, B)$ is the sum of the inner products between the corresponding vectors in the matrices. In this way, the more similar the matrices A and B the larger Γ . Since similar graphs will have similar modal matrices, then, the more similar two graphs the larger Γ . In other words, maximizing the term Γ should lead to minimize the distance between two graphs. Note that to be able to maximize

this term, the first thing to be done is to put in correspondence (i.e. to align) the eigenvectors of each modal matrix in L with the candidate spectral median graph. As we have seen in the previous section, this can be done using the Umeyama's method.

Conceptually, finding the set median eigenmode is a relative simple task. That is, using the Umeyama's method we can put in correspondence all the graphs in the set, either in the graph domain or in the spectral domain. Then, once all the graphs are aligned, the modal matrix that maximizes the projection (i.e. Γ) to all the modal matrices in L is the set median eigenmode.

Conversely, finding the generalized median eigenmode is a more complex task. In this case we are in a continuous domain (i.e. the values of the eigenvalues and eigenvectors belong to \mathbb{R} and \mathbb{R}^n respectively). Thus, the optimal modal matrix has to be found in this domain. In other words, we are looking for a real matrix \mathbf{U} which maximizes the term $\sum \Gamma(U, U_i)$ for all the matrices U_i in L . This is a more difficult problem, and to solve it we propose an approximate incremental algorithm.

4.2.1 Synthesis of the Generalized Median Eigenmode

Two different methods of synthesizing a median from a set of elements are used in the literature [95], namely, the incremental method and the hierarchical method. Therefore, two sub-optimal methods to synthesize \bar{U} could be defined. In the former, \bar{U} is iteratively updated according to the input graphs, which are sequentially considered. The advantage of this method is that if a new graph is added to the learning set, the new median can be re-computed considering only this new graph. The main drawback of this incremental approach is that different median eigenmodes can be synthesized from a set of unlabeled graphs depending on the order of presentation of the graphs. To infer a unique \bar{U} , a hierarchical method must be used, which consists of successively merging pairs of graphs with minimal distance. The drawback here is that the full ensemble of graphs is needed to generate the median eigenmode.

We adopted the incremental approach to compute the generalized median eigenmode. Given a set of graphs $S = \{g_1, g_1, \dots, g_n\}$, which are supposed to belong to the same class, we do not, in general, have any way of synthesizing the generalized median eigenmode \bar{U} that represents the ensemble unless we can first establish a common labeling of their vertices. We would like to choose the common labeling so as to minimize the dissimilarity between the given graphs and so to maximize the correlation between the eigenmodes $L = \{U_1, U_2, \dots, U_n\}$, extracted from the adjacency matrices. To this end, we first maximize the correlation between the modal matrices of the first two graphs in the set using the Umeyama's method [108]. This will give us a permutation matrix P . This permutation matrix is then used to put in correspondence the eigenmodes of the corresponding modal matrices. Then, an intermediate median graph that maximizes the term Γ with these two modal matrices is obtained by taking the mean of their eigenvectors. This maximization is done under two assumptions. First of all, we suppose our graphs are nearly isomorphic (this assumption is already made by the Umeyama's method). Since the graphs are nearly isomorphic, their corresponding eigenvectors (once aligned) will be close to each other. Then, for each pair of aligned eigenvectors, their mean is supposed to be the eigenmode in between them which maximizes their dot product. After these steps, the modal

matrix of this intermediate median graph is used to maximize the correlation to the next graph in the set using the same steps finding a new intermediate median graph. The process is repeated iteratively until the last graph in the set is processed, giving the final median graph.

This process is shown in **Algorithm 1**, and it is composed of three main procedures:

1. *Eigen-Decomposition*: Obtains the eigendecomposition of the adjacency matrix A_{g_k} of the graph g_k that is considered at the current iteration of the algorithm.
2. *Matching*: Finds a sub-optimal labeling between the current graph g_k and the intermediate candidate median graph \bar{g} using the Umeyama's method.
3. *Median_Graph*³: The nodes of the current graph g_k are reordered according with the node correspondence f^k found in the previous step. Then, the median graph \bar{g} is updated by computing the mean of the intermediate median \bar{g} and the reordered version of the current graph g_k .

Algorithm 1 Generalized Spectral Median Graph Algorithm

Require: A set of graphs $S = \{g_1, g_2, \dots, g_n\}$

Ensure: The generalized median eigenmode of S

- 1: $\{\bar{U}, \bar{\lambda}\} := \text{Eigen_Decomposition}(g_1) // \bar{g}$ is composed only of g_1
 - 2: **for** $k := 2$ to n **do**
 - 3: $\{U_k, \lambda_k\} := \text{Eigen_Decomposition}(g_k) // \bar{g}$ is composed only of g_1
 - 4: let $f_k : g_k \rightarrow \bar{g}$ be the labeling found by $\text{Matching}(U_k, \bar{U})$
 - 5: $\bar{g} := \text{Median_Graph}(\bar{g}, g_k, f_k)$
 - 6: $\{\bar{U}, \bar{\lambda}\} := \text{Eigen_Decomposition}(\bar{g})$
 - 7: **end for**
-

Example

An example of the generalized spectral median graph computation is given in Figure 4.5. In this example the set S of input graphs is composed of g_1 , g_2 and g_3 , i.e. $S = \{g_1, g_2, g_3\}$, where each one is a slightly distorted and rotated version of the other ones. To compute the generalized spectral median graph the algorithm takes g_1 and g_2 and compute an intermediate spectral graph $g_{1,2}$ by means of their respective eigendecompositions, \mathbf{U}_1 and \mathbf{U}_2 . This step includes:

1. Put the nodes of both graphs in correspondence (by extension their modal matrices are also put in correspondence).
2. Compute the mean of their modal matrices (which leads to obtain approximately the mean of the labels of the nodes).

³We have to mention that the *Median_Graph* function is not transitive. For this reason, the procedure has to keep the information of a graph that is the addition of all the graphs used to compute the median.

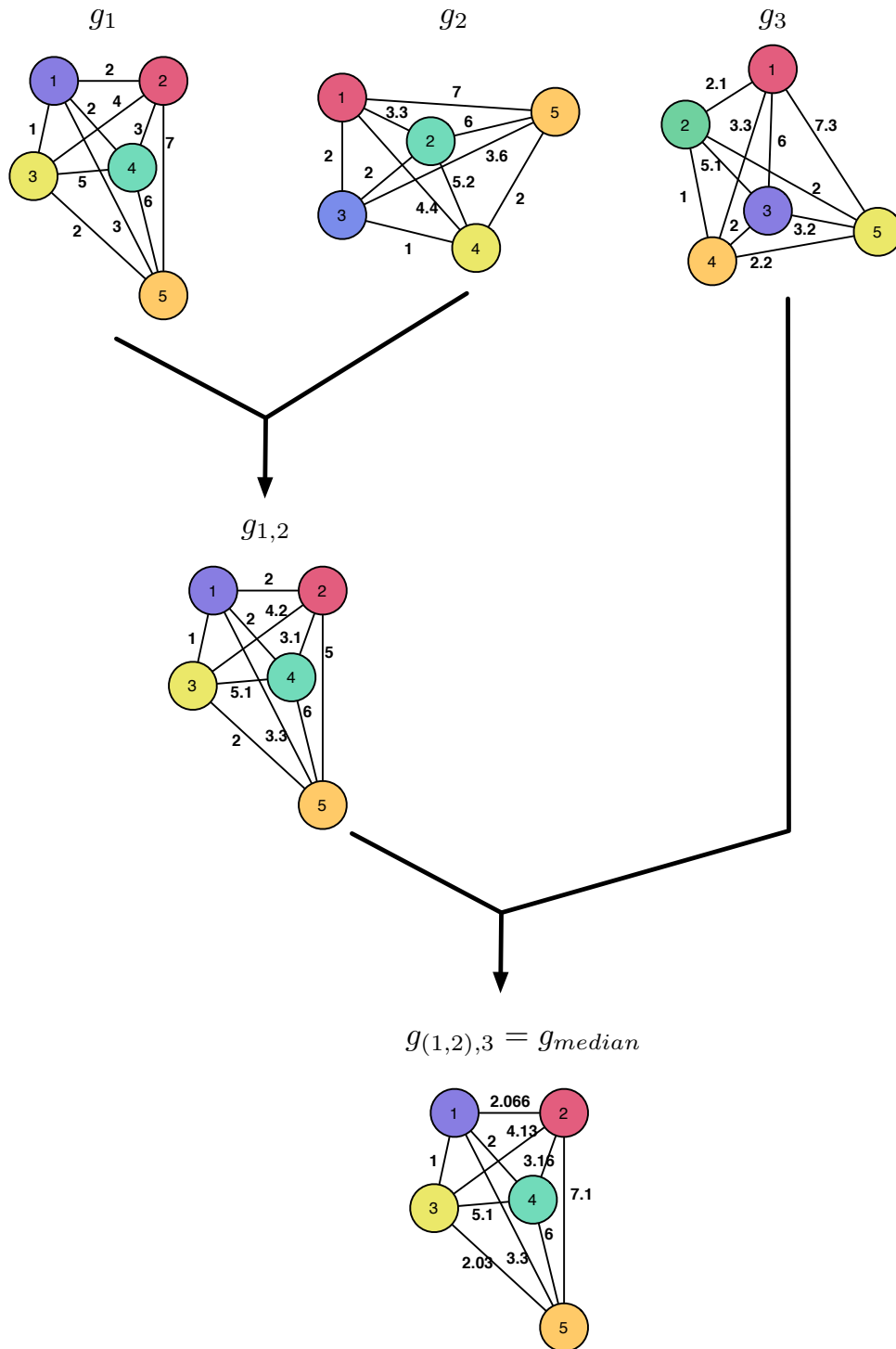


Figure 4.5: Example of the synthesis of the generalized spectral median graph.

Then, this intermediate spectral median graph $g_{1,2}$ (which is the representation of the intermediate median eigenmode $\mathbf{U}_{1,2}$) is combined with g_3 by means of their eigendecompositions $\mathbf{U}_{1,2}$ and \mathbf{U}_3 to obtain $g_{(1,2),3} = g_{median}$ ($\mathbf{U}_{(1,2),3} = \mathbf{U}_{median}$) which is the spectral median graph of g_1 , g_2 and g_3 .

We must notice the strengths and weaknesses of such algorithm. On one hand the use of an incremental method to compute the median graph, allows us to synthesize an approximation of the generalized-spectral graph with linear complexity with respect to the number of graphs. In addition, the Umeyama's method used in the matching process is able to find an approximate node correspondence between two graphs with $O(n^3)$ time complexity where n is the number of nodes of each graph. Unfortunately, the use of Umeyama's method imposes two limitations: a) the number of nodes of the graphs must be equal, and b) only one numerical attribute in the nodes and edges can be used (this is a common limitation to all spectral methods).

4.2.2 Experimental Results

In order to measure the quality of the medians we obtain with this method we conducted the following experiment using the GREC-1 database explained in Section A.2.1. Recall that in this database graphs represent graphical symbols. Graphs are distorted by moving every node within a radius r . Then, in the set of labels will most probably be as many labels as nodes in the set S . That is, we will rarely find a repeated label in the set L . This fact introduces a difficulty in the synthesis process, since the number of labels in L can be large. In addition, recall from the GREC-1 database that the nodes are labeled with a (x, y) attribute. Since the Umeyama's method can only deal with one dimensional attributes, each graph g is split into two graphs, g_x and g_y , one containing the x value and the other containing the y value. Then the distance between two graphs g_1 and g_2 , $d(g_1, g_2)$ is computed taking the mean of the x and y parts. That is, $d_x = d(g_{1x}, g_{2x})$, $d_y = d(g_{1y}, g_{2y})$ and $d(g_1, g_2) = \frac{d_x + d_y}{2}$.

The experiment consisted in computing the median graph using different number of graphs for each class and then assessing the quality of the approximation by analyzing its SOD. Since the number of graphs used in this experiment is too large to use any exact method to obtain the median graph, the results are compared with those obtained using the respective set median graph. In addition, we have performed the same operations using the genetic approach presented in [53], since it is the unique algorithm able to deal with relatively large sets of graphs. The results are shown in Figures 4.6 and 4.7 for the genetic algorithm and the spectral approach, respectively. For simplicity, in these figures we will refer to the genetic algorithm, the set median and the spectral median graph as GA, SM and SMG respectively.

It is important to note that while the spectral method could be applied to all the 32 classes existing in the GREC-1 database, the genetic algorithm was applied only using 4 classes in the database. The reason is that of the size of graphs. Within the 32 classes there are graphs with a number of nodes ranging from 4 to 28. For the genetic approach only those classes having graphs with 4 and 5 nodes could be used. The results in the figures are the mean values obtained over all these classes (4 for the genetic algorithm and 32 for the spectral median graph approach).

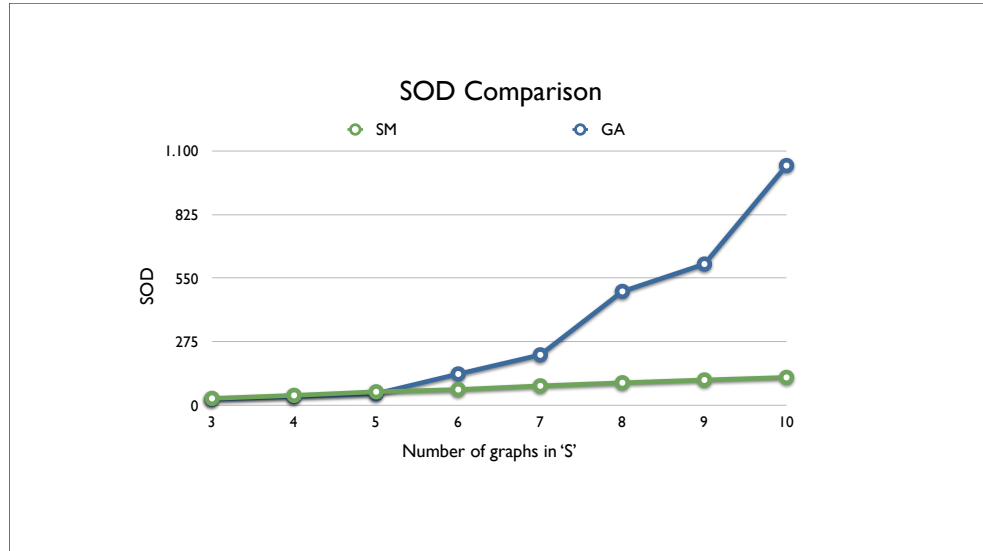


Figure 4.6: SOD comparison using the genetic approach of [53].

Results show that, in general, both methods are able to obtain good representatives (in terms of SOD) of the corresponding sets, specially for a small number of graphs in the set. In this sense, for small sets (up to 5 graphs/set), the GA approach obtains graphs with slightly better SODs than the set median graph. But its response becomes worse rapidly as the number of graphs in S increases. This can be explained by the fact that in these cases the search space also increases rapidly and it is more difficult to find a good representative with the genetic approach. On the other hand, the SM approach always obtains median graphs with SODs slightly worse than the set median graph, but the tendency is to remain very close to the set median regardless of the number of graphs used to compute the median. This suggests that our method is able to obtain good approximations even with a large number of graphs in the set.

Although the spectral median graph approach is not able to obtain better representatives than the set median graph, the method is able to work with large sets of graphs with reasonable sizes. The results shown in Figure 4.7 (SM approach) are computed with sets containing from 5 to 50 graphs with sizes from 4 to 28 nodes per graph, while the GA approach (Figure 4.6) could only be applied to sets up to 10 graphs and with sizes from 4 to 5 nodes per graph. This suggests that our method is potentially applicable to real problems. In Chapter 7 we will conduct a classification experiment using this approach and with a large amount of data.

4.3 Discussion

In this chapter we have studied the median graph using a concise class of graphs, the weighted graphs. Spectral techniques have been shown as a promising approach

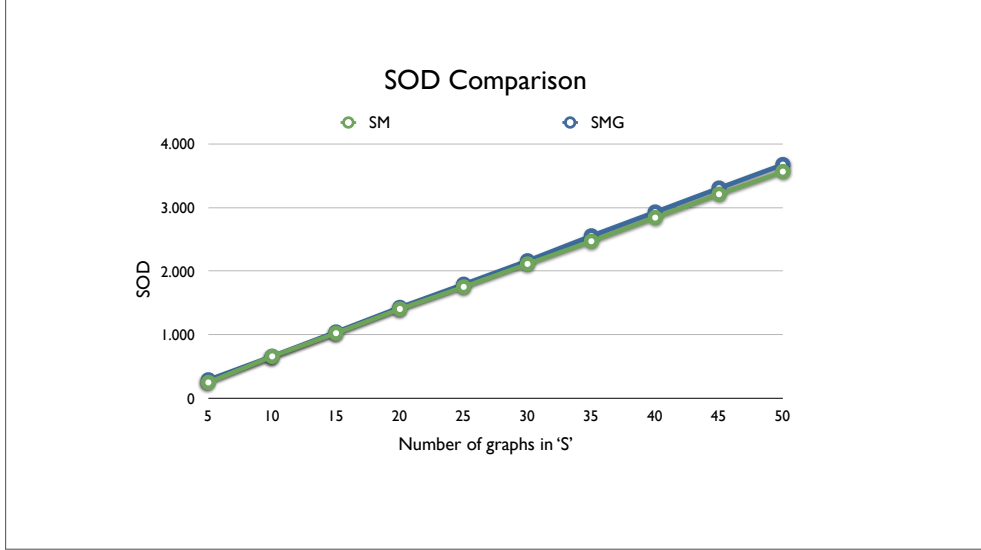


Figure 4.7: SOD comparison using the generalized median graph approach.

to solve graph matching problems with weighted graphs. For this reason, we have used this approach to compute the median of a set of weighted graphs. The main contributions of this chapter can be summarized as follows:

- **Section 4.2:** We have presented the novel theoretical concepts of *set* and *generalized spectral median graph* combining the spectral graph theory and the definition of the median graph. With these definitions, we have proposed a new way to obtain the median of a set of weighted graphs.
- **Section 4.2.1:** In order to obtain approximate medians according to these new definitions, we have provided an incremental algorithm to obtain good approximations of the *generalized median graph*. This algorithm has linear complexity with respect to the number of graphs used to compute the median.
- **Section 4.2.2:** We have conducted some preliminary experiments to measure the quality of the medians obtained with the GREC-1 database composed of 32 different classes. The results have been compared with those obtained with the genetic approach presented in [53]. Overall results show that although the genetic approach performs slightly better with very small sets, our method is able to keep a better approximation of the median graph as the size of graphs becomes large, even in big sets of graphs. This suggests that our method can be potentially exploited in real applications with large amounts of data.

Thus, with the use of the novel concept of the *spectral median graph* we can obtain good approximate solutions of the median graph in polynomial time complexity. Although this method can deal with a large number of graphs it suffers from two main

limitations. The first limitation is that the method can only be used with weighted graphs. Thus, complex attributes on nodes and edges, that are often crucial in pattern recognition applications, can not be exploited. The second major limitation is that the method can only work with graphs with the same number of nodes.

Although these limitations may seem too restrictive and therefore, make the concept of *spectral median graph* practically unusable in real situations, there exist important industrial processes where weighted graphs with the same size could appear. For instance, printed electronic circuits often include the so-called *fiducial points*, used to determine their position in a conveyor belt. The number of these fiducial points in an electronic circuit is fixed. So each electronic circuit can be represented as a graph where the fiducial points are nodes and the attributes of the nodes are the position in the conveyor belt. Thus, this kind of processes make the use of the weighted graphs and the Spectral Median Graph approach not very far from real world applications.

Chapter 5

Median Graph under a Particular Cost Function

In the definition of *median graph* given in Chapter 3, a distance $d(g, g_i)$ between a candidate median g and every graph g_i in the set S must be computed. Nevertheless, the definition of median graph is sufficiently general to avoid any assumption about this distance. That is, it potentially allows the use of any of the techniques explained in Chapter 2 to measure the similarity between two graphs. One possible choice is the graph edit distance. In the graph edit distance, the costs of the edit operations must be explicitly set through a cost function. In this chapter we will deeply study the concept of median graph under a particular cost function. Although this condition might seem quite restrictive, we will show later in this chapter that this cost function has important both theoretical and practical implications that permit to reduce the computation time of the median graph. Also this cost function has other important implications in other graph matching problems [10]. In this sense, the study of the median graph under this cost function is fully justified.

The chapter is organized as follows. In Section 5.1 we will introduce the particular cost function that will be used throughout the whole chapter. Then, in Section 5.2, we will introduce new theoretical properties about the median graph. In particular we will define new bounds on the size and the SOD of the median graph more accurate than those presented in Chapter 3. After that, in Section 5.3, we will show that using these new bounds and some implications derived from this cost function, the search space where the median is searched can be drastically reduced. This reduced search space can therefore be used to derive new algorithms for the median graph computation. This is precisely what we will present in Sections 5.4 and 5.5. In these sections, we will explain two different strategies to explore the new reduced search space. First, in Section 5.4 we will propose a new and more efficient exact algorithm to compute the median graph. This new algorithm will allow us to extend the applicability of the median graph (in a limited way) to real problems where never before the exact median computation could be applied. In addition, in Section 5.5 we will introduce a new approximate algorithm based on genetic search, giving quite good approximate solutions for the median graph in a reasonable time. With this

new algorithm we will be able to compute the median graph using real data.

5.1 A Particular Cost Function

The particular cost function that will be used throughout this chapter was firstly introduced in [10]. In this cost function, deletions and insertions of nodes have always a cost of 1, deletions and insertions of edges have always a cost of 0, and node and edge substitutions take the values of 0 or ∞ depending on whether the substitution is identical or not, respectively. Table 5.1 summarizes this cost function.

Table 5.1: Detail of the cost function

Operations on nodes		Cost
Node deletion $c_{nd}(u)$		1
Node insertion $c_{ni}(u)$		1
Node substitution $c_{ns}(u, v)$	0 if identical; ∞ otherwise	
Operations on edges		Cost
Edge deletion $c_{ed}(e)$		0
Edge insertion $c_{ei}(e)$		0
Edge substitution $c_{es}(e_1, e_2)$	0 if identical; ∞ otherwise	

Despite the simplicity of this cost function, it has important implications in different graph matching problems. The first important result about this cost function was presented in [10]. It states that, under such cost function, the edit distance $d(g_1, g_2)$ between two given graphs g_1 and g_2 is related to their maximum common subgraph $mcs(g_1, g_2)$ as expressed in Equation 2.1. To make this chapter self-contained, that equation is repeated here.

$$d(g_1, g_2) = |g_1| + |g_2| - 2|mcs(g_1, g_2)| \quad (5.1)$$

This result demonstrates the intuitive idea that the more two graphs have in common, the lower their distance. Beyond the theoretical implications of this result there are also important practical implications. For instance, an immediate consequence of this result is that any algorithm that computes the graph edit distance may be used for maximum common subgraph computation if it is run under this cost function.

More recently, the same cost function has been used in [18], to establish the relation between the maximum common subgraph and the mean of two graphs. Concretely, it is shown that, under this cost function, the maximum common subgraph of two graphs is also the mean, that is the graph that minimizes the sum of distances to these two graphs.

Finally, in [17] it is shown the relation between the maximum common subgraph and the minimum common supergraph of two graphs. In that work, it is demonstrated that under an extensive family of cost functions, including this particular cost

function, the computation of the minimum common supergraph can be done from the computation of the maximum common subgraph.

It is therefore clear that this cost function has important potential applications in different graph matching problems. In the subsequent sections we will explore its application to the theory and computation of the median graph.

5.2 New Theoretical Properties on the Median Graph

Let us recall that the theoretical properties introduced in Chapter 3, can be used to bound the search space of the median, either by limiting the size of the candidate median or by discarding some of these candidate medians according to the bounds of the SOD. Nevertheless, as mentioned in [53], these bounds are sometimes too coarse and may not be very useful to reduce the complexity of the median graph computation.

In the following, two improvements of these theoretical properties will be presented. The first one shows that the bounds on the size of the median graph can be reduced drastically. The new bounds are related to the Maximum Common Subgraph and the Minimum Common Supergraph of a set of graphs, defined in Section 5.2.1. The second contribution is related to the upper bound of the SOD of the median graph. In this case we will also reduce this bound relating it to the Maximum Common Subgraph of a set of graphs. Beyond the theoretical result, we will show later in the next section that these new bounds have important implications on the search space.

5.2.1 Maximum Common Subgraph and Minimum Common Supergraph of a Set of Graphs

The following definitions are the respective generalizations of the concepts of *Maximum Common Subgraph* and *Minimum Common Supergraph* presented in Chapter 2, but extended to a set of graphs S , instead of simply two graphs.

Definition 5.1 (*Maximum Common Subgraph of a Set of Graphs*) Let $S = \{g_1, g_2, \dots, g_n\}$ be a set of graphs. A graph $g_m(S)$ (also denoted by $mcs(S)$) is called a *maximum common subgraph* of S if $g_m(S)$ is a common subgraph of $\{g_1, g_2, \dots, g_n\}$ and there is no other common subgraph of $\{g_1, g_2, \dots, g_n\}$ with more nodes than $g_m(S)$.

Definition 5.2 (*Minimum Common Supergraph of a Set of Graphs*) Let $S = \{g_1, g_2, \dots, g_n\}$ be a set of graphs. A graph $g_M(S)$ (also denoted by $MCS(S)$) is called a *minimum common supergraph* of S if $\{g_1, g_2, \dots, g_n\}$ are subgraphs of $g_M(S)$ and there is no other common supergraph of $\{g_1, g_2, \dots, g_n\}$ with less nodes than $g_M(S)$.

In general, for a given set S , more than one $g_m(S)$ and $g_M(S)$ may exist. The computation of these graphs is a challenging problem, which usually suffers from a large complexity. For their computation, we have used a similar approach as that introduced in [15].

5.2.2 New Bounds on the Size of the Median Graph

This first result relates the size of the median graph to the Maximum Common Subgraph and the Minimum Common Supergraph of a set of graphs. In the bounds for the size of the median graph presented in Chapter 3 the lower bound was set to 0, while the upper bound was set to $\sum |g_i|$. In the following we will prove that the lower bound can be substituted by $|g_m(S)|$ (which is always greater or equal than 0) and the upper bound can be substituted by $|g_M(S)|$ (which is always less or equal than $\sum |g_i|$).

Theorem 1: Let $S = \{g_1, g_2, \dots, g_n\}$ be a set of graphs and \bar{g} a possible median graph of S . Under the cost function and the distance measure given in Section 5.1, the number of nodes of \bar{g} falls between the following limits,

$$|g_m(S)| \leq |\bar{g}| \leq |g_M(S)| \quad (5.2)$$

Proof: To demonstrate the first part of the Equation 5.2 (i.e. $|g_m(S)| \leq |\bar{g}|$), let us suppose the contrary, that is $|\bar{g}| < |g_m(S)|$ and let us analyze the relation between $SOD(\bar{g})$ and $SOD(g_m(S))$. If we compute the term $SOD(g_m(S))$, we will arrive to the next expression:

$$SOD(g_m(S)) = \sum_{i=1}^n d(g_i, g_m(S)) = \sum_{i=1}^n |g_i| + |g_m(S)| - 2|g_m(S)| = \sum_{i=1}^n |g_i| - n|g_m(S)| \quad (5.3)$$

Notice that $g_m(S)$ is the maximum common subgraph of S and, then, it is a subgraph of any graph g_i in S . Therefore, if we compute $d(g_i, g_m(S))$ using Equation 5.1 the term $|mcs(g_1, g_2)|$ is exactly $|g_m(S)|$.

For the computation of $SOD(\bar{g})$ we will follow a similar reasoning. Assuming that $|\bar{g}| < |g_m(S)|$, we can determine the minimum value for $SOD(\bar{g})$:

$$SOD(\bar{g}) = \sum_{i=1}^n d(g_i, \bar{g}) \geq \sum_{i=1}^n |g_i| + |\bar{g}| - 2|\bar{g}| = \sum_{i=1}^n |g_i| - n|\bar{g}| \quad (5.4)$$

Notice that, in this case, if $|\bar{g}| < |g_m(S)|$ then $|\bar{g}| < |g_i|$. Consequently the maximum value for $|mcs(g_1, g_2)|$ in (5.1) will be precisely $|\bar{g}|$ and the minimum value for $SOD(\bar{g})$ will be obtained when $|\bar{g}| = |mcs(g_1, g_2)|$ as expressed in Equation 5.4.

At this point, using Equations 5.3 and 5.4 and assuming that $|\bar{g}| < |g_m(S)|$ we arrive to the following conclusion:

$$SOD(\bar{g}) \geq \sum_{i=1}^n |g_i| - n|\bar{g}| > \sum_{i=1}^n |g_i| - n|g_m(S)| = SOD(g_m(S)) \quad (5.5)$$

But this is a contradiction because, by definition of the median, $SOD(\bar{g})$ must be minimum. Thus $|\bar{g}|$ must be greater or equal than $|g_m(S)|$.

Let us now proof the second part of Equation (5.2) (i.e. $|\bar{g}| \leq |g_M(S)|$). Again, let us suppose the contrary, that is $|\bar{g}| > |g_M(S)|$. In this case the term $SOD(g_M(S))$ will take this value:

$$SOD(g_M(S)) = \sum_{i=1}^n |g_i| + |g_M(S)| - 2|g_i| = n|g_M(S)| - \sum_{i=1}^n |g_i| \quad (5.6)$$

Again, Equation 5.6 holds because if $g_M(S)$ is the minimum common supergraph of S , the maximum common subgraph between g_i and g will be precisely g_i . Consequently the term $|mcs(g_1, g_2)|$ in (Equation 5.1) is exactly $|g_i|$.

Now, let us compute the minimum value of $SOD(\bar{g})$. If $|\bar{g}| > |g_M(S)|$ then every graph g_i can have at most $|g_i|$ nodes in common with \bar{g} and then the maximum value for $|g_m|$ in Expression 5.1 is $|g_i|$. Then:

$$SOD(\bar{g}) \geq \sum_{i=1}^n |g_i| + |\bar{g}| - 2|g_i| = n|\bar{g}| - \sum_{i=1}^n |g_i| \quad (5.7)$$

Then, from Equations 5.6 and 5.7, and assuming that $|\bar{g}| > |g_M(S)|$ we obtain:

$$SOD(\bar{g}) \geq n|\bar{g}| - \sum_{i=1}^n |g_i| > n|g_M(S)| - \sum_{i=1}^n |g_i| = SOD(g_M(S)) \quad (5.8)$$

Again, this is a contradiction and, thus $|\bar{g}|$ must be less or equal than $|g_M(S)|$. ■

5.2.3 Reducing the Upper Bound of the SOD of the Median Graph

The next result relates the upper bound of the SOD of the median graph to the Maximum Common Subgraph of a set of graphs. In the bounds for the SOD of the median graph given in Chapter 3 the upper bound was set to $\min\{SOD(\bar{g}_e), SOD(\bar{g}_u)\}$. In the following we will show that this limit can be substituted by $SOD(g_m(S))$ (which is always less or equal than $\min\{SOD(\bar{g}_e), SOD(\bar{g}_u)\}$).

Theorem 2: Let $S = \{g_1, g_2, \dots, g_n\}$ be a set of graphs and \bar{g} a possible median graph of S . Given the cost function and the distance measure presented in section 5.1, the upper bound for the term $SOD(\bar{g})$ is:

$$SOD(\bar{g}) \leq SOD(g_m(S)) \leq \min\{SOD(\bar{g}_e), SOD(\bar{g}_u)\} \quad (5.9)$$

Proof: First, we start by computing the term $\min\{SOD(\bar{g}_e), SOD(\bar{g}_u)\}$:

$$SOD(\bar{g}_e) = \sum_{i=1}^n d(g_i, \bar{g}_e) = \sum_{i=1}^n |g_i| + |\bar{g}_e| - 2|\bar{g}_e| = \sum_{i=1}^n |g_i|$$

Notice that, in this expression \bar{g}_e is the empty graph. Then, the mcs between any graph g_i and \bar{g}_e in expression 5.1 is \bar{g}_e , and $|\bar{g}_e| = 0$. A similar reasoning can

be done for $SOD(\bar{g}_u)$. In this case, the *mcs* between any graph g_i and \bar{g}_u is g_i , and $|\bar{g}_u| = \sum_{i=1}^n |g_i|$. Therefore,

$$SOD(\bar{g}_u) = \sum_{i=1}^n d(g_i, \bar{g}_u) = \sum_{i=1}^n |g_i| + |\bar{g}_u| - 2|g_i| = (n-1) \sum_{i=1}^n |g_i|$$

Thus, for $n \geq 2$

$$\min\{SOD(\bar{g}_e), SOD(\bar{g}_u)\} = \min\left\{\sum_{i=1}^n |g_i|, (n-1) \sum_{i=1}^n |g_i|\right\} = \sum_{i=1}^n |g_i| \quad (5.10)$$

Now we derive an expression for the term $SOD(g_m(S))$. If $g_m(S)$ is the maximum common subgraph of S , then any g_i will have precisely $g_m(S)$ as the maximum common subgraph between itself and $g_m(S)$. Therefore,

$$SOD(g_m(S)) = \sum_{i=1}^n d(g_i, g_m(S)) = \sum_{i=1}^n |g_i| + |g_m(S)| - 2|g_m(S)| = \sum_{i=1}^n |g_i| - n|g_m(S)| \quad (5.11)$$

Thus, from Expressions 5.10 and 5.11 we can easily see that $SOD(g_m(S)) \leq \min\{SOD(\bar{g}_e), SOD(\bar{g}_u)\}$. In addition, by the definition of median graph, the inequality $SOD(\bar{g}) \leq SOD(g_m(S))$ must be satisfied. Consequently, Equation 5.9 holds. ■

5.3 New Search Space

The new theoretical bounds on the size of the median graph introduced in the previous section establish that the candidate medians must have a size greater or equal than $|g_m(S)|$ and less or equal than $|g_M(S)|$. Thus, the search space for the median graph will be composed of graphs in between these sizes. In addition, as we will show in the next section, the nature of the cost function presented in Section 5.1 will imply that not all the graphs within these limits will need to be taken into account as candidate medians. The new bounds together with this implication of the cost function will serve us to present the new search space later in Section 5.3.2.

5.3.1 Influence of the Cost Function on the Search Space

As we have seen in Section 5.1, this particular cost function has both theoretical and practical important implications. But, beyond these results, in [17] another important consequence of this cost function has been introduced. It states that, under this cost function, there always exists an optimal edit path between two given graphs that never implies neither non-identical node substitutions nor non-identical edge substitutions. That is, in the best path to transform g_1 into g_2 there are always only node and edge identical substitutions, deletions and insertions. This can be understood by the fact that a non-identical substitution (which implies an infinity

cost) can be always simulated by a deletion followed by an insertion (which has a finite cost). Consequently these non-identical operations will never be applied.

This result can be seen in the next example. Suppose the graphs g_1 and g_2 of Figure 5.1(a) and 5.1(b) are given. Recall that in the definition of graph given in Section 2.1, it is assumed that graphs are fully connected. That is, there is always an edge linking any two nodes of the graph. If an edge does not exist between two nodes it is modeled using the "null" label, and it is depicted using dashed lines in the figure. In this situation, one could suppose that the cheapest way to convert g_1 into g_2 is by deleting the edge linking the white and black nodes in g_1 . Thus, in fact, deleting this edge is equivalent to substitute it by an edge with the "null" label. But this operation is not allowed because it would imply a non-identical edge substitution. Then, a possible alternative sequence of edit operations (see Figure 5.1(d)) consists of deleting the white node from g_1 (including the deletion of its adjacent edges) and inserting the same node in g_2 together with their incident edges. This sequence of edit operations on g_1 and g_2 , has a cost equal to 2, one node deletion and one node insertion (notice that edge deletions and insertions have a zero cost). The same result will be obtained applying Equation 5.1. For instance, in Figure 5.1(c) we can see a possible mcs between g_1 and g_2 . We can easily verify that applying this equation we obtain the same distance value as applying the edit operations.

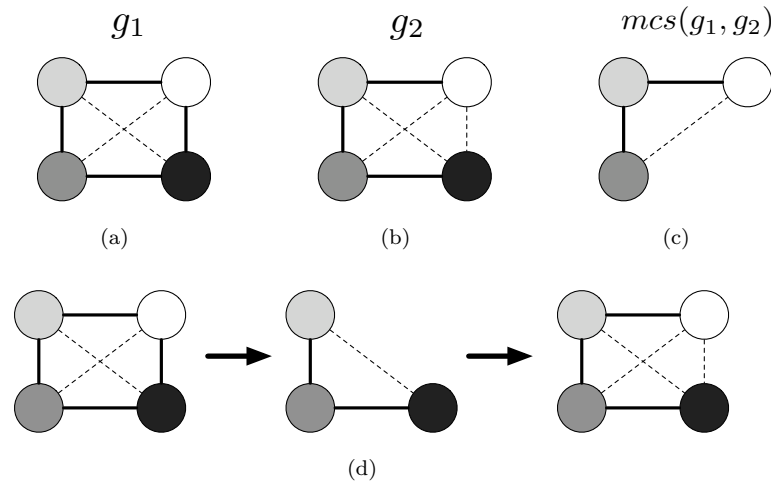


Figure 5.1: Two graphs g_1 (a) and g_2 (b), a possible $mcs(g_1, g_2)$ (c) and a possible edit path between g_1 and g_2 (d).

5.3.2 The New Search Space

Let us now introduce the new search space. From the upper bound of the size of the median graph derived in Section 5.2, we will suppose that the largest candidate median is precisely $g_M(S)$. We will assume that for a given set of graphs any of the possible $g_M(S)$ are equivalent. From this first candidate median and, with the

implications of the cost function explained in Section 5.3.1, where only node and edge deletions, insertions and identical substitutions are allowed, we can say that only the induced subgraphs of $g_M(S)$ are valid candidate medians. In fact, it is easy to see that any induced subgraph of $g_M(S)$ will imply only node deletions and insertions along the graphs in S . On the contrary, a non-induced subgraph of $g_M(S)$ will imply not only node deletions and insertions but also a non-identical edge substitution at some point, since there will be at least one graph in S that will need this operation. This non-identical operation will imply an extra node insertion and deletion (as we have seen in the example of Section 5.3.1) with respect to an induced subgraph of $g_M(S)$ with the same set of nodes. In other words, if we generate a non-induced subgraph of $g_M(S)$ as a possible candidate median, it will generate at some point, a sequence of edit operations with a cost larger than those generated by the induced subgraph of $g_M(S)$ with the same nodes.

Let us further analyze this search space. This new search space can be seen as a rhombus (see Figure 5.2). At the top of the rhombus (level 0) there is $g_M(S) = MCS(S)$. Let p be $|g_M(S)|$. Immediately below, at level 1, there are all the possible induced subgraphs of $g_M(S)$ with size $p-1$. Concretely there are $C_{p-1}^p = p$ graphs (level 1). At the next level, all of the induced subgraphs of $g_M(S)$ with $p-2$ nodes are generated (C_{p-2}^p graphs). As new levels are generated, the number of combinations will increase until the central row of the rhombus is reached. At this point we have the maximum number of combinations ($C_{p/2}^p$ for p even and $C_{(p+1)/2}^p$ for p odd). From this point to the bottom, the number of combinations decreases until the last row (level p) is reached. At the last level, there is only one combination ($C_0^p = 1$), corresponding to the empty graph g_e .

In this search space, the total number of possible candidate medians can be easily computed. For a given i , with $0 \leq i \leq p$, the number of possible candidate median graphs with size i , is $C_i^p = \frac{p!}{i!(p-i)!}$. That is, all the possible combinations of i nodes from a number of p nodes. As the search space is composed of all the induced subgraphs of $g_M(S)$ between the sizes 0 and p , the total number of possible different candidates corresponds to the sum $\sum_{i=0}^p C_i^p$, which is exactly 2^p .

Let us now integrate in this result the lower bound on the size of the median graph. Thus, in the search space illustrated in the Figure 5.2, we can drop the part corresponding to all the graphs with size lower than $|g_m(S)|$. These graphs would correspond to the white part of the rhombus in Figure 5.3, that shows the final search space.

Thus, the grey part in Figure 5.3 is the part of the rhombus search space that is still necessary to explore in the search of the median graph. The total number of possible candidate medians is defined by,

$$\sum_{i=|g_m(S)|}^p C_i^p \quad (5.12)$$

which is always less or equal than 2^p .

Now, the question is how to explore this new search space. The rest of this chapter is devoted to present two different strategies for that. The first strategy is

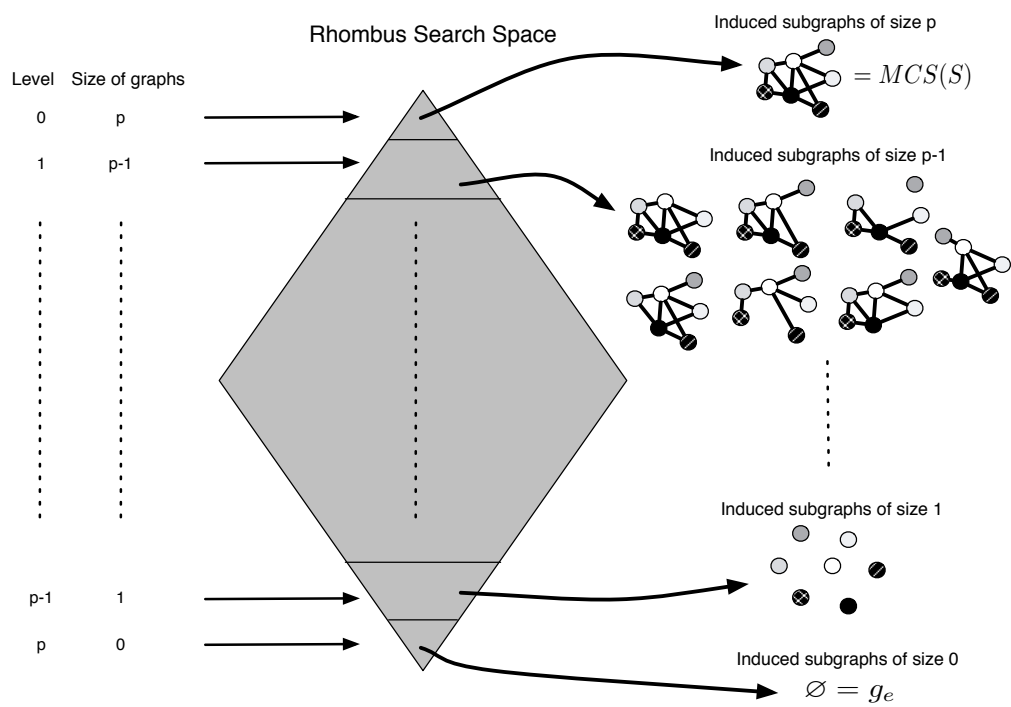


Figure 5.2: Detail of the rhombus search space.

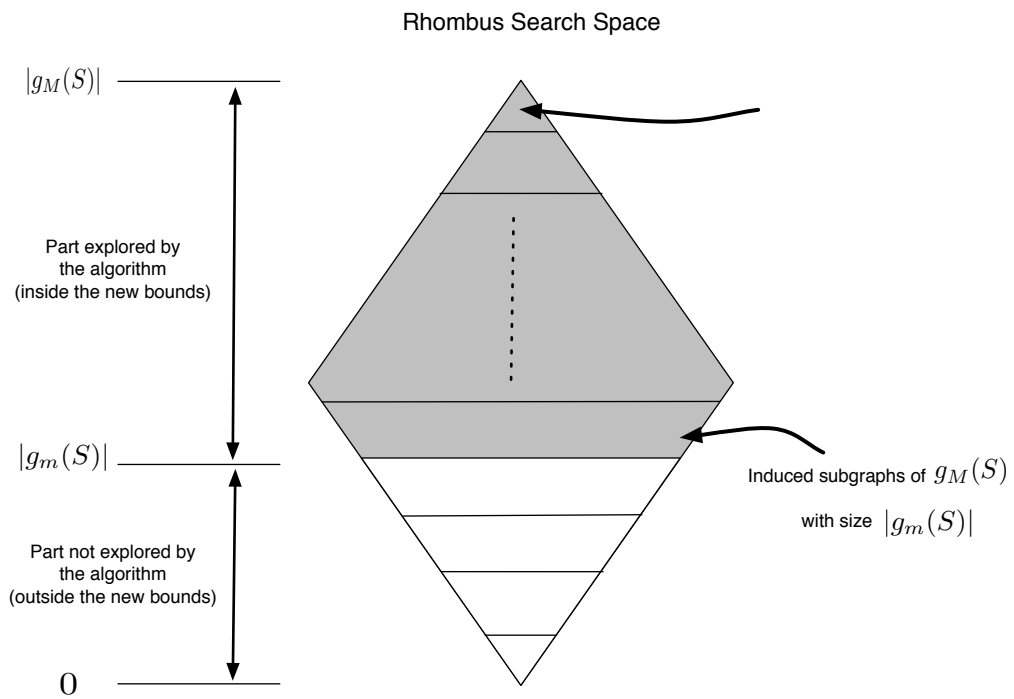


Figure 5.3: Reduction in the search space due to the new bounds.

an exhaustive search that will lead to present a new exact algorithm. The second approach is carried out by a new genetic algorithm for the approximate median graph computation.

5.4 New Exact Algorithm

In this section we will present a new exact algorithm for the median graph computation. This new algorithm will take advantage of all the results obtained previously to carry out an exhaustive search, but only on the reduced search space (grey zone in Figure 5.3). Nevertheless, despite the important reduction of the search space, its exhaustive exploration may still suffer from a large complexity. Thus, it would be still desirable to discard as many combinations to explore as possible. To this end, in Section 5.4.1, we will present an heuristic prediction function that will permit to avoid the evaluation of some states in the new search space. The new algorithm will be described in Section 5.4.2. Finally, we provide an experimental result both on synthetic and real data where the improvements in the performance of this new algorithm with respect to the Multimatch algorithm will be shown.

5.4.1 Prediction of the SOD

In this section we will explain how we can reduce even more the number of evaluations in the search space. We will show that, from a given candidate median in the search space, we can introduce a prediction of the cost associated to the candidate medians generated at the next level of the search space. Thus, evaluating this prediction function we will be able to decide whether it is worth to take into account the candidate medians at the next level.

Let us start by defining f^* as the evaluation function of a candidate median, \bar{g}^* . This function is just the sum of distances of the candidate median to all the graphs g_i in the set S . Then, taking the definition of the graph distance (Equation 5.1), we can express $f^*(\bar{g}^*)$, in the following way:

$$\begin{aligned} f^*(\bar{g}^*) &= SOD(\bar{g}^*) = \sum_{i=1}^n d(g_i, \bar{g}^*) = \sum_{i=1}^n (|g_i| + |\bar{g}^*| - 2|mcs(g_i, \bar{g}^*)|) \\ &= n|\bar{g}^*| + \sum_{i=1}^n |g_i| - 2 \sum_{i=1}^n |mcs(g_i, \bar{g}^*)| \end{aligned} \quad (5.13)$$

The complexity of this expression depends on the cost of computing the mcs , which, in the general case, is exponential in the size of the involved graphs. It is therefore desirable to avoid as many evaluations of this function as possible. To this end, we can try to infer the value of this function as we traverse the search space, without having to compute the mcs between the candidate median and all the graphs. Let us take two candidate median graphs, \bar{g}_1 and \bar{g}_2 . As we will be traversing the search space by generating all the induced subgraphs of $g_M(S)$, let us suppose too that \bar{g}_2 is an induced subgraph of \bar{g}_1 . Now, let us try to find some relation between $f^*(\bar{g}_1)$ and

$f^*(\bar{g}_2)$. Let us denote by $h^*(\bar{g}_1, \bar{g}_2)$ the function that computes the difference between both evaluation functions,

$$\begin{aligned} h^*(\bar{g}_1, \bar{g}_2) &= f^*(\bar{g}_2) - f^*(\bar{g}_1) = \\ &= n(|\bar{g}_2| - |\bar{g}_1|) - 2 \sum_{i=1}^n |mcs(g_i, \bar{g}_2)| + 2 \sum_{i=1}^n |mcs(g_i, \bar{g}_1)| \end{aligned} \quad (5.14)$$

We will call $h^*(\bar{g}_1, \bar{g}_2)$ the *prediction* function as we can express the evaluation function of any graph \bar{g}_2 in the search space in terms of the evaluation function of a graph \bar{g}_1 and the prediction $h^*(\bar{g}_1, \bar{g}_2)$.

Now, let us analyze this prediction function. If we are traversing the search space from graph \bar{g}_1 to graph \bar{g}_2 and we have already evaluated $f^*(\bar{g}_1)$, we know the value of all the terms in $h^*(\bar{g}_1, \bar{g}_2)$ except for $|mcs(g_i, \bar{g}_2)|$. However, we can infer an upper limit for this term and therefore, we can define an estimation of the prediction function, $h(\bar{g}_1, \bar{g}_2)$.

Let us observe that the *mcs* between two graphs will never have more nodes than any of them. In addition, if we are traversing the search space from \bar{g}_1 to \bar{g}_2 , then \bar{g}_2 will be a subgraph of \bar{g}_1 and $|mcs(g_i, \bar{g}_2)| \leq |mcs(g_i, \bar{g}_1)|$. Therefore, the following condition holds:

$$|mcs(g_i, \bar{g}_2)| \leq \min(|g_i|, |\bar{g}_2|, |mcs(g_i, \bar{g}_1)|) \quad (5.15)$$

Then, combining Equations 5.14 and 5.15 we can obtain the following estimation of the prediction function:

$$\begin{aligned} h^*(\bar{g}_1, \bar{g}_2) &\geq h(\bar{g}_1, \bar{g}_2) = \\ &= n(|\bar{g}_2| - |\bar{g}_1|) - 2 \sum_{i=1}^n \min(|g_i|, |\bar{g}_2|, |mcs(g_i, \bar{g}_1)|) + 2 \sum_{i=1}^n |mcs(g_i, \bar{g}_1)| \end{aligned} \quad (5.16)$$

Finally, as we know the value of all the terms involved in this equation, $h(\bar{g}_1, \bar{g}_2)$ can be used to obtain an estimation of the evaluation function of node \bar{g}_2 , $f(\bar{g}_2)$:

$$f(\bar{g}_2) = f^*(\bar{g}_1) + h(\bar{g}_1, \bar{g}_2) \quad (5.17)$$

Using this estimation and the upper bound for the SOD derived in Section 5.2, we can reduce the computation of the median graph by reducing the number of candidate graphs whose evaluation function needs to be explicitly computed. Given a candidate median graph, \bar{g}^* , we can compute this estimation for all their induced subgraphs. All graphs \bar{g}_j whose estimation $f(\bar{g}_j)$ is either greater than the actual evaluation function of the current median graph \bar{g}^* , $f^*(\bar{g}^*)$, or greater than the term $SOD(g_m(S))$ (the new upper limit derived in Section 5.2) can be automatically discarded and do not have to be evaluated.

One difficulty to apply this strategy is that, given a graph in the search space, we cannot guarantee that the prediction function is either positive or negative for all its

induced subgraphs. Therefore, we cannot discard all its remaining induced subgraphs in the search space. We can only discard nodes at the next level of the rhombus. This strategy will be used in the algorithm presented in the next section.

5.4.2 The New Exact Algorithm

Keeping in mind all the reductions of the search space proposed so far including those that are consequence of the heuristic function explained in the previous section, we are able to present a new and more efficient exact algorithm to compute the generalized median graph. For the sake of completeness the algorithm is described as **Algorithm 2**.

Algorithm 2 Exact-median Algorithm

Require: A set of graphs $S = \{g_1, g_2, \dots, g_n\}$

Ensure: A list L of true median graphs

```

1:  $g_M(S) = \text{Compute\_MCS}(S)$ 
2:  $g_m(S) = \text{Compute\_mcs}(S)$ 
3:  $\text{ComputeSOD}(g_M(S), S)$ 
4:  $\text{ComputeSOD}(g_m(S), S)$ 
5: Let  $\min\{\text{SOD}(g_m(S)), \text{SOD}(g_M(S))\}$  be the minimum SOD
6: if  $\text{SOD}(g_m(S)) \leq \text{SOD}(g_M(S))$  then
7:   Insert pair  $(g_m(S), \text{SOD}(g_m(S)))$  in L
8: else
9:   Insert pair  $(g_M(S), \text{SOD}(g_M(S)))$  in L
10: end if
11: Expand( $g_M(S)$ )
12: for size =  $|g_M(S)| - 1$  to  $|g_m(S)|$  do
13:   while  $\text{RemainInducedSubGraph}(\text{size})$  do
14:      $g = \text{GetNextInducedSubgraph}()$ 
15:     Expand( $g$ )
16:     if IsValid( $g$ ) then
17:        $\text{ComputeSOD}(g, S)$ 
18:       if  $\text{SOD}(g)$  is less than the minimum SOD then
19:         Let  $\text{SOD}(g)$  be the minimum SOD
20:         Delete L
21:         Insert pair  $(g, \text{SOD}(g))$  in L
22:       else
23:         if  $\text{SOD}(g)$  is equal to the minimum SOD then
24:           Insert pair  $(g, \text{SOD}(g))$  in L
25:         end if
26:       end if
27:     end if
28:   end while
29: end for
30: Return L

```

The algorithm receives a set S of graphs as input and returns a list of valid median graphs. The first task is to compute both the $g_m(S)$ and $g_M(S)$. Then, the algorithm computes the terms $SOD(g_M(S))$ and $SOD(g_m(S))$. This is accomplished by the function *ComputeSOD*, where the first parameter is a graph and the second parameter is a set of graphs. The distance between two graphs $d(g_1, g_2)$ is computed using the Equation 5.1. The graph (along $g_M(S)$ and $g_m(S)$) with minimum SOD is inserted in the candidate median list, and its SOD is set as the minimum SOD. After that, all the possible induced subgraphs of $g_M(S)$ are generated using the function *Expand*, and they are marked as valid or not by using both the heuristic function explained in the previous section and the upper bound for the SOD derived in Section 5.2. In order to avoid the generation of repeated combinations of nodes we keep a hash table at each level of the search space that permits to know if a given combination has already been generated. Then, all these graphs are visited using the functions *RemainInducedSubGraph* and *GetNextInducedSubGraph*. If the graph is valid, then its SOD is computed and compared with the current best SOD. Otherwise, the term SOD is not computed and the next graph in the level is processed in the same way. Anyway, even if the graph is not valid, all its induced subgraphs are generated and marked as valid or not valid. The process is repeated until the level corresponding to the graphs with size equal to $|g_m(S)|$ is reached.

5.4.3 Experimental Setup

In this section we will provide the results of an experimental evaluation of this algorithm in order to validate whether we can improve the existing algorithms. We have separated such experimental setup in two different parts. Firstly, we will compare our new algorithm with the Multimatch algorithm. Since the Multimatch algorithm has a very limited applicability, this comparison will be done using the synthetic Letter-1 database described in Section A.1.1. In a second set of experiments we will compare it against the genetic algorithm presented in [53] using the Molecule dataset described in Section A.3¹.

Scenario 1. Application to Synthetic Data

In this section, the Multimatch algorithm and our new exact algorithm for the median graph computation will be compared. For simplicity they will be referred as *MM* and *ER* respectively. The experiments consisted of the computation of several generalized median graphs using different number of graphs. For each median, the elapsed time and the number of SOD computations needed to compute it were recorded. In these experiments, the Letter-1 graph dataset (see Section A.1.1 for a detailed explanation of this database) will be used. We defined 24 sets of graphs. For each of the 6 classes of the database, we formed 4 different sets composed of 2, 3, 4 and 5 graphs, respectively. Table 5.2 shows, for each letter (first column), the number of nodes of the graph (in brackets) and, for each of the 4 sets, the total sum of nodes of the set.

Notice that, because of computation time, not all the possible combinations could

¹All these experiments were run on an Apple iMac computer equipped with an Intel Core 2 Duo processor and 2Gb of main memory.

Table 5.2: Sum of nodes in the set S function of the class and the number of nodes in S .

Letters (<i>Nodes/graph</i>)	N° graphs in S			
	2	3	4	5
L,V (3)	6 •	9 •	12 •	15
N,T (4)	8 •	12 •	16	20
K,M (5)	10 •	15	20	25

- *Combinations used in the Multimatch algorithm*

be used to compute the generalized median graph using the Multimatch algorithm. The combinations where the Multimatch algorithm could be applied are marked with the symbol • in Table 5.2. In contrast, our new exact algorithm could be applied to all the possible combinations.

Experiment 1. Computation Time and Number of SOD Computations

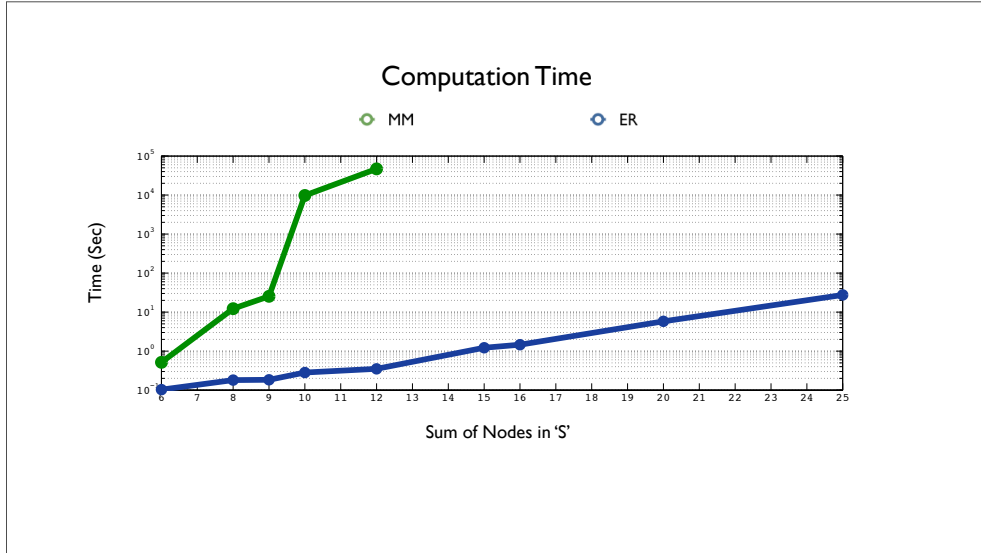
The results of the computation time and the number of SOD computations required to calculate the median graph as a function of the total number of nodes in S are shown in Figures 5.4(a) and 5.4(b) respectively. Note that the results are the mean values obtained for all the sets S with the same number of nodes.

The first important conclusion is that, due to the combinatorial explosion of the Multimatch algorithm it could be only applied to sets of graphs whose sum of nodes is up to 12. Beyond this limit, the time required for this algorithm is unfeasible. This result is consistent with the results presented in [53]. In contrast, our new exact algorithm could be applied obtaining reasonable computation times to sets having up to 25 nodes. In addition, the computation time required by our algorithm is quite lower than the time required by the Multimatch algorithm, even for small sum of nodes in S . Such difference in time is more evident when the sum of nodes in S becomes larger. All these facts can be appreciated in the results showed in Figure 5.4(a).

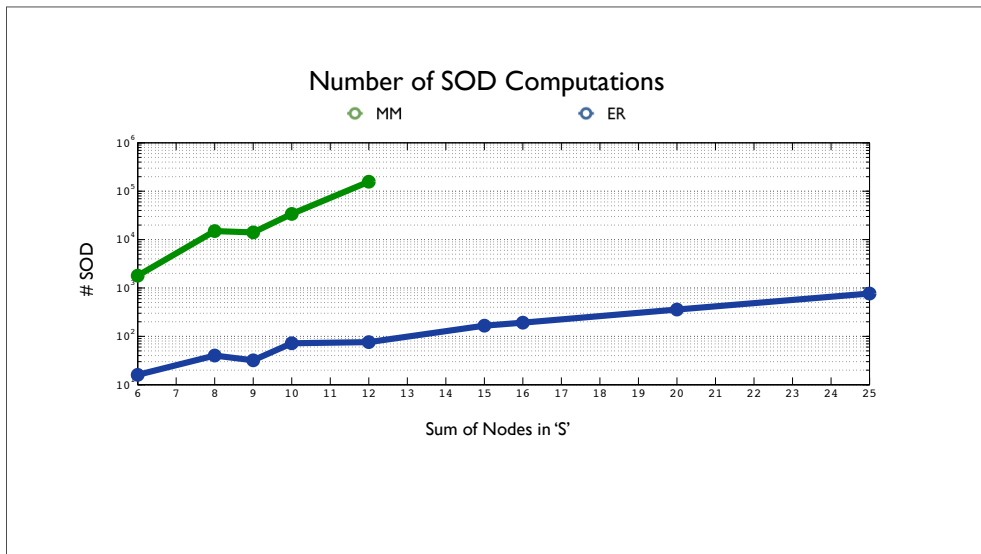
A similar behavior can be deduced from the number of SOD computations needed for the two algorithms (see Figure 5.4(b)). Again, a significant difference between the number of SOD computations required by our algorithm can be seen in the results. This reduction of the number of SOD computations can be associated to both the reduction of the search space and the heuristic function we presented in Sections 5.3.2 and 5.4.1 respectively.

Experiment 2. Reduction in Computation Time and Number of SOD Computations

In order to be able to quantify the gain achieved by our algorithm with respect to the Multimatch algorithm, we present in Table 5.3 the time and number of SOD computations required by both algorithms to compute the generalized median graphs.



(a)



(b)

Figure 5.4: Computation time (a) and number of SOD computations (b) as a function of the total number of nodes of the graphs in S .

The results for the computation time are shown in columns 2 and 3. The next column (labelled as *%Reduction*) represents the reduction of time (in percentage) of our algorithm taking the value of the Multimatch algorithm as reference. The same reasoning respect the number of SOD computations can be done for the rest of the table.

Table 5.3: Percentage of time and SOD computations required for the new exact algorithms with respect to the Multimatch algorithm.

$\sum g_i $	Time (sec)			SOD computations		
	<i>MM</i>	<i>ER</i>	<i>%Reduction</i>	<i>MM</i>	<i>ER</i>	<i>%Reduction</i>
6	0.51	0.103	79.81	1,794	16	99.11
8	12.25	0.181	98.53	2,943	32	98.92
9	25.40	0.184	99.28	14,092	40	99.72
10	9712	0.283	99.98	33,961	72	99.79
12	46,794.85	0.352	99.99	157,176	76	99.95

A significant reduction in the time needed for the median graph computation is achieved by our new algorithm with respect to the Multimatch algorithm, as shown in Table 5.3. This reduction increases as the sum of nodes in the set also increases. Notice that the mean of the time percentage needed for our algorithm with respect to the Multimatch algorithm is about 4.5% (or a 95.5% of reduction). A similar reasoning can be done for the number of SOD computations. In this case the mean percentage of the required SOD computations is about 0.5% (99.5% of reduction).

Scenario 2. Application to Real Data

In the previous section, it has been shown that the computation of the median graph by means of our new algorithm is not restricted to a small number of graphs with a small number of nodes as in the case of the Multimatch algorithm. In this section we will go a step further and we will extend the exact computation of the median graph to real data. In particular we will use our algorithm to obtain prototypes of certain molecules, using the molecule database described in Section A.3. As the Multimatch algorithm cannot be applied here due to the size and the number of graphs, we will compare the results with those obtained by the genetic algorithm presented in [53] (referred as GA in the experiments).

Two different classes (Active and Inactive compounds) conform the database. For each class we have 50 different instances or molecules. The experiments consisted in generating, for each class, 40 sets with a different number of graphs. The graphs in each set had different sizes and were chosen randomly from the original set of 50 instances. In this case, for each set, the time required and the SOD of the median graph obtained by each algorithm were recorded. It is important to notice that the sum of nodes in the sets ranged from 4 to 23 nodes. The results are shown in the next sections.

Experiment 1. Computation Time and Number of SOD Computations

The computation time required for both active and inactive compounds is shown in Figures 5.5(a) and 5.5(b) respectively. In both charts, the *x-axis* represents the size of the minimum common supergraph of the set, independently of the sum of nodes in the set.

First of all it is important to notice that for minimum common supergraphs of sizes up to 10-11, the time required by our algorithm is lower than the time required by the genetic algorithm. Such difference is specially relevant for small sizes of the minimum common supergraph (up to 8), and also for the case of inactive molecules, where the time required by our exact algorithm is two orders of magnitudes less than the genetic algorithm. It is important to mention that, for sizes of the minimum common supergraph up to 11 nodes, we can find sets whose sum of nodes was up to 20 nodes. This conclusion suggests that for sets of graphs that share large structures (and consequently the minimum common supergraph tends to the mean size of the graphs in the set), our algorithm may outperform the GA algorithm.

As our algorithm is an optimal one, the computation time for larger sizes of the minimum common supergraph increases rapidly. Nevertheless, the computation time in these cases does not make the application of our algorithm completely unfeasible (the sum of nodes of the sets with minimum common supergraph of 14 nodes were 23 nodes).

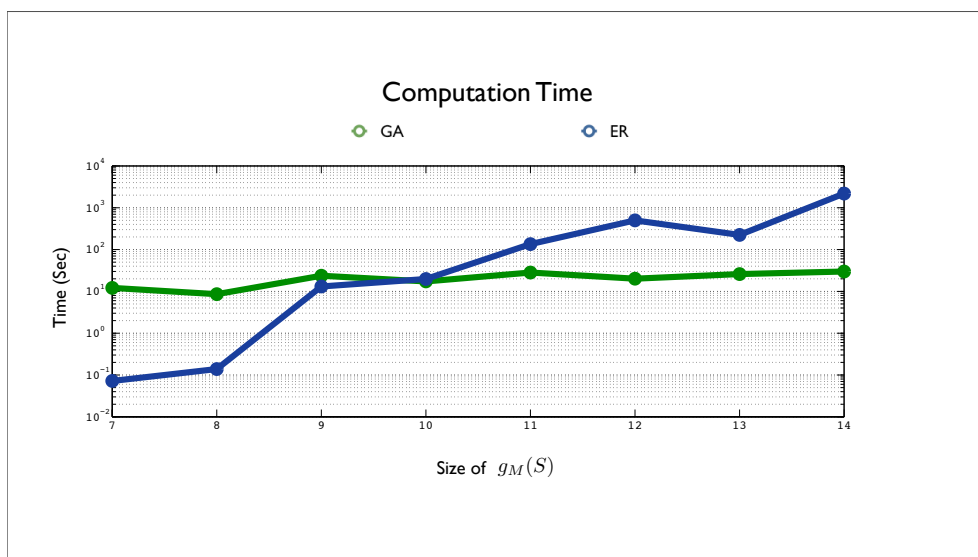
Experiment 2. SOD Comparison

The definition of the median graph implies the computation of the sum of distances of the candidate median to all the graphs in the set. In this sense, a measure of how good the median graph is, can be obtained by computing the term SOD. A measure of the SOD for the medians computed with both algorithms is shown in Figures 5.6(a) and 5.6(b), for the active and inactive compounds respectively. Again, the *x-axis* represents the size of the minimum common supergraph of the set, independently of the sum of nodes in the set.

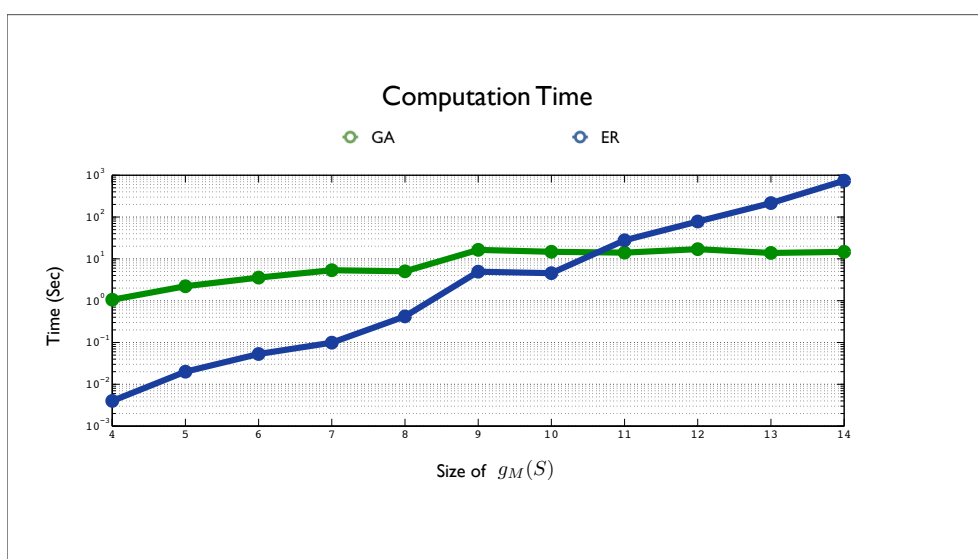
As expected, the SOD for the genetic algorithm is always greater than or equal to the SOD achieved by the exact algorithm. This behavior is more clear in the case of inactive compounds, where for sizes of the minimum common supergraph greater than 9, the difference increases significantly (Figure 5.6(b)). The difference is less evident in the case of active compounds.

Such difference in the SOD suggests that the medians obtained by the genetic algorithm, although they are quite accurate, tend to diverge as the size of the minimum common supergraph increases (the graphs in the set are more dissimilar). This may lead to obtain median graphs that do not represent accurately the set of graphs. In contrast, the exact algorithm always finds an optimal median graph. Thus it always finds the best representative in the set (following the criteria established in the definition of the median graph).

Combining the results of Figures 5.5 and 5.6 we can observe that there is a trade-off between the computation time of the algorithms and the accuracy in the SOD. That is, from a certain point on, the GA algorithm outperforms the ER approach in computation time. Nevertheless, Figure 5.6 shows that the GA approach diverges

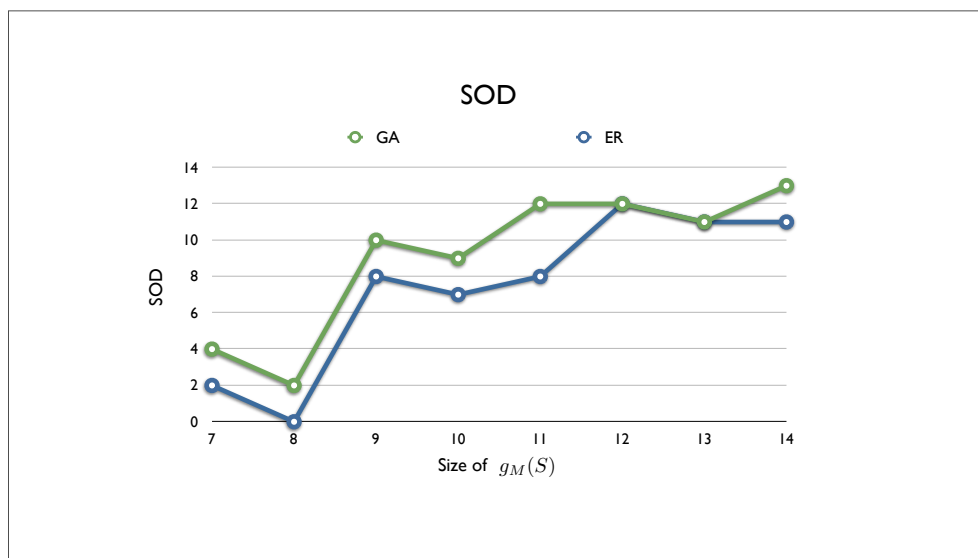


(a)

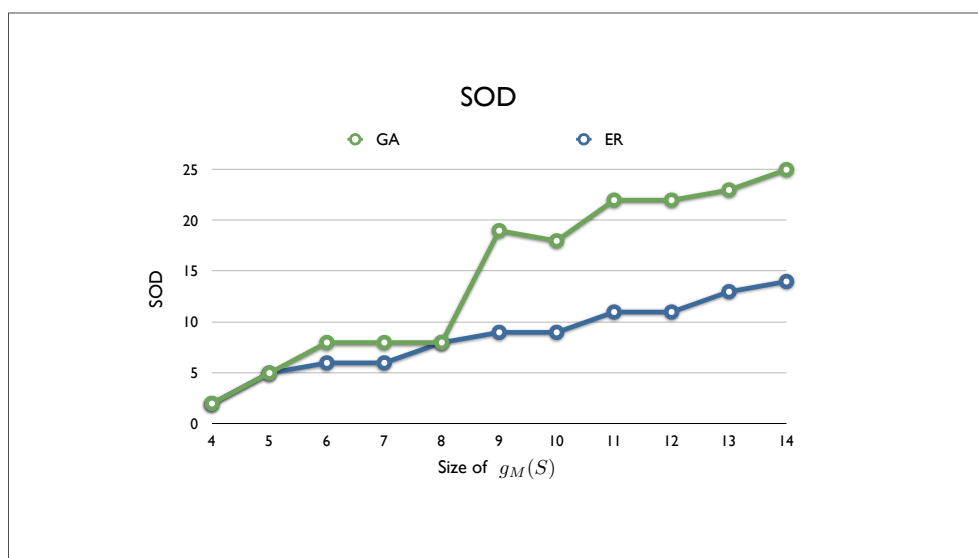


(b)

Figure 5.5: Computation time for Active (a) and Inactive (b) compounds as a function of the size of $MCS(S)$.



(a)



(b)

Figure 5.6: SOD of computed median for Active (a) and Inactive (b) compounds as a function of the size of $MCS(S)$.

from the exact solution when the size of the $g_M(S)$ increases. Thus, there exists a trade-off between these two factors. Depending on the application, it could be desirable to spend a bit more of time but obtaining more accurate solutions or on the contrary, to obtain approximate solutions with a fast response. Thus, although still limited, the exact algorithm may be useful in some scenarios.

5.5 New Genetic Algorithm

In the last section, we have presented an algorithm to explore the search space in an exhaustive way. The results have shown that this approach outperforms the previous existing exact algorithm in terms of time and number of SOD computations needed to find the true median. Nevertheless, the size of the search space may still be too large to perform an exhaustive search on it. In this section, we propose a genetic approach to explore the search space. In this way, the search space is not explored exhaustively but in a pseudo-random way. Although the genetic approaches are not deterministic we will show we are able to still obtain good approximations of the median graph on relatively large sets of graphs with a reasonable time.

5.5.1 Basics on Genetic Search

Genetic search techniques are general-purpose optimization methods inspired in the theory of the biological evolution. They have been successfully applied to difficult search tasks, optimization problems, machine learning, etc. It has also been shown that they are good candidates to give good approximate solutions to general NP-complete problems [54]. They have been applied to solve graph matching problems [3, 30, 111] and to compute approximate solutions for the generalized median graph [53].

The basics of genetic algorithms are as follows. A possible solution of the problem is encoded using chromosomes. Each chromosome has a cost. Such a cost is computed by means of a fitness function. Given an initial population of chromosomes, genetic algorithms use genetic operators to alter chromosomes in the population, generating a new population. The genetic operators are typically the crossover and mutation. In the former, a pair of chromosomes of the current population are randomly chosen and some of their positions are interchanged. The latter takes only one chromosome and alter some of its positions randomly. The algorithm tends to favor the most promising chromosomes (in terms of the fitness function). Thus, these promising chromosomes will have higher probabilities to be in the next generation. The process is iteratively repeated until one or more stop conditions are satisfied. For more information about genetic algorithms the reader is referred to [73].

5.5.2 Our Approach

Chromosome Representation

The results presented in Section 5.2 have shown that the candidate medians are the induced subgraphs of $g_M(S)$ with sizes between $|g_M(S)|$ and $|g_m(S)|$. Then, the

chromosome representation should be able to encode one of these induced subgraphs of $g_M(S)$. Thus, we have chosen the size of the chromosome equal to the size of the $g_M(S)$. Each position in the chromosome is associated to one node of $g_M(S)$, and may store either a value of "1" or a value of "0" depending on whether that node appears or not in the candidate median. By the definition of induced subgraph given in Section 2.1, a subset of nodes of a given graph uniquely defines a subgraph. In order to clarify such representation an example is given in Figure 5.7.

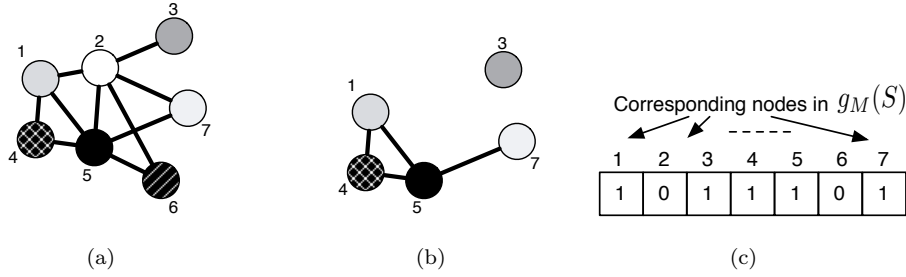


Figure 5.7: A supposed $g_M(S)$ (a), an induced subgraph g of $g_M(S)$ (b) and the chromosome representing g (c).

Assume that $g_M(S)$ is the graph shown in Figure 5.7(a). As we can see, a number is assigned to each node of $g_M(S)$. A possible induced subgraph of $g_M(S)$ is shown in Figure 5.7(b), which is composed only of the nodes 1, 3, 4, 5 and 7. Then, the chromosome representation of such an induced subgraph is shown in the figure 5.7(c). In such chromosome the total number of positions is equal to the number of nodes of $g_M(S)$. Notice that the chromosome has only set to "1" the positions of the nodes which are also in the induced subgraph.

Fitness Function

The fitness function of each chromosome corresponds to the SOD of the induced subgraph of $g_M(S)$ encoded by the chromosome. That is, if the chromosome c represents a graph g , then its fitness function $f(c)$ is:

$$f(c) = SOD(g, S) = \sum_{i=1}^n d(g, g_i) = \sum_{i=1}^n (|g| - |g_i| - 2|mcs(g, g_i)|) \quad (5.18)$$

Clearly, the lower its fitness function is, the better the chromosome is. The computational complexity of this fitness function is related to the computational complexity of the maximum common subgraph of two graphs, which is NP-complete in the general case. Nevertheless, such computational complexity becomes polynomial under some special classes of graphs.

Genetic Operators

We apply the classical operators of the genetic algorithms adapted to this particular case in order to include the new bounds presented in Section 5.2. In our algorithm, the roulette wheel sampling implementing fitness-proportionate selection is chosen to create the descendants (also called offspring). Conceptually, it is equivalent to give a slice of a circular roulette wheel to each chromosome, proportional in area to the fitness of the chromosome. The crossover operator simply interchanges an arbitrary position of two chromosomes (selected with a uniform probability) to form two offsprings. Mutation is accomplished by changing randomly a number in the array with a mutation probability. After the genetic operators have been applied and a new population is created, every chromosome is checked in order to validate whether it fulfils the bounds of the size and the upper bound of the SOD given in Section 5.2 (that is, if the number of ones in the chromosome is in between $|g_m(S)|$ and $|g_M(S)|$, and if its SOD is less than $SOD(g_m(S))$). If the generated chromosome does not hold these conditions, it is randomly altered until it fulfils them. This procedure has two effects. On the one hand it reduces the search space from all the possible induced subgraphs of $g_M(S)$ to only the induced subgraphs that fulfil the conditions given in Section 5.2. On the other hand, as the search space is reduced and the non-admissible candidate medians will never appear in the population, the convergence of the algorithm is expected to be faster compared to the same algorithm without taking into account the new limits.

Population Initialization

The length of the initial population is set according to a predefined value K , determined empirically. Then, the first n chromosomes (with $K \geq n$) are set with the n graphs in S . It assures that the initial population includes the set median graph, which is a potential generalized median graph. The remaining $K-n$ chromosomes are generated randomly but all of them fulfil the new bounds given in Section 5.2.

Parameters of the Genetic Algorithm

Table 5.4 shows the basic configuration parameters for the genetic algorithm.

Table 5.4: Configuration parameters for the genetic algorithm.

Parameter	Value
<i>Mutation probability</i>	0.1
<i>Crossover probability</i>	0.9
<i>Initial population size (K)</i>	20
<i>Maximum number of iterations</i>	400

Termination Condition

The evolution of the population continues until one of the two following conditions is fulfilled. The first criterion is that the maximum number of generations (which is set according to a predefined constant at the beginning of the algorithm) is reached. The second stop condition is related to the best *SOD* in the population. If a chromosome in the population has a *SOD* less than the *SOD* of the set median graph, then the algorithm finishes too.

5.5.3 Experimental Setup

In this section we provide the results of an experimental evaluation of the proposed algorithm. To this end, the dataset composed of graph-based representations of web-pages explained in Section A.4 is used. Such graphs have a large number of nodes (around 200) but they are a particular class of graphs with unique node labels. Such kind of graphs allow the computation of the maximum common subgraph of two graphs in polynomial time [57]. This condition makes the computation of the edit distance based on the maximum common subgraph (and for extension, the computation of the median graph) applicable to large graphs. Due to the large size of the graphs, all other methods for the median graph computation are not applicable. In this sense, in the next experiments we will give only some measures about the quality of the median graph with respect to the set median graph.

As shown in Table A.9, the class with the smallest number of graphs is the class T, with 60 graphs. For this reason, in the next experiments we will use this number of graphs for each class. The 60 graphs of each class are chosen randomly except for the class T obviously.

The experiments consisted in the computation of the median graph using different number of graphs (from 3 to 7) using the genetic approach presented before. Since this approach is non-deterministic, the experiments were repeated 10 times and the best values were taken. Then we compared the *SOD* of the medians obtained using this method with the *SOD* of the set median. For simplicity they will be referred to as *AG* (for the genetic algorithm) and *SM* (for the set median).

Experiment 1. Median Evaluation

Tables 5.5 and 5.6 show some interesting results of the median graph computation. In both tables, the first row represents the sum of nodes of the graphs used to compute the median, the second row depicts the number of iterations needed to achieve a graph with a *SOD* better than the set median graph and the third row shows the computation time. While in the first table the results are grouped by class, in the second table the results are shown as a function of the number of graphs used to compute the median. In both cases, the results are the mean values over each class or over each number of graphs respectively.

The results show that the number of populations needed to find out a median better than the set median graph is very low (less than 100 in all cases). It means that the genetic algorithm always finds a graph with a *SOD* better than the *SOD* of the set median graph. Of course it does not imply that the algorithm finds the true

Table 5.5: Statistics for median graph grouped by class.

	Class					
	B	E	H	P	S	T
$\sum g_i $	1,021.2	777	845	940.4	806.2	566.2
<i># iterations</i>	66.40	8.40	2.40	11.80	32.20	3.20
<i>Computation time (sec)</i>	4,636	274.1	65.057	179.6	1,428.748	75.8

Table 5.6: Statistics for median graph grouped by the number of graphs used to compute the median.

	Number of graphs in S				
	3	4	5	6	7
$\sum g_i $	467.1	701.1	813.3	1,041.1	1,107.1
<i># iterations</i>	1.5	58.50	2.83	21.6	19.1
<i>Computation time (sec)</i>	13.8	3,362.4	82.3	1,278.2	2,159.6

median, but it finds a median better than the set median.

It is also important to remark that we have been able to apply the median computation to real problems, as it is demonstrated with the results of the first and third rows. As we can see in both tables, the sum of nodes of the graphs ranged from 400 to 1,000, while the computation times ranged from 13 to 4,600 seconds. Such numbers show the application of the median graph computation to real data with reasonable computation times. It is important to notice that previously existing methods could not be applied in this case due to their high computational requirements.

Experiment 2. SOD Comparison

The results shown above are suitable to quantitatively evaluate the algorithm. That is, they give an idea of the power of the new algorithm in terms of computation time, number of iterations needed to compute the median and the size of the set S . Nevertheless, it is also of interest to qualitatively evaluate the medians obtained by the algorithm. One choice could be to compare the SOD with the SOD of the median graph obtained with other methods. However, in this case, due to the size and the number of graphs it is not possible to perform such an experiment, since the existing methods cannot deal with such large sets. Another approach is to compare the SOD of the approximate median with the SOD of the set median. In this way, we can evaluate whether the median is better than the set median and consequently it is potentially a good median.

Figures 5.8 and 5.9 show the results of this comparison as a function of the classes in the dataset, and the number of graphs in the set S respectively.

The results of Figure 5.8 show that we obtain medians with a SOD lower than the set median SOD for all the classes. It suggests the method is able to obtain good

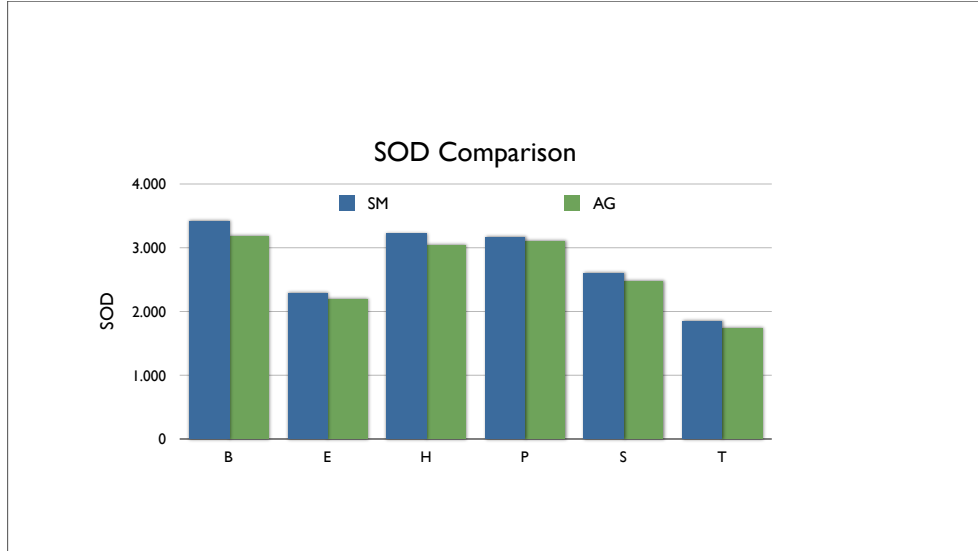


Figure 5.8: SOD comparison for the Set Median (SM) and the Approximate Genetic (AG) algorithm, function for the different classes.

approximations of the median graph regardless of the class.

Figure 5.9 shows that we obtain a better SOD with our method than with the set median. What is important in this result is the tendency in the difference between the set median SOD and the SOD of the approximate median. This difference increases as the number of graphs in S increases. This tendency suggests that the more information of the class the method has (more elements in S), better representations is able to obtain.

With these results we can conclude that we obtain good approximations of the median graph with this new genetic approach.

5.6 Discussion

In this chapter we have studied the median graph under a particular cost function. The main contributions of this chapter can be summarized as follows:

- **Section 5.2:** The main contribution from a theoretical point of view is that we have shown that using this particular cost function and the distance measure based on the maximum common subgraph, the original bounds for the median graph related to its size and its sum of distances can be reduced. Such reductions can be used either to obtain a better knowledge of the median graph or can be used to present more efficient and accurate algorithms. This is precisely, the contribution of the next section.
- **Section 5.3:** As a first point and from a theoretical point of view, we have shown that using this particular cost function, the distance measure based on

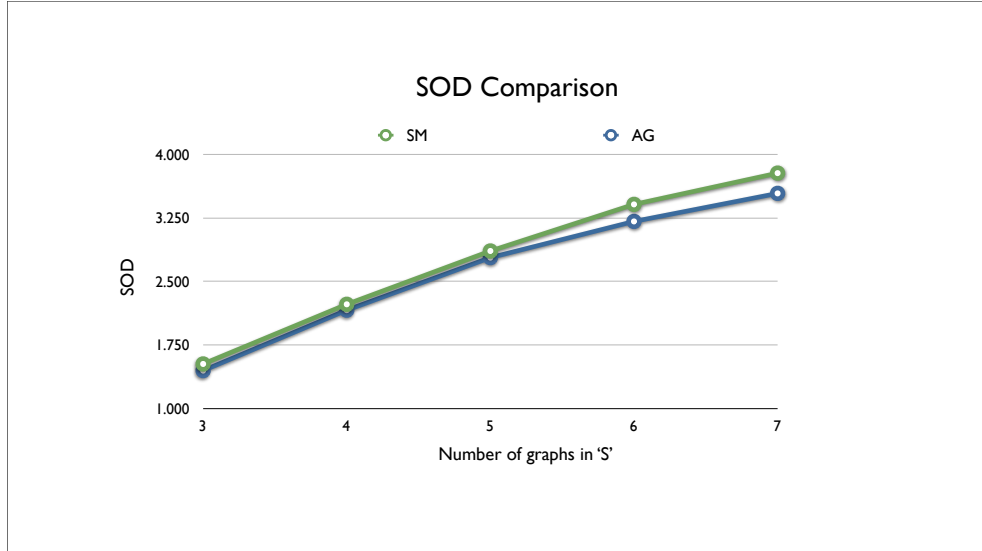


Figure 5.9: SOD comparison function of the number of graphs in S .

the maximum common subgraph and the new theoretical properties introduced in Section 5.2, the size of the search space for the median can be drastically reduced.

- **Section 5.4:** Based on the new theoretical results, a new exact algorithm for the computation of the generalized median graph has been presented. In addition to that, an heuristic strategy has also been introduced in order to avoid the evaluation of some states in the search space. We have compared this new exact algorithm to the previous existing exact algorithm using synthetic data. The results show that our algorithm clearly outperforms the previous existing algorithm, in terms of the computation time and the number of SOD computations needed to obtain the final solution. Encouraged by these results, we have applied our algorithm to the computation of median graphs using real data, and we have compared the results with those obtained by an approximate algorithm based on the genetic search. The results show that, although the application of the exact algorithm is limited, it can be used in real problems.
- **Section 5.5:** Using the same theoretical results, we have presented in this section a new approximate algorithm for the median graph computation based on genetic search. With this new algorithm we have shown that we are able to extend the applicability of the median graph to a real application. In particular, we have used this algorithm using a database composed of webpages extracted from real data. A particularity of these graphs is that they have unique node labels, which allow to compute the *mcs* in polynomial time with respect to the size of graphs. Despite this condition, since the graphs in this database are very large, and all the existing algorithms are not able to work with this kind

of graphs, we have compared the results with those of the set median graph. Values of SOD show that we obtain better approximations of the median than with the set median graph.

Chapter 6

Median Graph by Means of Graph Embedding in Vector Spaces

In the introductory chapter of this thesis we have compared feature vectors and graphs in terms of both their representational power and the easiness to implement mathematical operations such as sums, means, etc., over them. We have seen that graphs offer a better way to represent structured objects, whereas with feature vectors, a lot of interesting mathematical operations can be easily implemented.

In this chapter we will take advantage of the strengths of both worlds to present a new technique to compute both exact and approximate solutions for the median graph. In this way we can keep the representational power of graphs while being able to operate in a vector space. The basic idea underlying such technique is composed of three main steps. The first step is to embed the graphs into a vector space, that is, each graph becomes a point in a particular vector space. After that, once in the vector domain, we compute the median vector of this set of vectors, which is easier than computing the median in the graph domain. The median vector is then interpreted as the point in the vector domain corresponding to the median graph. The last step is going back to the graph domain, obtaining the corresponding graph from the median vector. This graph is taken as the median graph. With this new embedding approach we are able to compute both exact and good approximate solutions for the median graph.

This chapter is organized as follows. In the next section we will present an overview of the different approaches for graph embedding. This introduction will serve in Section 6.2 to introduce the new approach for the median computation. After that, in Section 6.3, we particularize this technique for the cost function used in Chapter 5, and we give a new method for the exact median computation. In Section 6.4, we extend this work to potentially any cost function and we propose a new method to obtain approximate solutions for the median graph. Experiments on these methods show that the proposed technique outperforms the previous existing methods both in computation time and median quality. Finally, this chapter ends with a discussion of these methods and their potential applications.

6.1 Graph Embedding

Graph embedding aims to convert graphs into another structure, such as real vectors, and then operate in the associated space to make easier some typical graph-based tasks, such as matching and clustering. To this end, different graph embedding procedures have been proposed in the literature so far. Some of them are based on the spectral graph theory. Others take advantage of typical similarity measures to perform the embedding tasks. In the following, a brief review of some strategies for graph embedding will be outlined.

A relatively early approach based on the adjacency matrix of a graph is proposed in [69]. In this work, graphs are converted into a vector representation using some spectral features extracted from the adjacency matrix of a graph. Then, these vectors are embedded into eigenspaces with the use of the eigenvectors of the covariance matrix of the vectors. This approach is then used to perform graph clustering experiments. Another similar approach has been presented in [115]. This work is similar to the previous one, but in this case they use the coefficients of some symmetric polynomials constructed from the spectral features of the Laplacian matrix, to represent the graphs into a vectorial form. Finally, in a recent approach [85], the idea is to embed the nodes of a graph into a metric space and view the graph edge set as geodesics between pairs of points in a Riemannian manifold. Then, the problem of matching the nodes of a pair of graphs is viewed as the alignment of the embedded point sets.

In this thesis we will use a new class of graph embedding procedures based on the selection of some prototypes and graph edit distance computation. This approach was first presented in [83], and it is based on the work proposed in [79]. The basic intuition of this work is that the description of the regularities in observations of classes and objects is the basis to perform pattern classification. Thus, from the selection of concrete prototypes, each point is embedded into a vector space by taking its distance to all these prototypes. Assuming these prototypes have been chosen appropriately, each class will form a compact zone in the vector space. An extension to map string representations into vector spaces using a similar approach was later proposed in [102].

In the Section 6.2.1, we will explain in detail the approach proposed in [83], since it will form the basis to construct our method.

6.2 Median Graph via Graph Embedding in Vector Spaces: An Overview

In this section we present an overview of the proposed embedding technique for the median graph computation. This general embedding procedure will be applied to both exact and approximate median graph computation, and is composed of three main steps, assuming we have a set of n graphs $S = \{g_1, g_2, \dots, g_n\}$ for the median graph computation.

- **Graph Embedding in a Vector Space:** Each graph in the set S is embedded into an n -dimensional vector space. That is, each graph becomes a point in

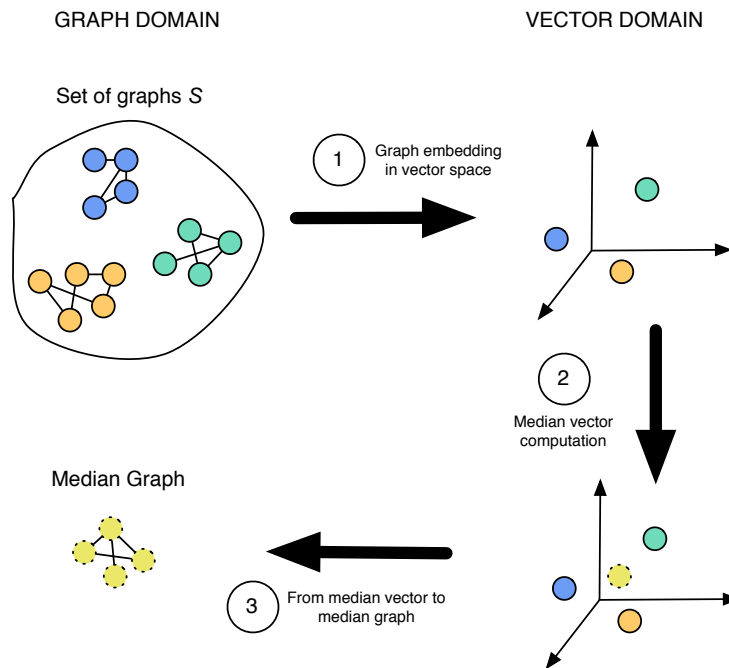


Figure 6.1: Overview of the general embedding procedure for median graph computation.

an n -dimensional space. The vector representation of a graph is obtained by computing the distance between all the graphs in the set.

- **Median Vector Computation:** This step consists in computing the median vector using the points obtained in the first step. The median vector can be obtained using different approaches. Anyway, the task of computing the median in a vector space is easier than computing it in the graph domain.
- **Going Back to the Graph Domain:** The last step consists in going back to the graph domain converting the median vector into a graph. This graph is taken as the median graph of the set.

For clarity these three main steps of the general embedding procedure are depicted in Figure 6.1.

Using this general approach we will present in the following sections two new methods for both the exact and approximate median computation. We will particularize the two last steps in a different way for each method. Nevertheless the first step is almost identical in both methods, with the unique difference in the way the similarity between two graphs is computed. For this reason, this first step is further explained in the following, and then, for each method we will only explain the small

differences between them concerning the distance computation.

6.2.1 Graph Embedding in a Vector Space

As we have shown in Section 6.1, several techniques for embedding graphs into vector spaces have been proposed. We will use the novel and very promising procedure proposed in [83]. For the sake of completeness, we briefly describe this approach in the following lines.

Assume we have a set of training graphs $T = \{g_1, g_2, \dots, g_n\}$ and a graph similarity measure $d(g_i, g_j)$ ($i, j = 1 \dots n; g_i, g_j \in T$). Then, a set $P = \{p_1, \dots, p_m\} \subseteq T$ of m prototypes is selected from T (with $m \leq n$). After that, the similarity between a given graph $g \in T$ and every prototype $p \in P$ is computed. This leads to m dissimilarity values, d_1, \dots, d_m where $d_k = d(g, p_k)$. These dissimilarities can be arranged in a vector form (d_1, \dots, d_m) . In this way, we can transform any graph of the training set T into an m -dimensional vector using the prototype set P . More formally this embedding procedure can be defined as follows:

Definition 6.1 (Graph Embedding in Vector Spaces) *Given a set of training graphs $T = \{g_1, g_2, \dots, g_n\}$ and a set of prototypes $P = \{p_1, \dots, p_m\} \subseteq T$, the embedding:*

$$\psi : T \longrightarrow \mathbb{R}^n \quad (6.1)$$

is defined as the function:

$$\psi(g) \longrightarrow (d(g, p_1), d(g, p_2), \dots, d(g, p_m)) \quad (6.2)$$

where $g \in T$, and $d(g, p_i)$ is a graph dissimilarity measure.

We perform the graph embedding step according to this definition, but we let the training set T and the prototype set P be the same, i.e, the set S for which the median graph is to be computed. So, we compute the distance between every pair of graphs in the set S . These distances are arranged in a distance matrix. Each row (column) of the matrix can be seen as an n -dimensional vector. Since each row (column) of the distance matrix is assigned to one graph, such an n -dimensional vector is the vectorial representation of the corresponding graph. Figure 6.2 illustrates this procedure. In this example, it is assumed that the first row in the matrix corresponds to the distances of the blue graph in the set to all the graphs in S . This first row is interpreted as an n -dimensional point in a vector space (this n -dimensional space is represented here as a 3D space for obvious reasons).

At the end, each graph in S has a corresponding point (n -dimensional vector) in the vector space. What is important to remark here is the meaning of each position in this vector. If a vector \vec{v}_i corresponds to the graph $g_i \in S$, then the coordinate j (with $j = 1 \dots n$) of this vector is the distance from the graph g_i to the graph g_j , that is $d(g_i, g_j)$. This fact will be used later to obtain the median graph.

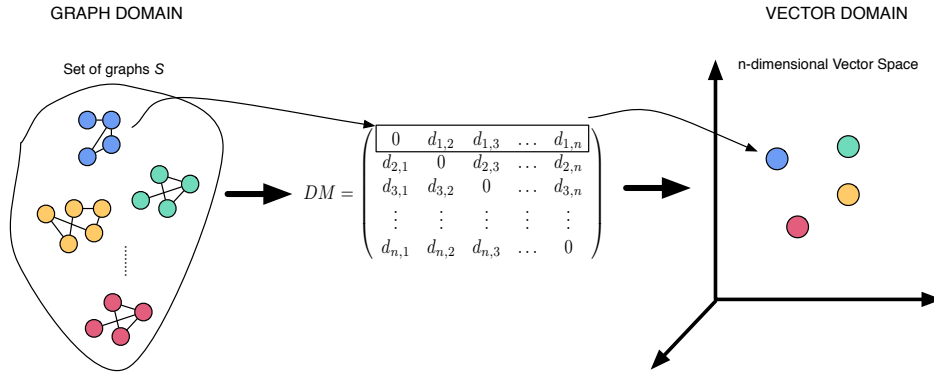


Figure 6.2: Detail of the first step (Graph embedding).

6.3 Exact Median Graph via Embedding

In this section, we will use the embedding technique presented in Section 6.1 to obtain exact solutions for the median graph. The embedding technique is applied in this case assuming that the same cost function introduced in Chapter 5 is used. Thus we will work on the rhombus search space explained in the same chapter. For the sake of completeness the rhombus search space is shown in Figure 6.3, and briefly reminded here. At the top of the rhombus there is the minimum common supergraph of S , $MCS(S)$. At the bottom, there is the empty graph $g_e = \phi$. Inside the rhombus there are all the possible induced subgraphs of $MCS(S)$, but recall that we have only to take into account those induced subgraphs of $g_M(S)$ with a size greater or equal than $|g_m(S)|$ (the grey part in Figure 6.3). Then, each graph in S is located somewhere in this search space, as it is shown by means of the dashed lines in Figure 6.3.

6.3.1 Graph Embedding in Vector Spaces

The graph embedding in a vector space is performed using the procedure explained in Section 6.2.1. In this case the distance between graphs is computed according to the Equation 5.1 [10].

Due to the nature of the cost function we can assume that the vector space where the graphs are embedded follows the properties of a L_1 geometry (also called taxicab geometry). This can be explained by the fact that under this cost function, the distance between two graphs will always be a multiple of the cost of the node insertion (or deletion), since we only apply node insertions (or deletions) to transform one graph into the other. The L_1 geometry operates in similar terms, the distance between two points is always conformed by discrete steps. Thus, in this geometry, the usual Euclidean metric is replaced by the L_1 metric in which the distance between two points is the sum of the (absolute) differences of their coordinates. The fact that we are in a L_1 geometry will be crucial in the third step, when the median vector will be

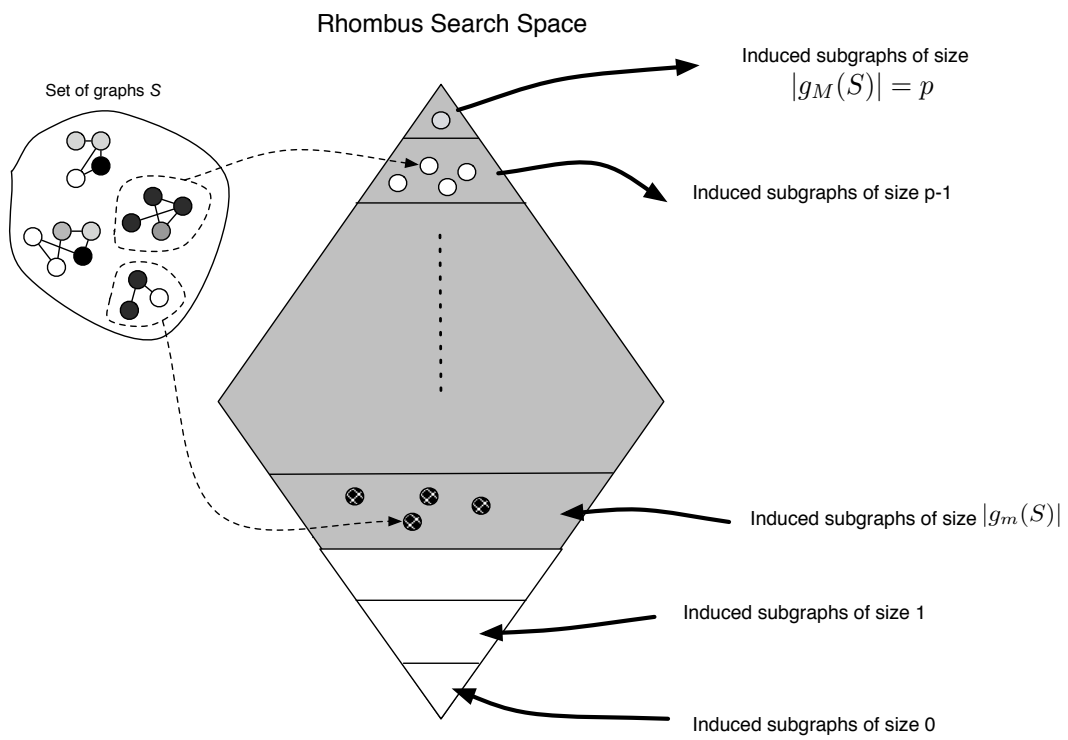


Figure 6.3: Review of rhombus search space.

used to obtain the median graph.

6.3.2 Median Vector Computation

Once all the graphs have been embedded in the vector space, and every graph has its corresponding point in this vector space, then the median vector of all these points is computed. As we are in a L_1 geometry the median vector can be computed by means of the *Manhattan median*, defined as follows:

Definition 6.2 (Manhattan Median) Given a set $X = \{x_1, x_2, \dots, x_m\}$ of m points with $x_i \in \mathbb{R}^n$ for $i = 1 \dots m$, the *Manhattan median* is defined as

$$\text{Manhattan median} = \arg \min_{y \in \mathbb{R}^n} \sum_{i=1}^m |x_i - y| \quad (6.3)$$

where $|x_i - y|$ denotes the Manhattan distance between the points $x_i, y \in \mathbb{R}^n$.

From this definition and the properties of the L_1 geometry, it follows that the *Manhattan median* is simply the median of each coordinate of the vector. That is, we can compute separately, the median of each coordinate without taking into account the other coordinates. This property makes the *Manhattan median* computation really simple.

6.3.3 Back to the Graph Domain

The median vector is used to go back to the graph domain and obtain the median graph. Note that using the embedding procedure explained before (Section 6.2.1), every component of the vector associated to a particular graph corresponds to the distance between this graph and another graph in the set S . The median vector can be interpreted in a similar way, that is, every component represents the distance between the median graph and one of the graphs in the set S (see Figure 6.4). In this example, the median graph would have a distance of 2 to the graph $g_1 \in S$, a distance of 4 to the graph $g_2 \in S$, and so on.

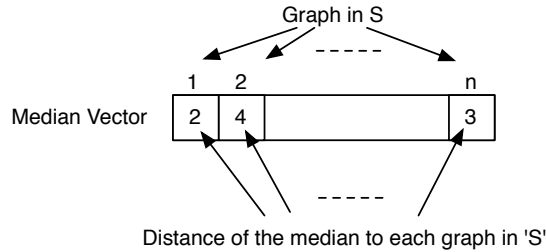


Figure 6.4: Example of a median vector and its interpretation.

Knowing the distance from the median to one graph in S , we can draw an interval in the search space around this graph according to the distance. Recall that in the distance we are using only insertions and deletions of nodes will appear in the edit path transforming one graph into another. An insertion or deletion of a node has a cost of 1. Thus, a distance x from the median to the graph g_i will imply that we would have to take into account graphs around g_i with size $|g_i| \pm x$, since at most the distance x will add (or remove) x nodes to g_i . Figure 6.5 shows an example of a simple situation where the set S is composed of 4 graphs. For clarity, in this example only the interval for g_2 is shown.

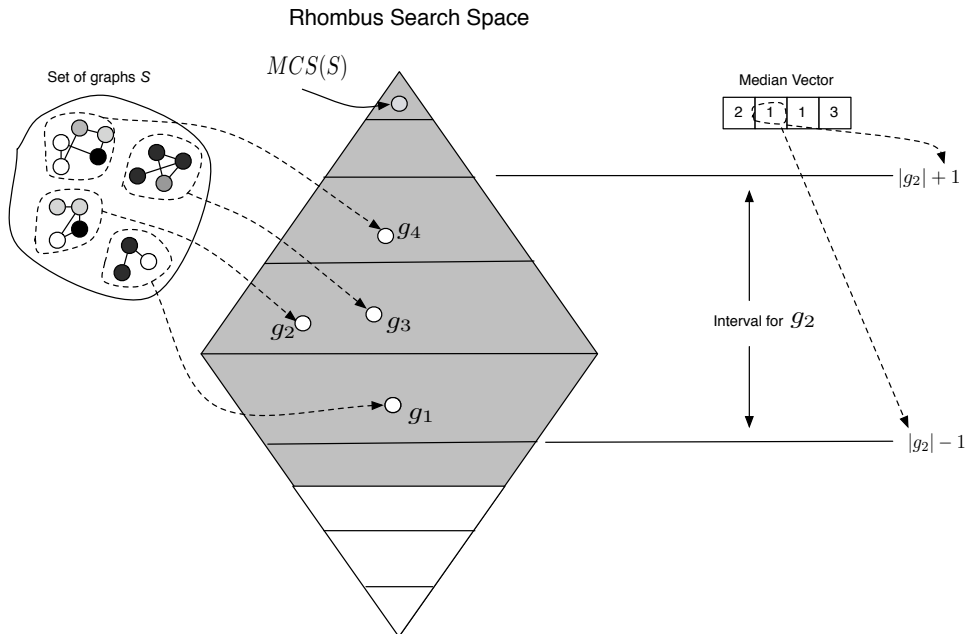


Figure 6.5: Interpretation of the median vector.

Once the intervals are set, a straightforward approach could be to search the median graph in all these intervals, and take the graph with minimum SOD over all of these intervals. Nevertheless, we have chosen a more efficient approach. Instead of taking all the intervals, we choose only the smallest interval and its corresponding graph (this corresponds to the graphs 2 or 3 in Figure 6.5). Then, the median is chosen as the graph with minimum SOD only in this interval.

With this approach the search space for the median graph is reduced to only this interval (grey part in Figure 6.6), and can be explored using the same approach as in Chapter 5. This reduction will lead to obtain exact solutions for the median graph with small computation times as we will see in the next section.

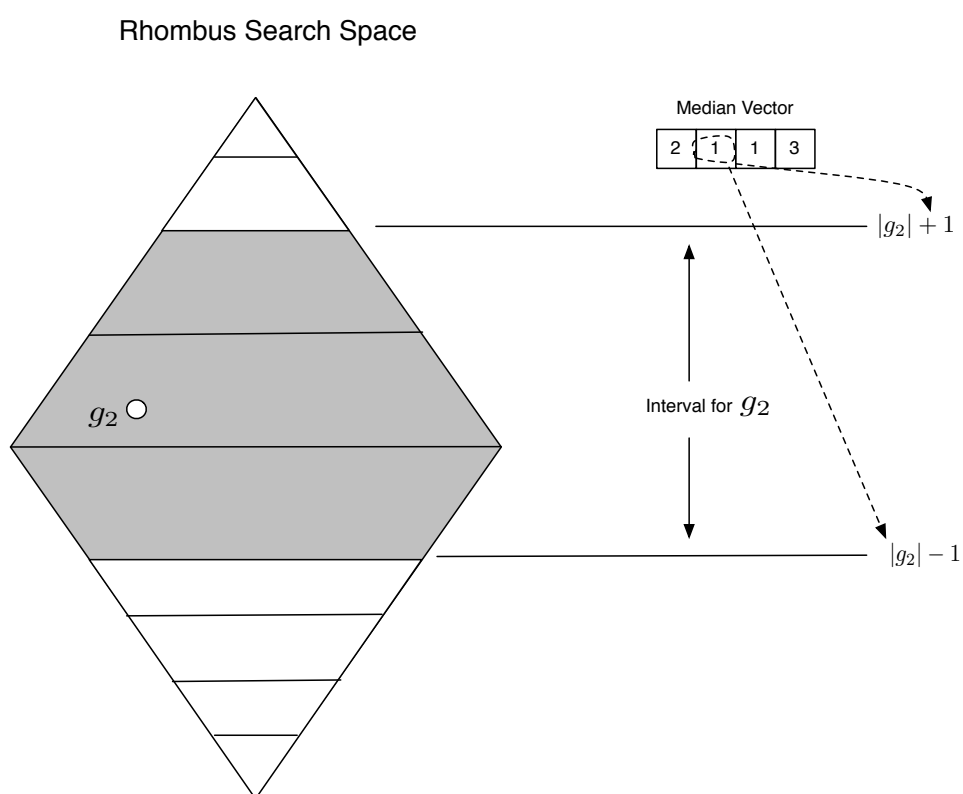


Figure 6.6: Explored part of the search space (grey part).

6.3.4 Experimental Setup

In order to evaluate the proposed method, we present in this section, the same experiment as in Section 5.4.3. That is, with the Letter-1 dataset described in Section A.1.1, we have computed several medians, using different number of graphs. The time and the number of SOD computations needed to compute the medians have been recorded. In this case, the results are compared with those obtained in the exact algorithm presented in the previous chapter. For simplicity these algorithms will be referred to as *ER* (for the exact algorithm presented in Chapter 5) and *EE* (for the exact embedding algorithm).

Computation Time and Number of SOD Computations

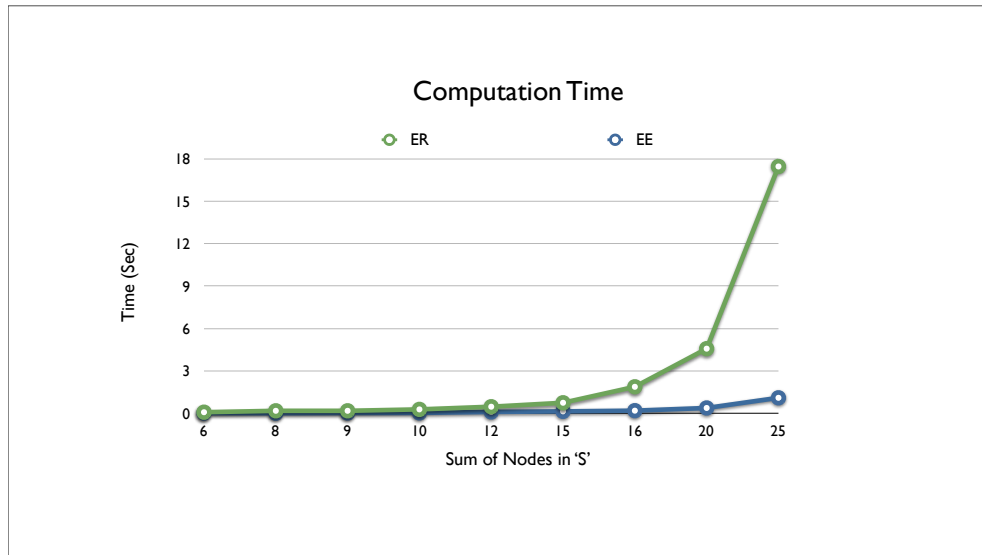
The results for both the computation time and the SOD computations required as a function of the total number of nodes in S are shown in Figures 6.7(a) and 6.7(b) respectively. Notice that the results are the mean values obtained for all sets S having the same number of nodes.

First of all it is important to remark that, in both methods and in all cases we obtained the same median graphs, which are in fact the true medians of the set S . Nevertheless, both figures show a substantially improvement, both in the computation time and the number of SOD computations, in the *EE* algorithm. The results show that while for the *ER* algorithm, the computation time for sets of graphs with 25 nodes is around 18 seconds, the *EE* algorithm only needs less than 3 seconds. Similar results can be extracted from the number of SOD computations. For the *ER* algorithm this number grows up to 500 SOD computations (for sets of graphs with 25 nodes), while for the *EE* algorithm it is only 75 in the same case.

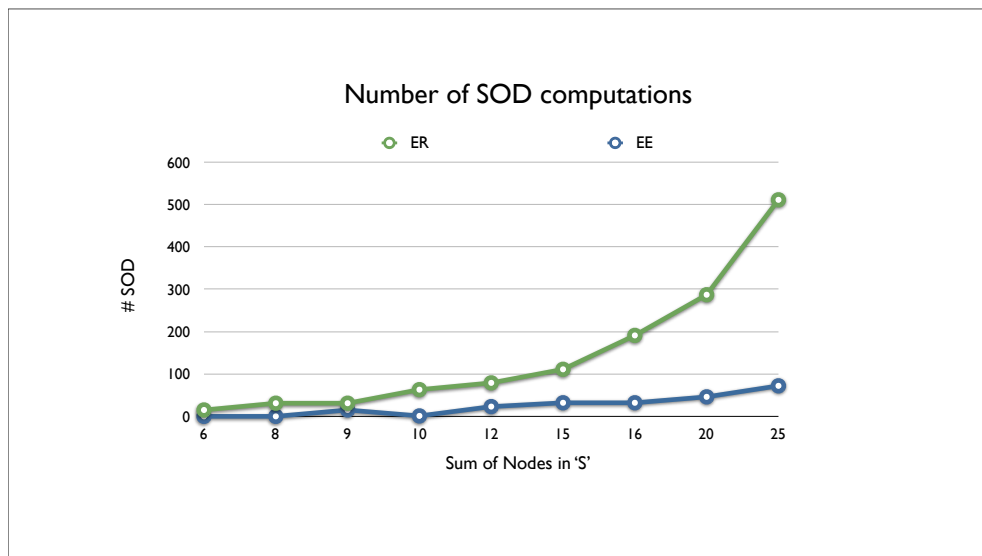
As a final conclusion, we can say that with the embedding approach for the median graph computation we are able to obtain exact solutions for the median graph with relatively big sets of graphs with a low computation time. This suggests this method may be used to extend the exact median graph computation to real applications as we will see later in Chapter 7.

6.4 Approximate Median Graph via Embedding

In the last section we have adapted the embedding procedure explained at the beginning of this chapter to obtain exact solutions for the median graph. The potential of this method has been shown in the experiments, since it clearly improves the exact method presented in Chapter 5. In this section we will provide an extension of the embedding method to potentially any kind of cost function, which makes this method applicable to any kind of graph with any kind of information in its nodes and edges. Differently, the new proposed method will find approximate solutions for the median graph. Nevertheless, we will show some results on both synthetic and real data showing that the method finds good approximations of the median graph. In this case, the three main steps of the general embedding procedure are based on the graph edit distance (first step), the Euclidean median vector computation (second step) and the weighted mean of a pair of graphs (third step).



(a)



(b)

Figure 6.7: Computation time (a) and number of SOD computations (b) as a function of the total number of nodes of the graphs in S .

6.4.1 Graph Embedding in a Vector Space

The graph embedding in a vector space is also performed using the procedure explained in Section 6.2.1. In this case the distance between graphs is the *graph edit distance* [13]. The use of the graph edit distance, potentially allows the application of this method to graphs with any kind of information in the nodes and the edges. That is, the distance may be computed over discrete data, continuous data, symbolic data, etc. This fact allows also to obtain continuous values of the distance. So, in this case we are in a typical Euclidean space, and the assumption that we are in a L_1 geometry is not valid here. This condition will make the third step more complicated.

6.4.2 Median Vector Computation

Once all the graphs have been embedded in the vector space, the median vector is computed. To this end we use the concept of *Euclidean Median*.

Definition 6.3 (*Euclidean Median*) Given a set $X = \{x_1, x_2, \dots, x_m\}$ of m points with $x_i \in \mathbb{R}^n$ for $i = 1 \dots m$, the *Euclidean median* is defined as

$$\text{Euclidean median} = \arg \min_{y \in \mathbb{R}^n} \sum_{i=1}^m \|x_i - y\| \quad (6.4)$$

where $\|x_i - y\|$ denotes the *Euclidean distance* between the points $x_i, y \in \mathbb{R}^n$.

That is, the *Euclidean Median*, is a point $y \in \mathbb{R}^n$ that minimizes the sum of the Euclidean distances to all the points in X . The Euclidean Median has been chosen as the representative in the vector domain for two reasons. The first reason is that the median of a set of objects is one of the most promising ways to obtain the representative of such a set. The second is that, since the median graph is defined in a very close way to the median vector we expect the median vector to represent accurately the vectorial representation of the median graph, and then, from the median vector to obtain good median graphs. The Euclidean Median cannot be calculated in a straightforward way. The exact location of the Euclidean Median can not be found when the number of elements in X is greater than 5 [4]. No algorithm in polynomial time is known, nor has the problem been shown to be NP-hard [44]. In this work we will use the most common approximate algorithm for the computation of the Euclidean Median, that is, the Weiszfeld's algorithm [112]. It is an iterative procedure that converges to the Euclidean Median. To this end, the algorithm first selects an initial estimate solution y (this initial solution is often chosen randomly). Then, the algorithm defines a set of weights that are inversely proportional to the distances from the current estimate to the samples, and creates a new estimate that is the weighted average of the samples according to these weights. The algorithm may finish when a predefined number of iterations is reached, or under some other criteria, such as that the difference between the current estimate and the previous one is less than a established threshold.

6.4.3 Back to the Graph Domain

The last step in order to obtain the median graph is to transform the Euclidean median into a graph. Such a graph will be considered as an approximation of the median graph of the set S . To this end we will use two different procedures based on the weighted mean of a pair of graphs [16] and the edit path between two given graphs. For the sake of completeness the definition of the weighted mean of a pair of graphs is included here.

Definition 6.4 (Weighted Mean of a Pair of Graphs) Let g and g' be graphs. The weighed mean of g and g' is a graph g'' such that,

$$d(g, g'') = a \quad (6.5)$$

$$d(g, g') = a + d(g'', g') \quad (6.6)$$

where a , with $0 \leq a \leq d(g, g')$, is a constant.

That is, the graph g'' is a graph in between the graphs g and g' along the edit path between them. Furthermore, if the distance between g and g'' is a and the distance between g'' and g' is b , then the distance between g and g' is $a + b$. Figure 6.8 illustrates this idea.

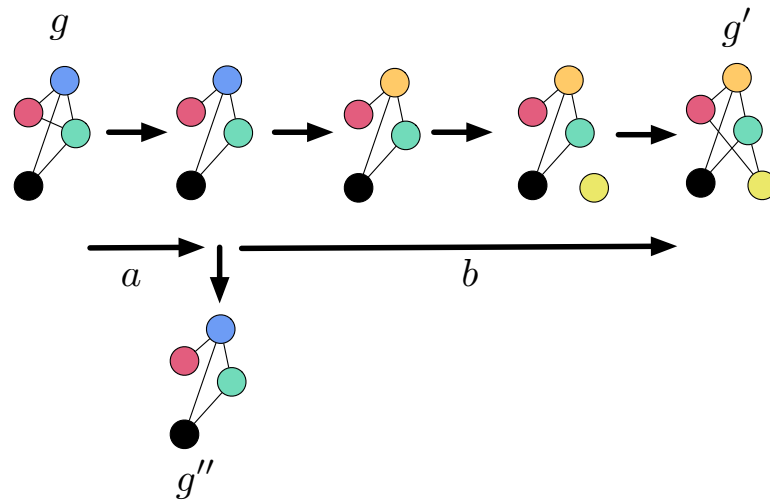


Figure 6.8: Example of the weighted mean of a pair of graphs

In the following we will present two different strategies to obtain the median graph from the median vector based on the weighted mean of a pair of graphs. The first solution will use three different points in the vector space to recover the median graph. For this reason we have called this approach *Triangulation Procedure*. In the second

solution we will use only two points in the vector space to recover the median graph. This approach will be called *Linear Interpolation Procedure*.

Triangulation Procedure

This procedure, illustrated in Figure 6.9(a), is based on the triangulation among three points in the vector space as follows. Given the n -dimensional points representing every graph in S (represented as white dots in Figure 6.9(a)), and the Euclidean Median vector v_m obtained using the Weiszfeld method (represented as a grey dot in Figure 6.9(a)), we first select the three closest points to the Euclidean median (v_1 to v_3 in Figure 6.9(a)). Notice that we know the corresponding graph of each of these points. We have indicated this fact by labelling them with the pair v_j, g_j with $j = 1 \dots 3$, in Figure 6.9(a). Then, we compute the median vector v'_m of these three points (represented as a black dot in Figure 6.9(a)). Notice that v'_m is in the plane formed by v_1, v_2 and v_3 . With v_1 to v_3 and v'_m at hand (Figure 6.9(b)), we arbitrarily choose two out of these three points (without loss of generality we can assume that we select v_1 and v_2) and we project the remaining point v_3 onto the line joining v_1 and v_2 . In this way, we obtain a point v_i in between v_1 and v_2 (Figure 6.9(c)). With this point at hand, we can compute the percentage of the distance in between v_1 and v_2 where v_i is located (Figure 6.9(d)). As we know the corresponding graphs of the points v_1 and v_2 we can obtain the graph g_i corresponding to v_i by applying the weighted mean procedure explained before (Figure 6.9(e)). Once g_i is known, then we can obtain the percentage of distance in between v_i and v_3 where v'_m is located and obtain g'_m applying again the weighted mean procedure (Figure 6.9(f)). Finally, g'_m is chosen as the approximation for the generalized median of the set S .

Linear Interpolation Procedure

In this case, once the median vector v_m is computed, we propose to choose only the two closest points to obtain the approximate median. The procedure is shown in Figure 6.10. With the median vector v_m at hand, we first choose the two closest points (v_1 and v_2 in Figure 6.10(a)). Then we compute the median vector of these two points to obtain v'_m (Figure 6.10(b)). This point v'_m will be used to obtain the approximate median. To this end, we first compute the distance of each point to v'_m (Figure 6.10(c)), and then, with these distances we apply the weighted mean of a pair of graphs to obtain g'_m , the approximate median (Figure 6.10(d)).

6.4.4 Discussion on the Approximations

This approximate embedding procedure is composed of three steps: the graph embedding into a vector space, the median vector computation and the return to the graph domain. Each of these steps introduces some kind of approximation to the final solution. In the first step, in order to deal with large graphs an approximate edit distance algorithm is normally used. Thus, each vector representing a graph includes small errors in its coordinates with respect to the optimal distance between two graphs. Also the median vector computation introduces a certain amount of error, since the Weiszfeld method obtains approximations for the median vector. This factor may

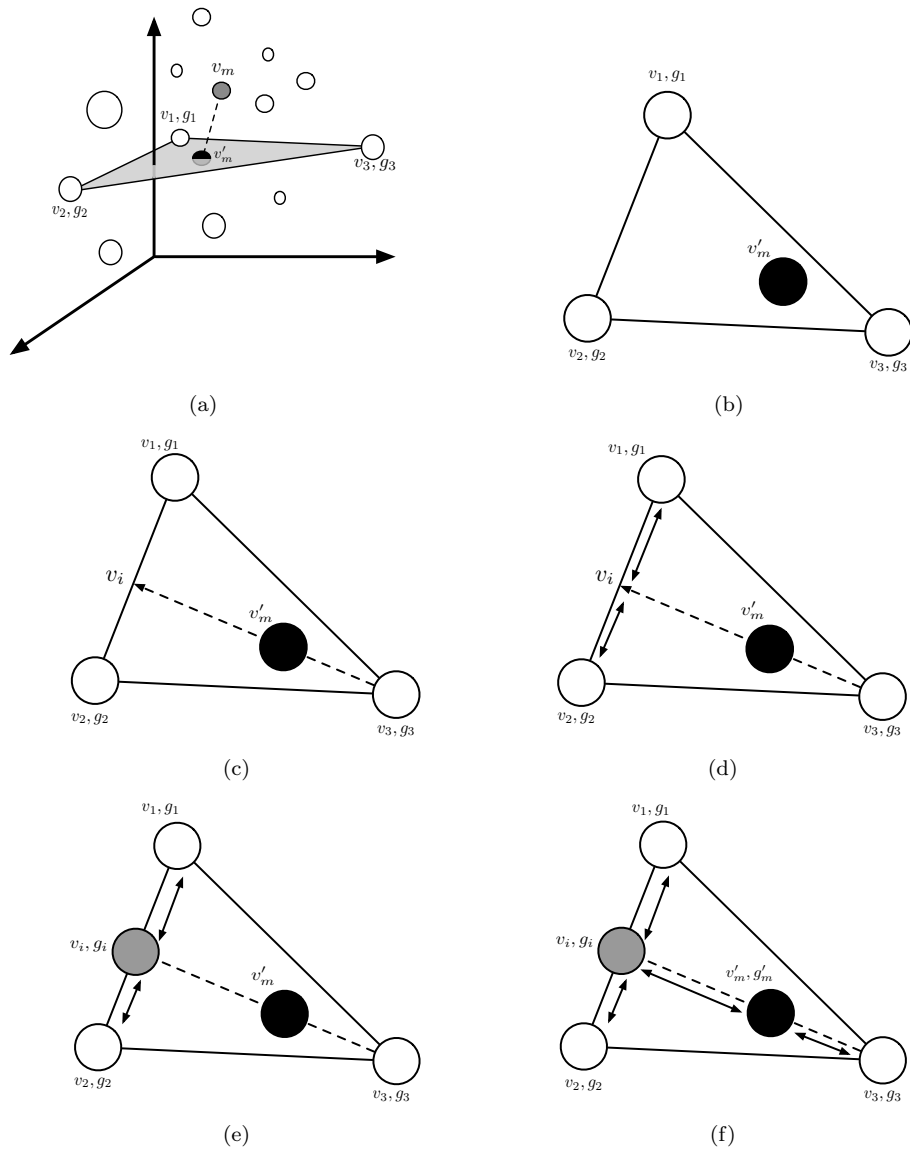


Figure 6.9: Illustration of the triangulation procedure.

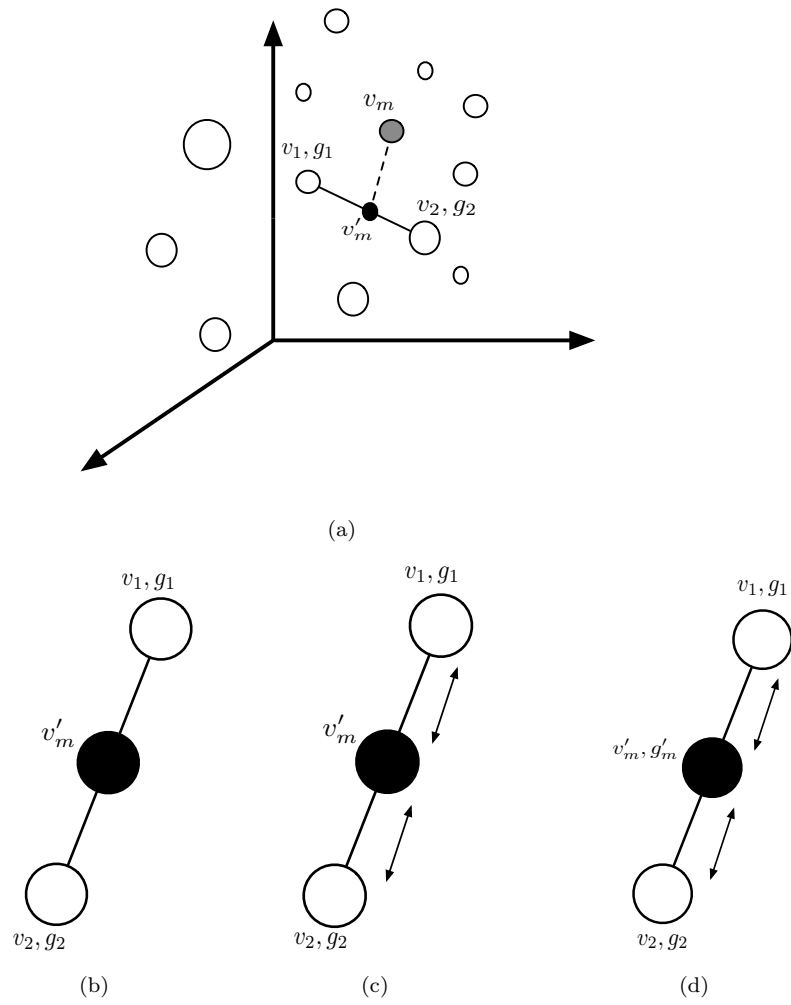


Figure 6.10: Illustration of the linear interpolation procedure.

lead to choose three (or two) points that might not be the best points to go back to the graph domain. In addition, small errors may be introduced when choosing v'_m instead of directly v_m to perform the weighted mean of a pair of graphs. Finally, when the weighted mean between two points is computed, the graph edit path is composed of a set of discrete steps, each of them with its own cost. In the return to the graph domain, the percentage of distance needed to obtain the weighted mean of a pair of graphs may fall in between two of this edit operations. Since we can choose only one of them, small errors may also be introduced in this step.

Although all these approximations may seem too severe to obtain good medians, in the next experiments we will show that this method is able to obtain reasonable good approximations for the median graph. But, at the same time, these well defined entry points of approximation can be used to improve the medians obtained. That is, each of these three steps can be further studied and other options for each of them can be proposed in order to minimize the error introduced and obtain better medians, as we will comment on the proposals for future work in Chapter 8.

6.4.5 Assessment of the Median Quality

In this section we propose an experiment to evaluate the quality of the median graphs obtained using the two variations that we have presented (i.e. using the triangulation and the linear interpolation procedures). It is remarkable that in this experiment we have used three different datasets: the Webpage dataset (see Section A.4 for more details); the Molecule dataset (see Section A.3 for more details) and the GREC-2 dataset (see Section A.2.2 for more details). But, in this case, differently from other experiments, we have used these datasets without any restriction in the number of graphs or in the kind of distortions and variability.

The experiment consisted in comparing the SOD of the obtained median with the SOD of the set median graph. We do not use other approximations of the median graph as a reference for two reasons. First, the existing methods are not able to compute the median graph with these large graphs and datasets we are dealing with. Secondly, as the set median graph is the graph belonging to the training set with minimum SOD, it is a good reference to evaluate the generalized median graph quality. To compute the distance between graphs we have used the approach introduced in [78] for the GREC-2 dataset and the approach presented in [82] for the Molecule and Webpage dataset.

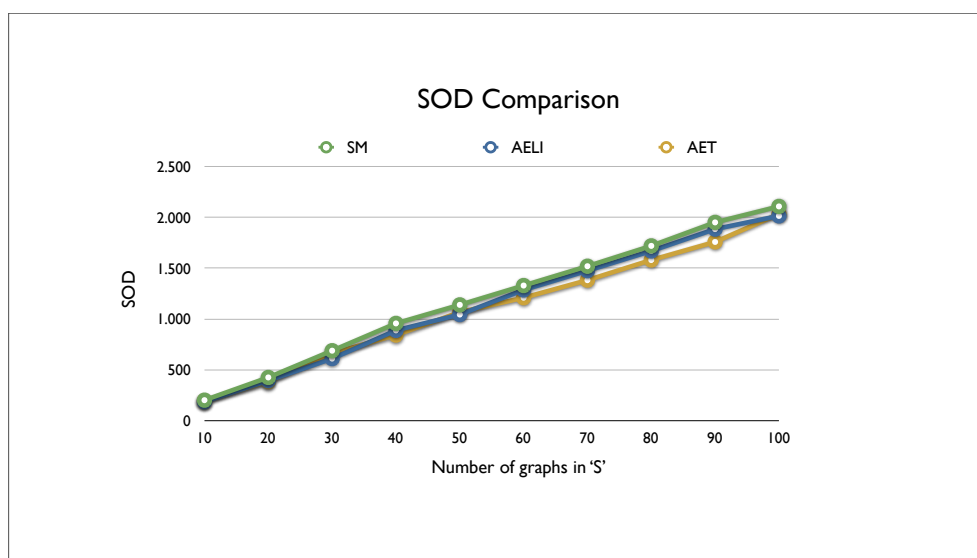
For this experiment we have randomly chosen an increasing number of graphs from the training set, and we have computed the median graph of each of these sets. The different number of graphs chosen from the training set for each database is shown in Table 6.1.

Results for the Molecule, Webpage and GREC datasets are shown in Figures 6.11, 6.12 and 6.13 respectively. In these figures, for simplicity we will refer to the set median graph simply as SM , the approximate embedding method with linear interpolation will be referred to as $AELI$ and the same method using the triangulation procedure will be denoted by AET .

First of all, it is important to notice that in the two first cases, the SOD of the

Table 6.1: Number of Graphs in S for each database.

Class	Number of Graphs in S
Molecules	10, 20, 30, ..., 100
Webpages	5, 10, 15, ..., 30
GREC	5, 10, 15, 20

**Figure 6.11:** SOD evolution on the Molecule dataset

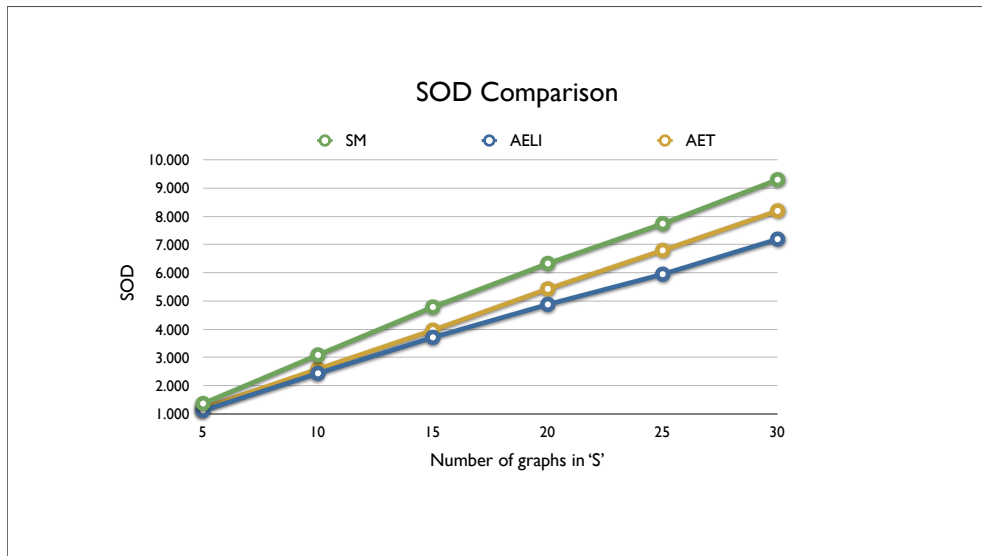


Figure 6.12: SOD evolution on the Webpage dataset

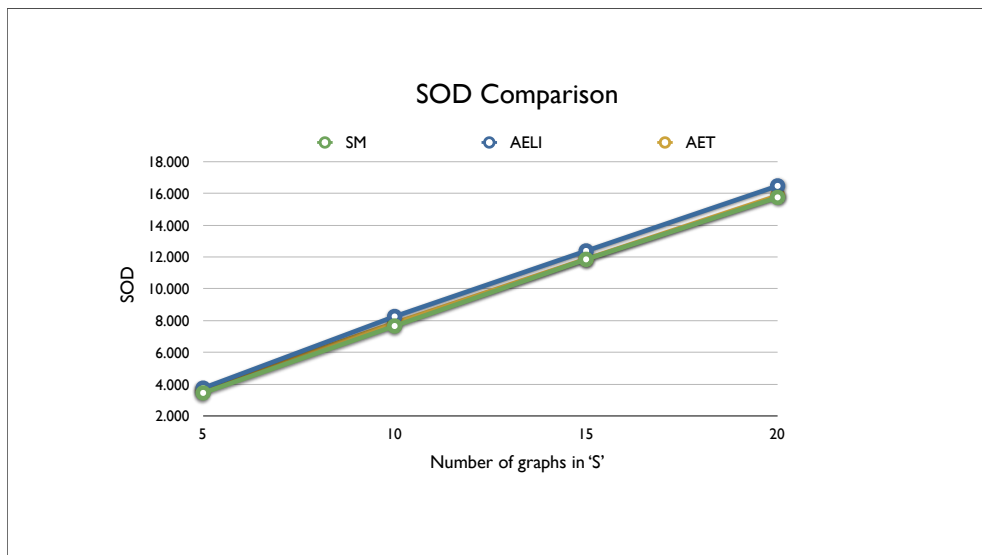


Figure 6.13: SOD evolution on the GREC dataset

approximate generalized median is lower than the SOD of the set median for any number of graphs in S . Intuitively, this result could mean that the obtained median is "located" more accurately in the center of the class than the set median.

Differently from the two previous results, in the GREC-2 database the SOD of the set median is slightly better than the SOD of the medians obtained using the triangulation procedure. However, when the size of S is 20 both values are equal. Conversely, the results for the medians obtained using the linear interpolation procedure, although they are very close to the SOD of the set median, are slightly worse in all the cases. The results obtained for this database may be explained by the fact that the distortion introduced in the GREC-2 database is very high (see Figure A.5). With such a strong distortion it is really difficult to keep the structural information shared among all the elements in the class.

Nevertheless, with these results at hand, we can conclude that our method achieves, in general, good approximations of the generalized median graph, independently of the size of the training set. That means that the generalized median adapts well to the increasing variability and distortion as the number of graphs in the training set increases.

6.5 Discussion

The main conclusions of this chapter can be summarized as follows:

- **Section 6.2:** We have proposed a novel general technique for the median graph computation. This new approach is based on the embedding of graphs into a vector space. First, the graphs are turned into points of an n -dimensional vector space using the graph edit distance paradigm. Then, the crucial point of obtaining the median of the set is carried out in the vector space, not in the graph domain, which dramatically simplifies this operation. Finally we transform the obtained median vector to a graph. This embedding approach allows us to get the main advantages of both the vector and graph representations. That is, we compute the more complex parts in real vector spaces but keeping the representational power of graphs. With this new technique we have presented two algorithms for the median graph computation, one exact and one approximate.
- **Section 6.3:** A new exact algorithm for the median graph computation has been introduced based on the embedding procedure. This algorithm uses the same cost function and the distance introduced in Chapter 5. But with the embedding approach we have shown that the part of the search space that must be explored to obtain the median can be reduced even more. With this new reduction, we have presented a very fast exact algorithm for the median graph computation. We have compared this new approach with the exact algorithm we presented in Chapter 5. Results have shown a significant improvement on both the computation time and the number of SOD computations needed to obtain the median graph. With these new promising results, the use of the exact median graph could be extended, in a limited way, to real machine learning algorithms.

- **Section 6.4:** In the last part of this chapter we have presented the most powerful technique for the approximate median graph computation. This technique is based on three main pillars, the graph edit distance, the median vector computation and the weighted mean of a pair of graphs. To compute the median graph we have analyzed two alternatives namely the triangulation procedure and the linear interpolation procedure. In addition we have provided some experiments using large amounts of real data to compute approximations of the median graph. The results comparing the SOD with that of the set median show that this method is able to obtain good approximations of the median graph even under hard conditions, with no restrictions neither on the size nor the number or the kind of graphs.

Finally we want to remark two additional properties of the general embedding approach (both the exact and the approximate) that can be interesting for its application to machine learning algorithms.

- The method can be used to compute the median in an incremental form. That is, imagine we have computed the median using n graphs. Then, the distance matrix (step 1) and all the edit paths between these n graphs are already computed and can easily be recorded. Imagine now another graph is included in the set and we want to recompute the median using these $n + 1$ graphs. With traditional methods we should re-start the computation of the median from the scratch. However, with this embedding procedure, we can recover the distance matrix and the edit paths stored previously. Then we only need to compute the distance and the edit paths between the new graph and the initial set of n graphs. Thus, addition of new graphs to the training set has a lower impact than in other methods.
- One of the critical points in the median computation is the need of computing the distance between the candidate median and all the graphs in the set. In the set median computation this number of pairwise computations is upper bounded by $\frac{n(n-1)}{2}$. In the median computation this quantity is usually much larger, even in the approximate methods. But in the embedding approach we have presented, we only need to compute the same number of pairwise distances as in the case of the set median graph, to compute the initial distance matrix. After that, the method, only includes a small overhead introduced by the median vector computation. In general, the computation time is not very far away from the time required to compute the set median.

Chapter 7

Application of the Median Graph

In this chapter, an experimental evaluation of the proposed methods for the median graph computation will be given. The aim is to demonstrate that beyond its theoretical properties, the median graph can be used in classic machine learning algorithms to solve real problems. The chapter is divided into two different parts. In the first part, the main objective is to confirm empirically that the proposed methods are applicable to relatively difficult graph-based classification problems. In the second part of the chapter we extend the applicability of the median graph to real clustering problems, using the approximate embedding approach presented in Chapter 6.

The chapter is organized as follows. First in Section 7.1 we will perform a series of classification experiments. First we will carry out three experiments using a limited amount of data. In these experiments we will test the methods presented in Sections 4.2, 5.5 and 6.3. Then, we will extend the experiments to large amounts of data using the approximate methods presented in Section 6.4. In all these experiments, the proposed methods will be compared with the performance of the traditional graph-based nearest neighbor classifier. After that, using again the two approximate embedding methods we will perform some clustering experiments in Section 7.2. In this case we will compare the results with those obtained with the set median graph. Finally, the chapter concludes summarizing the contributions of these experiments.

7.1 Classification Experiments

In this section we will perform some classification experiments. In order to put this experimental part in context, we will briefly review the basics of graph-based classification in the next section. Then in Section 7.1.2 we will explain the experimental framework we will use along all the experiments. Finally, in Section 7.1.3, the results will be presented.

7.1.1 Graph-based classification

Roughly speaking, classification is the task of assigning a class out of a set of possible classes to an unknown input pattern. There are several approaches for this task, such

as neural networks, Bayes classifiers, decision trees and others [35]. In the context of graphs, nearest-neighbor classifiers are almost the unique alternative. The main reason is that only a pattern dissimilarity measure is needed, unlike other approaches where the underlying pattern space needs to be rich in mathematical operations.

Nearest neighbor classifiers are a supervised learning task based on a labeled training set or set of prototypes. Then, given a test set composed of all the patterns to be classified, each pattern in this set is compared to all the elements in the training set and labelled with the class of the most similar element.

The formal definition of a nearest-neighbor classifier can be as follows. Assume a pattern space \mathcal{X} , a space of class labels \mathcal{Y} and a labeled training set of m patterns $\{(x_i, y_i)\} \subseteq \mathcal{X} \times \mathcal{Y}$ with $i = 1 \dots m$, is given. The *1-nearest-neighbor* classifier (usually denoted by 1NN) is defined by assigning a test pattern $x \in \mathcal{X}$ to the class of its most similar training pattern. Thus, the 1NN classifier can be defined as,

$$f : \mathcal{X} \mapsto \mathcal{Y} \quad (7.1)$$

where $f(x) = y$ is defined as,

$$f(x) = y_j \text{ where } j = \arg \min_{i=1 \dots m} d(x, x_i) \quad (7.2)$$

and $d(x, x_i)$ is a pattern similarity measure. To make the 1NN classifier less vulnerable to the outliers, the nearest-neighbor classifier can be extended to consider not only the most similar pattern in the training set but the k closest patterns, and we have a k NN classifier. In a k NN classifier the test pattern is assigned to the class that occurs most frequently among its k closest training patterns.

Algorithm 3 shows the classical k NN classification approach adapted to work with graphs.

Algorithm 3 Graph-based k NN Algorithm

Require: A set of pre-classified (training) instances, a query q and a parameter k defining the number of nearest neighbors to use.

Ensure: Label indicating the class of the query q

- 1: Find the k closest training instances to q according to a distance measure.
 - 2: Select the class of q to be the class held by the majority of the k nearest training instances.
-

As it can be appreciated in **Algorithm 3** only a similarity (or distance) measure is needed to classify the input pattern. For this reason the k NN approach is very suitable to work with graphs.

7.1.2 Experimental Setup

In the following we will perform a series of classification experiments using the median graph and we will compare them with the k NN approach. The dataset used in each experiment will be split into a training set and a test set. Our reference system will be a 1NN classifier using the whole training set as labeled patterns. Then, in each experiment we will obtain a representative of each class in the dataset using one of

the proposed methods for the median graph computation with some graphs of the training set. Thus, the graphs in the test set will be compared only with the median graph instead of the whole training set as in the 1NN approach.

In the three first experiments we will test the *Exact Embedding* approach explained in Section 6.3, the *Approximate Genetic* algorithm presented in Section 5.5 and the *Spectral Median Graph* method proposed in Section 4.2. All these experiments will be carried out using a relatively small amount of data, due to the limitations of these methods. The scenario will be selected according to the constraints of the method, such as the size of graphs, the nature of the labels, etc. Nevertheless, both approximate embedding approaches presented in Section 6.4, that is, the *Triangulation Approximate Embedding* and the *Linear Interpolation Approximate Embedding* can be applied to any kind of graphs and scenario. Therefore we also include them in these three experiments to have a graph-based reference system.

After that, we will extend the classification experiments using a large amount of data. For this last experiment only the approximate embedding methods will be used, as they are the only ones that can be applied to the datasets used. In this case we will also include the results obtained using the set median graph as a reference system.

For simplicity, all these methods will be referred to as EE (Exact Embedding), AG (Approximate Genetic), AS (Approximate Spectral), AET (Approximate Embedding with Triangulation), AELI (Approximate Embedding with Linear Interpolation) and SM (Set Median). Table 7.1 summarizes all these methods showing, for each of them, their acronym, the section where they have been presented and the median and method type.

Table 7.1: Summary of the tested methods and their characteristics.

Acronym	Method	Median Type	Method Type
EE	Section 6.3	Generalized Median	Exact
AG	Section 5.5	Generalized Median	Approximate
AS	Section 4.2	Generalized Median	Approximate
AELI	Section 6.4	Generalized Median	Approximate
AET	Section 6.4	Generalized Median	Approximate
SM	–	Set Median	–

At the beginning of each experiment, we also include a short table indicating the most important parameters of the experiment such as the dataset used, the number of elements in the training set and the test set, the methods compared and the number of graphs used to compute the median. In the presentation of the results, it is important to notice that due to the variability in the classification accuracy, the scales of the charts will be adapted in each case to better show the accuracy of each method.

7.1.3 Results

In the following the results obtained in the classification experiments for each method will be presented.

Experiment 1. Exact Embedding

In this first experiment we will test the classification accuracy for the *EE* method presented in Section 6.3. Since the method is based on a particular cost function and the labels for nodes and edges must have a discrete nature, in this experiment we have used the Molecule dataset described in Section A.3. We also compare this method with both the *AET* and *AELI* methods presented in Section 6.4. It is important to mention that the medians have been computed using different distance measures for each kind of method. Thus, the distance used in the *EE* method is that of Equation 5.1, while in the *AELI* and *AET* methods we have used the graph edit distance introduced in [82]. Table 7.2 summarizes all the parameters for this experiment.

Table 7.2: Configuration parameters for the experiment 1.

Dataset	Molecules
Number of classes	2
Num. of elements in the Training Set	20 (10/class)
Num. of elements in the Test Set	80 (40/class)
Compared Methods	1NN, EE, AELI, AET
Num. of Graphs to compute the Median Graph	5 and 10

The results for the classification accuracy are shown in Figure 7.1. First, it is important to note that the results for the 1NN classifier and the results for the EE method are very close to each other. In fact, the accuracy achieved by the EE method using 10 graphs to compute the median outperforms the accuracy of the 1NN classifier. We should also mention that while the training set for the 1NN classifier is composed of 20 elements (10 elements per class) and therefore the number of comparisons needed in the whole experiment is 1,600 (80x20), in the case of using the median graph there is only one representative per class, and the total number of comparisons is 160 (80x2). Thus, using the EE method we can achieve better classification results using 10 times less comparisons than using a classic 1NN classifier.

The second important result comes from the comparison between the EE method and the AET and AELI methods. For these two last methods, the classification accuracy decreases substantially with respect to the EE method. As the AET and AELI methods are approximate whereas the EE method is exact, this result reinforces the idea that the true median graph is the best representative of a given set. Thus, the convenience of exact methods for the median graph computation is fully justified with this result.

Experiment 2. Spectral Median Graph

In this experiment we will test the classification accuracy for the *AS* approach presented in Section 4.2. Since this method needs weighted graphs with the same number of nodes, we will use the GREC-1 dataset described in Section A.2.1. Although this dataset is composed of 32 different classes, we have only used 12 of these classes. All of them have 12 nodes per graph. This is a restriction of the spectral methods used

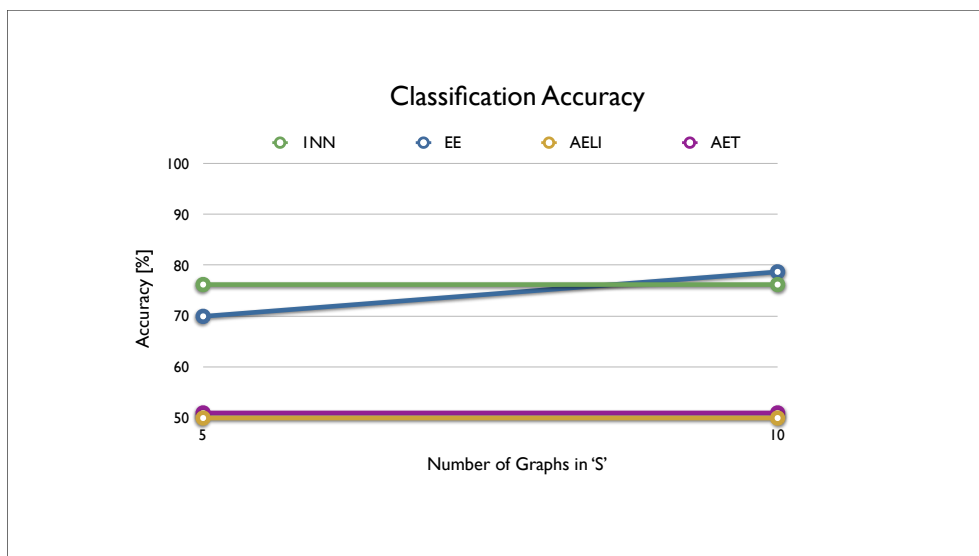


Figure 7.1: Classification accuracy for 1NN, EE, AELI and AET methods using the Molecule dataset.

in the AS algorithm. We also compare this method with both the *AELI* and *AET* methods presented in Section 6.4. In this case there is also a difference in the distance used to compute the median graph and to perform the comparison between the test set and the median. In the case of the AS method, the distance is computed using the Umeyama’s method (Section 4.1.4), while the distance used in the embedding methods is computed according to the procedure proposed in [78]. Table 7.3 summarizes all the parameters for this experiment.

Table 7.3: Configuration parameters for the experiment 2.

Dataset	GREC-1
Number of classes	12
Num. of elements in the Training Set	600 (50/class)
Num. of elements in the Test Set	600 (50/class)
Compared Methods	1NN, AS, AELI, AET
Num. of Graphs to compute the Median Graph	5, 10, 15, ..., 50

The results for this experiment are shown in Figure 7.2. First of all it is important to note that with medians computed using 10 graphs and more, all our methods outperform the classification accuracy with respect to the 1NN classifier. This gives the intuitive idea that the median graph is able to capture the essential information of each class. Again it is important to note the reduction in the number of comparisons. For the 1NN classifier this number grows up to 360,000 (600x600) while in the rest of

the methods the number of comparisons is only 7,200 (600x12).

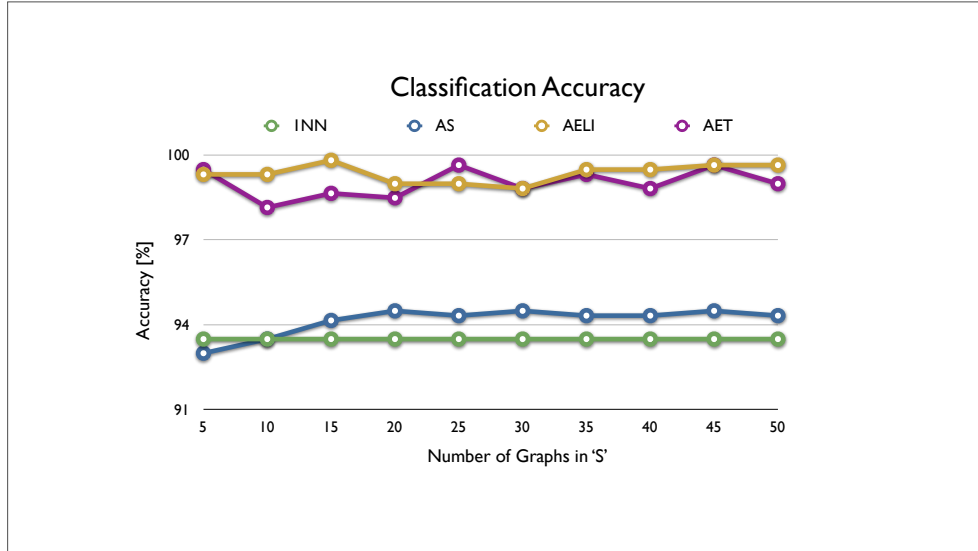


Figure 7.2: Classification accuracy for 1NN, AS, AELI and AET methods using the GREC-1 dataset.

In this case, the approximate methods based on the embedding procedure clearly outperform the spectral-based method. For all of them the training set and the test set were the same, and the medians were computed using the same graphs of the training set.

Experiment 3. Genetic Approach

In this experiment we will test the classification accuracy for the *AG* approach presented in Section 5.5. For this experiment we will use the Webpage dataset described in Section A.4. We also compare this method with both the *AELI* and *AET* methods presented in Section 6.3. The reasons to use this dataset are twofold. On one hand the size of the graphs in this dataset is very large. It implies the search space is also large, and the genetic algorithms are a good alternative to cope with this kind of search spaces. On the other hand, since the graphs in this dataset have unique node labels, the distance between two graphs can be computed in polynomial time. In this case the distance used to compute the medians in the *AG* method is that of Equation 5.1, while in the *AELI* and *AET* methods we use the distance introduced in [82]. Table 7.4 summarizes all the parameters for this experiment.

The results for this experiment are shown in Figure 7.3. First of all, it is important to note that in this case the 1NN classifier outperforms the rest of the methods in almost all cases. Only when the median is computed using 6 graphs, the *AG* method achieves slightly better results than the 1NN classifier. Although the results are not very good, it is important to mention that the number of comparisons needed in the 1NN approach were 28,800 (240x120) while the number of comparisons needed to

Table 7.4: Configuration parameters for the experiment 3.

Dataset	Webpages
Number of classes	6
Num. of elements in the Training Set	120 (20/class)
Num. of elements in the Test Set	240 (40/class)
Compared Methods	1NN, AG, AELI, AET
Num. of Graphs to compute the Median Graph	3, 4, 5, 6, 7

classify all the input patterns using the rest of the methods were only 1,440 (240x6).

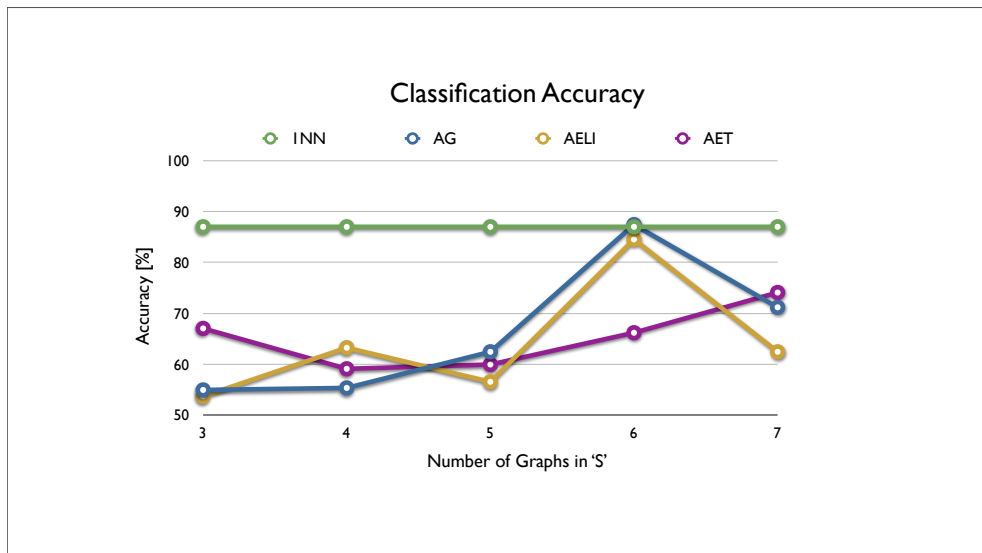


Figure 7.3: Classification accuracy for 1NN, AG, AELI and AET methods using the Webpage dataset.

One possible explanation for the poor results achieved by the median-based methods could be that, since the graphs in the webpage dataset have a large number of nodes and the number of graphs used to compute the medians is very low (due to the limitation of the AG method), the medians are not able to capture the essential information of the class. In this sense, it could be interesting to repeat this experiment using more data in order to see if the median is able to improve these results. That is precisely what we will do in the next section. Using the *AELI* and *AET* approaches presented in Section 6.4 we will perform classification experiments using a large amount of data.

Experiment 4. Approximate Embedding

The *AELI* and *AET* methods presented in Section 6.3 are able to cope with a large amount of data and they are not constrained in the kind of graphs and attributes of nodes and edges. In this section we will perform some classification experiments using the Molecule, the Webpage and the GREC-2 datasets with a large number of graphs in both the training and the test set. In all of them, we will use the 1NN classifier as a reference system and we will also use the set median (SM) as a graph-based reference system. This way, we can verify whether the median graphs we obtain can improve the results of the set median graph, which is also a potential representative of the set. In all the experiments the medians were computed using different number of graphs in order to generalize the results.

A. Molecule dataset

For this experiment we have used the Molecule dataset described in Section A.3. In all the methods the distance used to compute the medians was that presented in [82]. Table 7.5 summarizes all the parameters for this experiment.

Table 7.5: Configuration parameters for the experiment 4 using the Molecule dataset.

Dataset	Molecule
Number of classes	2
Num. of elements in the Training Set	200 (100/class)
Num. of elements in the Test Set	1,300 (200 and 1,100)
Compared Methods	1NN, SM, AELI, AET
Num. of Graphs to compute the Median Graph	10, 20, . . . , 100

Figure 7.4 shows the results for this experiment. The values are the mean values over the two classes and the x -axis represents the number of graphs from the training set used to compute the medians.

First of all it is important to notice that both kind of medians do not achieve good results for small numbers of graphs in S . A possible explanation is that with this small number of graphs (less than 30) there is not enough information in the graph structure to be able to keep the structural information of the class. Then in this cases, the median approach does not work well. Nevertheless, from a reasonable number of graphs to compute the median on (from 30 on), the classification accuracy achieves quite good levels. But at this point, it is relevant to mention that the classification accuracy of the computed median is, in general, better than that obtained for the set median. In addition it is remarkable the stability in the classification accuracy. This means that even for a large number of graphs the median is able to keep the basic information of the class. We can remark the direct correlation of these results with those obtained in the SOD evolution (see Section 6.4.5). That is, we obtain a better classification accuracy with the approximate median as it also achieves a better SOD

than the set median graph.

In spite of the small loss in the classification accuracy of the median-based methods with respect to the 1NN classifier, it should be remarked the difference in the number of comparisons. While the number of comparisons is 260,000 ($1,300 \times 200$) for the 1NN classifier, in the median-based classifiers the number of comparisons is only 2,600 ($1,300 \times 2$). This reduction may play an important role in graph-based applications, where the time needed for comparing two graphs is sometimes quite high.

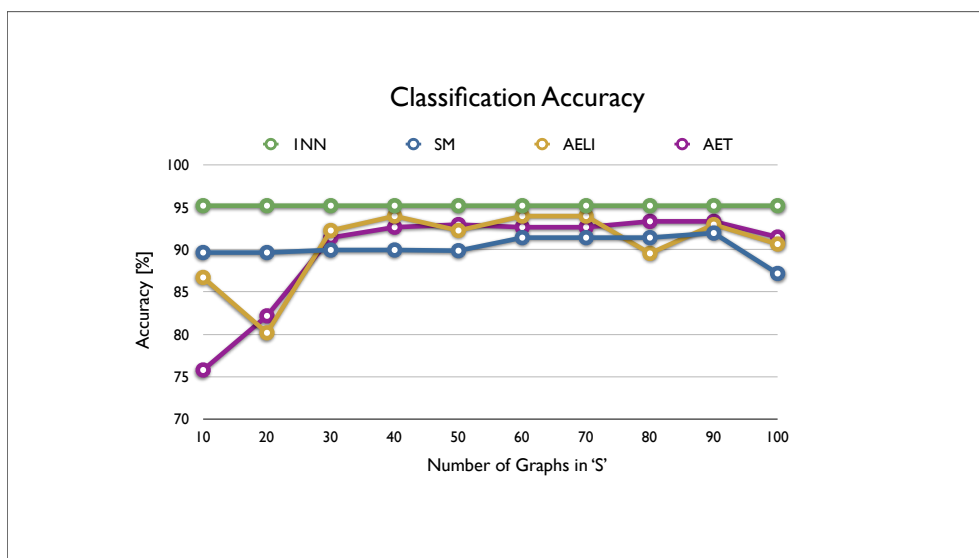


Figure 7.4: Classification accuracy for 1NN, SM, AELI and AET methods using the Molecule dataset and for different number of graphs to compute the median graph.

B. Webpages dataset

For this experiment we have used the Webpage dataset described in Section A.4. In all the methods, the distance used to compute the medians was that introduced in [82]. Table 7.6 summarizes all the parameters for this experiment.

Figure 7.5 shows the results for this experiment. Here, the classification accuracy is the mean over the 6 classes. Similar conclusions to the Molecule dataset can be drawn from these results. The classification accuracy achieved by the median approach is very close to the 1NN classifier. The minimum difference between the 1NN classifier and the median is only about 2% (better than in the Molecule database). Again it is important to mention the difference in the classification accuracy between the set median and the computed median. In this case our medians outperform the set median in the classification accuracy for sets of graphs up to 15. This difference is even higher than that obtained in the molecule dataset. A direct correlation between these results and the median evolution can be seen here as well.

Although the 1NN performs better, it is important to notice again the difference

Table 7.6: Configuration parameters for the experiment 4 using the Webpage dataset.

Dataset	Webpage
Number of classes	6
Num. of elements in the Training Set	180 (30/class)
Num. of elements in the Test Set	2,160 (112, 1359, 464, 84, 111 and 30)
Compared Methods	1NN, SM, AELI, AET
Num. of Graphs to compute the Median Graph	5, 10, 15, . . . , 30

between the number of comparisons needed for both classifiers. While the 1NN classifier needs 388,800 ($2,160 \times 180$) comparisons, the use of the median graph reduces such quantity to 12,960 ($2,160 \times 6$) comparisons.

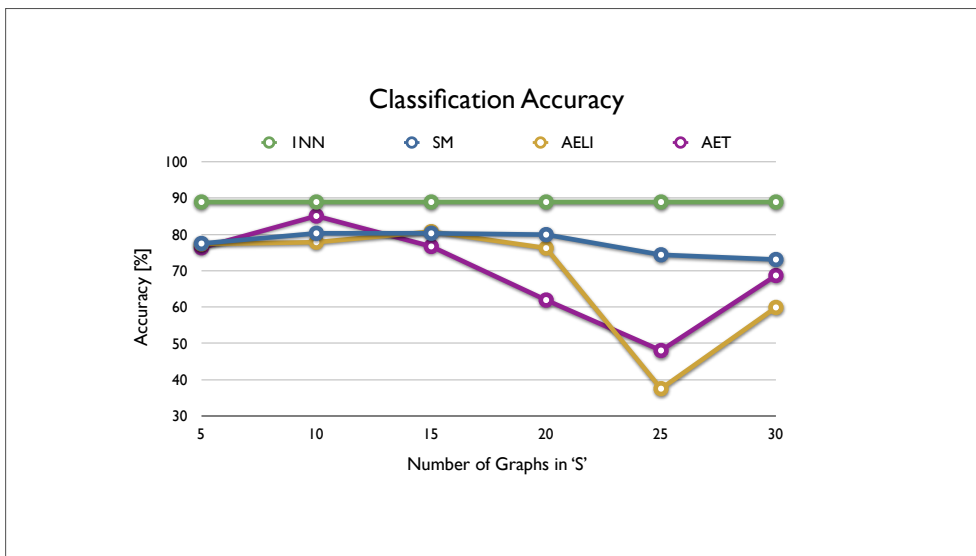


Figure 7.5: Classification accuracy for 1NN, SM, AELI and AET methods using the Webpage dataset and for different number of graphs to compute the median graph.

With the fact that the median achieves a classification accuracy quite close to the 1NN classifier, we can think of using the median to filter out the number of possible classes before applying the 1NN approach. With this approach we aim to improve the performance of the 1NN classifier and reduce the number of comparisons needed.

To this end, we propose first to measure the appearance frequency of the correct class within a predefined number of retrieved classes. That is, for every input pattern

we will rank all classes in a decreasing order based on the distance of the input pattern to the median of each class. Then, we will set a depth (the predefined number of classes) and we will see if the correct class appears in this set. In this experiment, we have used a depth of 3 and 5 classes.

In Figures 7.6 and 7.7, we can see the percentage of the appearance frequency of the correct class using depths of 3 and 5 respectively. Results show that in general, the appearance frequency of the correct class using the computed median outperforms the appearance frequency using the set median. It is important to note that even for a number of classes equal to 3 we have in general more than 95% of frequency appearance in both medians, higher than the classification accuracy using the 1NN. The results with 5 classes are obviously close to 100%. The low values achieved when the number of graphs in S is 25 can be a consequence of the randomness of the data, and consequently it can be seen as a spurious phenomenon. As a matter of fact, for the rest of sizes of S , the results show a high stability in the response, which reinforces the fact this is a punctual situation.

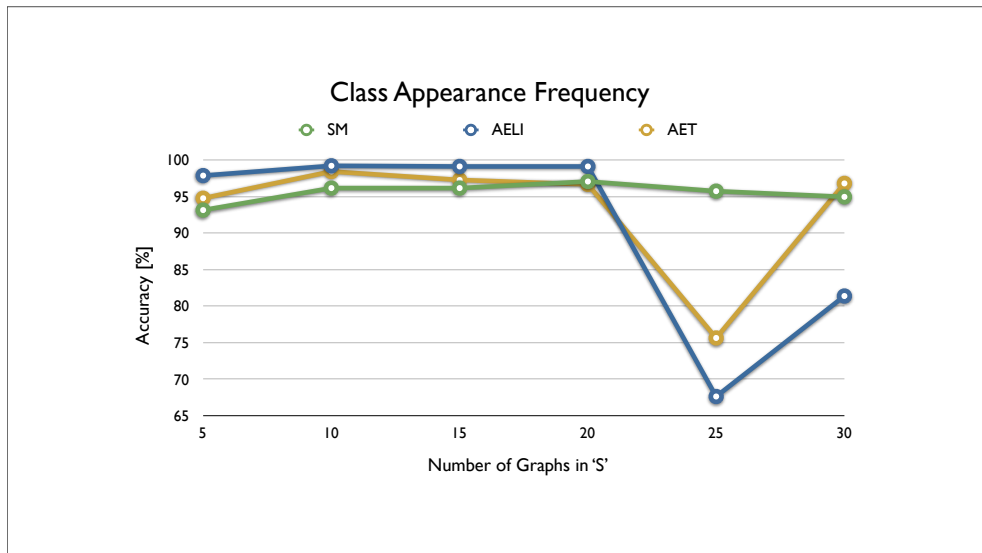


Figure 7.6: Appearance frequency of the correct class using 3 classes.

Finally, with these results at hand, we performed a modified classification experiment, mixing the median-based methods with the classic 1NN classifier. The objective of this experiment is to show whether it is possible to increase the classification accuracy of the 1NN classifier but using a lower number of comparisons. The basic idea is as follows. First, we compare each element of the test set against the medians, and rank the classes from the most similar to the less similar. After that, the same element in the test set is compared against the training set but using only the elements of the best k (3 for instance) classes according to the previous ranking instead of using all classes as in the 1NN classical approach. The input query is assigned to the class

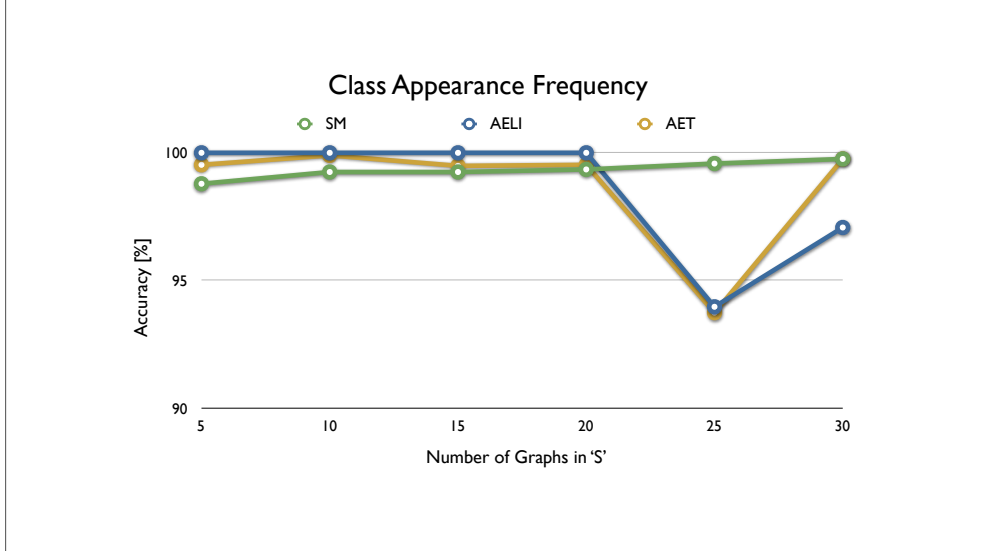


Figure 7.7: Appearance frequency of the correct class using 5 classes.

of the most similar element within this reduced number of classes. It is clear in this experiment that if k is set to 1 then the results are the same as those obtained with the classification using simply the median, and if $k = 6$ then the results are the same as in the 1NN classifier. Figure 7.8 shows the results using the median computed with 10 elements and for k ranging from 1 (the minimum number of classes) to 6 (all the possible classes).

As we have mentioned for a number of classes equal to 1 the results for both the set and the generalized median are the same as those obtained using simply the median, and the results for 6 classes are the same as those of the 1NN classifier. As we can see, the generalized median always outperform the set median in the results. This reinforces again the hypothesis that the generalized median keeps better the basic information of the class. But what is important to note is that even for $k = 2$, the classification accuracy is better than that of the 1NN classifier. The maximum classification accuracy is achieved for a number of classes equal to 3. This means that we can obtain better results than the 1NN classifier computing less than half the number of comparisons needed by the 1NN classifier (for $k = 2$).

C. GREC dataset

For this experiment we have used the GREC-2 dataset described in Section A.2.1. In all the methods, the distance used to compute the median was that introduced in [78]. Table 7.7 summarizes all the parameters for this experiment.

Figure 7.9 shows the results of classification accuracy for the GREC database. The

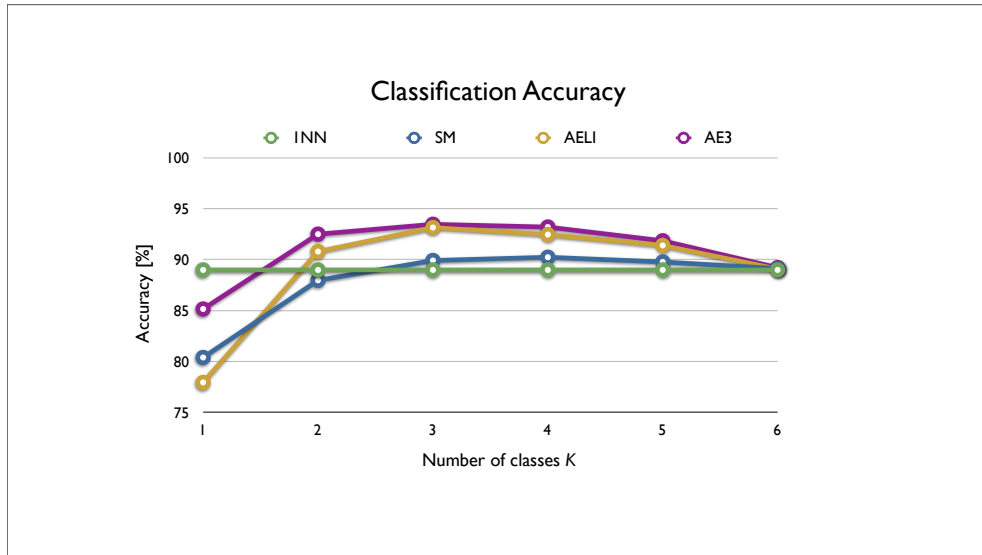


Figure 7.8: Multiclass classification accuracy for the Webpage database with an increasing number of graphs classes taken into account (number of graphs to compute the median = 10).

Table 7.7: Configuration parameters for the experiment 4 using the GREC-2 dataset.

Dataset	GREC-2
Number of classes	32
Num. of elements in the Training Set	640 (20/class)
Num. of elements in the Test Set	960 (30/class)
Compared Methods	1NN, SM, AELI, AET
Num. of Graphs to compute the Median Graph	5, 10, 15, and 20

values are the mean values over all the classes and the x -axis represents the number of graphs from the training set used to compute the medians.

As we can see in the figure, the 1NN classifier outperforms in all cases the classification accuracy for both the set and the generalized median. It is important to notice that in this case, the classification accuracy for the set median outperforms also the results for the generalized medians. This result is consistent with the SOD comparison results (see Section 6.4.5), where the set median obtained also better results. It gives the idea that there is a direct correlation between the SOD of the median and the classification accuracy. That is, the best the SOD of the median, the best the classification accuracy. It suggests that if we are able to obtain better approximate medians we will be able to obtain better classification results as well.

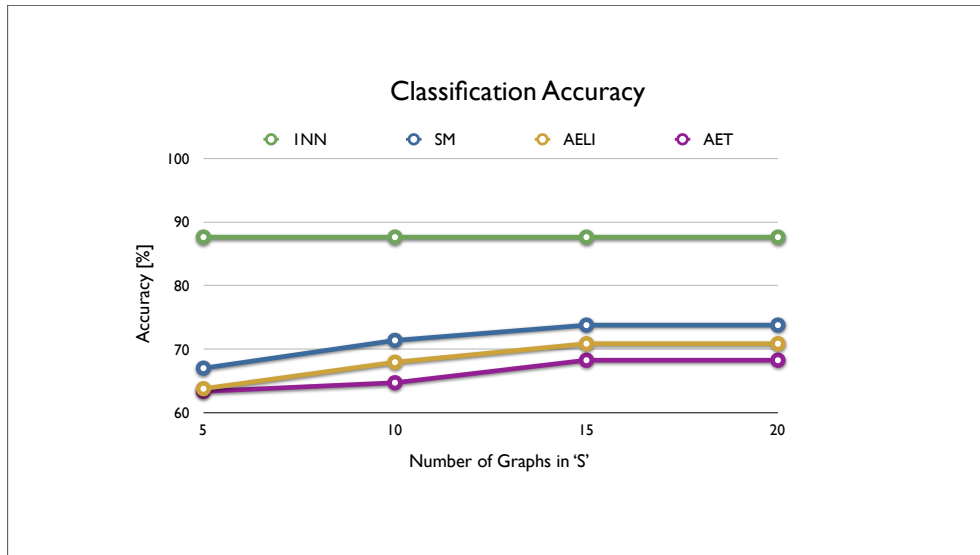


Figure 7.9: Classification accuracy for 1NN, SM, AELI and AET methods using the GREC-2 dataset and for different number of graphs to compute the median graph.

Again, as in the webpage experiment, in order to improve the classification results, a frequency appearance experiment has been done. In this case the results are shown in Figures 7.10 and 7.11 for depths of 5 and 10 respectively. The results show that, in contrast with the Webpage dataset, the set median represents better the classes. Nevertheless, it is important to note that for a number of classes equal to 10 the percentage of times the correct class appears in the selected set of classes is close to 100%.

Finally, we also performed the same modified classification experiment as in the Webpage dataset. Results are shown in Figure 7.12, for a number of classes k ranging from 1 to 32 and for a number of graphs in S equal to 15. In this case, the results obtained with the medians are in the best case equal to those of the 1NN classifier. Again the set median slightly outperforms the generalized median. But, what is important to notice is that using about 15 classes we achieve the same classification

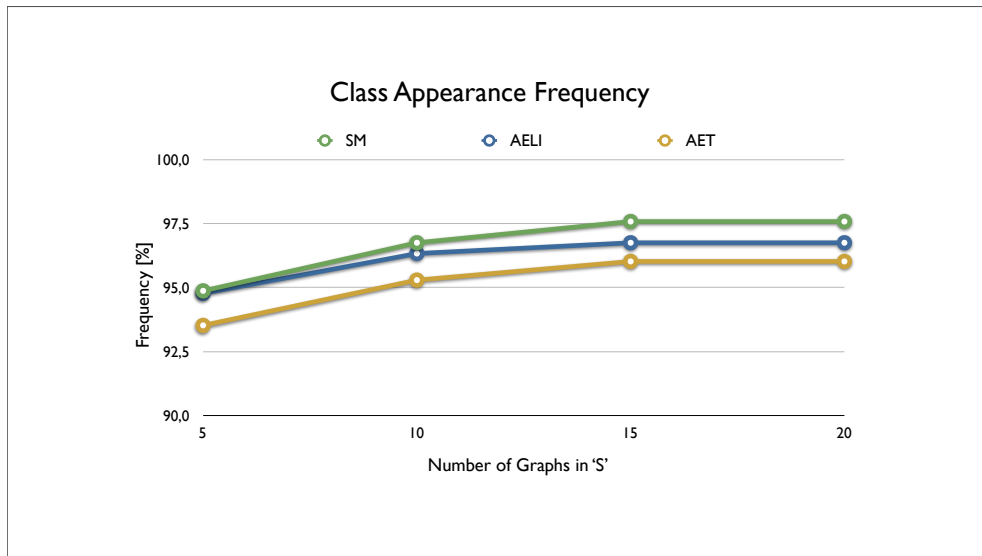


Figure 7.10: Appearance frequency of the correct class using 5 elements.

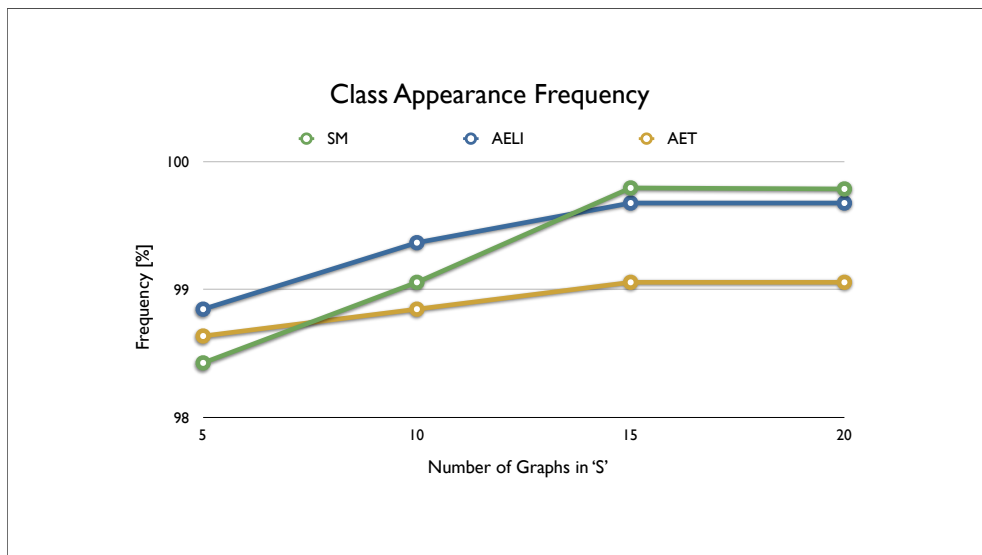


Figure 7.11: Appearance frequency of the correct class using 10 elements.

accuracy as with the 1NN classifier but using nearly half the number of comparisons. Thus, the use of the median is fully justified for large databases. It also suggests that most probably if we obtain better approximations of the median we can improve these results and achieve better results than the 1NN classifier.

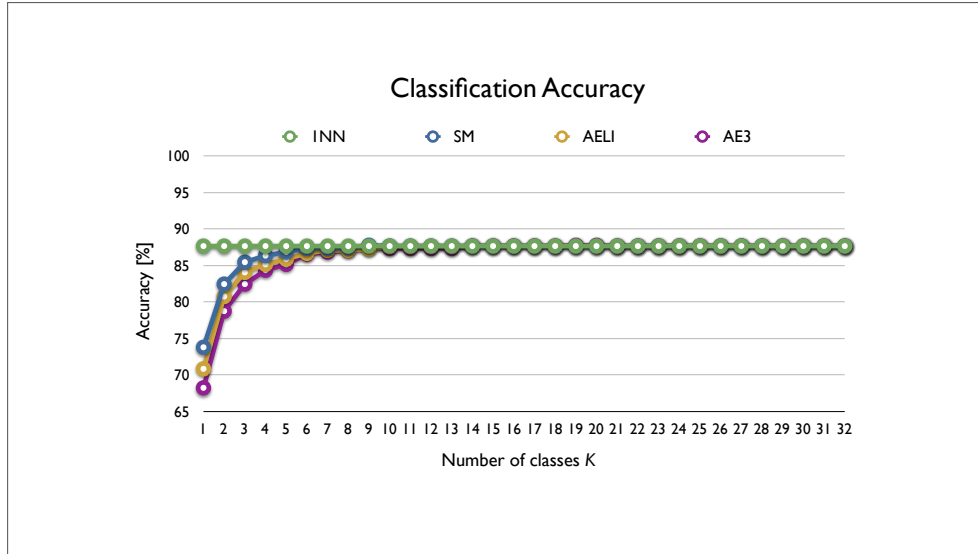


Figure 7.12: Multiclass classification accuracy for the GREC database with an increasing number of classes taken into account (number of graphs to compute the median = 15).

7.1.4 Discussion on Graph-based Classification

In the previous sections we have performed several classification experiments. The first important result is that with the use of the median graph we can achieve, in general, similar results as with the 1NN classifiers but with a reduced number of comparisons (even in the case where a limited amount of data is used the results obtained with the median approach outperform in some cases the 1NN classifier). This reduction in the number of comparisons can be useful in large databases, where the number of comparisons in the 1NN classifier may be unfeasible. In addition, in the experiments done with a large amount of data and using the *AELI* and *AET* methods, we have shown that a mixed solution in between the median and the 1NN classifier is able to obtain a better classification accuracy than the 1NN classifier with a reduced number of comparisons. These last classification results are similar to those obtained in [83]. This makes the graph embedding in vector spaces a very promising way to improve the classical learning algorithms using the power of representation of graphs.

7.2 Clustering

In addition to the classification experiments, in this section we will extend the use of the median graph to clustering experiments. In particular, we will use the *AELI* and *AET* methods presented in Section 6.4 to compute the centers of the clusters in a k -means clustering algorithm. Traditionally these centers have been computed using the set median, but in this section we will extend this computation to the generalized median using our new approximate algorithm. First, in the next section, we will introduce the concept of graph-based clustering and, in Section 7.2.2, we will give two numerical measures to evaluate the quality of the clustering. After that, in section 7.2.3 we will provide the results of clustering using the Molecule, the Webpage and the GREC datasets.

7.2.1 Graph-Based Clustering

Clustering with graphs is a well studied topic in the literature, and some different approaches have been presented up to now. The classical paradigm in those approaches has been to treat the entire clustering problem as a graph, that is, each element to be clustered is represented as a node and the distance between two elements is modeled by means of a certain weight in the edge linking the nodes. Then, with this graph at hand, the usual procedure is to create a minimum spanning tree of the graph and then, from the remaining edges, iteratively remove those with the largest weights, until the desired number of connected components matches the number of desired clusters. That is, each connected component represents a cluster. The nodes of each connected component indicate which original elements belong to each cluster [50].

Conversely, some recent approaches propose to perform clustering directly on graph-based data. For instance in [43], the graph edit distance and the weighted mean of a pair of graphs were used to cluster graph-based data under an extension of self-organizing maps (SOMs). In [88], the authors investigated the clustering of attributed graphs by means of the Function-Described Graphs (FDGs) to obtain the representative of a cluster. Trees have also been used for clustering purposes. For instance, in [67], the clustering of shock trees using the tree edit distance was introduced. Finally, the extension of the well-known k -means algorithm for clustering to graph based representations was introduced in [89]. In the following, this approach will be further explained.

The Graph-Based k -means Clustering Algorithm

The k -means clustering algorithm is one of the most simple and straightforward methods for clustering data [74]. The usual way is to represent the data items as a collection of n numeric values usually arranged into a vector form in the space \mathbb{R}^n . Then, the *euclidean distance* in this space and the *centroid* of a set of vectors are used to compute the mean of the data in the cluster. The classical version of the k -means algorithm is presented in the following.

Nevertheless, in [89], the graph-based version of the classic k -means clustering algorithm is presented by introducing some changes in this algorithm: 1. replacing the vector-based distance with any graph-based distance and 2. replacing the centroid

Algorithm 4 Classic k -means Clustering Algorithm

Require: A set of n data items and the number of clusters k to create**Ensure:** The centroids of the cluster and for each data item an integer $[1, k]$ indicating the cluster the item belongs to

- 1: Assign randomly each data item to a cluster.
 - 2: Using this initial assignment, compute the centroid of each cluster.
 - 3: With the computed centroids, assign each data item to be in the cluster of its closest centroid.
 - 4: Recompute the centroids as in Step 2.
 - 5: Repeat Steps 3 and 4 until the centroids do not change.
-

with the median of a set of graphs. This median is usually the set median. The graph-based version of the k -means algorithm is presented in the following as **Algorithm 5**,

Algorithm 5 Graph-based k -means Clustering Algorithm

Require: A set of n data items (represented as graphs) and the number of clusters k to create**Ensure:** The centroids of the cluster (represented by the median graphs) and for each data item an integer $[1, k]$ indicating the cluster the item belongs to

- 1: Assign randomly each data item to a cluster.
 - 2: Using this initial assignment, compute the median graph of each cluster.
 - 3: With the computed medians, assign each data item to be in the cluster of its closest median using a graph-based distance.
 - 4: Recompute the medians as in Step 2.
 - 5: Repeat Steps 3 and 4 until the set medians do not change.
-

In [89], the set median graph is used to represent the center of the cluster. We will extend the algorithm to the generalized median graph using the approximate embedding technique introduced in Section 6.4.

7.2.2 Clustering Performance Measures

In order to evaluate the cluster performance, different measures exist. We will present two of these measures, namely the *Rand Index* and the *Dunn Index*, that have been also used in previous graph-based clustering experiments [89].

A. Rand Index

This index measures the matching of the obtained clusters to the "ground truth" clusters (i.e. the accuracy). That is, the *Rand index* measures how closely the clusters created by the clustering algorithm match the ground truth, and is defined as,

$$R_I = \frac{A}{A + D}$$

where A means the number of agreements and D means the number of disagreements. After the clustering process, a pair-wise comparison between all pairs of items in the data set is computed. If the two elements are in the same class both in the clustering result and in the ground truth, it counts as an agreement. In the same way, if the two elements belong to different classes both in the clustering and the ground truth it counts as a disagreement too. Otherwise, it counts as an agreement. The *Rand Index* produces measures in the interval $[0, 1]$, with 1 meaning a perfect match between the result of the algorithm and the ground truth.

B. Dunn Index

The *Dunn Index* is a measure of the compactness and separation of the clusters. It is not an accuracy measure like the *Rand Index*. It is rather based on the assumption that in a "perfect" clustering, items in the same cluster should be similar (i.e. small distances between them) and items in different clusters should be dissimilar (i.e. large distances between them). It is defined as,

$$D_I = \frac{d_{min}}{d_{max}}$$

where d_{min} is the minimum distance between any two objects in different clusters and d_{max} is the maximum distance between any two objects in the same cluster. Thus, d_{min} measures the worst case of separation between clusters and d_{max} captures the worst case of compactness among all the clusters. Higher values of the *Dunn Index* indicates better clustering. Unlike the *Rand Index*, the *Dunn Index* is not bounded in the interval $[0, 1]$.

7.2.3 Experimental Setup

In the following we will perform a number of clustering experiments based on the graph-based k -means approach explained before. We will use the Molecule, the Webpage and the GREC-2 datasets. For each dataset, the experiments consisted in computing the centers of the clusters using a certain number of k clusters according to the number of classes in the dataset. The initial elements in each cluster were chosen randomly. Table 7.8 summarizes the basic parameters (number of classes and the number of items to be clustered) used for each dataset.

Table 7.8: Number of classes and number of elements per class for each database.

Class	Num. of Classes	Items/Class
<i>Molecules</i>	2	100
<i>Webpages</i>	6	30
<i>GREC</i>	32	20

The centers of the clusters were computed using the set median (SM) and both the AELI and AET methods introduced in Section 6.4. In order to evaluate the obtained

results we performed 10 repetitions of each experiment and then we computed the *Rand Index* and the *Dunn Index* for each of them.

7.2.4 Results

The results for this experiment are summarized in Tables 7.9 and 7.10. In each table the minimum, mean and maximum values for the *Rand Index* (Table 7.9) and the *Dunn Index* (Table 7.10) for each dataset are shown. In both tables, the best results are marked in **bold** style.

A. Rand Index

Results of the *Rand Index* show that in almost all cases the *AELI* and *AET* methods obtain better results than the set median graph. More concretely, six out of the nine best results in Table 7.9 correspond to the *AET* method. Since the *Rand Index* is a measure of how similar to the ground truth the clusters are, these overall results demonstrate again the idea that the median graph computed with this method is a good representative of a given set, better than the set median graph.

Table 7.9: Minimum, average and maximum values of the Rand index for different datasets.

	Minimum		
	SM	AELI	AET
<i>Molecule</i>	0.5072	0.5545	0.5544
<i>Webpages</i>	0.6841	0.8332	0.7758
<i>GREC</i>	0.9410	0.9340	0.9500
	Average		
	SM	AELI	AET
<i>Molecule</i>	0.56207	0.5952	0.6352
<i>Webpages</i>	0.8083	0.8773	0.8786
<i>GREC</i>	0.9506	0.9513	0.9537
	Maximum		
	SM	AELI	AET
<i>Molecule</i>	0.6205	0.6860	0.7178
<i>Webpages</i>	0.8558	0.9133	0.9506
<i>GREC</i>	0.9602	0.9566	0.9587

Looking more accurately the average results of the *Rand Index* we can see again a direct correlation between these results and the results obtained in the SOD comparison experiments performed in Section 6.4.5. In those results, the SOD of the generalized medians of the *Molecule* and the *Webpage* datasets were clearly better

with respect to the set median graph, while the results of the GREC-2 dataset were in some cases slightly better. A similar behavior can be extracted from the results of Table 7.9. The *Rand Index* of the Molecule and the Webpage datasets are clearly better in the AELI and AET methods than the SM approach, while the results of the GREC-2 dataset are practically the same in all the cases. Thus, if we are able to obtain better medians in the case of the GREC-2 database, we would probably improve this results as well.

B. Dunn Index

Results of the *Dunn Index* are shown in table 7.10. Differently from the *Rand Index*, which is bound in between 0 and 1, the *Dunn Index* is not bounded. Thus, for the *Rand Index* it is relatively easy to interpret the value, because 0 means a completely uncorrelated result with respect to the groundtruth and 1 means a perfect match between the result and the groundtruth independently of the dataset used. Nevertheless, the same reasoning is not possible for the *Dunn Index*. That is, we cannot say how good a result x for the *Dunn Index* is unless the *Dunn Index* for the groundtruth is given. For this reason, we have also computed the *Dunn Index* for the groundtruth.

The results for each method are shown in Table 7.10. In this case the majority of the best results correspond to the set median. In a first moment, these results could be interpreted in the sense that the set median reflects better the ideal cluster. Actually, they show that the set median graph obtains a better separation of the data into compact clusters. However, the results of the *Dunn Index* for the groundtruth (see Table 7.11), show very low values. That means that original datasets have low separability and compactness. In this sense, the *AELI* and *AET* methods have more similar results to the groundtruth than the set median. That means that they are able to better capture the original information of the cluster.

7.3 Discussion

In this chapter we have given an experimental evaluation of the proposed methods for the median graph computation. To this end we have performed graph-based classification and clustering experiments. The main conclusions of this chapter can be summarized as follows:

- **Section 7.1:** We have performed several classification experiments using different methods. The first three experiments performed using a limited amount of data have shown that with the median approach we can outperform in some cases the classification accuracy of the 1NN approach reducing the number of comparisons needed in the classification. In a second part, and using the *AELI* and *AET* methods presented in Section 6.4, we performed classification experiments using a large amount of data. The results have shown that, although the median is not able to outperform the 1NN approach, we can obtain similar results but with a significant reduction in the number of comparisons. In addition, we have presented a slightly more complex classification scheme where the results obtained using the median approach outperform the 1NN classifier

Table 7.10: Minimum, average and maximum values of the Dunn index for different datasets.

Minimum			
	SM	AELI	AET
<i>Molecule</i>	0.0113	0.0272	0.0272
<i>Webpages</i>	0.2039	0.1028	0.0769
<i>GREC</i>	0.0411	0.0423	0.0423

Average			
	SM	AELI	AET
<i>Molecule</i>	0.034	0.0288	0.0272
<i>Webpages</i>	0.2448	0.2027	0.1711
<i>GREC</i>	0.0503	0.0507	0.0498

Maximum			
	SM	AELI	AET
<i>Molecule</i>	0.0909	0.0431	0.0272
<i>Webpages</i>	0.6046	0.5784	0.2055
<i>GREC</i>	0.0651	0.0569	0.0618

Table 7.11: Dunn Index for the groundtruth for each dataset.

Dataset	GT Dunn Index
<i>Molecule</i>	0.0182
<i>Webpages</i>	0.1835
<i>GREC</i>	0.0619

reducing also the number of comparisons.

- **Section 7.2:** Using the *AELI* and *AET* methods presented in Section 6.4, we performed a series of clustering experiments and we have used the median graph to obtain the representative of each cluster. Results in terms of the *Rand Index* show that with the median graph we obtain clusters closer to the groundtruth than using the set median graph. In addition, results given by the *Dunn Index* show that, although the set median graph obtains higher scores, the median graph obtains again results closer to the groundtruth.
- Finally, from these results we can give two more general conclusions. The former is that the median graph has been used to perform, for the first time, classification and clustering under real conditions, that is, using a large amount of real data with no constraints. From these experiments, we have shown em-

pirically, that the concept of the median graph is really suitable to obtain the representative of a given set.

Chapter 8

Conclusions

This thesis has addressed the concept of the *median graph*. Given a set of objects, the *median* is defined as the object which has the smallest sum of distances to all the objects in the set. The median of a set of objects has been shown as a good option to obtain a representative of the set. Already in the vector domain, the definition of the median vector has been presented as a suitable form to capture the characteristics of the set into a single vector. The definition of the *median graph* can be seen as the analogous definition of the *median* but in the graph domain. That is, the *median graph* can be seen as a good way to obtain a representative of a given set of graphs.

Despite its simple but powerful definition, the median graph has the lack of its complex computation. Given a set of graphs, the median graph computation is exponential both in the number of graphs and their size. A number of algorithms have been presented up to now, but all of them have been only applied to very small sets of graphs with small sizes. Thus, in spite of the potential of the concept of the median graph, its applicability has been limited due to the computational complexity underlying this concept.

The main objective of this thesis was to try to give the median graph the opportunity to leave the box of the only theoretical concepts and to be applied to real problems. To this end the thesis has addressed the problem from different points of view, theoretic and practical. This last chapter summarizes the main conclusions of this work and poses new possible open questions to be studied in the future.

8.1 Conclusions

The basic conclusions of this work can be summarized from three points of view, theory, algorithms and applications.

8.1.1 Theory

This thesis has addressed theoretical aspects of the median graph. From this theoretical point of view we can highlight two major contributions: the improvement on the existing theoretical properties of the median graph and the novel concept of spectral

median graph.

- **Spectral Median Graph:** With the use of the spectral graph theory we have presented the novel concept of the *Spectral Median Graph*. With this new concept we are able to compute good approximations of the median graph in polynomial time. Nevertheless, this concept is limited to operate only with weighted graphs having the same size.
- **New Theoretical Properties:** We have theoretically demonstrated that the existing bounds on the size and the SOD of the median graph can be improved under the assumption that a particular cost function is used. The previous bounds were too coarse to be really useful for practical purposes. With these new bounds we have shown that the search space of the median graph can be drastically reduced. Thus, these new bounds affect directly the computation of the median graph and therefore they can be used to improve it.

8.1.2 Algorithms

Beyond the theoretical aspects of the median graph we have also proposed five different algorithms and strategies for its computation. These different approaches can be summarized as follows:

- **Spectral-based Algorithm**

- Using the concept of the *Spectral Median Graph*, we have provided an incremental algorithm for its computation. With this new algorithm the computation of the median graph becomes linear with respect to the number of graphs. Preliminary results have shown that we obtain good approximations of the median graph even using a large number of graphs (up to 50).

- **Algorithms Based on a Particular Cost Function**

With the new theoretical properties based on a particular cost function and the reduced search space, we have developed two different approaches for the median graph computation.

- First, we have presented a new and more efficient exact algorithm. Preliminary experiments have shown that with this approach we can extend the exact computation of the median graph to real data, with a limited number of graphs. The results show that it not only improves the results of the previous exact algorithm but it can also be compared with one of the previous approximate algorithms.
- The second approach is based on genetic search. We have shown that with this new algorithm we can obtain good approximations of the median graph while keeping reasonable computation times. We have also shown its application to real data, with a limited number of graphs, but extending this number with respect to the exact algorithm.

- **Embedding-based Algorithms**

In the last part of this thesis we have proposed a new technique for the median graph computation based on graph embedding into vector spaces. In this new technique graphs are embedded into a vector space using the graph edit distance. Then, the median is computed in this vector space instead of doing that in the graph domain. Finally, the median graph is recovered from the median vector. With the embedding technique we have presented two new algorithms, one exact and one approximate.

- A new exact algorithm for the median graph computation has been introduced. With the embedding approach we have shown that the part of the search space that must be explored to obtain the median can be reduced even more. With this new reduction, we have presented the fastest exact algorithm for the median graph computation. With these new promising results the exact computation of the median graph have been extended to real machine learning algorithms.
- The approximate algorithm constitutes the more general and powerful option for the median graph computation. Experiments have shown its application to three different databases with a large number of graphs with large sizes. In spite of being an approximate algorithm we have shown the algorithm is able to obtain good median graphs.

8.1.3 Applications

Finally, we have used the median graph in two real graph-based applications: classification and clustering.

- **Graph-based Classification:** We have tested several of our algorithms for classification. In a first set of experiments we have used a limited amount of data and we have tested three of our new algorithms. All of them have been compared with the approximate embedding approach. The results have shown, in some cases, an improvement in the classification accuracy with respect to a classical nearest-neighbor classifier also reducing the number of comparisons. In a second stage, we have extended the classification tasks using a large amount of real data using the approximate embedding approach. The results have shown that the classification accuracy is very close to that obtained by the nearest-neighbor classifier but with a significant reduction in the number of comparisons. Nevertheless, we have also shown a second strategy, a more sophisticated classification scheme, that improves the results of the nearest-neighbor classifier and still keeps a lower number of comparisons.
- **Graph-based Clustering:** In the last part of this thesis and using the approximate embedding approach we have used the median graph to perform clustering tasks using the k -means approach. In particular we have used our approach to obtain the centers of the clusters. Traditionally, these centers have

been represented by the set median. This task is more challenging than classification since the medians are computed using elements from different classes which, in the median computation, can be seen as a form of noise. Results based on two standard measures (the *Rand Index* and the *Dunn Index*) have shown a certain improvement of the median graph over the set median graph. This results reinforces the thesis that the generalized median graph is a better representative than the set median graph.

8.2 Discussion

In this thesis we have made some contributions about the median graph in three different aspects: Theory, Algorithms and Applications. In the algorithmic part, we have presented five new algorithms, two exact and three approximate. All these solutions should not be seen as five separated boxes without any connection between them, but as an evolution from more limited methods (spectral approach) to more general methods (approximate embedding).

Thus, our first algorithm (spectral-based algorithm) is limited to operate only with weighted graphs with the same number of nodes. However, it has no constraints concerning the size of the graphs or the number of graphs in the set. After that, we overcame some of those limitations and we presented two new algorithms, one exact and one approximate, able to work with graphs with different sizes and with any kind of information both in the nodes and the edges, with the unique constraint that the information of the labels and edges must be of discrete nature. The only restriction was the dependency on the concept of the maximum common subgraph and the minimum common supergraph. This condition imposed to work with a low number of graphs and with low sizes. Nevertheless, we showed that even with these limitations the algorithms could be applied using real data. Finally, we went one step further and we presented a new technique based on graph embedding in vector spaces, and we introduced two more algorithms, one exact and one approximate. With the exact algorithm (still depending on the *mcs* and *MCS*) we were able to improve the previous results obtained with the exact algorithm, in terms of computation time. And finally, the approximate approach was shown as the more general and powerful approach for the median computation, applicable to a large number of graphs representing real data.

Nevertheless, the limitations in some of the methods should not be completely seen as a disadvantage. Conversely, this wide offer in the number of algorithms gives the opportunity to choose the best option (algorithm) in each case. That is, depending on the application one of these algorithms may perform better than the others. For instance, whenever applicable, exact algorithms are supposed to obtain always a better representative of a set of graphs.

In order to synthesize the properties and limitations of all the methods, we provide in Table 8.1 a brief summary of each of the five methods we have presented.

It is important to remark, as a final conclusion, that the median graph has been used for the first time in two classical machine learning algorithms using a large number of graphs with no restrictions in their labels or size.

Table 8.1: Number of graphs in each class.

<p>Method: <i>Spectral Median Graph</i></p> <p>Summary: Approximate. Based on Spectral Graph Theory.</p> <p>Restrictions: Only weighted graphs with the same number of nodes.</p>
<p>Method: <i>Exhaustive Search</i></p> <p>Summary: Exact. Based on a particular cost function.</p> <p>Restrictions: Computation of $g_M(S)$ and $g_m(S)$. Small number of graphs with small size and discrete labels.</p>
<p>Method: <i>Genetic Approach</i></p> <p>Summary: Approximate. Based on a particular cost function.</p> <p>Restrictions: Computation of $g_M(S)$ and $g_m(S)$. Small number of graphs with small size and discrete labels.</p>
<p>Method: <i>Exact Embedding</i></p> <p>Summary: Exact. Based on graph embedding into vector spaces, and on a particular cost function.</p> <p>Restrictions: Computation of $g_M(S)$ and $g_m(S)$. Small number of graphs with small size and discrete labels.</p>
<p>Method: <i>Approximate Embedding</i></p> <p>Summary: Approximate. Based on graph embedding into vector spaces and graph edit distance computation.</p> <p>Restrictions: Not restricted neither in the number of graphs, their size nor the label nature.</p>

8.3 Future Work

In spite of the advances on the median graph we have introduced, we believe there are still some open questions that can be further developed in the future. The following list summarizes such possible lines of continuation of this work:

- In some parts of this work it is assumed the median is computed under a particular cost function. In spite of the good properties of such cost function, it would be desirable to extend those results to other cost functions. In this sense, in [12] a deep study of error-tolerant graph matching under an extensive family of cost functions has been presented. A future work could be the extension of our results to that family of cost functions in order to give them a more general framework of application.

- Regarding the theoretical properties we have presented, we have empirical evidence that the relation between the median graph, the $mcs(S)$ and the $MCS(S)$ can be extended in the following way:

$$g_m(S) \subseteq \bar{g} \subseteq g_M(S)$$

That is, the median graph would be a subgraph of $g_M(S)$ including also the $g_m(S)$. Some preliminary experiments show that this conditions holds. This result can be seen as an extension of the result reported in [18], that shows this relation, but only with a set of two graphs. If confirmed, this result would make the computation of the median graph even easier and faster than now.

- The spectral median graph has the limitations that it can only be applied to graphs with the same number of nodes. Thus, a future research line is to extend this work in order to be able to deal with graphs with different sizes. In this sense, some good works related to the spectral graph theory such as [68, 23, 58] can be a good starting point to reach this objective.
- Maybe the most promising method for the median graph computation we have presented is the approximate embedding method. This approach is based on three steps: the embedding of the graphs into a vector space using the graph edit distance, the median vector computation and the conversion of the median vector to a graph. In all of these three steps some form of distortion is introduced. In order to improve the medians obtained, we can study some strategies to improve the two last steps:
 - We think that a deep research on different forms to obtain a representative in the vector domain, the mean vector for instance, would lead to improve the results of our method.
 - Another possibility to reduce the distortion introduced in the median computation is the investigation of new ways to go back to the graph domain. We have proposed the weighted mean of a pair of graphs for that purpose, but the research on new ways may improve the accuracy of the median.
- One of the results of this work is the empirical demonstration that the median graph is a good choice to obtain a representative of a set of graphs. The graph-based classification and clustering experiment reinforces this idea. In this sense, future work on the median graph could include:
 - Large-scale application of the median graph. That is, to extend the median graph computation to more sophisticated classification or clustering schemes. Eventually, the median graph computation should be tested not only in these frameworks but in any application where a representative of a set of graphs is needed.
 - With the extension of the median graph to a wide spectrum of applications, we can start thinking in the evaluation of the median graph as a representative of a set of graphs. That is, since the median graph is not

the unique option to obtain a representative of a set of graphs, a comparison with other methods could be useful to place the median within this context.

List of Publications

Journal Papers

- M. Ferrer, E. Valveny, F. Serratosa, K. Riesen and H. Bunke. "**Generalized Median Graph Computation by Means of Graph Embedding in Vector Spaces**". Submitted to *Journal of Machine Learning Research*.
- M. Ferrer, E. Valveny and F. Serratosa. "**Median Graphs: A Genetic Approach Based on New Theoretical Properties**". Submitted to *Pattern Recognition*.
- M. Ferrer, E. Valveny and F. Serratosa. "**Median Graphs: A New Exact Algorithm Based on the Maximum Common Subgraph**". Submitted to *Pattern Recognition Letters*.

Conference Contributions

- M. Ferrer, E. Valveny, F. Serratosa, K. Riesen and H. Bunke. "**An Approximate Algorithm for Median Graph Computation using Graph Embedding**". Submitted to *ICPR 2008*.
- D. Karatzas, M. Rusiñol, C. Antens and M. Ferrer. "**Segmentation Robust to the Vignette Effect for Machine Vision Systems**". Submitted to *ICPR 2008*.
- M. Ferrer, E. Valveny, F. Serratosa and H. Bunke. "**Exact Median Graph Computation via Graph Embedding**". Submitted to *SSPR 2008*.
- M. Ferrer, E. Valveny, F. Serratosa. "**Theoretical Advances on Median Graphs**". *2nd CVC Research and Development Internal Workshop*. October 19th, 2007.
- M. Ferrer, E. Valveny. "**Combination of OCR Engines for Page Segmentation Based on Performance Evaluation**". *9th International Conference on Document Analysis and Recognition, ICDAR*, Curitiba (Brazil), September 2007. pp. 784-788

- M. Ferrer, E. Valveny, F. Serratosa. "**Bounding the Size of the Median Graph**". *3rd Iberian Conference on Pattern Recognition and Image Analysis, IbPRIA 2007*. pp. 491-498, LNCS 4478.
- M. Ferrer, E. Valveny, F. Serratosa. "**Comparison Between two Spectral-based Methods for the Median Graph Computation**". *3rd Iberian Conference on Pattern Recognition and Image Analysis, IbPRIA 2007*. pp. 580-587, LNCS 4478.
- M. Ferrer, F. Serratosa, E. Valveny. "**On the Relation Between the Median Graph and the Maximum Common Subgraph of a Set of Graphs**". *6th IAPR-TC15 Workshop on Graph-Based Representations in Pattern Recognition, GbR 2007*. pp. 351-360, LNCS 4538.
- M. Ferrer, E. Valveny, F. Serratosa. "**Spectral Median Graphs and its Application to Graphical Symbol Recognition**". *1st CVC Research and Development Internal Workshop*. October 6th, 2006.
- M. Ferrer, E. Valveny, F. Serratosa. "**Spectral Median Graphs Applied to Graphical Symbol Recognition**". *XI Iberoamerican Congress on pattern Recognition, CIARP 2006*, Cancun, Mexico. Lecture Notes in Computer Science, vol 4225, pp.774-783. Springer-Verlag, 2006.
- M. Ferrer, F. Serratosa, A. Sanfeliu. "**Synthesis of Median Spectral Graph**". *2nd Iberian Conference on Pattern Recognition and Image Analysis, IbPRIA 2005*, Estoril, Portugal. Lecture Notes in Computer Science, vol 3523, pp.139-146. Springer-Verlag, 2005.

Technical Reports

- M. Ferrer, E. Valveny and F. Serratosa. "**A New Optimal Algorithm for the Generalized Median Graph Computation Based on the Maximum Common Subgraph**". *CVC Technical Report* Num. 109. July 2007.
- M. Ferrer. "**Spectral Median Graphs and its Application to Graphical Symbol Recognition**". *CVC Technical Report* Num. 95. July 2006.

Appendix A

Databases

Along this work, several databases have been used, namely the *Letter Graph Dataset*, the *GREC dataset* [34], the *Molecule Dataset* [1] and the *Webpage Dataset* [89]. In this appendix we will explain in detail all of these databases. For each one we will explain the kind of data that composes the database, the graph-based representation of such data and some relevant characteristics such as the number of objects in the database, the maximum number of nodes in the graphs, etc. In addition, due to the restrictions imposed by some of the presented methods for the median graph computation, for some experiments we have used variations of these datasets, basically applying different distortions to the data. These variations are also explained in this appendix. Finally, it is important to mention that the two first databases contain synthetic data whereas the two last databases are composed of real world data.

A.1 Letter Database

The Letter dataset was originally created at the University of Bern. It is composed of 15 capital letters (classes) of the Roman alphabet (only those which are composed of straight lines), which are *A, E, F, H, I, K, L, M, N, T, V, W, X, Y* and *Z*. For each class, an original prototype was constructed in a manual fashion. An illustration of these prototypes is shown in Figure A.1.

The letters are represented by graphs as follows. The straight lines are represented by edges and the terminal points of the lines by the nodes. Nodes are labelled by a two-dimensional attribute that represents the position (x,y) of the terminal point in the plane. Edges have a one-dimensional and binary attribute that represents the existence or non-existence of a line between two terminal points. Graph-based representations of the prototypes are shown in Figure A.2.

A.1.1 Variation Letter-1

From the original set composed of 15 letters, we only used a subset of 6 letters namely *L, V, N, T, K* and *M*. Then, from each original model, we manually generated 4 distorted instances. Therefore, this variation is composed of 30 elements and 6

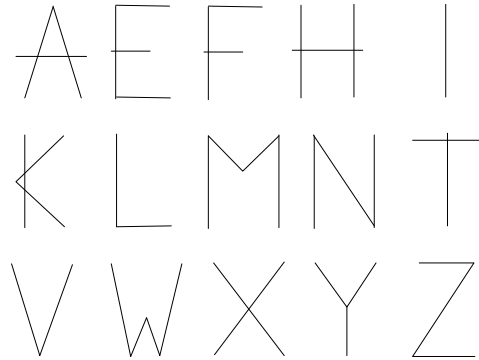


Figure A.1: Original prototypes for the Letter dataset.

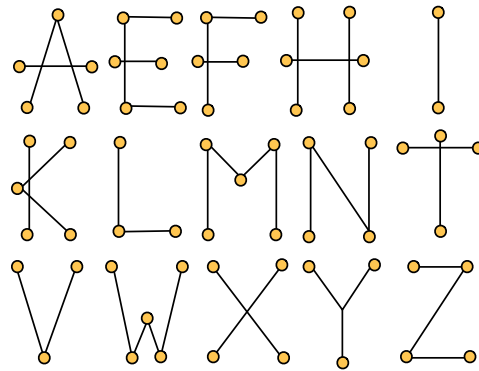


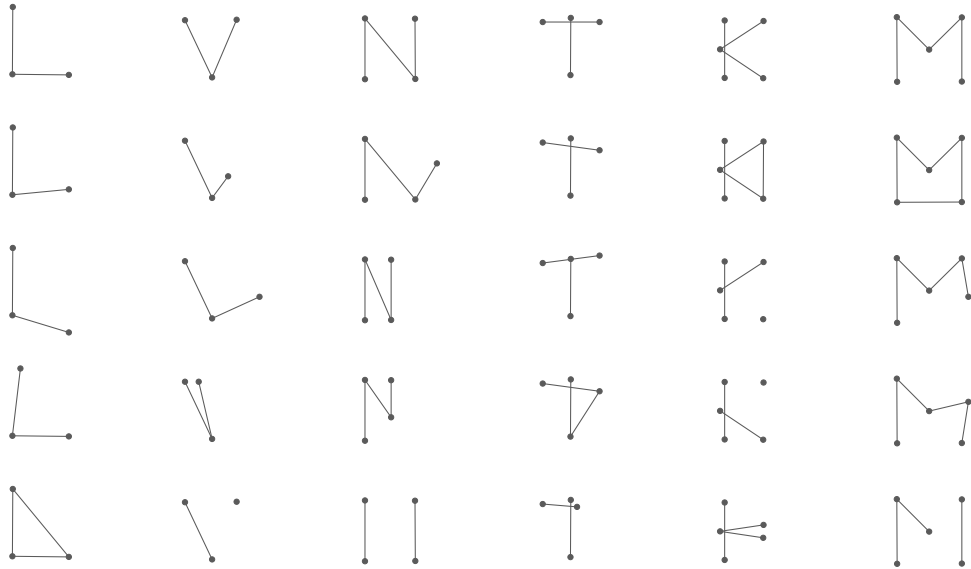
Figure A.2: Graph-based representations of the original prototypes.

classes. Each class represents a letter and contains 5 different instances of the letter. Table A.1 shows the original letters in the first row and the four distorted letters in the rest of the rows. Notice that, in the distorted letters, some lines have been erased or moved, but the number of terminal points has been kept unchanged. Note that there are two classes (L and V) whose elements have 3 nodes each, two classes (N and T) whose elements have four nodes each and two classes (K and M) whose elements have 5 nodes each.

The number of graphs in this variation and the maximum and the mean number of nodes in the graphs are shown in Table A.2.

A.2 GREC 2005 Database

The GREC database is composed of a set of graphical symbols coming from different technical fields. It was originally created for the GREC contest.

Table A.1: Entire Letter-1 database.

The GREC Contest

The main goal of the contest is to provide a framework for the evaluation of different methods for symbol recognition in graphic documents. This framework is intended to be general and flexible so that it can be used to evaluate a wide range of symbol recognition methods. The contest is based on a pre-defined set of symbols (tables A.3 to A.5). Using this set of symbols, different tests were generated, consisting of several images of each symbol with increasing levels of degradation and distortion. In the 2005 edition, several tests were available to the participants through the web site (<http://www.cs.cityu.edu.hk/grec2005/>). These tests could be used as a learning set to train the methods participating in the contest. For real contest execution, other

Table A.2: Some characteristics of Letter-1 dataset.

Property	Value
<i>Number of classes</i>	6
<i>Total number of elements</i>	30 (5 elements/class)
<i>Max. number of nodes in a graph</i>	5
<i>Min. number of nodes in a graph</i>	3
<i>Mean number of nodes</i>	4

tests, similar to the sample tests, containing different images were generated. As a final result of the contest, several tables were provided, one for each kind of test, showing the performance of the methods taking part in those tests. Performance was measured in terms of recognition accuracy.

Description of Symbols

The symbols of the GREC database are extracted from different technical fields such as architecture and electronics. All these symbols have common features, that can be summarized as follows:

- They come from two different domains such as architecture and electronics
- They are composed of two types of primitives: lines and arcs
- They do not contain filled regions, text or any other type of primitive

The complete collection of 150 symbols can be seen in Tables A.3 to A.5.

These images are converted into graphs by assigning a node to each junction or terminal point and an edge to each line. The labels for the nodes are coordinates in a 2-dimensional space corresponding to the location of the point. An edge exists between two given nodes if there is a line between the corresponding terminal points. Edges are labeled with a number indicating its existence. Figure A.3 shows an example of the graph-based representation of a symbol.

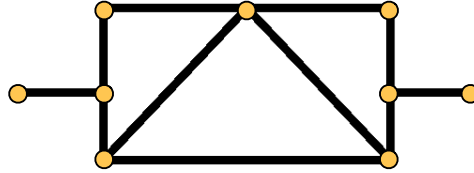


Figure A.3: Graph-based representation of a GREC symbol.

A.2.1 Variation GREC-1

In this variation we have used a subset of 32 different symbols of the original database. All these symbols are composed only of straight lines. In order to work with a large dataset, with arbitrarily strong distortions, we have generated 100 instances of each symbol by moving the junction or terminal points between two lines within a pre-defined radius r . Notice that this distortion keeps unchanged the number of nodes within a class. That is, all the generated instances of a certain class have the same number of nodes. Figure A.4 shows an example of two symbols and different distorted instances of them.







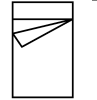
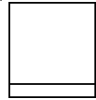
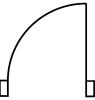
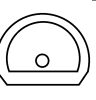
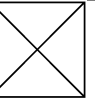
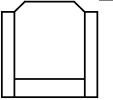
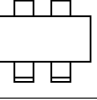
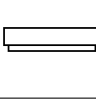
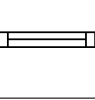
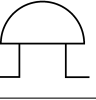

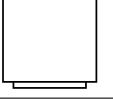





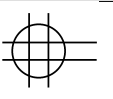
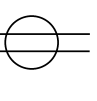
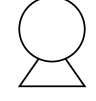
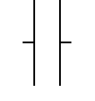
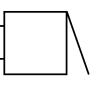
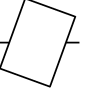
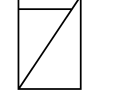





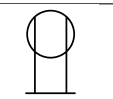





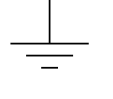

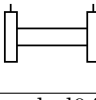
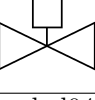
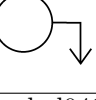
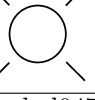
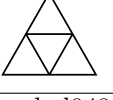
					
symbol001	symbol002	symbol003	symbol004	symbol005	symbol006
					
symbol007	symbol008	symbol009	symbol010	symbol011	symbol012
					
symbol013	symbol014	symbol015	symbol016	symbol017	symbol018
					
symbol019	symbol020	symbol021	symbol022	symbol023	symbol024
					
symbol025	symbol026	symbol027	symbol028	symbol029	symbol030
					
symbol031	symbol032	symbol033	symbol034	symbol035	symbol036
					
symbol037	symbol038	symbol039	symbol040	symbol041	symbol042
					
symbol043	symbol044	symbol045	symbol046	symbol047	symbol048

Table A.3: GREC 2005 Database (1)

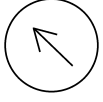
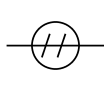
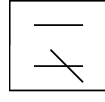
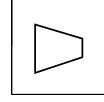
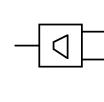
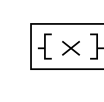
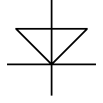
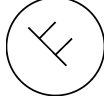
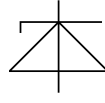
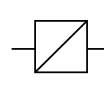
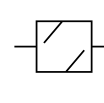
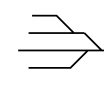
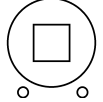
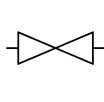
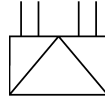
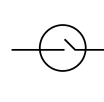
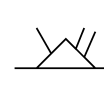
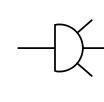
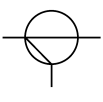
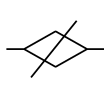
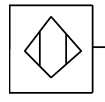
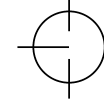
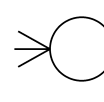
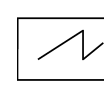

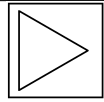
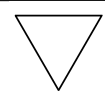
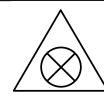
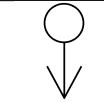
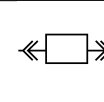
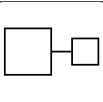
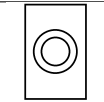
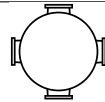
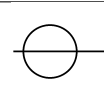
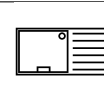
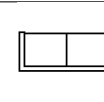
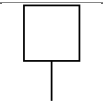
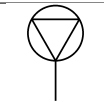
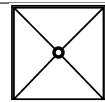
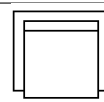
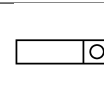
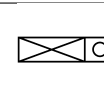
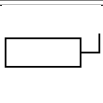
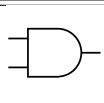
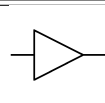
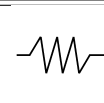
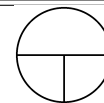
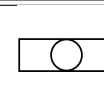
					
symbol049	symbol050	symbol051	symbol052	symbol053	symbol054
					
symbol055	symbol056	symbol057	symbol058	symbol059	symbol060
					
symbol061	symbol062	symbol063	symbol064	symbol065	symbol066
					
symbol067	symbol068	symbol069	symbol070	symbol071	symbol072
					
symbol073	symbol074	symbol075	symbol076	symbol077	symbol078
					
symbol079	symbol080	symbol081	symbol082	symbol083	symbol084
					
symbol085	symbol086	symbol087	symbol088	symbol089	symbol090
					
symbol091	symbol092	symbol093	symbol094	symbol095	symbol096

Table A.4: GREC 2005 Database (2)

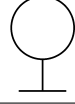
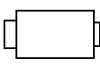


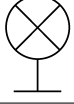
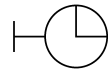
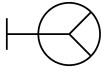
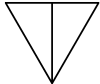

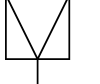
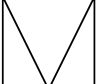
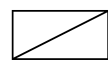
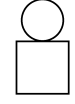
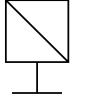

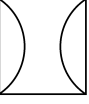


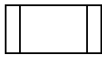

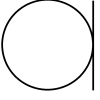
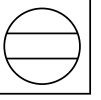
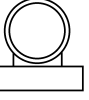
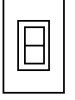
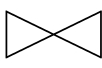

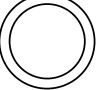
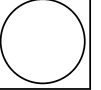
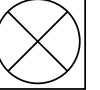







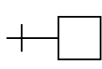

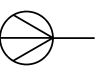



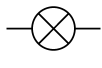
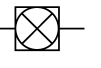
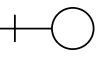

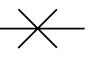

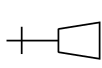
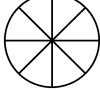
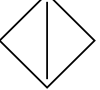
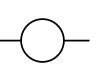
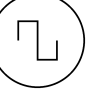

					
symbol097	symbol098	symbol099	symbol100	symbol101	symbol102
					
symbol103	symbol104	symbol105	symbol106	symbol107	symbol108
					
symbol109	symbol110	symbol111	symbol112	symbol113	symbol114
					
symbol115	symbol116	symbol117	symbol118	symbol119	symbol120
					
symbol121	symbol122	symbol123	symbol124	symbol125	symbol126
					
symbol127	symbol128	symbol129	symbol130	symbol131	symbol132
					
symbol133	symbol134	symbol135	symbol136	symbol137	symbol138
					
symbol139	symbol140	symbol141	symbol142	symbol143	symbol144
					
symbol145	symbol146	symbol147	symbol148	symbol149	symbol150

Table A.5: GREC 2005 Database (3)

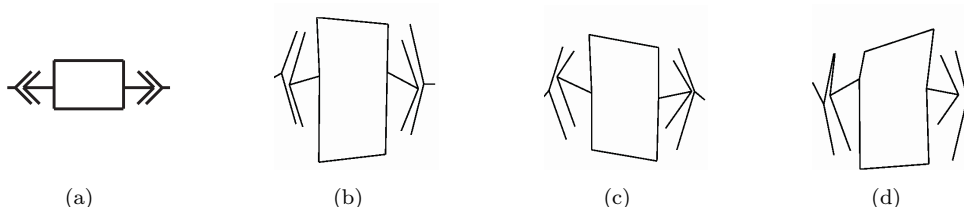


Figure A.4: Example of distorted symbols.

The number of graphs in this variation and the maximum and the mean number of nodes in the graphs are shown in Table A.6. Note that within these classes the number of nodes of each graph ranges from 4 to 28.

Table A.6: Some characteristics of GREC-1 dataset.

Property	Value
<i>Number of classes</i>	32
<i>Total number of elements</i>	3,200 (100 elements/class)
<i>Max. number of nodes in a graph</i>	28
<i>Min. number of nodes in a graph</i>	4
<i>Mean number of nodes</i>	8.8

A.2.2 Variation GREC-2

In this variation we have used the same subset of 32 different symbols as in the GREC-1 database. In order to work with a large dataset, with arbitrarily strong distortions, we have generated 50 instances of each symbol applying different distortion operators to the original images. Such distortion operators include moving the junction points between two lines within a predefined radius r , splitting junction points and deleting some lines. Notice that in this case, two elements in the same class may have different number of nodes and edges. Figure A.5 shows an example of two symbols and different distorted instances of them.

The number of graphs in this variation and the maximum and the mean number of nodes in the graphs are shown in Table A.7. Note that within these classes the number of nodes of each graph ranges from 3 to 38.

A.3 Molecule Database

The molecule database consists in graphs representing molecular compounds. These graphs are extracted from the AIDS Antiviral Screen Database of Active Compounds [1]. Such database consists of two different classes of molecules: active and inactive, depending on whether they show activity against HIV or not respectively. The

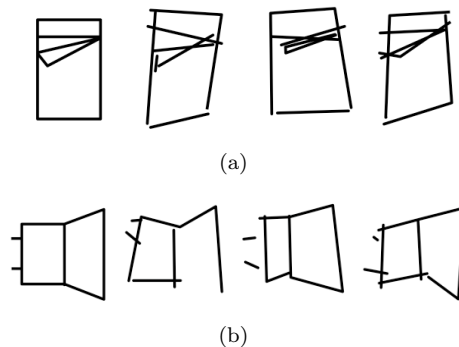


Figure A.5: Examples of GREC symbols with some distortions. (a) Architectural symbol (b) Electrical symbol.

Table A.7: Some characteristics of GREC-2 dataset.

Property	Value
<i>Number of classes</i>	32
<i>Total number of elements</i>	1,600 (50 elements/class)
<i>Max. number of nodes in a graph</i>	38
<i>Min. number of nodes in a graph</i>	3
<i>Mean number of nodes</i>	11.2

molecules are converted into graphs in a straightforward way, representing its atoms as nodes and its covalent bonds as edges. The nodes are labelled with the number of the corresponding chemical symbol. The edges are labelled with the valence of the linkage. Some examples of each class are represented in Figure A.6. In order to simplify the representation, in such examples, different chemical symbols are represented using different gray tonality.

In this database there is a total number of 1,500 graphs, 300 corresponding to active molecules and 1,200 to inactive ones. Table A.8 shows some characteristics of this database.

Table A.8: Some characteristics of Molecule dataset.

Property	Value
<i>Number of classes</i>	2
<i>Total number of elements</i>	1,500 (200+1,300)
<i>Max. number of nodes in a graph</i>	63
<i>Min. number of nodes in a graph</i>	1
<i>Mean number of nodes</i>	9.5

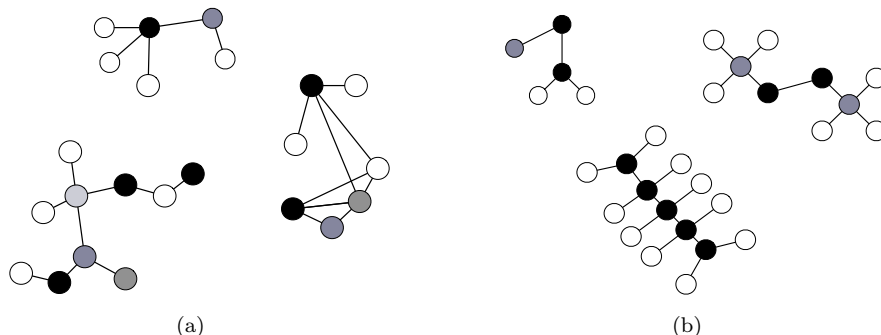


Figure A.6: Example of some active compounds (a) and some inactive compounds (b)

A.4 Webpage Database

The last database is composed of a collection of webpages. In [89] several methods to create graph-based representations of webpages are introduced. In our case, graphs representing webpages are constructed as follows. First, all words appearing in the web document are converted into nodes in the web graph, except for stop words which contain little information. The nodes are attributed with the corresponding word and its frequency. That is, even if a word appears more than once in the web document, only one node with this word label is added to the graph, and the frequency of the word is used as an additional attribute. Then, if a word w_i immediately precedes another word w_j in the document, a directed edge from the node corresponding to the word w_i to the node corresponding to the word w_j is added to the graph. In order to keep the essential information of the document, only the most frequently used words (nodes) are kept in the graph and the terms are combined to the most frequently occurring form. Figure A.7 shows an example of a webgraph.

The dataset is composed of 2,340 documents belonging to 20 different categories (Business, Health, Politics, Sports, Technology, Entertainment, Art, Cable, Culture, Film, Industry, Media, Multimedia, Music, Online, People, Review, Stage, Television and Variety). The last 14 categories are sub-categories of Entertainment. These web documents were originally hosted at Yahoo as news pages (<http://www.yahoo.com>). For simplicity, from now on, the 6 main classes will be referred as B, H, P, S, T and E for Business, Health, Politics, Sports, Technology and Entertainment, respectively.

Note that, not all the classes have the same number of graphs. Table A.9 shows the number of graphs in each class.

Table A.9: Number of graphs in each class.

	Class					
	B	E	H	P	S	T
Number of graphs	142	1,389	494	114	141	60

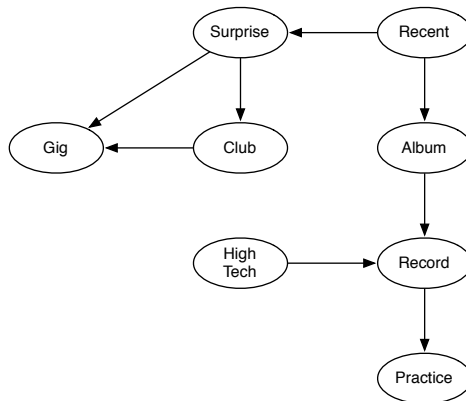


Figure A.7: Webgraph example.

Table A.10 shows some characteristics of this database.

Table A.10: Some characteristics of Webpage dataset.

Property	Value
<i>Number of classes</i>	6
<i>Total number of elements</i>	2,340 (distribution shown in Table A.9)
<i>Max. number of nodes in a graph</i>	834
<i>Min. number of nodes in a graph</i>	43
<i>Mean number of nodes</i>	180.06

Acknowledgment

We would like to thank Kaspar Riesen from the University of Bern for his help with the databases and to make them available to us.

Bibliography

- [1] Development Therapeutics Program DTP. AIDS Antiviral Screen. http://dtp.nci.nih.gov/docs/aids/aids_data.html, 2004.
- [2] Robert Allen, Luigi Cinque, Steven L. Tanimoto, Linda G. Shapiro, and Dean Yasuda. A parallel algorithm for graph matching and its maspar implementation. *IEEE Trans. Parallel Distrib. Syst.*, 8(5):490–501, 1997.
- [3] Surapong Auwatanamongkol. Inexact graph matching using a genetic algorithm for image recognition. *Pattern Recognition Letters*, 28(12):1428–1437, 2007.
- [4] Chanderrjit Bajaj. The algebraic degree of geometric optimization problems. *Discrete Comput. Geom.*, 3(2):177–191, 1988.
- [5] Egon Balas and Chang Sung Yu. Finding a maximum clique in an arbitrary graph. *SIAM J. Comput.*, 15(4):1054–1068, 1986.
- [6] Stefano Berretti, Alberto Del Bimbo, and Enrico Vicario. The computational aspect of retrieval by spatial arrangement. In *15th International Conference on Pattern Recognition*, pages 5047–5051, 2000.
- [7] Stefano Berretti, Alberto Del Bimbo, and Enrico Vicario. A look-ahead strategy for graph matching in retrieval by spatial arrangement. In *IEEE International Conference on Multimedia and Expo (III)*, pages 1721–1724, 2000.
- [8] Stefano Berretti, Alberto Del Bimbo, and Enrico Vicario. Efficient matching and indexing of graph models in content-based retrieval. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(10):1089–1105, 2001.
- [9] C. Bron and J. Kerbosch. Finding all the cliques in an undirected graph. *Communication of the ACM*, 16:189–201, 1973.
- [10] Horst Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters*, 18(8):689–694, 1997.
- [11] Horst Bunke. Error-tolerant graph matching: A formal framework and algorithms. In *SSPR '98/SPR '98: Proceedings of the Joint IAPR International Workshops on Advances in Pattern Recognition*, pages 1–14, London, UK, 1998. Springer-Verlag.

- [12] Horst Bunke. Error correcting graph matching: On the influence of the underlying cost function. *IEEE Trans. Pattern Anal. Mach. Intell.*, 21(9):917–922, 1999.
- [13] Horst Bunke and G Allerman. Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters*, 1(4):245–253, 1983.
- [14] Horst Bunke, Pasquale Foggia, C. Guidobaldi, Carlo Sansone, and Mario Vento. A comparison of algorithms for maximum common subgraph on randomly connected graphs. In *Structural, Syntactic, and Statistical Pattern Recognition, Joint IAPR International Workshops SSPR 2002 and SPR 2002, Windsor, Ontario, Canada, August 6-9, 2002, Proceedings. Lecture Notes in Computer Science Vol. 2396*, pages 123–132, 2002.
- [15] Horst Bunke, Pasquale Foggia, C. Guidobaldi, and Mario Vento. Graph clustering using the weighted minimum common supergraph. In *Graph Based Representations in Pattern Recognition, 4th IAPR International Workshop, GbRPR 2003, York, UK, June 30 - July 2, 2003, Proceedings. Lecture Notes in Computer Science Vol. 2726*, pages 235–246, 2003.
- [16] Horst Bunke and Simon Günter. Weighted mean of a pair of graphs. *Computing*, 67(3):209–224, 2001.
- [17] Horst Bunke, Xiaoyi Jiang, and Abraham Kandel. On the minimum common supergraph of two graphs. *Computing*, 65(1):13–25, 2000.
- [18] Horst Bunke and Abraham Kandel. Mean and maximum common subgraph of two graphs. *Pattern Recognition Letters*, 21(2):163–168, 2000.
- [19] Horst Bunke, Andreas Münger, and Xiaoyi Jiang. Combinatorial search versus genetic algorithms: A case study based on the generalized median graph problem. *Pattern Recognition Letters*, 20(11-13):1271–1277, 1999.
- [20] Horst Bunke and Kim Shearer. A graph distance metric based on the maximal common subgraph. *Pattern Recognition Letters*, 19(3-4):255–259, 1998.
- [21] Terry Caelli and Serhiy Kosinov. An eigenspace projection clustering method for inexact graph matching. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(4):515–519, 2004.
- [22] Virginio Cantoni, Luigi Cinque, Concettina Guerra, Stefano Levialdi, and Luca Lombardi. 2-D object recognition by multiscale tree matching. *Pattern Recognition*, 31(10):1443–1454, 1998.
- [23] Marco Carcassoni and E. R. Hancock. Weighted graph-matching using modal clusters. In *CAIP '01: Proceedings of the 9th International Conference on Computer Analysis of Images and Patterns*, pages 142–151, London, UK, 2001. Springer-Verlag.

- [24] William J. Christmas, Josef Kittler, and Maria Petrou. Structural matching in computer vision using probabilistic relaxation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 17(8):749–764, 1995.
- [25] Fan R. K. Chung. *Spectral Graph Theory*. AMS Publications, 1997.
- [26] Donatello Conte, Pasquale Foggia, Carlo Sansone, and Mario Vento. Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(3):265–298, 2004.
- [27] Donatello Conte, Pasquale Foggia, and Mario Vento. Challenging complexity of maximum common subgraph detection algorithms: A performance analysis of three algorithms on a wide database of graphs. *Journal of Graph Algorithms and Applications*, 11(1):99–143, 2007.
- [28] Luigi P. Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. Fast graph matching for detecting cad image components. In *15th International Conference on Pattern Recognition*, pages 6034–6037, 2000.
- [29] Luigi P. Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. Learning structural shape descriptions from examples. *Pattern Recognition Letters*, 23(12):1427–1437, 2002.
- [30] Andrew D. J. Cross, Richard C. Wilson, and Edwin R. Hancock. Inexact graph matching using genetic search. *Pattern Recognition*, 30(6):953–970, 1997.
- [31] D. M. Cvetkovic, M. Doob, and H. Sachs. *Spectra of graphs, theory and applications*. Academic Press, 1980.
- [32] Colin de la Higuera and Francisco Casacuberta. Topology of strings: Median string is NP-complete. *Theor. Comput. Sci.*, 230(1-2):39–48, 2000.
- [33] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1977.
- [34] Philippe Dosch and Ernest Valveny. Report on the second symbol recognition contest. In Wenyin Liu and Josep Lladós, editors, *GREC*, volume 3926 of *Lecture Notes in Computer Science*, pages 381–397. Springer, 2005.
- [35] R. Duda, P. Hart, and D. Stork. *Pattern Classification*. Wiley Interscience. 2nd Edition, 2000.
- [36] 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 *11th International Conference on Pattern Recognition*, pages III:439–442, 1992.
- [37] Paul J. Durand, Rohit Pasari, Johnnie W. Baker, and Chun che Tsai. An efficient algorithm for similarity analysis of molecules. *Internet Journal of Chemistry*, 2(17), 1999.

- [38] M. A. Eshera and K. S. Fu. A graph distance measure for image analysis. *IEEE Transactions on Systems, Man and Cybernetics*, 14:398–408, 1984.
- [39] Mirtha-Lina Fernández and Gabriel Valiente. A graph distance metric combining maximum common subgraph and minimum common supergraph. *Pattern Recognition Letters*, 22(6/7):753–758, 2001.
- [40] M. Friedman and A. Kandel. *Introduction to Pattern Recognition*. World Scientific, 1999.
- [41] Steven Gold and Anand Rangarajan. A graduated assignment algorithm for graph matching. *IEEE Trans. Pattern Anal. Mach. Intell.*, 18(4):377–388, 1996.
- [42] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 3rd Edition, Baltimore, Maryland, 1996.
- [43] Simon Günter and Horst Bunke. Self-organizing map for clustering in the graph domain. *Pattern Recognition Letters*, 23(4):405–417, 2002.
- [44] S. L. Hakimi. *Location Theory*. CRC Press., 2000.
- [45] Adel Hlaoui and Shengrui Wang. A new median graph algorithm. In *Graph Based Representations in Pattern Recognition, 4th IAPR International Workshop, GbRPR 2003, York, UK, June 30 - July 2, 2003, Proceedings. Lecture Notes in Computer Science Vol. 2726*, pages 225–234, 2003.
- [46] Adel Hlaoui and Shengrui Wang. Median graph computation for graph clustering. *Soft Comput.*, 10(1):47–53, 2006.
- [47] J. E. Hopcroft and J. K. Wong. Linear time algorithm for isomorphism of planar graphs (preliminary report). In *STOC '74: Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 172–184, New York, NY, USA, 1974. ACM Press.
- [48] Benoit Huet and Edwin R. Hancock. Relational object recognition from large structural libraries. *Pattern Recognition*, 35(9):1895–1915, 2002.
- [49] M. Budunuch. P. M. Paralos I. M. Bomze and M. Pelillo. *The Maximum Clique Problem*. Kluwer Academic Publisher, 1999.
- [50] Anil K. Jain, M. Narasimha Murty, and Patrick J. Flynn. Data clustering: A review. *ACM Comput. Surv.*, 31(3):264–323, 1999.
- [51] Xiaoyi Jiang, Andreas Münger, and Horst Bunke. Computing the generalized median of a set of graphs. In *Proceedings of the 2nd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition*, pages 115–124, 1999.
- [52] Xiaoyi Jiang, Andreas Münger, and Horst Bunke. Synthesis of representative graphical symbols by computing generalized median graph. In Atul K. Chhabra and Dov Dori, editors, *GREC*, volume 1941 of *Lecture Notes in Computer Science*, pages 183–192. Springer, 1999.

- [53] Xiaoyi Jiang, Andreas Munger, and Horst Bunke. On median graphs: Properties, algorithms, and applications. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(10):1144–1151, 2001.
- [54] Kenneth A. De Jong and William M. Spears. Using genetic algorithms to solve NP-complete problems. In J. David Schaffer, editor, *ICGA*, pages 124–132. Morgan Kaufmann, 1989.
- [55] Alfons Juan and Enrique Vidal. Fast median search in metric spaces. In *SSPR '98/SPR '98: Proceedings of the Joint IAPR International Workshops on Advances in Pattern Recognition*, pages 905–912, London, UK, 1998. Springer-Verlag.
- [56] Derek Justice and Alfred O. Hero. A binary linear programming formulation of the graph edit distance. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(8):1200–1214, 2006.
- [57] A. Kandel, H. Bunke, and M. Last. *Applied Graph Theory in Computer Vision and Pattern Recognition (Series in Computational Intelligence)*. Springer-Verlag, Berlin, 2007.
- [58] Serhiy Kosinov and Terry Caelli. Inexact multisubgraph matching using graph eigenspace and clustering models. In *Structural, Syntactic, and Statistical Pattern Recognition, Joint IAPR International Workshops SSPR 2002 and SPR 2002, Windsor, Ontario, Canada, August 6-9, 2002, Proceedings. Lecture Notes in Computer Science Vol. 2396*, pages 133–142, 2002.
- [59] Evgeny B. Krissinel and Kim Henrick. Common subgraph isomorphism detection by backtracking search. *Softw., Pract. Exper.*, 34(6):591–607, 2004.
- [60] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, (2):83–97, 1955.
- [61] D. C. Lay. *Linear Algebra and Its Applications*. Addison Wesley, 3rd Edition, 2005.
- [62] S.W. Lee, J.H. Kim, and F.C.A. Groen. Translation-, rotation-, and scale invariant recognition of hand-drawn symbols in schematic diagrams. *International Journal of Pattern Recognition and Artificial Intelligence*, 4:1–15, 1990.
- [63] Josep Llados, Enric Martı, and Juan Jose Villanueva. Symbol recognition by error-tolerant subgraph matching between region adjacency graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(10):1137–1143, 2001.
- [64] C. Sansone L.P Cordella, P. Foggia and M. Vento. An improved algorithm for matching large graphs. In *Proc. 3rd IAPR Workshop Graph-Based Representations in Pattern Recognition*, pages 149–159, 2001.
- [65] Si Wei Lu, Ying Ren, and Ching Y. Suen. Hierarchical attributed graph representation and recognition of handwritten chinese characters. *Pattern Recognition*, 24(7):617–632, 1991.

- [66] E. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of Computer and System Sciences*, 25:42–65, 1982.
- [67] B. Luo, A. Robles-Kelly, A. Torsello, R. C. Wilson, and E. Hancock. Clustering shock trees. In *Proc. 3rd IAPR Workshop Graph-Based Representations in Pattern Recognition*, pages 217–228, 2001.
- [68] Bin Luo and Edwin R. Hancock. Structural graph matching using the EM algorithm and singular value decomposition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(10):1120–1136, 2001.
- [69] Bin Luo, Richard C. Wilson, and Edwin R. Hancock. Spectral embedding of graphs. *Pattern Recognition*, 36(10):2213–2230, 2003.
- [70] James J. McGregor. Backtrack search algorithms and the maximal common subgraph problem. *Software - Practice and Experience*, 12(1):23–24, 1982.
- [71] B.D. McKay. Practical graph isomorphism. *Congressus Numerantium*, 30:45–87, 1981.
- [72] Bruno T. Messmer and Horst Bunke. A new algorithm for error-tolerant subgraph isomorphism detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(5):493–504, 1998.
- [73] M. Mitchel. *An Introduction to Genetic Algorithms*. MIT Press., 1996.
- [74] T. M. Mitchell. *Machine Learning*. McGraw-Hill., 1997.
- [75] Bojan Mohar. The laplacian spectrum of graphs. *Graph Theory, Combinatorics and Applications*, 2:871–898, 1991.
- [76] Andreas Munger. Synthesis of prototype graphs from sample graphs. In *Diploma Thesis, University of Bern (in German)*, 1998.
- [77] Michel Neuhaus and Horst Bunke. An error-tolerant approximate matching algorithm for attributed planar graphs and its application to fingerprint classification. In Ana L. N. Fred, Terry Caelli, Robert P. W. Duin, Aurelio C. Campilho, and Dick de Ridder, editors, *SSPR/SPR*, volume 3138 of *Lecture Notes in Computer Science*, pages 180–189. Springer, 2004.
- [78] Michel Neuhaus, Kaspar Riesen, and Horst Bunke. Fast suboptimal algorithms for the computation of graph edit distance. In *Structural, Syntactic, and Statistical Pattern Recognition, Joint IAPR International Workshops, SSPR 2006 and SPR 2006, Hong Kong, China, August 17-19, 2006, Proceedings. Lecture Notes in Computer Science 4109*, pages 163–172, 2006.
- [79] Elzbieta Pekalska, Robert P. W. Duin, and Pavel Paclık. Prototype selection for dissimilarity-based classifiers. *Pattern Recognition*, 39(2):189–208, 2006.
- [80] Marcello Pelillo, Kaleem Siddiqi, and Steven W. Zucker. Matching hierarchical structures using association graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 21(11):1105–1120, 1999.

- [81] John W. Raymond and Peter Willett. Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *Journal of Computer-Aided Molecular Design*, 16(7):521–533, 2002.
- [82] Kaspar Riesen, Michel Neuhaus, and Horst Bunke. Bipartite graph matching for computing the edit distance of graphs. In Francisco Escolano and Mario Vento, editors, *Graph-Based Representations in Pattern Recognition, 6th IAPR-TC-15 International Workshop, GbRPR 2007, Alicante, Spain, June 11-13, 2007, Proceedings*, volume 4538 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2007.
- [83] Kaspar Riesen, Michel Neuhaus, and Horst Bunke. Graph embedding in vector spaces by means of prototype selection. In Francisco Escolano and Mario Vento, editors, *Graph-Based Representations in Pattern Recognition, 6th IAPR-TC-15 International Workshop, GbRPR 2007, Alicante, Spain, June 11-13, 2007, Proceedings*, volume 4538 of *Lecture Notes in Computer Science*, pages 383–393. Springer, 2007.
- [84] Antonio Robles-Kelly and Edwin R. Hancock. Graph edit distance from spectral seriation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(3):365–378, 2005.
- [85] Antonio Robles-Kelly and Edwin R. Hancock. A Riemannian approach to graph embedding. *Pattern Recognition*, 40(3):1042–1056, 2007.
- [86] Jairo Rocha and Theodosios Pavlidis. A shape analysis model with applications to a character recognition system. *IEEE Trans. Pattern Anal. Mach. Intell.*, 16(4):393–404, 1994.
- [87] A. Sanfeliu and K.S. Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE Transactions on Systems, Man and Cybernetics*, 13(3):353–362, May 1983.
- [88] Alberto Sanfeliu, René Alquézar, and Francesc Serratosa. Clustering of attributed graphs and unsupervised synthesis of function-described graphs. In *15th International Conference on Pattern Recognition*, pages 6022–6025, 2000.
- [89] A. Schenker, H. Bunke, M. Last, and A. Kandel. *Graph-Theoretic Techniques for Web Content Mining (Machine Perception and Artificial Intelligence) (Series in Machine Perception and Artificial Intelligence)*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2005.
- [90] Douglas C. Schmidt and Larry E. Druffel. A fast backtracking algorithm to test directed graphs for isomorphism using distance matrices. *J. ACM*, 23(3):433–445, 1976.
- [91] Kuntal Sengupta and Kim L. Boyer. Organizing large structural modelbases. *IEEE Trans. Pattern Anal. Mach. Intell.*, 17(4):321–332, 1995.
- [92] D.S. Seong, H.S. Kim, and K.H. Park. Incremental clustering of attributed graphs. *T-SMC*, 23:1399–1411, 1993.

- [93] F. Serratos, R. Alquezar, 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.
- [94] Francesc Serratos, René Alquézar, and Alberto Sanfeliu. Estimating the joint probability distribution of random vertices and arcs by means of second-order random graphs. In *Structural, Syntactic, and Statistical Pattern Recognition, Joint IAPR International Workshops SSPR 2002 and SPR 2002, Windsor, Ontario, Canada, August 6-9, 2002, Proceedings. Lecture Notes in Computer Science Vol. 2396*, pages 252–262, 2002.
- [95] Francesc Serratos, René Alquézar, and Alberto Sanfeliu. Synthesis of function-described graphs and clustering of attributed graphs. *International Journal of Pattern Recognition and Artificial Intelligence*, 16(6):621–656, 2002.
- [96] Francesc Serratos, René Alquézar, and Alberto Sanfeliu. Function-described graphs for modelling objects represented by sets of attributed graphs. *Pattern Recognition*, 36(3):781–798, 2003.
- [97] Kim Shearer, Horst Bunke, and Svetha Venkatesh. Video indexing and similarity retrieval by largest common subgraph detection using decision trees. *Pattern Recognition*, 34(5):1075–1091, 2001.
- [98] Ali Shokoufandeh and Sven J. Dickinson. A unified framework for indexing and matching hierarchical shape structures. In Carlo Arcelli, Luigi P. Cordella, and Gabriella Sanniti di Baja, editors, *IWVF*, volume 2059 of *Lecture Notes in Computer Science*, pages 67–84. Springer, 2001.
- [99] Ali Shokoufandeh, Diego Macrini, Sven J. Dickinson, Kaleem Siddiqi, and Steven W. Zucker. Indexing hierarchical structures using graph spectra. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(7):1125–1140, 2005.
- [100] M. Singh, A. Chatterjee, and S. Chaudhury. Matching structural shape descriptions using genetic algorithms. *Pattern Recognition*, 30(9):1451–1462, September 1997.
- [101] Daniel A. Spielman. *Spectral graph theory and its applications*, 2004.
- [102] Barbara Spillmann, Michel Neuhaus, Horst Bunke, Elzbieta Pekalska, and Robert P. W. Duin. Transforming strings to vector spaces using prototype selection. In *Structural, Syntactic, and Statistical Pattern Recognition, Joint IAPR International Workshops, SSPR 2006 and SPR 2006, Hong Kong, China, August 17-19, 2006, Proceedings*, pages 287–296, 2006.
- [103] Ponnuthurai N. Suganthan. Structural pattern recognition using genetic algorithms. *Pattern Recognition*, 35(9):1883–1893, 2002.
- [104] Hiroyuki Yoshioka Sumio Masuda and Eiichi Tanaka. Algorithm for finding one of the largest common subgraphs of two three-dimensional graph structures.

- Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, 81(9):48–53, 1998.
- [105] Y. Takahashi, Y. Satoh, H. Suzuki, and S. Sasaki. Recognition of largest common structural fragment among a variety of chemical structures. *Analytical Sciences*, 3(1):23–28, 1987.
- [106] W. H. Tsai and K. S. Fu. Error-correcting isomorphisms of attributed relational graphs for pattern analysis. *IEEE Transactions on Systems, Man and Cybernetics*, 9:757–768, 1979.
- [107] J. R. Ullman. An algorithm for subgraph isomorphism. *Journal of ACM*, 23(1):31–42, January 1976.
- [108] Shinji Umeyama. An eigendecomposition approach to weighted graph matching problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(5):695–703, September 1988.
- [109] W. D. Wallis, P. Shoubridge, M. Kraetz, and D. Ray. Graph distances using graph union. *Pattern Recognition Letters*, 22(6/7):701–704, 2001.
- [110] Yu Wang and Carsten Maple. A novel efficient algorithm for determining maximum common subgraphs. In *9th International Conference on Information Visualisation, IV 2005, 6-8 July 2005, London, UK*, pages 657–663. IEEE Computer Society, 2005.
- [111] Yuan-Kai Wang, Kuo-Chin Fan, and Jorng-Tzong Horng. Genetic-based search for error-correcting graph isomorphism. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 27(4):588–597, 1997.
- [112] E. Weiszfeld. Sur le point pour lequel la somme des distances de n points donnés est minimum. *Tohoku Math. Journal*, (43):355–386, 1937.
- [113] Richard C. Wilson and Edwin R. Hancock. Structural matching by discrete relaxation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19(6):634–648, 1997.
- [114] Richard C. Wilson and Edwin R. Hancock. Levenshtein distance for graph spectral features. In *17th International Conference on Pattern Recognition*, pages 489–492, 2004.
- [115] Richard C. Wilson, Edwin R. Hancock, and Bin Luo. Pattern vectors from algebraic graph theory. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(7):1112–1124, 2005.
- [116] A.K.C. Wong and M. You. Entropy and distance of random graphs with application to structural pattern recognition. *Transactions on Pattern Analysis and Machine Intelligence*, 7:599–609, 1985.
- [117] E. K. Wong. Model matching in robot vision by subgraph isomorphism. *Pattern Recognition*, 25(3):287–303, 1992.

- [118] Lei Xu and Irwin King. A PCA approach for fast retrieval of structural patterns in attributed graphs. *IEEE Transactions on Systems, Man and Cybernetics - Part B*, 31(5):812–817, October 2001.

Final Acknowledgment

This work has been partially supported by the research Fellowship number 401-027 (UAB) and the CICYT projects TIN2006-15694-C02-02 and DPI2007-614452 (Ministerio Ciencia y Tecnología) and by the Spanish research programme Consolider Ingenio 2010: MIPRCV (CSD2007-00018).
