



UNIVERSITAT POLITÈCNICA  
DE CATALUNYA  
BARCELONATECH

UNIVERSITAT POLITÈCNICA DE CATALUNYA

---

# A MULTI-RESOLUTION APPROACH FOR SPECTRAL GRAPH MATCHING

---

A Degree Thesis Submitted to the Faculty of the Escola Tècnica  
d'Enginyeria de Telecomunicació de Barcelona, Universitat  
Politécnica de Catalunya by

**Víctor González Navarro**

In partial fulfillment of the requirements for the degree in  
Telecommunications Technologies and Services Engineering

Supervised by:  
Prof. Antonio Ortega  
Prof. Philippe Salembier

**Barcelona, August 2018**



# A Multi-Resolution Approach for Spectral Graph Matching

by Víctor GONZÁLEZ NAVARRO

## *Abstract*

This project presents an innovative way of solving the inexact graph matching problem for weighted graphs. The goal is to find the correspondence between the vertices of two similar graphs. This problem is solved through a multi-resolution approach in the spectral domain. Spectral graph matching provides a framework that allows to represent both graphs in a space of different dimensions where the problem is more tractable. Furthermore, the multi-resolution approach improves the performance of the graph matching algorithm, since lower resolutions reveal new structural patterns. To obtain lower resolutions of both graphs, the proposed graph downsampling methods must make sure that the vertices selected for the lower resolutions should be the ones that are more likely to be correctly matched. If this property is not accomplished, the matching in this lower resolution will not provide useful information for the final matching. A novel solution to common challenges in graph matching, i.e., sign ambiguity of the eigenvectors, is provided in order to improve the correspondence between the vertices of both graphs. Different graph matching applications are presented to illustrate the effectiveness of the proposed technique. A comparison with other spectral graph matching algorithms demonstrates the benefits of the proposed approach.



# Acknowledgements

I would like to gratefully acknowledge some people who have journeyed with me as I have worked in this thesis. First, I would like to thank my supervisors, who have encouraged and challenged me with innovative ideas to fulfill the requirements of this project. I also owe an enormous debt of gratitude to the students of my laboratory at the University of Southern California, who have always given me good advice to try new ideas for the development of this project. Lastly, and most of all, I would like to thank all the professors I have had the recent years at Universitat Politècnica de Catalunya, since they taught me all the concepts I required to build this project. Without them, this would have not been possible. It has been an enormous privilege to be a student of this university. Thank you.



# Contents

List of Figures	i
List of Tables	iii
<b>1 Introduction</b>	<b>1</b>
<b>2 Background</b>	<b>5</b>
2.1 Graphs and Graph Signals	5
2.2 Frequency in Graphs	7
<b>3 A Multi-Resolution Approach</b>	<b>9</b>
3.1 Graph Downsampling	9
3.1.1 Maximum Degree Gap (MDG)	11
3.1.2 Corrected Second Eigenvector (CSE)	14
3.2 Graph Reduction	18
3.3 Graph Sparsification	19
<b>4 Weighted Graph Matching Problem</b>	<b>21</b>
4.1 Isomorphic Graphs	21
4.2 Non Isomorphic Graphs	24
4.3 Proposed Approach	25
4.3.1 Step 1: Compute and order the eigenfunctions	26
4.3.2 Step 2: Select the first K eigenfunctions	26
4.3.3 Step 3: Remove the less similar eigenfunctions	27
4.3.4 Step 4: Find the best alignment between two graphs	28
4.3.5 Step 5: Combining multi-resolution information	29
4.3.6 Step 6 (optional): Improvement against graph deformations	30
<b>5 Point Set Registration</b>	<b>33</b>
<b>6 Experimental Results</b>	<b>37</b>
6.1 Random graphs	37
6.2 3D point-clouds	39
6.3 Video compression of point-clouds	41
<b>7 Comparison With Previous Approaches</b>	<b>43</b>
7.1 Comparison with other spectral graph matching algorithms	43
7.2 Sensitivity Analysis	44
<b>8 Conclusions</b>	<b>47</b>
<b>Bibliography</b>	<b>49</b>
<b>Glossary</b>	<b>55</b>



# List of Figures

2.1	Applications where the information is represented by graphs: a) a subset of the web, b) an online social network, c) an image and d) a chemical compound. . . . .	6
2.2	Representation of the graph signal's variation (low to high) defined by the entries of four different eigenvectors: $\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_{15}$ and $\mathbf{u}_{30}$ . . . . .	8
3.1	Example of partitioning a bipartite graph into two sets (red and grey) according to the polarity of the largest eigenvector of the graph Laplacian. . . . .	10
3.2	Example of the <i>Maximum Degree Gap</i> method applied to two similar graphs. . . . .	13
3.3	Example of the variability of the degree of a particular vertex due to Gaussian noise. . . . .	14
3.4	Example of the variability of the degree due to Gaussian noise. Note that this is a simplified figure, since the base of each triangle should have different length. . . . .	14
3.5	Example of the <i>Corrected Second Eigenvector</i> method applied to two similar graphs but using histograms to correct the sign ambiguity. . . . .	16
3.6	Example of the <i>Corrected Second Eigenvector</i> method applied to two similar graphs. . . . .	17
3.7	Example of graph reduction. In this case, since there is a path between node 1 and node 4 (selected vertices) and along the path there are vertices which have not been selected, there is an edge between node 1 and node 4. . . . .	18
3.8	Graphical demonstration that shows why graph sparsification modifies the structure of two similar graphs. . . . .	19
3.9	Graphical demonstration that shows the effect of Kron reduction over multiple resolutions. . . . .	20
4.1	Illustration of the spectral graph matching problem. From the permutation matrix we obtain the final alignment: node 1 of $G_1$ is aligned to node 3 of $G_2$ , node 2 of $G_1$ is aligned to node 1 of $G_2$ , node 3 of $G_1$ is aligned to node 2 of $G_2$ , and node 4 of $G_1$ is aligned to node 4 of $G_2$ . . . . .	22
4.2	Representation of the graph signal defined by the entries of the Fiedler eigenvector of two similar graphs. Node 1 of $G_1$ will be assigned to node 2 of $G_2$ , node 2 of $G_1$ will be assigned to node 3 of $G_2$ and node 3 of $G_1$ will be assigned to node 1 of $G_2$ . . . . .	25
4.3	A multi-resolution approach for spectral graph matching algorithm. . . . .	26
4.4	Graphs $G_1^{(i)}$ and $G_2^{(i)}$ embedded with the first three eigenfunctions. . . . .	27
4.5	Graphs $G_1^{(i)}$ and $G_2^{(i)}$ embedded with the the first two eigenfunctions. . . . .	28

5.1	Original and transformed model set and data set . . . . .	33
6.1	Performance of our graph matching algorithm for a) identical graphs; b) very similar graphs; c) similar graphs. . . . .	38
6.2	Details of the resolution levels using the multi-resolution approach for spectral graph matching. . . . .	38
6.3	Performance of the graph matching algorithm with very similar 3D point-clouds ( <i>3 different perspectives of bunny</i> ). . . . .	39
6.4	Performance of the graph matching algorithm with similar 3D point- clouds ( <i>3 different perspectives of bunny</i> ). . . . .	40
6.5	Performance of the graph matching algorithm with not similar 3D point-clouds ( <i>3 different perspectives of bunny</i> ). . . . .	40
6.6	Performance of the graph matching algorithm with 3D point-clouds ( <i>3 different perspectives of dragon</i> ). . . . .	40
6.7	Performance of the graph matching algorithm using the <i>CSE</i> down- sampling method with 3D point-clouds taken from a video ( <i>3 dif- ferent perspectives of human</i> ). . . . .	41
6.8	Performance of the graph matching algorithm using the <i>MDG</i> down- sampling method with 3D point-clouds taken from a video ( <i>3 dif- ferent perspectives of human</i> ). . . . .	42

# List of Tables

7.1	Comparison of the graph matching error $J(\mathbf{P})$ for different spectral graph matching algorithms . . . . .	43
7.2	Comparison of the graph matching error $J(\mathbf{P})$ for different graph downsampling methods . . . . .	45
7.3	Comparison of the graph matching error $J(\mathbf{P})$ for different settings	45
7.4	Comparison of the graph matching error $J(\mathbf{P})$ with and without point set registration . . . . .	45



# Chapter 1

## Introduction

Many problems in areas such as 2D/3D image analysis, document processing, biometric identification, biological and biomedical applications, image databases and video analysis can be formulated as a graph matching problem. A graph is used to represent information about elements (concepts, locations, etc.) and their relationships. The set of elements is represented by vertices and the relationships between pairs of elements are denoted by edges.

The large number of applications where graph matching is used has motivated us to investigate this problem. For instance, in the field of 2D image analysis, previous works [31] have proposed a method for pattern recognition that divides a scene graph into many sub-scene graphs, and an algorithm to get local matches between each sub-scene graph and the model graph. Regarding 3D image analysis, graph matching has been proposed for 3D object recognition [3] and 3D object reconstruction [18]. Graph matching has also been used in document processing applications such as handwritten recognition [16]. In the field of biometric applications, graph matching has been widely used for face authentication [13] and facial expression recognition [50]. Graph matching is also employed in biological applications to identify diatoms [17]. Graph based techniques have also been employed for image databases for indexing and data retrieval [40], [22]. Finally, in the video analysis area, motion estimation, object tracking and retrieval from video databases have been also addressed through graph matching techniques [15], [9], [46].

The graph matching problem [11], [21], [24] consists of finding the optimum alignment (a.k.a., matching, mapping, correspondence) between two weighted graphs. That is, certain substructures in one graph are mapped to similar substructures in the other graph. In many applications, the graph matching algorithm needs to be efficient and robust in the presence of structural noise or deformations, both in the weights of the nodes and the edges. In this case, the two graphs to be matched are not identical and the problem is referred to as inexact graph matching. Inexact graph matching is an important problem in practice since in most applications the graphs to be matched have different sizes, as they are constructed from real data such as visual sensors that incur in some noise and deformations [34].

Finding the best correspondence between the vertices of two graphs according to a score function has been proven to be an NP-hard problem with exponential computational complexity [37]. Most of the previous works have focused on solving

this problem using heuristics [6], [14]. Unfortunately, some of these techniques incur in significant errors. For this reason, we propose to use spectral graph theory combined with Graph Signal Processing (GSP) as a powerful approach to find a better solution within an acceptable computational cost. The algorithm for solving the alignment between two graphs is based on a quadratic programming formulation that generalizes the classical graph matching problem.

In this thesis we present a novel approach for inexact graph matching based on matching representations of both graphs using multiple resolutions. The proposed method combines the information provided by all the resolution levels in order to find a better alignment between the two graphs being matched. Moreover, once the correspondence in each resolution has been decided by the algorithm, we measure objectively the quality of the alignment, so that we give additional weight to the information provided by the resolutions with better alignment. In this way, the multi-resolution approach results in improved matching accuracy: if the alignment achieved in lower resolutions is not good enough, this resolution is given lower weight in the final matching.

To define the resolution levels of each graph, two new graph downsampling methods are presented. These graph downsampling methods are designed so that if two vertices are correctly matched in a specific resolution level, either both of them or none of them should be selected for the lower resolution.

Our proposed multi-resolution approach can be used in combination with any existing graph matching methods. Nonetheless, in this project we present a novel spectral graph matching algorithm. This new technique, which is applied in each resolution level, efficiently tackles the common challenges in graph matching, and it results in an increased performance when compared with previous methods using random graphs and 3D point-clouds. The main difference between the spectral graph matching algorithm we use in each resolution level and existing algorithms is how the alignment is decided, which in our proposal is based on weighting more the frequencies less affected by the dissimilarity between the two graphs.

There are other techniques that seek to find the best mapping between two graphs. A popular approach is based on the graph edit distance, which searches for the minimum distortion that transforms one graph into another one [29]. Each operation (i.e., insertions, substitutions and deletions of graph vertices and edges) has an associated cost. The transformation with lower cost determines the correspondence between the nodes. Other techniques are based on the Expectation Maximization algorithm [32], replicator equations [39], and graduated assignments [20]. Some of these techniques are critically dependent on initial model estimates, and others suffer from their computational complexity. On the other hand, since the computation of eigenvectors used in spectral graph matching is a well studied problem that can be solved in polynomial time, there is a great interest in using them for graph matching [11]. Our proposal is the first attempt to the best of our knowledge that combines a spectral analysis with a multi-resolution approach to solve this problem.

---

The rest of this thesis is organized as follows. Chapter 2 reviews basic concepts of graphs and defines the notion of frequency in graphs. Chapter 3 describes the proposed multi-resolution technique including the novel graph downsampling methods. Chapter 4 presents the new spectral graph matching scheme. Chapter 5 describes a point set registration algorithm that is used with the spectral graph matching algorithm. Chapter 6 shows the performance of the proposed scheme for a variety of graphs. Chapter 7 compares our framework with some of the most popular spectral graph matching algorithms and Chapter 8 summarizes the main conclusions of this work.



## Chapter 2

# Background

### 2.1 Graphs and Graph Signals

A graph  $G$  is defined as  $G = (V, E)$ , where  $V$  corresponds to the set of vertices,  $V = \{v_1, v_2, \dots, v_N\}$ , and  $E \subseteq V \times V$  corresponds to the set of edges,  $E = \{e_1, e_2, \dots, e_M\}$ . The  $i$ -th vertex can be denoted as  $v_i \in V$  and the  $i$ -th edge as  $e_i \in E$ . An edge represents a connection between two vertices. If we have a graph with no *directed* edges (edges that have a direction associated with them), the graph is called *undirected*. Otherwise, the graph is called *directed*. In our case, we work with *undirected* graphs. Furthermore, we do not consider graphs with self-loops, meaning that an edge will never connect a vertex with itself.

A graph can have some weights assigned to the vertices and the edges. Vertex weighting is a function  $\mu : V \rightarrow \mathbb{R}$ , whereas edge weighting is a function  $w : E \rightarrow \mathbb{R}$ . Consequently, the weight of a vertex is denoted as  $\mu(v_i)$  and the weight of an edge is denoted as  $w(v_i, v_j)$  or  $w_{ij}$  for short. The aggregation into a vector,  $\mathbf{f}_s$ , of all the weights of the vertices of the graph defines a *graph signal*: if we work with graphs of  $N$  vertices,  $\mathbf{f}_s \in \mathbb{R}^N$ . These *graph signals* can be analyzed in order to derive different interpretations depending on the particular graph.

There are many applications where the information is represented by a graph. For example, a graph can represent a set of temperature sensors of a particular city, and the information defined by the *graph signal* could be the temperature that each sensor measures. On the other hand, the weights assigned to the edges can be a function of the distance between the temperature sensors, giving higher values to edges connecting closer nodes. Other applications where the information is represented by graphs are: a subset of the web, online social networks, an image and a chemical compound. These examples are shown in Figure 2.1. Note that classical signals cannot represent a set of elements and a set of pairwise relationships among them, as required by the examples mentioned above. On the other hand, graphs and *graph signals* can efficiently represent this type of information and Graph Signal Processing provides methods to analyze these systems.

The problem of finding a graph matching between two graphs  $G_1 = (V_1, E_1, \mu_1, w_1)$  and  $G_2 = (V_2, E_2, \mu_2, w_2)$  consists of finding a function  $f : V_1 \rightarrow V_2$  that minimizes a given error cost. If the two graphs have the same number of vertices,  $f$  is a bijection (one-to-one matching). In case that the number of nodes is different, the algorithm must find a many-to-many vertices correspondence. In the former case,

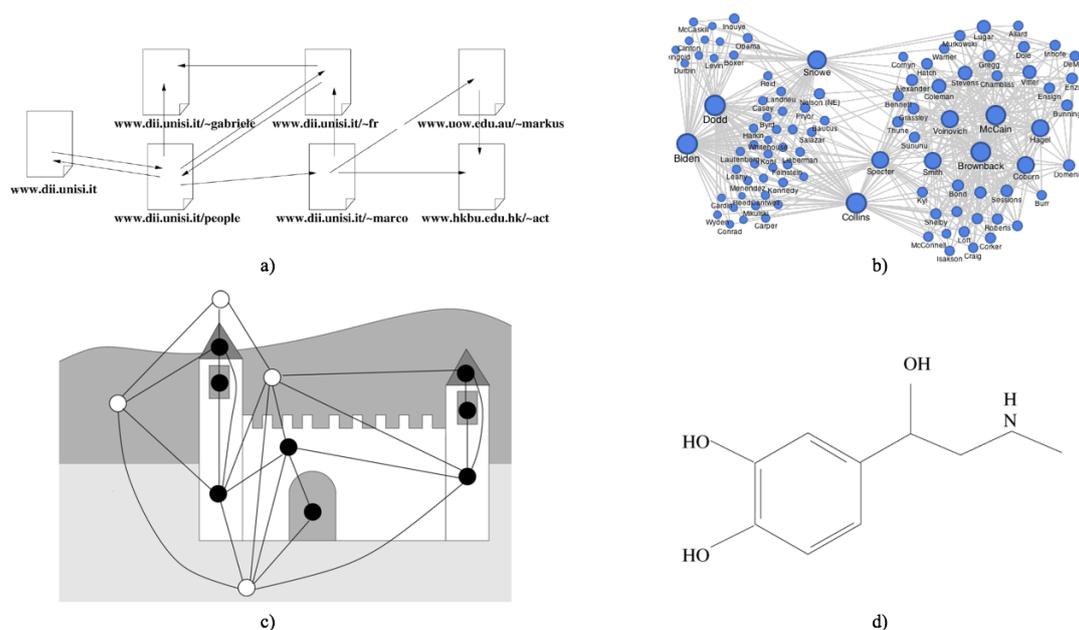


Figure 2.1: Applications where the information is represented by graphs: a) a subset of the web, b) an online social network, c) an image and d) a chemical compound.

this problem is referred to as exact graph matching, whereas in the latter case, it is called inexact graph matching.

The problem of exact graph matching is also known as subgraph isomorphism. It aims at finding a mapping between the nodes of the first graph and the nodes of the second graph such that the edge structure of the graphs is preserved. For example, if there is an edge between two nodes of the first graph, there must be an edge between the mapped nodes of the second graph. Nonetheless, in real world applications, the constraint of exact graph matching is too rigid. In many applications, the observed graphs are subject to deformations due to the intrinsic variability of the data or noise in the acquisition process. For this reason, inexact graph matching or error-tolerant graph matching tries to find ‘the best’ alignment by relaxing the constraints of exact graph matching. The matching between two nodes that do not satisfy the edge-preservation requirements is not forbidden. Instead, it is penalized by assigning to it a cost that takes into account their differences. That is, the goal is to find a mapping  $f$  from one graph to another one such that the overall cost is minimized.

In this thesis, we present a novel effective solution for the inexact graph matching problem that makes use of spectral graph matching. Background on this technique is provided in Chapter 4. Spectral graph matching is still on an early stage and some of its challenges have not been correctly solved yet. In Chapter 4 we also explain how we effectively solve these challenges in a more effective way than previous approaches.

## 2.2 Frequency in Graphs

The proposed graph matching algorithm makes use of the frequency of a *graph signal* derived from the original graph. We measure the frequency of a *graph signal* as the variation of the signal in adjacent nodes. In particular, the variation of the *graph signal* for a node  $v_i$  ( $\Delta v_i$ ) is defined as the difference between the signal at this node and the signal at its neighbors ( $N_i$ , set of vertices connected to vertex  $i$  by an edge), with each difference being multiplied by the weight of the corresponding edge. That is:

$$\Delta v_i = \sum_{k \neq i} w_{ik} \mathbf{f}_s(i) - \sum_{k \neq i} w_{ik} \mathbf{f}_s(k) \quad (2.1)$$

$$\Delta v_i = d_i \mathbf{f}_s(i) - \sum_{k \neq i} w_{ik} \mathbf{f}_s(k) \quad (2.2)$$

where  $d_i = \sum_{k \neq i} w_{ik}$  is defined as the degree of vertex  $i$ . Note that the variation is the difference in the signal amplified by the weight of the edges. Therefore, the variation over vertices connected by an edge with a high weight will contribute more to the final variation.

The variation of the *graph signal* for each individual node of the graph with their respective neighbors can be represented by a vector  $\Delta = [\Delta v_1, \Delta v_2, \dots, \Delta v_N]$ :

$$\Delta(i) = \sum_{k \in N_i} w_{ik} (\mathbf{f}_s(i) - \mathbf{f}_s(k)) = \mathbf{L} \mathbf{f}_s \quad (2.3)$$

where  $\mathbf{L}$  is called the Laplacian matrix. The graph Laplacian is a difference operator and it is defined as:

$$\mathbf{L}_{ij} = \begin{cases} d_i & \text{if } i = j \\ -w_{ij} & \text{if } e_{ij} \in E \\ 0 & \text{otherwise} \end{cases} \quad (2.4)$$

Finally, the total variation of a *graph signal* associated to the graph  $G$  is defined as:

$$\Delta_G(\mathbf{f}_s) = \mathbf{f}_s^t \mathbf{L} \mathbf{f}_s \quad (2.5)$$

Due to the fact that the graph Laplacian is a real symmetric matrix, it has a complete set of orthonormal eigenvectors  $\{\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_N\}$ . Furthermore, if we order the eigenvectors according to the magnitude of their eigenvalues, from smaller to larger, the variation of the graph signal defined by the entries of the eigenvectors (given by (2.5)) is also ordered from lower to higher [48], [26, Theorem 4.2.11]. That is,

$$0 = \mathbf{u}_1^t \mathbf{L} \mathbf{u}_1 \leq \mathbf{u}_2^t \mathbf{L} \mathbf{u}_2 \leq \dots \leq \mathbf{u}_N^t \mathbf{L} \mathbf{u}_N = \lambda_{max} \quad (2.6)$$

In Fig. 2.2 we can see an example of this notion of variation over a random graph with 30 vertices. In this example, the *graph signal* is defined by the entries

of four different eigenvectors:  $\mathbf{u}_1$ ,  $\mathbf{u}_2$ ,  $\mathbf{u}_{15}$  and  $\mathbf{u}_{30}$ . Positive *graph signal* values are shown in red while negative *graph signal* values are shown in blue. Depending on the eigenvector, we can see a substantial difference in the variation of the *graph signal* [41], the smallest being for  $\mathbf{u}_1$  and increasing successively until  $\mathbf{u}_{30}$ .

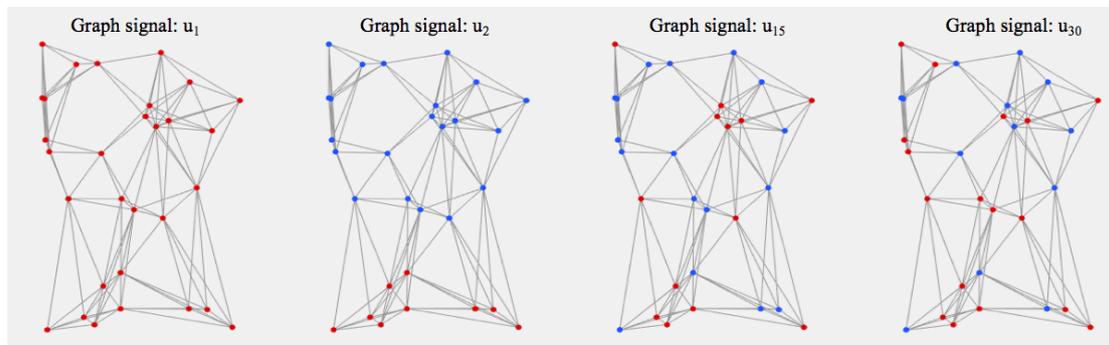


Figure 2.2: Representation of the graph signal's variation (low to high) defined by the entries of four different eigenvectors:  $\mathbf{u}_1$ ,  $\mathbf{u}_2$ ,  $\mathbf{u}_{15}$  and  $\mathbf{u}_{30}$ .

This notion of variation of the eigenvectors of the graph Laplacian is essential for our graph matching technique. In particular, our algorithm exploits the fact that for two similar graphs with minor changes in their edge weights, the entries of their respective eigenvectors, ordered according to their eigenvalue, are very similar [49]. That is,

$$\mathbf{u}_i \approx \mathbf{P}_i \mathbf{u}'_i \quad (2.7)$$

where  $\mathbf{P}_i$  is a permutation matrix that minimizes the Euclidean distance between  $\mathbf{u}_i$  and  $\mathbf{P}_i \mathbf{u}'_i$ , and  $\mathbf{u}_i$  and  $\mathbf{u}'_i$  are the  $i$ -th eigenvectors of  $G_1$  and  $G_2$  respectively.

Furthermore, if we compare the eigenvectors of the two graphs being matched, we have experimentally observed that the similarity of the eigenvectors associated to small eigenvalues (i.e., those with low variation) is higher than for those associated to large eigenvalues (those with high variation). This is due to the fact that: a) low frequencies are more global while high frequencies are more local and b) the graphs being matched, since one of them is a noisy version of the other one, are more similar in a global scale than in a local scale. In other words,

$$\text{dist}(\mathbf{u}_i, \mathbf{P}_i \mathbf{u}'_i) < \text{dist}(\mathbf{u}_{i+k}, \mathbf{P}_{i+k} \mathbf{u}'_{i+k}) \quad (2.8)$$

where the Euclidean distance is used and  $k$  is a positive integer.

These two properties, (2.7) and (2.8), will be used to improve the alignment between both graphs, as detailed in Chapter 4.

## Chapter 3

# A Multi-Resolution Approach

We propose a graph matching algorithm using the eigenvectors of the graph Laplacians of multiple resolutions of the two graphs being matched. This is in contrast with most existing methods, which consider only matching the original graphs. Consequently, if for example we use three different resolution levels for  $G_1$  and  $G_2$ , then we compute a matching between  $G_1^{(1)}$  and  $G_2^{(1)}$ ,  $G_1^{(2)}$  and  $G_2^{(2)}$ ,  $G_1^{(3)}$  and  $G_2^{(3)}$ , where  $(i)$  indicates the resolution level.

In each resolution level a probability matrix is computed,  $\mathbf{Pr}^{(i)}$ . If  $G_1$  has  $M$  nodes and  $G_2$  has  $N$  nodes, then  $\mathbf{Pr}^{(i)} \in \mathbb{R}^{M \times N}$ , and  $\mathbf{Pr}^{(i)}(m, n)$  indicates the certainty that the node  $v_m$  of  $G_1^{(i)}$  is assigned to the node  $v'_n$  of  $G_2^{(i)}$ , being a low value very provable. The final alignment between the vertices of  $G_1$  and  $G_2$  will be given by a linear combination of the probability matrices computed in all the resolution levels.

The reason for combining the graph matching result of different resolutions is the fact that multi-resolution methods can reveal structural information such as irregular structural patterns. At the same time, they provide a way to reduce the complexity and dimensionality of the graph matching task, resulting in a better matching.

Computing different resolutions of a particular graph involves three steps: graph downsampling, graph reduction and graph sparsification. We describe these three steps next.

### 3.1 Graph Downsampling

To obtain a lower resolution of a graph  $G$ , it is necessary to decide which nodes from the vertex set  $V = \{v_1, v_2, \dots, v_N\}$  will be maintained in the lower resolution. To downsample classical signals, such as a discrete-time signals, we remove every other component of the signal. Nevertheless, since there is not a predefined order of the vertices of a graph, it is not obvious which nodes should be selected [47], [38].

There are two desirable properties for the graph downsampling method in the specific case of graph matching:

- Remove approximately the same percentage of vertices (e.g., half) in both graphs.
- The vertices selected from  $G_1^{(i)}$  and  $G_2^{(i)}$  should be the ones that are more likely to be correctly matched in the next resolution level. That is, if  $v_i$  and  $v'_j$  belong to  $V_1^{(i)}$  and  $V_2^{(i)}$  respectively and are the correct matching, then either both of  $v_i$  and  $v'_j$  belong to  $V_1^{(i+1)}$  and  $V_2^{(i+1)}$  or neither of them are.

Clearly, the second property is the most challenging, since we do not know the correct correspondence between the vertices of  $G_1$  and  $G_2$  ( $f_{correct}$ ). In fact, this is the problem we are trying to solve. Nonetheless, with the spectral information of both graphs we will be able to approximate a good solution to achieve this property. Note that this property, although being essential for our graph downsampling algorithm, is not required in other existing graph signal sampling methods.

One way to downsample graphs, applied for example in compression of *graph signals* defined by data acquired from different applications (weather data, sensor networks data, social network data, transportation data), is the spectral method, which selects the vertices to keep based on the polarity of their corresponding entries in the largest eigenvector of the graph Laplacian ( $u_{max}^{(i)}$ ):

$$V^{(i+1)} = \{v_l \in V^{(i)} : u_{max}^{(i)}(l) > 0\} \quad (3.1)$$

This is a popular method because the polarity of the largest eigenvector splits the graph into two sets of similar size. In the particular case where the graph is bipartite, it selects all the vertices from one subset [47]. A *bipartite* graph is a graph such that the set of vertices  $V$  can be partitioned into two subsets  $V_a$  and  $V_b$  so that there are only edges between the vertices of  $V_a$  and  $V_b$ , and no edges within the nodes of each of the two subsets. For example, in Fig. 3.1 we can observe the result of downsampling a graph representing a 2D image using (3.1).

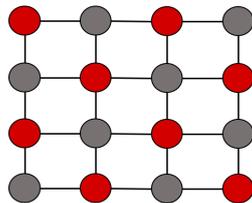


Figure 3.1: Example of partitioning a bipartite graph into two sets (red and grey) according to the polarity of the largest eigenvector of the graph Laplacian.

Note, however, that the spectral downsampling method does not fulfill the second property required for the graph matching problem.

Other graph sampling methods exist as well [1], [19], [10]. Nevertheless, none of these algorithms are designed to accomplish the second property. Instead, we

propose two new algorithms to perform downsampling for the specific case of graph matching focusing on solving the two properties mentioned above. The most effective one depends on the type of graph, as we will show later. We refer to them as: *Maximum Degree Gap (MDG)* and *Corrected Second Eigenvector (CSE)*. The first one is very robust against noise in the edge weights of small graphs (e.g. graph matching of random graphs or graph matching of 3D point-clouds). The second one is more efficient when dealing with deformations, that is, the two graphs are similar but some of the edge weights have changed substantially (e.g. graph matching of articulated objects represented by point-clouds). Noise is common in applications that use a graph to represent a point-cloud of a rigid object; for example, when we try to match two similar cars represented as two point-clouds. On the other hand, deformations may appear when working with point-clouds that represent articulated objects or persons, since some of their parts can be in a completely different position.

A comparison between the two graph downsampling methods we propose and other existing graph sampling methods is provided in Chapter 7.

### 3.1.1 Maximum Degree Gap (MDG)

In order to generate a new resolution level  $G^{(i+1)}$ , we must decide which vertices we keep from  $G^{(i)}$ . That is, we must select a subset of vertices from  $V_1^{(i)}$  and  $V_2^{(i)}$  to generate  $V_1^{(i+1)}$  and  $V_2^{(i+1)}$  respectively.

The *Maximum Degree Gap (MDG)* is based on selecting the vertices with higher degree ( $d_i$ ) of the graphs in the original resolution,  $G_1^{(i)}$  and  $G_2^{(i)}$ . Since a high degree node of  $G_1^{(i)}$  is normally associated with a high-degree node in  $G_2^{(i)}$ , this heuristic tends to favour that in the lower resolution, the vast majority of the nodes will be able to find their right pair.

More specifically, if  $G_1^{(i)}$  and  $G_2^{(i)}$  have  $N$  nodes:

$$V_1^{(i)} = \{v_1, v_2, \dots, v_N\}$$

$$V_2^{(i)} = \{v'_1, v'_2, \dots, v'_N\}$$

The first step of the MDG is to compute the degree of each set of vertices:

$$\mathbf{d}_1^{(i)} = \{d_1(v_1), d_1(v_2), \dots, d_1(v_N)\}$$

$$\mathbf{d}_2^{(i)} = \{d_2(v'_1), d_2(v'_2), \dots, d_2(v'_N)\}$$

Then, for one of the two graphs being matched, we sort the degree vector from bigger to smaller and compute the differences between consecutive elements:

$$\mathbf{sd}_1 = \text{sort}(\mathbf{d}_1) = \{a_1, b_1, \dots, z_1\}$$

$$\mathbf{rest}_1 = \{(a_1 - b_1), (b_1 - c_1), \dots, (y_1 - z_1)\}$$

The last step is to select approximately half of the vertices with higher degree from the set  $V_1^{(i)}$ . Note that instead of selecting exactly  $N/2$ , it is better to stop the selection at a node whose degree is significantly higher than for the next node. This results in improved robustness to noise. Since there is a significant difference in the degree of the selected vertices with respect to the non-selected ones, even in the presence of noise, it is highly likely that we select in both graphs the nodes that form each one of the pairs. Finally, we select the same number of vertices of  $G_2^{(i)}$ , by choosing those with the highest degree. In case there is not a node whose degree is significantly higher than for the next node, the algorithm may not perform as expected. For this reason, in some types of graphs, the other graph downsampling method we propose obtains better results.

The proposed method is summarized in Algorithm 1, where arrows denote which variables are needed to compute the variable on the left.

---

**Algorithm 1** MDG Graph downsampling algorithm
 

---

**Input:**  $G_1^{(i)} = (V_1^{(i)}, E_1^{(i)}, w_1^{(i)})$ ,  $G_2^{(i)} = (V_2^{(i)}, E_2^{(i)}, w_2^{(i)})$

**Output:**  $V_1^{(i+1)}, V_2^{(i+1)}$

- 1:  $\mathbf{d}_1^{(i)} \leftarrow G_1^{(i)}$
  - 2:  $\mathbf{d}_2^{(i)} \leftarrow G_2^{(i)}$
  - 3:  $\mathbf{sd}_1 = \text{sort}(\mathbf{d}_1^{(i)})$
  - 4:  $\mathbf{rest}_1 \leftarrow \mathbf{sd}_1$
  - 5:  $[V_1^{(i+1)}, V_2^{(i+1)}] \leftarrow \mathbf{rest}_1$
- 

An example that illustrates this graph downsampling method is shown in Fig. 3.2, where each column represents a different resolution level of two similar graphs. The edges are not plotted since they would make it more difficult to visualize the results. As can be seen in the last column (resolution 3), the selected vertices in both the top graph and the bottom graph are very similar, meaning that nodes are likely to be correctly matched. Furthermore, matching the vertices in this resolution simplifies the graph matching problem, since there are fewer nodes to decide their alignment.

Next we show a concrete example to give a better intuition of why this method works. In the case of graphs representing 2D point-clouds, we can determine when the selected vertices from the two graphs being matched are the correct pairs with very high probability. For this purpose, we first define *GAP* as the difference between the degree of the vertex with lowest degree from the selected set of vertices and the degree of the vertex with highest degree from the non selected vertices. If one of the graphs is created based on the other graph but has Gaussian noise added to the x-y coordinates (bounded by  $3\sigma$ ), then for the worst case scenario

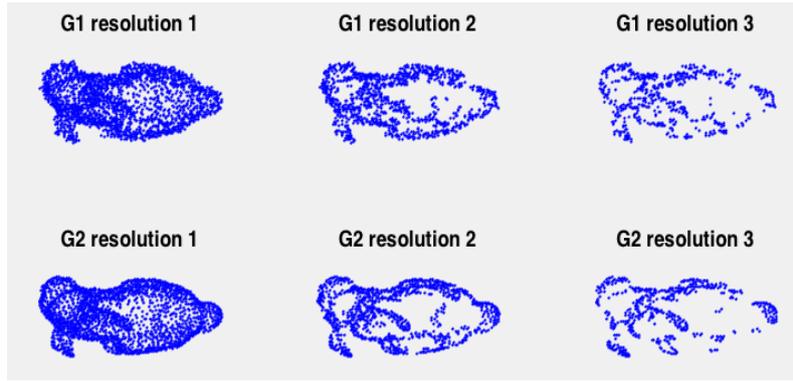


Figure 3.2: Example of the *Maximum Degree Gap* method applied to two similar graphs.

(which is very unlikely to occur):

$$\begin{aligned} \text{dist}(v', v'_i) &= \text{dist}(v, v_i) \pm 2\sqrt{(3\sigma)^2 + (3\sigma)^2} = \\ &= \text{dist}(v, v_i) \pm 8.49\sigma \end{aligned} \quad (3.2)$$

where  $v, v_i$  are two vertices of  $G_1$  and  $v', v'_i$  are the corresponding vertices of  $G_2$ .

Consequently, using a Gaussian kernel to compute the weights, we can define the degree of vertex  $v$  and  $v'$ , which are connected to its  $k$  closer vertices, as follows:

$$d(v) = \sum_{i=1}^k w(v, v_i) = \sum_{i=1}^k \exp(-\text{dist}(v, v_i)^2 / \sigma') \quad (3.3)$$

$$\begin{aligned} d(v') &= \sum_{i=1}^k w(v', v'_i) = \sum_{i=1}^k \exp(-\text{dist}(v', v'_i)^2 / \sigma') = \\ &= \sum_{i=1}^k \exp(-(\text{dist}(v, v_i) \pm 8.49\sigma)^2 / \sigma') \end{aligned} \quad (3.4)$$

Therefore, if  $G_2$  is a noisy version of  $G_1$ , the degree of a node in  $G_2$  ( $d(v')$ ) is bounded around the degree of the correct node in  $G_1$  ( $d(v)$ ), as shown in Fig. 3.3.

In the case that  $GAP$  is greater than  $\gamma$  (parameter shown in Fig. 3.4), then we can be sure that the selected vertices of both graphs have their corresponding pairs. Otherwise, we still may have selected the correct pairs since the above case is the very worst scenario. Finally, note that even in the case that a few nodes do not have their corresponding pairs, the final outcome of the graph matching may not be the best one but it will be very close to it.

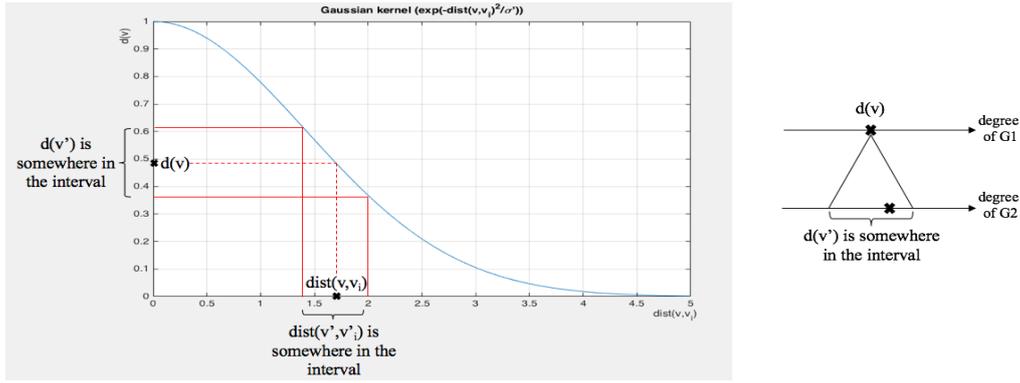


Figure 3.3: Example of the variability of the degree of a particular vertex due to Gaussian noise.

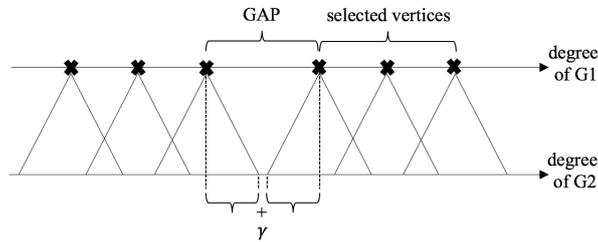


Figure 3.4: Example of the variability of the degree due to Gaussian noise. Note that this is a simplified figure, since the base of each triangle should have different length.

### 3.1.2 Corrected Second Eigenvector (CSE)

The other method we propose to downsample graphs for graph matching consists of selecting the vertices based on the polarity (sign) of the components of the second eigenvector of the graph Laplacian, also known as the Fiedler eigenvector:

$$\mathbf{u}_2^{(i)} = \text{Fiedler eigenvector of } G_1^{(i)}$$

$$\mathbf{u}'_2^{(i)} = \text{Fiedler eigenvector of } G_2^{(i)}$$

$$V_1^{(i+1)} = \{v_l \in V_1^{(i)} : \mathbf{u}_2^{(i)}(l) > 0\} \quad (3.5)$$

$$V_2^{(i+1)} = \{v_l \in V_2^{(i)} : \alpha \mathbf{u}'_2^{(i)}(l) > 0\} \quad (3.6)$$

where the value of  $\alpha$  is +1 or -1 and it is decided according to a criteria that is explained later in this section. With this downsampling method, if the value of  $\alpha$  is correctly chosen, we will select a connected region from the graph  $G_1$  and the same connected region for  $G_2$ . This is due to the fact that the *graph signal* defined by the entries of the Fiedler eigenvector for each node of  $G_1$  and  $G_2$  is approximately the same, even if the labels assigned to the nodes are not the same.

Furthermore, for a more mathematical justification, the best permutation matrix that decides the alignment (if  $\mathbf{P}_{ij} \neq 0$  then  $(v_i \in V_1) \rightarrow (v'_j \in V_2)$ ) can be

found by:  $\mathbf{P} = \mathbf{U}_1 \mathbf{S} \mathbf{U}_2^T$  (detailed in Chapter 4.1), where  $\mathbf{S} = \{\text{diag}(s_1, s_2, \dots, s_N) | s_i = 1 \text{ or } -1\}$ . This implies that  $\mathbf{U}_1 \mathbf{S} = \mathbf{P} \mathbf{U}_2$ . Therefore, since  $\mathbf{u}_2 = \alpha \mathbf{P} \mathbf{u}'_2$  ( $\alpha = \pm 1$ ), the vertices with a positive graph signal defined by the Fiedler eigenvector of  $\mathbf{u}_2$  and  $\alpha \mathbf{u}'_2$  are the same for isomorphic graphs even if the vertices in each graph have different labels, and are likely to be the same for non isomorphic graphs.

Nevertheless, the novelty of this algorithm is not the fact of selecting the vertices based on the polarity of the Fiedler eigenvector, but how we combine it with a sign ambiguity detector to correctly decide the sign of the eigenvector of  $G_1^{(i)}$  to be equal to the sign of the eigenvector of  $G_2^{(i)}$  (parameter  $\alpha$ ). In fact, since in our case we are downsampling the two graphs being matched, it is a must to correct the sign of the Fiedler eigenvector of both graphs in order to select the right pairs of nodes for the lower resolution.

As shown in Fig. 2.2, the second eigenvector of the graph Laplacian is the one with less variation over the vertices of the graph, if we ignore the first eigenvector, since this eigenvector has all the components equal, and does not allow us to discriminate among them. The Fiedler eigenvector divides the graph in two subsets where each of the subsets is connected with edges of high weight, while the weight of the edges crossing the two subsets is low.

The two main challenges to apply this method are: a) the arbitrary determination of the signs of the components of the eigenvectors by the eigenvalue program (also known as sign ambiguity, i.e., an eigenvector can be multiplied by -1 and it will result in an equivalent eigenvector); and b) the permutation of the entries of the Fiedler eigenvectors of the two graphs. This latter problem occurs because the vertices in each graph have arbitrary labels which are different for each of them. The two Fiedler vectors are expected to be similar only if the labels are the same in both graphs. In other words,

$$\mathbf{u}_2^{(i)} \approx \{\mathbf{P} \mathbf{u}'_2^{(i)} \text{ or } \mathbf{P}(-\mathbf{u}'_2^{(i)})\} \quad (3.7)$$

In order to solve the sign ambiguity problem for  $\mathbf{u}_2$  and  $\mathbf{u}'_2$ , some techniques have been proposed. All of them may lack good performance in particular cases. A comparison of the performance between these techniques and the one we present is shown later in this section. Caelli and Kosinov propose a dominant sign correction [7], assuming that there are different number of positive and negative entries in the Fiedler eigenvector. If  $\mathbf{u}_2$  has more positive entries and  $\mathbf{u}'_2$  has more negative entries, they flip the sign of one of the Fiedler eigenvectors, either  $\mathbf{u}_2$  or  $\mathbf{u}'_2$ . This is unreliable since most of the Fiedler eigenvectors have about the same number of positive and negative entries, and given the presence of noise, a small difference between the number of positive and negative entries may not be due to the sign of the vector but to the noise. Other techniques [34] propose to compute three histograms:  $\text{hist}(\mathbf{u}_2)$ ,  $\text{hist}(\mathbf{u}'_2)$  and  $\text{hist}(-\mathbf{u}'_2)$ . Then, they calculate a similarity function and decide whether  $\text{hist}(\mathbf{u}'_2)$  or  $\text{hist}(-\mathbf{u}'_2)$  is more similar to  $\text{hist}(\mathbf{u}_2)$ . This technique, although having a better performance than the dominant sign correction method, is very sensitive to the number of bins one chooses in each

resolution to represent the histogram, which often results in unsatisfactory results, as depicted in Fig. 3.5. In this example, the nodes in the third resolution have not been correctly selected, which will make the graph matching algorithm fail in this resolution.

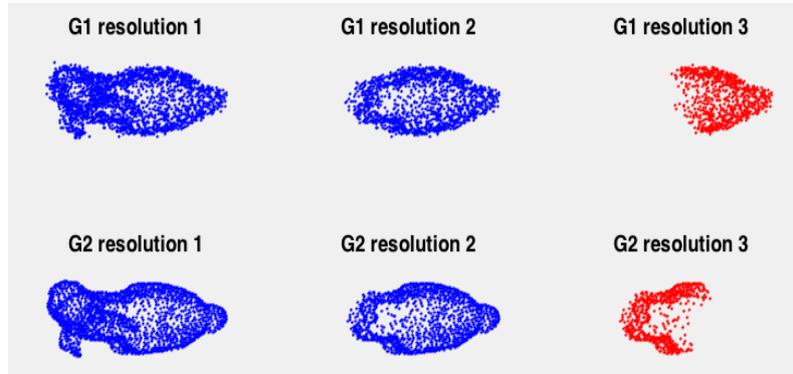


Figure 3.5: Example of the *Corrected Second Eigenvector* method applied to two similar graphs but using histograms to correct the sign ambiguity.

We propose a novel scheme to solve at the same time the sign ambiguity problem and the fact that the order of the entries of the Fiedler eigenvector are arbitrary. It is based on sorting the entries of the eigenvectors and computing a similarity measure of the sorted vectors, for the original Fiedler vectors  $\mathbf{u}$  and  $\mathbf{u}'$ . In other words,

$$C_1 = \left\| \text{sort}(\mathbf{u}^{(i)}) - \text{sort}(\mathbf{u}'^{(i)}) \right\|^2 \quad (3.8)$$

$$C_2 = \left\| \text{sort}(\mathbf{u}^{(i)}) - \text{sort}(-\mathbf{u}'^{(i)}) \right\|^2 \quad (3.9)$$

Then, if  $C_1 > C_2$ , we multiply the Fiedler eigenvector of  $G_2$  by -1. Otherwise, the vector is left unchanged. The two resulting Fiedler eigenvectors are used to select the vertices according to (3.5) and (3.6).

The proposed method is summarized in Algorithm 2.

An example of the performance of this graph downsampling method is shown in Fig. 3.6, where each column represents a different resolution of two similar graphs. Note that for each resolution, the two graphs are very similar, which means that the vertices selected for each resolution are the ones that should be correctly matched. Note also that the vertices selected in each resolution using the *Maximum Degree Gap* method for the same graph (see Fig. 3.2) are not the same as the ones selected by the *Corrected Second Eigenvector* method. In *MDG*, the selected vertices are distributed over the graph whereas in *CSE* the selected vertices are all from a sub-area of the graph.

We evaluate our novel *CSE* method using the above three different techniques for the sign ambiguity detector: the dominant sign correction, the comparison of the histograms and our approach based on sorting each vector. The evaluation

**Algorithm 2** CSE Graph downsampling algorithm**Input:**  $G_1^{(i)} = (V_1^{(i)}, E_1^{(i)}, w_1^{(i)})$ ,  $G_2^{(i)} = (V_2^{(i)}, E_2^{(i)}, w_2^{(i)})$ **Output:**  $V_1^{(i+1)}$ ,  $V_2^{(i+1)}$ 

- 1:  $L_1^{(i)} \leftarrow G_1^{(i)}$
- 2:  $L_2^{(i)} \leftarrow G_2^{(i)}$
- 3:  $\mathbf{u}_2^{(i)} \leftarrow L_1^{(i)}$
- 4:  $\mathbf{u}'_2^{(i)} \leftarrow L_2^{(i)}$
- 5:  $C_1 = \left\| \text{sort}(\mathbf{u}_2^{(i)}) - \text{sort}(\mathbf{u}'_2^{(i)}) \right\|^2$
- 6:  $C_2 = \left\| \text{sort}(\mathbf{u}_2^{(i)}) - \text{sort}(-\mathbf{u}'_2^{(i)}) \right\|^2$
- 7:  $V_1^{(i+1)} = \{v_l \in V_1^{(i)} : \mathbf{u}_2^{(i)}(l) > 0\}$
- 8: **if**  $C_1 > C_2$
- 9:      $V_2^{(i+1)} = \{v_l \in V_2^{(i)} : \mathbf{u}'_2^{(i)}(l) < 0\}$
- 10: **else**
- 11:      $V_2^{(i+1)} = \{v_l \in V_2^{(i)} : \mathbf{u}'_2^{(i)}(l) > 0\}$
- 12: **end**

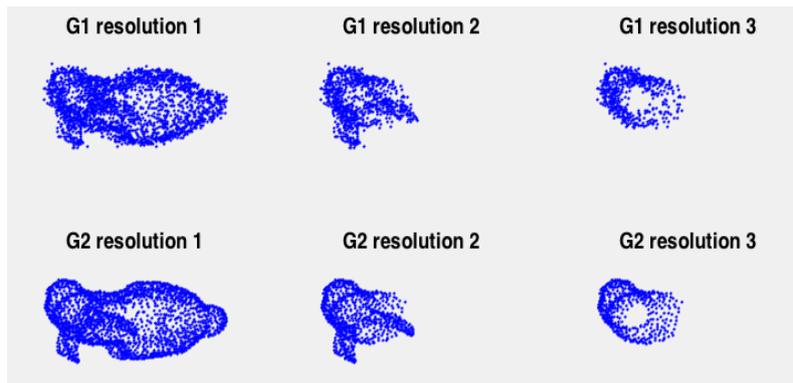


Figure 3.6: Example of the *Corrected Second Eigenvector* method applied to two similar graphs.

is done over different random graphs for a maximum of five resolution levels, and each of the experiments uses graphs with different number of vertices (ranging from 30 vertices to 5000 vertices). In each experiment we decide which technique has better performance (e.g. Fig. 3.5 has better performance than Fig. 3.6, since the highest correct resolution is 2 for Fig. 3.5 whereas it is 3 for Fig. 3.6). Indeed, this is a binary decision (right or wrong) for each resolution level since  $\mathbf{u}_2^{(i)} \approx \{\mathbf{P}\mathbf{u}'_2^{(i)} \text{ or } \mathbf{P}(-\mathbf{u}'_2^{(i)})\}$ . That is, as mentioned before, the Fiedler eigenvector divides the graph in two sets. Since we are matching two similar graphs, it is straightforward to decide if the selected set is the same for  $G_1^{(i)}$  and  $G_2^{(i)}$ . If the algorithm has correctly chosen the sign of  $\mathbf{P}\mathbf{u}'_2^{(i)}$  to be the same as  $\mathbf{u}_2^{(i)}$ , the nodes selected will be the ones that are more likely to be correctly aligned. After 200 experiments, the results are the following: for 70% of the graphs our approach has achieved better performance, whereas for 18% of the graphs the technique based

on the comparison of the histograms performs better, and in 12% of them our approach was the best but equal to some of the other two techniques.

### 3.2 Graph Reduction

Once we have selected the nodes that are kept in the lower resolution, we need to compute the edges that connect these vertices. This is done by computing the graph Laplacian of this lower resolution graph, which includes the edge weight information. That is, given  $\mathbf{L}^{(i)} \in \mathbb{R}^{M \times M}$ , the Laplacian of the lower resolution will be defined as  $\mathbf{L}^{(i+1)} \in \mathbb{R}^{M' \times M'}$ , where  $M' < M$ .

An effective method for computing the Laplacian of the reduced graph is described by Shuman et al. [47]. Among the properties described in that method, the ones that are relevant to the specific case of the graph matching problem are the following:

- The graph defined by the new Laplacian has to preserve connectivity. That is, if the original graph is connected, the reduced graph is also connected:

$$\lambda_1(\mathbf{L}^{(i)} > 0) \rightarrow \lambda_1(\mathbf{L}^{(i+1)} > 0) \quad (3.10)$$

- Edges in the original graph that connect vertices that are also present in the reduced graph are preserved.
- There is a correspondence between the spectral properties of the reduced graph Laplacian and the original one. For example, the spectral interlacing theorem [23] is preserved.
- Two vertices of the reduced graph are connected if there is a path between them in the original graph where the vertices along the path have not been selected by the graph downsampling method (Figure 3.7).

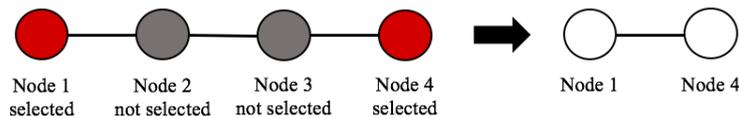


Figure 3.7: Example of graph reduction. In this case, since there is a path between node 1 and node 4 (selected vertices) and along the path there are vertices which have not been selected, there is an edge between node 1 and node 4.

One method that satisfies these properties is the *Kron Reduction*. If we have a graph  $G$  and its set of vertices  $V = V_k \cup V_{nk}$ , where  $V_k$  are the subset of vertices

selected by the graph downsampling method, then the *Kron Reduction* of  $\mathbf{L}$  is the Schur complement of  $\mathbf{L}$  relative to  $\mathbf{L}_{V_k, V_k^c}$  [12], [8]. That is,

$$\mathbf{L}^{(i+1)} = \mathbf{L}_{V_k, V_k}^{(i)} - \mathbf{L}_{V_k, V_{nk}}^{(i)} (\mathbf{L}_{V_{nk}, V_{nk}}^{(i)})^{-1} \mathbf{L}_{V_{nk}, V_k}^{(i)} \quad (3.11)$$

where  $\mathbf{L}_{A,B}$  denotes the submatrix consisting of all entries of  $\mathbf{L}$  whose row index is in  $A$  and whose column index is in  $B$ .

### 3.3 Graph Sparsification

Due to the last property of the graph reduction method, repeated applications of Kron Reduction, one for each lower solution, results in graphs that become denser (the number of edges increases). To counter this effect, different algorithms have been proposed to remove the less important edges from the reduced graph [42]. Nevertheless, graph sparsification harms the performance of the graph matching algorithm, since the deleted edges should be the same for the two graphs being matched and this is difficult to achieve. For this reason, no graph sparsification has been applied in our multi-resolution approach.

Figure 3.8 illustrates how when performing graph sparsification to two similar graphs, the graphs in the lower resolution are not similar any more.

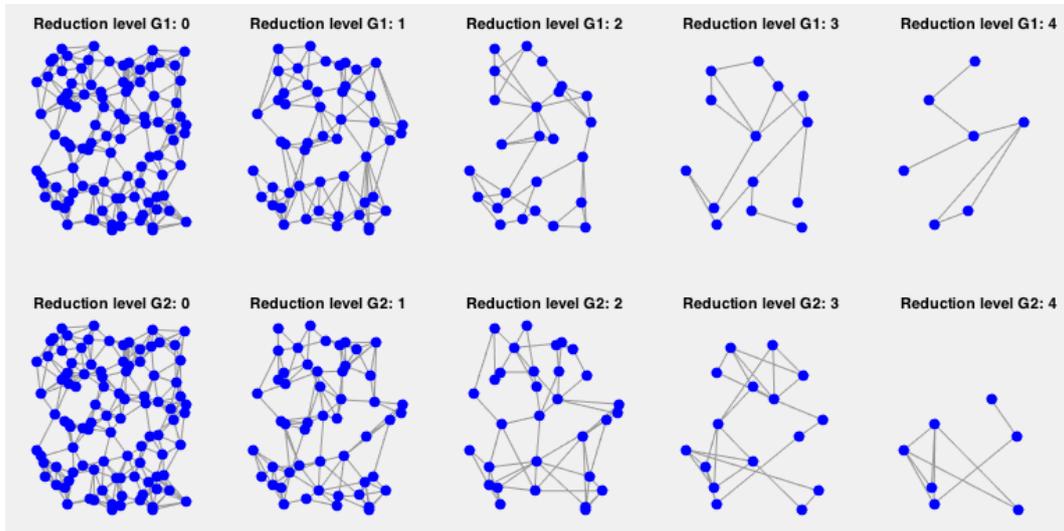


Figure 3.8: Graphical demonstration that shows why graph sparsification modifies the structure of two similar graphs.

Figure 3.9 illustrates how when performing repeated applications of Kron Reduction, one for each lower solution, results in graphs that get every time denser.

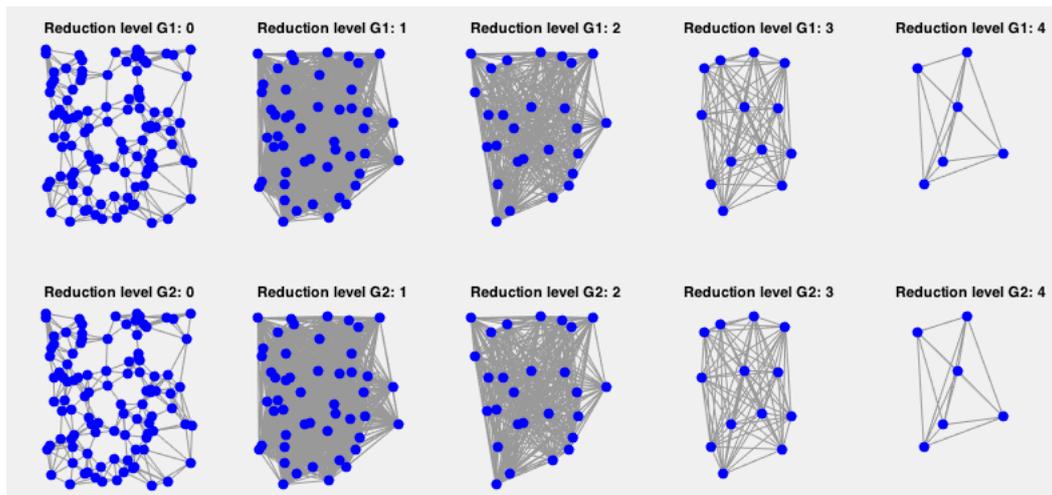


Figure 3.9: Graphical demonstration that shows the effect of Kron reduction over multiple resolutions.

## Chapter 4

# Weighted Graph Matching Problem

### 4.1 Isomorphic Graphs

This chapter describes how to decide the alignment in each resolution level. This task is solved using the spectral graph matching technique. Consider two identical graphs  $G_1$  and  $G_2$  where each of them has an Adjacency matrix  $\mathbf{W}$  defined as:

$$\mathbf{W}_{ij} = \begin{cases} w_{ij} & \text{if } e_{ij} \in E \\ 0 & \text{otherwise} \end{cases} \quad (4.1)$$

Since the labels given to the nodes may not match, if  $G_1$  and  $G_2$  are isomorphic weighted graphs (graphs which contain the same number of vertices connected in the same way), this will imply that the Adjacency matrix of  $G_2$  is a permutation of rows and columns of the Adjacency matrix of  $G_1$ . If  $\mathbf{W}_1$  and  $\mathbf{W}_2$  are their Adjacency matrices respectively, then the problem of finding a one-to-one correspondence between  $V_1$  and  $V_2$  can be formulated as the problem of finding a permutation matrix  $\mathbf{P}$  that minimizes the following error:

$$J(\mathbf{P}) = \|\mathbf{P}\mathbf{W}_2\mathbf{P}^T - \mathbf{W}_1\|^2 \quad (4.2)$$

where  $\|\cdot\|$  is the Frobenius norm and  $\mathbf{P}$  is a matrix with just one entry equal to 1 per row and column and the rest of the entries equal to 0 [49], [45] (example shown in Figure 4.1). In the case of matching two isomorphic graphs,  $J(\mathbf{P}) = 0$  for the best  $\mathbf{P}$ . If  $G_1$  and  $G_2$  are not isomorphic, the goal of the graph matching problem is still to find  $\mathbf{P}$  that minimizes  $J(\mathbf{P})$ . Trying all the possible permutation matrices is computationally unfeasible since, if  $\mathbf{P}$  is a permutation matrix of dimension  $N \times N$ , it entails the computation of  $N!$  different error functions.

Having computed the permutation matrix with an heuristic algorithm such as in [49] or [45], it is straightforward to find the alignment between the vertices,  $f: V_1 \rightarrow V_2$ , since:

$$\text{if } \mathbf{P}_{ij} \neq 0 \text{ then } (v_i \in V_1) \rightarrow (v'_j \in V_2) \quad (4.3)$$

In the case of matching two isomorphic graphs, the best permutation matrix that minimizes the error  $J(\mathbf{P})$  can be found by

$$\mathbf{P} = \mathbf{U}_1 \mathbf{S} \mathbf{U}_2^T \quad (4.4)$$

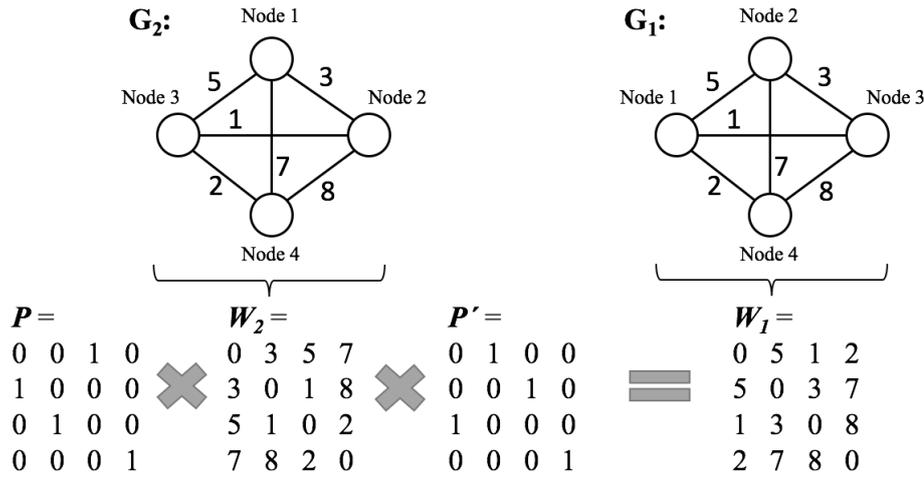


Figure 4.1: Illustration of the spectral graph matching problem. From the permutation matrix we obtain the final alignment: node 1 of  $G_1$  is aligned to node 3 of  $G_2$ , node 2 of  $G_1$  is aligned to node 1 of  $G_2$ , node 3 of  $G_1$  is aligned to node 2 of  $G_2$ , and node 4 of  $G_1$  is aligned to node 4 of  $G_2$ .

where  $\mathbf{U}_i$  is an orthogonal matrix whose columns correspond to the eigenvectors of the Adjacency matrix  $\mathbf{W}_i$  or the graph Laplacian  $\mathbf{L}_i$ . In this thesis we will work with the eigenvectors of  $\mathbf{L}_i$ . Furthermore,  $\mathbf{S}$  is defined as:

$$\mathbf{S} = \{\text{diag}(s_1, s_2, \dots, s_N) | s_i = 1 \text{ or } -1\} \quad (4.5)$$

and it is necessary to take into account the arbitrary determination of the signs of the eigenvectors by the eigenvalue program (sign ambiguity). Finding the right  $\mathbf{S}$  is not an easy task.

In summary, the graph matching problem can be reformulated as the problem of finding the right sign of the eigenvectors and the permutation of rows in the matrix  $\mathbf{U}_2$  that results in  $\mathbf{U}_1$ .

---

#### Proof Eq. 4.4:

The graph matching problem consists of finding:

$$\mathbf{P}^* = \arg \min_{\mathbf{P}} \|\mathbf{P}\mathbf{W}_2\mathbf{P}^T - \mathbf{W}_1\|^2$$

but can also be reformulated as

$$\mathbf{P}^* = \arg \min_{\mathbf{P}} \|\mathbf{P}\mathbf{L}_2\mathbf{P}^T - \mathbf{L}_1\|^2$$

Below we proof that  $\mathbf{P}^* = \mathbf{U}_1\mathbf{S}\mathbf{U}_2^T$ , where  $\mathbf{U}_i$  is an orthogonal matrix whose columns correspond to the eigenvectors of the graph Laplacian  $\mathbf{L}_i$ .

*Theorem 1* [25]: If  $\mathbf{A}$  and  $\mathbf{B}$  are Hermitian matrices with respective eigenvalues  $\alpha_1 \geq \alpha_2 \geq \dots \geq \alpha_N$  and  $\beta_1 \geq \beta_2 \geq \dots \geq \beta_N$ , then:

$$\|\mathbf{A} - \mathbf{B}\|^2 \geq \sum_{i=1}^N (\alpha_i - \beta_i)^2 \quad (4.6)$$

*Theorem 2:* Let  $A$  and  $B$  be real symmetric matrices with n distinct eigenvalues and their eigendecomposition be given by:

$$\mathbf{A} = \mathbf{U}_A \mathbf{\Lambda}_A \mathbf{U}_A^T \quad (4.7)$$

$$\mathbf{B} = \mathbf{U}_B \mathbf{\Lambda}_B \mathbf{U}_B^T \quad (4.8)$$

where  $\mathbf{U}_A$  and  $\mathbf{U}_B$  are orthogonal matrices and  $\mathbf{\Lambda}_A = \text{diag}(\alpha_i)$  and  $\mathbf{\Lambda}_B = \text{diag}(\beta_i)$ . Then, the minimum value of  $\|\mathbf{R}\mathbf{A}\mathbf{R}^T - \mathbf{B}\|^2$  (where  $\mathbf{R}$  is an orthogonal matrix) is given by any  $\mathbf{R}$  that can be decomposed as:

$$\mathbf{R} = \mathbf{U}_B \mathbf{S} \mathbf{U}_A^T \quad (4.9)$$

where  $\mathbf{S} = \{\text{diag}(s_1, s_2, \dots, s_N) | s_i = 1 \text{ or } -1\}$ .

Proof *Theorem 2:* Since the eigenvalues of  $\mathbf{R}\mathbf{A}\mathbf{R}^T$  are the same as the eigenvalues of  $\mathbf{A}$ , then (using *Theorem 1*):

$$\|\mathbf{R}\mathbf{A}\mathbf{R}^T - \mathbf{B}\|^2 \geq \sum_{i=1}^N (\alpha_i - \beta_i)^2 \quad (4.10)$$

Then,

$$\begin{aligned} \|\mathbf{R}\mathbf{A}\mathbf{R}^T - \mathbf{B}\|^2 &= \|(\mathbf{U}_B \mathbf{S} \mathbf{U}_A^T)(\mathbf{U}_A \mathbf{\Lambda}_A \mathbf{U}_A^T)(\mathbf{U}_B \mathbf{S} \mathbf{U}_A^T)^T - (\mathbf{U}_B \mathbf{\Lambda}_B \mathbf{U}_B^T)\|^2 = \\ &= \|(\mathbf{U}_B \mathbf{S} \mathbf{U}_A^T)(\mathbf{U}_A \mathbf{\Lambda}_A \mathbf{U}_A^T)(\mathbf{U}_A \mathbf{S} \mathbf{U}_B^T) - (\mathbf{U}_B \mathbf{\Lambda}_B \mathbf{U}_B^T)\|^2 = \\ &= \|\mathbf{U}_B \mathbf{S} \mathbf{\Lambda}_A \mathbf{S} \mathbf{U}_B^T - \mathbf{U}_B \mathbf{\Lambda}_B \mathbf{U}_B^T\|^2 = \|\mathbf{U}_B (\mathbf{S} \mathbf{\Lambda}_A \mathbf{S} - \mathbf{\Lambda}_B) \mathbf{U}_B^T\|^2 = \\ &= \|\mathbf{S} \mathbf{\Lambda}_A \mathbf{S} - \mathbf{\Lambda}_B\|^2 = \|\mathbf{\Lambda}_A - \mathbf{\Lambda}_B\|^2 = \sum_{i=1}^N (\alpha_i - \beta_i)^2 \end{aligned}$$

Therefore, when  $\mathbf{R} = \mathbf{U}_B \mathbf{S} \mathbf{U}_A^T$ , the lower bound of  $\|\mathbf{R}\mathbf{A}\mathbf{R}^T - \mathbf{B}\|^2$  is achieved.

The graph matching problem is based on finding the permutation matrix  $\mathbf{P}$  that minimizes  $\|\mathbf{P}\mathbf{W}_2\mathbf{P}^T - \mathbf{W}_1\|^2$ . Furthermore,

$$\mathbf{P}\mathbf{U}_A = \mathbf{U}_B \mathbf{S} \rightarrow \mathbf{P} = \mathbf{U}_B \mathbf{S} \mathbf{U}_A^T$$

because the eigenvectors of a matrix are uniquely determined except for their negative and positive directions (sign ambiguity) and rows are differently ordered due to the different labels assigned to the nodes of the two graphs being matched. Consequently, since  $\mathbf{R} = \mathbf{U}_B \mathbf{S} \mathbf{U}_A^T$  is a permutation matrix, the lower bound of the graph matching error is achieved for  $\mathbf{P} = \mathbf{U}_B \mathbf{S} \mathbf{U}_A^T$ .

There is a situation where the method of using  $\mathbf{P} = \mathbf{U}_B \mathbf{S} \mathbf{U}_A^T$  as the permutation matrix to solve the graph matching problem does not work. In case  $\mathbf{A}$  and  $\mathbf{B}$  do not have n distinct eigenvalues, we cannot guarantee that  $\mathbf{U}^T \mathbf{U} = \mathbf{I}$  (as has been assumed above).

Proof: If  $\mathbf{U}^T\mathbf{U} = \mathbf{I}$ , then the eigenvectors must be orthogonal among them. Moreover, for any real matrix  $\mathbf{A}$  and any vectors  $\mathbf{x}$  and  $\mathbf{y}$  we have that:

$$\langle \mathbf{A}\mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{x}, \mathbf{A}^T\mathbf{y} \rangle$$

Now assume  $\mathbf{A}$  is symmetric, and  $\mathbf{x}$  and  $\mathbf{y}$  are distinct eigenvectors of  $\mathbf{A}$  corresponding to distinct eigenvalues  $\lambda$  and  $\mu$ . Then,

$$\lambda \langle \mathbf{x}, \mathbf{y} \rangle = \langle \lambda\mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{A}\mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{x}, \mathbf{A}^T\mathbf{y} \rangle = \langle \mathbf{x}, \mu\mathbf{y} \rangle = \mu \langle \mathbf{x}, \mathbf{y} \rangle$$

Therefore,

$$(\lambda - \mu) \langle \mathbf{x}, \mathbf{y} \rangle = 0 \rightarrow \langle \mathbf{x}, \mathbf{y} \rangle = 0$$

since  $(\lambda - \mu) \neq 0$ . Consequently, we have shown that different eigenvectors are orthogonal if they have different eigenvalues. Nevertheless, two eigenvectors associated to the same eigenvalue may not be orthogonal.

Fortunately, the fact of having repeated eigenvalues is highly unlikely for irregular graphs, which makes this problem a minor issue.

Instead of finding the correct sign of the eigenvectors which allow us to compute  $\mathbf{S}$  and then  $\mathbf{P}$  ( $\mathbf{P} = \mathbf{U}_1\mathbf{S}\mathbf{U}_2^T$ ), [49] provides another solution: the optimum permutation matrix is obtained as the one that maximizes  $tr(\mathbf{P}|\mathbf{U}_1||\mathbf{U}_2^T|)$ , where  $|\mathbf{U}|$  is the absolute value of all the entries. Nevertheless, as mentioned in [49], this way of finding  $\mathbf{P}$  only works for isomorphic graphs and it is far from optimal when the two graphs are not identical or when they have different number of nodes. A comparison between the method of [49] and our method can be found in Chapter 7.

## 4.2 Non Isomorphic Graphs

In this project we propose a different way of approaching the graph matching problem for non isomorphic graphs by exploiting the observation that the variation (frequency interpretation) over the graph, defined by the entries of the eigenvectors of two similar graphs, is almost the same:  $\mathbf{u}_i \approx \mathbf{P}_i\mathbf{u}'_i$  (see Chapter 2.2). Consequently, the vertex with the largest *graph signal* as defined by the Fiedler eigenvector of  $G_1$  is likely to be correctly aligned with the vertex with the largest *graph signal* as defined by the Fiedler eigenvector of  $G_2$  (example in Fig. 4.2). These two nodes are likely to be in different positions (i.e., different labels) in their respective graphs. The same happens for the rest of the nodes: the two nodes with the second highest magnitude in both graphs are aligned and so on for the following ones.

Furthermore, if in addition to using the frequency information given by the Fiedler eigenvector we use other frequencies, the alignment is more likely to be more precise. For example, the vertex with the highest *graph signal* defined by

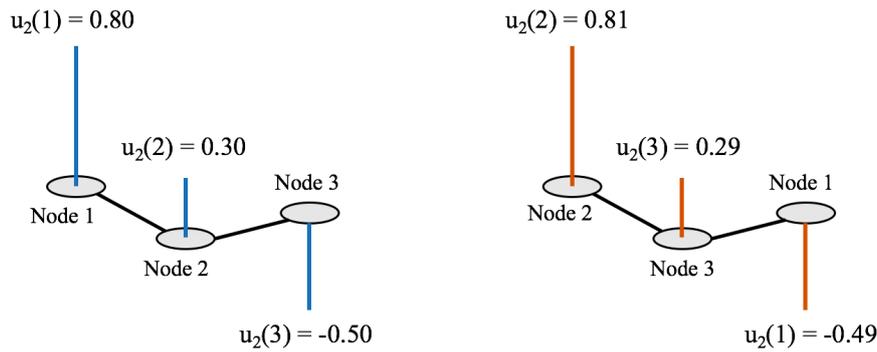


Figure 4.2: Representation of the graph signal defined by the entries of the Fiedler eigenvector of two similar graphs. Node 1 of  $G_1$  will be assigned to node 2 of  $G_2$ , node 2 of  $G_1$  will be assigned to node 3 of  $G_2$  and node 3 of  $G_1$  will be assigned to node 1 of  $G_2$ .

the Fiedler eigenvector and the third eigenvector of  $G_1$  is more likely to be correctly aligned with the vertex with the highest *graph signal* defined by the Fiedler eigenvector and the third eigenvector of  $G_2$ . In other words, what we propose is to embed the two graphs and use a distance measure in the embedded space.

A novel approach of our algorithm will be to put more weight on the information given by the frequencies less affected by the dissimilarity of the two graphs being matched. These frequencies are often the lowest frequencies (the ones associated to small eigenvalues). In a way this is expected, since when matching two similar graphs the major changes will be observed in the higher frequencies.

### 4.3 Proposed Approach

The algorithm begins either with two graphs, or with two point-clouds of 2- or 3-dimensional elements,  $\mathbf{X}_1$  and  $\mathbf{X}_2$ . In the latter case, the point-clouds are first transformed into a graph representation where the weights of the edges are defined as:

$$w_{ij} = \exp(-dist_{ij}^2/\sigma') \quad (4.11)$$

where  $dist()$  corresponds to the Euclidean distance. To avoid having an edge between all pairs of vertices of the graph, we only connect each node to its  $k$  closest neighbors.

Based on the above observations, below we present the steps to find a near-optimal correspondence between the graphs in each resolution level:  $G_1^{(i)}$  and  $G_2^{(i)}$  ( $f : V_1^{(i)} \rightarrow V_2^{(i)}$ ). This procedure is repeated in each resolution.

Fig. 4.3 illustrates a summary of the spectral graph matching algorithm through a multi-resolution approach, which is detailed below.

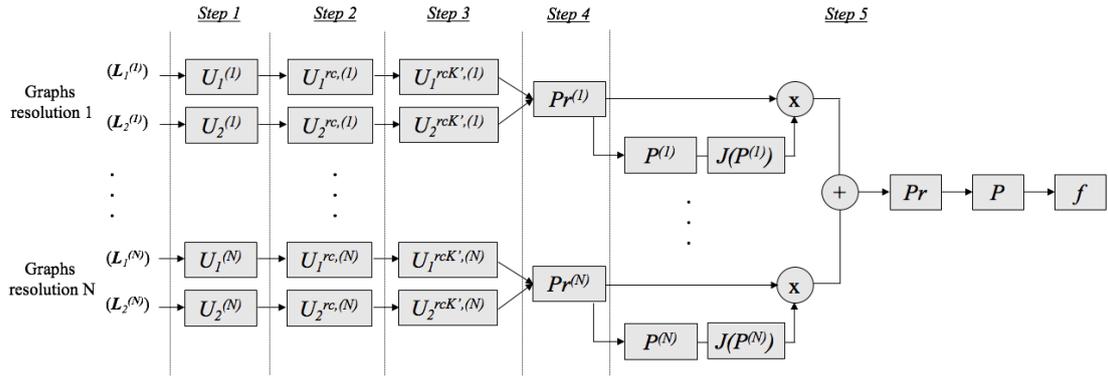


Figure 4.3: A multi-resolution approach for spectral graph matching algorithm.

### 4.3.1 Step 1: Compute and order the eigenfunctions

The first step is to compute the eigenfunctions of the graph Laplacian of the two graphs being matched. If  $G_1^{(i)}$  and  $G_2^{(i)}$  have  $M$  and  $N$  vertices respectively, then  $U_1 \in \mathbb{R}^{M \times M}$  and  $U_2 \in \mathbb{R}^{N \times N}$  are the matrices that store the eigenvectors by columns for each of the two graphs. Eigenvectors are ordered according to the magnitude of their associated eigenvalues, from smaller to larger, obtaining  $U_1^c \in \mathbb{R}^{M \times M}$  and  $U_2^c \in \mathbb{R}^{N \times N}$ , where

$$\begin{cases} U_1^c = [\mathbf{u}_1, \dots, \mathbf{u}_M] \\ U_2^c = [\mathbf{u}'_1, \dots, \mathbf{u}'_N] \end{cases} \quad (4.12)$$

### 4.3.2 Step 2: Select the first K eigenfunctions

Once the eigenvectors have been ordered, we select the first  $K$  ( $K \ll \min(M, N)$ ) eigenvectors from both  $U_1^c$  and  $U_2^c$ , obtaining  $U_1^{rc} \in \mathbb{R}^{M \times K}$  and  $U_2^{rc} \in \mathbb{R}^{N \times K}$ , where

$$\begin{cases} U_1^{rc} = [\mathbf{u}_1, \dots, \mathbf{u}_K] \\ U_2^{rc} = [\mathbf{u}'_1, \dots, \mathbf{u}'_K] \end{cases} \quad (4.13)$$

Therefore, its rows represent the coordinates of each of the two sets in the embedded  $K$ -dimensional space. We choose the first  $K$  eigenvectors of the graph Laplacian because the variation (frequency interpretation) of the first eigenvectors is often similar when matching similar graphs, while the last eigenvectors tend to be less consistent, providing wrong information, as defined in (2.8). In addition, it helps to reduce significantly the computational complexity of the algorithm.

### 4.3.3 Step 3: Remove the less similar eigenfunctions

Although the first  $K$  eigenvectors (the ones associated with smaller eigenvalues) tend to be more consistent between the two graphs being matched, still some of the eigenvectors of  $G_1^{(i)}$  do not match with the eigenvectors of  $G_2^{(i)}$ . There are two different reason why this may happen. On the one hand, some eigenvectors (i.e., frequencies) may have been strongly affected by the noise and deformations in the graphs, and they may be quite dissimilar, in spite of being associated to relatively low frequencies. For example, as depicted in Figure 4.4 and 4.5, aligning the points using the first three eigenvector (Fig. 4.4) of a particular graph is harder than aligning the points using only the first two eigenvectors (Fig. 4.5) [33]. In other words, the frequency information given by the third eigenvector has been strongly affected by the differences between the two graphs being matched (for instance, due to noise and graph deformations).

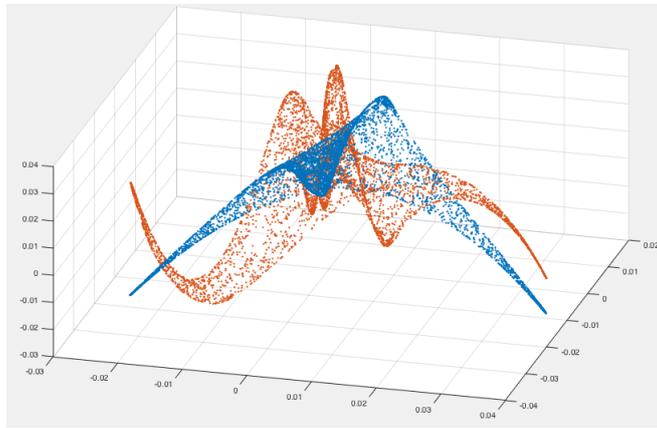


Figure 4.4: Graphs  $G_1^{(i)}$  and  $G_2^{(i)}$  embedded with the first three eigenfunctions.

On the other hand, although it is highly unlikely for irregular graphs, we may have repeated eigenvalues. The problem with repeated eigenvalues is the arbitrary determination of the eigenvectors by the eigenvalue program. That is, the eigenvectors associated to a multiple eigenvalue may be different in the two graphs, even if the graphs are identical. Therefore, an easy way to solve this problem is to remove the eigenvectors associated to repeated eigenvalues.

To remove these not well matched eigenfunctions between  $G_1^{(i)}$  and  $G_2^{(i)}$  (e.g., select an embedding based on only some frequencies), we compute a similarity measure between all pairs of eigenvectors. The most similar ones are kept and the other ones are excluded. Nevertheless, we face the same two obstacles as in the *SCE* graph downsampling method (see Chapter 3.1.2). That is, the entries of the eigenvectors are in different order due to the different indices associated to the two graphs being aligned. Besides, we have the arbitrary determination of the signs of the eigenvectors by the eigenvalue program (sign ambiguity). To overcome them, we use the same technique as in (3.8) and (3.9), where we sort the components of both vectors and compute two different costs. Therefore, we define

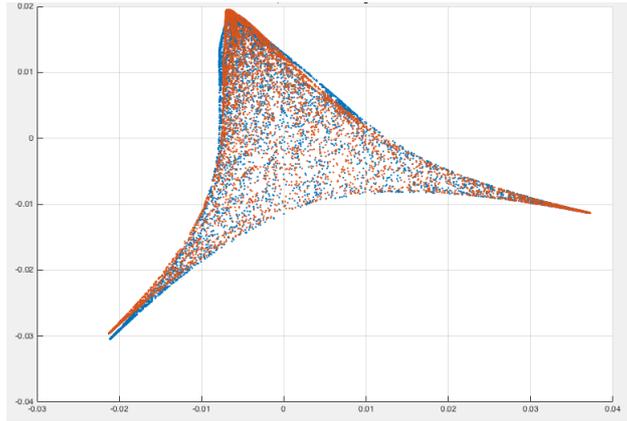


Figure 4.5: Graphs  $G_1^{(i)}$  and  $G_2^{(i)}$  embedded with the the first two eigenfunctions.

two new matrices  $\mathbf{H}$  and  $\mathbf{S}$  where  $\mathbf{H}$  measures the similarity between all pairs of eigenvectors and  $\mathbf{S}$  stores the sign that has given a lower cost. That is,

$$\mathbf{H}_{mn} = \min(\text{dist}(\mathbf{u}_m, \mathbf{u}'_n), \text{dist}(\mathbf{u}_m, -\mathbf{u}'_n)) \quad (4.14)$$

$$\mathbf{S}_{mn} = \begin{cases} +1 & \text{if } \mathbf{H}_{mn} = \text{dist}(\mathbf{u}_m, \mathbf{u}'_n) \\ -1 & \text{if } \mathbf{H}_{mn} = \text{dist}(\mathbf{u}_m, -\mathbf{u}'_n) \end{cases} \quad (4.15)$$

where

$$\text{dist}(\mathbf{a}, \mathbf{b}) = \|\text{sort}(\mathbf{a}) - \text{sort}(\mathbf{b})\|^2 \quad (4.16)$$

The lowest  $K'$  ( $K' < K$ ) values of  $\mathbf{H}_{mn}$  indicate the  $K'$  pairs of eigenvectors from  $G_1^{(i)}$  and  $G_2^{(i)}$  that are more similar. We use this information to order the  $K'$  eigenfunctions from both  $G_1^{(i)}$  and  $G_2^{(i)}$  so that eigenvectors in the same position are likely to match. Besides, the sign ambiguity problem is solved by using the information provided by  $\mathbf{S}$ . Combining all this, we generate the new matrices  $\mathbf{U}_1^{rcK'} \in \mathbb{R}^{M \times K'}$  and  $\mathbf{U}_2^{rcK'} \in \mathbb{R}^{N \times K'}$ , where

$$\begin{cases} \mathbf{U}_1^{rcK'} = \{[\mathbf{a}_1, \dots, \mathbf{a}_{K'}] : \mathbf{a}_i = \mathbf{u}_j\} \\ \mathbf{U}_2^{rcK'} = \{[\mathbf{b}_1, \dots, \mathbf{b}_{K'}] : \mathbf{b}_i = \mathbf{u}'_l * \mathbf{S}_{jl}\} \end{cases} \quad (4.17)$$

#### 4.3.4 Step 4: Find the best alignment between two graphs

As discussed in Chapter 4.1, to find the best alignment between the vertices of  $G_1^{(i)}$  and  $G_2^{(i)}$ , the graph matching problem can be reformulated as the problem of finding the right sign of the eigenvectors (already solved in the above step) and the permutation of rows in the matrix  $\mathbf{U}_2$  that makes it more similar to  $\mathbf{U}_1$  ( $\mathbf{P} = \mathbf{U}_1 \mathbf{S} \mathbf{U}_2^T$ ). To do so, one could compute the Euclidean distance between the rows of  $\mathbf{U}_1$  (which in our case corresponds to  $\mathbf{U}_1^{rcK'}$ ) and  $\mathbf{S} \mathbf{U}_2$  (which in our case corresponds to  $\mathbf{U}_2^{rcK'}$ ). Nonetheless, we compute a modified Euclidean distance

between the rows of  $\mathbf{U}_1^{rcK'}$  and  $\mathbf{U}_2^{rcK'}$  defined as

$$dist(\mathbf{U}_1^{rcK'}(m, :), \mathbf{U}_2^{rcK'}(n, :)) = \sum_{i=1}^{K'} \gamma_i (\mathbf{a}_i(m) - \mathbf{b}_i(n))^2 \quad (4.18)$$

where  $\gamma_i$  is directly proportional to the similarity between the pairs of eigenvectors of  $\mathbf{U}_1^{rcK'}$  and  $\mathbf{U}_2^{rcK'}$  (i.e.  $\mathbf{a}_i$  and  $\mathbf{b}_i$ ). The similarity information is provided by the matrix  $\mathbf{H}$ . In this way, we give more importance to the information provided by the eigenvectors that represent frequency information less affected by the dissimilarity of the two graphs being matched. That is, in the previous step we removed the less similar eigenvectors between  $G_1$  and  $G_2$  before computing the Euclidean distances between all the rows so as to find  $\mathbf{P}$ , which has the same meaning as using equation (4.18) with  $\gamma_i = 0$  for these dissimilar eigenvectors. Now, although having the most similar eigenvectors, some of the pairs are more similar than others. For this reason,  $\gamma_i$  is proportional to the similarity between each pair of eigenvectors ( $\mathbf{a}_i$ ,  $\mathbf{b}_i$ ). Experimentally, we have seen an increased performance of the graph matching when using  $\gamma_i$ .

Finally, we store the modified Euclidean distances between all the rows in a matrix  $\mathbf{Pr}^{(i)}$  where  $\mathbf{Pr}^{(i)}(m, n)$  indicates the certainty that node  $v_m$  of  $G_1^{(i)}$  is assigned to node  $v'_n$  of  $G_2^{(i)}$ , being a low value very provable. Therefore, using  $dist()$  defined in (4.18),

$$\mathbf{Pr}^{(i)}(m, n) = dist(\mathbf{U}_1^{rcK'}(m, :), \mathbf{U}_2^{rcK'}(n, :)) \quad (4.19)$$

### 4.3.5 Step 5: Combining multi-resolution information

The final goal of the matching algorithm is to compute the probability matrix  $\mathbf{Pr}$  that indicates the likelihood that each node of  $G_1$  is associated with a node in  $G_2$ . We will compute this probability matrix ( $\mathbf{Pr}$ ) by combining the probability matrix of each of the resolution levels ( $\mathbf{Pr}^{(i)}$ ). To compute the final probability matrix, for each pair of nodes we combine the probabilities computed in all resolutions where these pair of nodes appear. These probabilities are weighted according to the goodness of the alignment in each resolution.

More specifically, we first compute the permutation matrix for each level,  $\mathbf{P}^{(i)}$ . Each  $(i, j)$  entry of this matrix indicates whether node  $i$  of  $G_1^{(i)}$  is aligned with node  $j$  of  $G_2^{(i)}$ . If this is the case, the entry is set to 1; otherwise it is set to zero. To generate this matrix, we start by setting all entries equal to zero. Then, we search in  $\mathbf{Pr}^{(i)}$  for the smallest entry  $(i, j)$ , and we set to 1 the corresponding entry in  $\mathbf{P}^{(i)}$ . Then, we proceed to find the next smallest entry in  $\mathbf{Pr}^{(i)}$ , with the constraint that the nodes of the largest graph can have only a single assignment. We repeat this process until all nodes have been aligned.

Then, we compute the performance of the matching in each resolution, as

defined in (4.2):

$$\alpha_i = J(\mathbf{P}^{(i)}) = \left\| \mathbf{P}^{(i)} \mathbf{W}_2^{(i)} \mathbf{P}^{(i)T} - \mathbf{W}_1^{(i)} \right\|^2$$

Finally, the probability matrix that combines the information obtained in all the resolution levels is defined as:

$$\mathbf{Pr}(m, n) = \sum_{\{v_m \in res_{(i)} \mid v'_n \in res_{(i)}\}} \alpha_i \mathbf{Pr}^{(i)}(m, n) \quad (4.20)$$

That is, for each pair of vertices  $(v_m, v'_n)$ , we combine the probabilities of matching these nodes in all the resolution levels that they appear, each one weighted by the performance of the matching in that particular resolution.

Finally, we compute the permutation matrix of the complete graphs  $\mathbf{P}$  in the same way as explained above for each resolution, based on the probability matrix  $\mathbf{Pr}$ . Having computed the permutation matrix we find the alignment between the vertices,  $f : V_1 \rightarrow V_2$ , as:

$$\text{if } \mathbf{P}_{ij} \neq 0 \quad \text{then} \quad (v_i \in V_1) \rightarrow (v'_j \in V_2)$$

An important benefit of this approach is that multi-scale methods reveal structural information such as irregular structural patterns. Each different resolution provides complementary information that enrich the final alignment of the two graphs. Moreover, due to the graph downsampling method, the vertices in each resolution are often the ones that should be correctly matched, and this matching is easier since graphs are smaller.

Additionally, if we still want to reduce more the computational complexity of the algorithm, instead of computing a probability matrix in each resolution, we can fix the alignments already decided in lower resolutions, and simply compute the alignment for the new vertices.

### 4.3.6 Step 6 (optional): Improvement against graph deformations

In order to avoid poor results due to graph deformations, the final probability matrix  $\mathbf{Pr}$  can be computed slightly differently [28]. The goal is to perform a global alignment between some percentage ( $X$ ) of the vertices of  $G_1^{(i)}$  and  $G_2^{(i)}$ . These vertices, which we call *secure* vertices, are far from each other and are the ones that the algorithm is more sure of their correct matching. Then, as described later in this section, the vertices that have been more affected by the presence of noise or articulated objects (the ones that create the deformation of the graph) will be matched the last.

The first *secure* vertex  $\mathbf{sv}_1$  in resolution  $i$  can be computed as follows:

$$\mathbf{Y}^{(i)} = \mathbf{Pr}^{(i)} \quad (4.21)$$

$$\mathbf{sv}_1 = (a_1, b_1) = \arg \min_{(m,n)} (\mathbf{Y}^{(i)}) \quad (4.22)$$

Afterwards, to compute the second *secure* vertex  $\mathbf{sv}_2$ , we find a pair of vertices with high certainty of being mapped correctly but at the same time far away from the previous *secure* vertex  $\mathbf{sv}_1$ . Consequently,

$$\mathbf{Y}_{ij}^{(i)} = \mathbf{Y}_{ij}^{(i)} - \beta_1 (\text{dist}^{G_1^{(i)}}(i, a_1) + \text{dist}^{G_2^{(i)}}(j, b_1)) \quad (4.23)$$

$$\mathbf{sv}_2 = (a_2, b_2) = \arg \min_{(m,n)} (\mathbf{Y}^{(i)}) \quad (4.24)$$

where  $\text{dist}(x, y)$  corresponds to the geodesic distance and  $\beta_1$  is a parameter that decides how much we penalize vertices close to the *secure* vertex. This process is repeated in order to find more *secure* vertices. Once all *secure* vertices are found, we define a new matrix  $\mathbf{T}$  so as to reduce the certainty in the vertices that have been more affected by noise or articulated objects. That is,

$$\mathbf{T}_{i,j}^{(i)} = \sum_{l=1}^{|\mathbf{sv}|} |\text{dist}^{G_1^{(i)}}(i, a_l) - \text{dist}^{G_2^{(i)}}(j, b_l)| \quad (4.25)$$

Then, the final  $\mathbf{Pr}$  is computed as

$$\mathbf{Pr}(m, n) = \sum_{\{\forall res_{(i)} | v_m \& v'_n \in res_{(i)}\}} \left( \alpha_i \mathbf{Pr}^{(i)}(m, n) + \beta_2 \mathbf{T}^{(i)}(m, n) \right) \quad (4.26)$$

where  $\beta_2$  is a parameter that allow us to modulate how much we correct the probabilities of the alignments using the information provided by the *secure* nodes.

A summary of the graph matching method proposed in this project is shown in Algorithm 3.

**Algorithm 3** Graph matching algorithm

**Input:**  $\mathbf{X}_1$  and  $\mathbf{X}_2$  (2D/3D coordinates of  $G_1$  and  $G_2$ ),  $RN$  (number of resolutions),  $\beta_2$  (improvement against graph deformations),  $Z = X/100$  (% of *secure vertices*)

\*If the graphs are provided (instead of 2D/3D coordinates) jump to line 3.

**Output:**  $f : V_1 \rightarrow V_2$

```

1:  $\mathbf{W}_1^{(1)} \leftarrow \mathbf{X}_1$ 
2:  $\mathbf{W}_2^{(1)} \leftarrow \mathbf{X}_2$ 
3:  $\mathbf{L}_1^{(1)} \leftarrow \mathbf{W}_1^{(1)}$ 
4:  $\mathbf{L}_2^{(1)} \leftarrow \mathbf{W}_2^{(1)}$ 
5: for  $i = 1 : i < (RN + 1)$  do:
6:   % Compute and order the eigenfunctions
7:    $[\mathbf{U}_1^{(i)}, \mathbf{e}_1^{(i)}] \leftarrow \mathbf{L}_1^{(i)}$ 
8:    $[\mathbf{U}_2^{(i)}, \mathbf{e}_2^{(i)}] \leftarrow \mathbf{L}_2^{(i)}$ 
9:    $\mathbf{U}_1^{c,(i)} \leftarrow [\mathbf{U}_1^{(i)}, \mathbf{e}_1^{(i)}]$ 
10:   $\mathbf{U}_2^{c,(i)} \leftarrow [\mathbf{U}_2^{(i)}, \mathbf{e}_2^{(i)}]$ 
11:  % Select the first  $K$  eigenfunctions
12:   $\mathbf{U}_1^{rc,(i)} \leftarrow \mathbf{U}_1^{c,(i)}$ 
13:   $\mathbf{U}_2^{rc,(i)} \leftarrow \mathbf{U}_2^{c,(i)}$ 
14:  % Remove the less similar eigenfunctions
15:   $[\mathbf{H}^{(i)}, \mathbf{S}^{(i)}] \leftarrow [\mathbf{U}_1^{rc,(i)}, \mathbf{U}_2^{rc,(i)}]$ 
16:   $\mathbf{U}_1^{rcK',(i)} \leftarrow [\mathbf{H}^{(i)}, \mathbf{U}_1^{rc,(i)}]$ 
17:   $\mathbf{U}_2^{rcK',(i)} \leftarrow [\mathbf{H}^{(i)}, \mathbf{S}^{(i)}, \mathbf{U}_2^{rc,(i)}]$ 
18:  % Find the best alignment between  $G_1^{(i)}$  and  $G_2^{(i)}$ 
19:   $\mathbf{Pr}^{(i)} \leftarrow [\mathbf{U}_1^{rcK',(i)}, \mathbf{U}_2^{rcK',(i)}]$ 
20:  % Combining multi-resolution information
21:   $\mathbf{P} \leftarrow \mathbf{Pr}^{(i)}$ 
22:   $\alpha_i = \left\| \mathbf{P}^{(i)} \mathbf{W}_1^{(i)} \mathbf{P}^{(i)T} - \mathbf{W}_2^{(i)} \right\|^2$ 
23:  if  $\beta_2 = 0$ 
24:     $\mathbf{Pr} = \mathbf{Pr} + \alpha_i \mathbf{Pr}^{(i)}$ 
25:  else
26:    % Improvement against graph deformations
27:     $\mathbf{Y}^{(i)} = \mathbf{Pr}^{(i)}$ 
28:     $\mathbf{T}^{(i)} = 0$ 
29:    for  $j = 1 : j \leq Z[\text{size}((\mathbf{W})_1^{(i)}, 1)]$  do:
30:       $\mathbf{sv}_j = (a_j, b_j) = \text{argmin}_{(m,n)}(\mathbf{Y}^{(i)})$ 
31:       $\mathbf{Y}^{(i+1)} \leftarrow [\mathbf{Y}^{(i)}, \mathbf{sv}_j, \beta_1]$ 
32:       $\mathbf{T}^i \leftarrow [\mathbf{T}^i, \mathbf{sv}_j, \mathbf{W}_1^i, \mathbf{W}_2^i]$ 
33:    end for
34:     $\mathbf{Pr} = \mathbf{Pr} + (\alpha_i \mathbf{Pr}^{(i)} + \beta_2 \mathbf{T}^{(i)})$ 
35:  end
36:  % Graph downsampling method (MDG or CSE) & Kron Reduction
37:   $\mathbf{L}_1^{(i+1)} \leftarrow \mathbf{L}_1^{(i)}$ 
38:   $\mathbf{L}_2^{(i+1)} \leftarrow \mathbf{L}_2^{(i)}$ 
39: end for
40:  $\mathbf{P} \leftarrow \mathbf{Pr}$ 
41:  $f \leftarrow \mathbf{P}$ 

```

## Chapter 5

# Point Set Registration

In computer vision and pattern recognition, point set registration is the process of finding a spatial transformation that aligns two point sets. The problem can be summarized as follows: let define two finite size point sets  $\{S_1, S_2\}$  in a finite dimensional vector space  $\mathbb{R}^d$ , which contain  $M$  and  $N$  points respectively. The problem is to find a transformation to be applied to one of the point sets, for example  $S_1$ , such that the difference between the transformed  $S_1$  and  $S_2$  is minimized. Typically, the set that is transformed is called the data set while the other one is called the model set. The point set registration algorithm can be applied in addition to the spectral graph matching algorithm to slightly improve the performance in some cases of the final matching. Nevertheless, the computational complexity is increased due to the point set registration step.

There are two types of point set registration methods: rigid and non-rigid registration. On the one hand, rigid registration applies a rigid transformation which consists of a translation and a rotation. On the other hand, non-rigid registration uses a non-rigid transformation which includes affine transformations such as scaling and shear mapping. Figure 5.1 illustrates an example of a rigid registration.

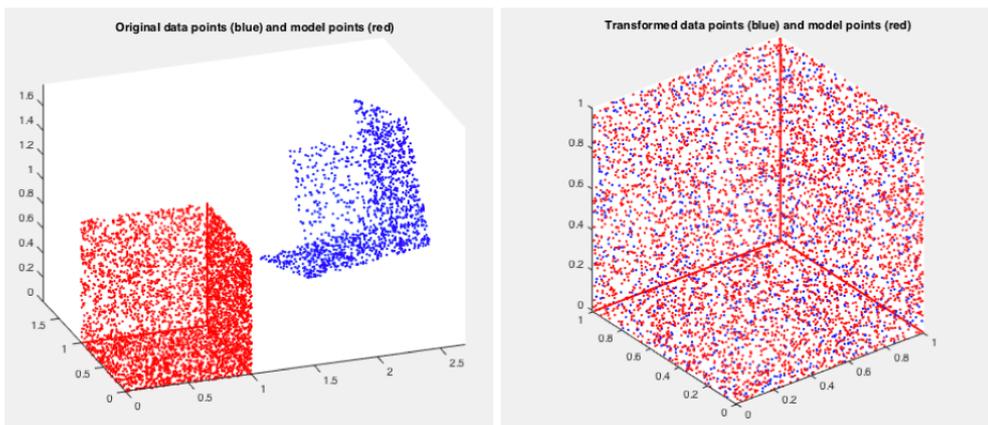


Figure 5.1: Original and transformed model set and data set

As defined in Chapter 4, we use (4.18) to compute the Euclidean distance between the rows of  $\mathbf{U}_1^{rcK'}$  and  $\mathbf{U}_2^{rcK'}$ , where  $\mathbf{U}_1^{rcK'} \in \mathbb{R}^{M \times K'}$  and  $\mathbf{U}_2^{rcK'} \in \mathbb{R}^{N \times K'}$ . This problem can be understood as finding the closest points between the two point sets defined by  $\mathbf{U}_1^{rcK'}$  and  $\mathbf{U}_2^{rcK'}$ , where each  $K'$ -dimensional point is defined by a row of these two matrices. Since  $\mathbf{U}_1^{rcK'}$  and  $\mathbf{U}_2^{rcK'}$  represent the frequency information of the two graphs being matched, the object created by the points

defined by  $\mathbf{U}_1^{rcK'}$  should be similar to the object created by the points of  $\mathbf{U}_2^{rcK'}$ . Nevertheless, if one of the objects is slightly displaced, the final matching using the euclidean distance can be considerably harmed. Consequently, the final performance can be improved by applying a point set registration algorithm to these two point sets before computing the Euclidean distance.

The method we use to perform the point set registration is called robust registration of point sets using iteratively reweighted least squares [4], which is a modified version of the Iterative Closest Point (ICP), one of the best known algorithms to perform a point set registration [5].

As mentioned above, to perform a rigid registration, we must find a rotation and translation that produces the best fit between a set of data points and a set of model points. To simplify the notation, we define the set of data points ( $\mathbf{U}_1^{rcK'}(i, :)$ ) as  $\mathbf{p}_i$  and the set of model points ( $\mathbf{U}_2^{rcK'}(i, :)$ ) as  $\mathbf{x}_i$ . Since we consider robust registration of point sets in  $\mathbb{R}^{K'}$ , the rotation matrix is defined as  $\mathbf{R} \in \{\mathbb{R}^{K' \times K'} | \mathbf{R}^T \mathbf{R} = \mathbf{I}, \det(\mathbf{R}) = +1\}$  and the translation vector is defined as  $\mathbf{t} \in \mathbb{R}^{K'}$ .

The point registration algorithm is iterative and consists of two principal stages in each iteration. The first step is to find the closest model points to the data points. The second step is to find a rigid body transformation such that the data points are fitted to the closest model points. Therefore, we begin initializing the parameters  $\mathbf{R}^0 = \mathbf{I}$  and  $\mathbf{t}^0 = \mathbf{0}$ . Then, for each data point  $\mathbf{p}_i$  we find which is the model point that is closest to it. We refer to this model point as  $\mathbf{y}_i = \mathbf{x}_{j(i)}$ . Next, we find  $\mathbf{R}$  and  $\mathbf{t}$  that minimizes the following error:

$$E = \sum_{i=1}^N \|(\mathbf{R}\mathbf{p}_i + \mathbf{t}) - \mathbf{y}_i\|^2 \quad (5.1)$$

Finally, we apply the rotation  $\mathbf{R}$  and translation  $\mathbf{t}$  to all data points  $\mathbf{p}_i$ . These two steps (finding the closest model points to the data points and applying the new  $\mathbf{R}$  and  $\mathbf{t}$  computed) are repeated using the new data points until the error  $E$  is below a given threshold or a maximum number of iterations is reached.

To find the rotation matrix and the translation vector that minimizes the error of equation (5.1) we need to define

$$\bar{\mathbf{p}} = \frac{1}{N} \sum_{i=1}^N \mathbf{p}_i \quad \bar{\mathbf{y}} = \frac{1}{N} \sum_{i=1}^N \mathbf{y}_i \quad (5.2)$$

Then, we define a new vector  $\mathbf{u}$  such as  $\mathbf{t} = -\mathbf{R}\bar{\mathbf{p}} + \bar{\mathbf{y}} + \mathbf{u}$ . Consequently, equation (5.1) can be reformulated as [4]:

$$E = \sum_{i=1}^N \|\mathbf{R}\mathbf{p}_i - \mathbf{R}\bar{\mathbf{p}} + \bar{\mathbf{y}} + \mathbf{u} - \mathbf{y}_i\|^2 =$$

$$= \sum_{i=1}^N (\mathbf{R}(\mathbf{p}_i - \bar{\mathbf{p}}) - (\mathbf{y}_i - \bar{\mathbf{y}}) + \mathbf{u})^T (\mathbf{R}(\mathbf{p}_i - \bar{\mathbf{p}}) - (\mathbf{y}_i - \bar{\mathbf{y}}) + \mathbf{u})$$

Since both the sum of all  $(\mathbf{p}_i - \bar{\mathbf{p}})$  and the sum of all  $(\mathbf{y}_i - \bar{\mathbf{y}})$  are equal to the zero vector, their scalar product with the  $\mathbf{u}$  vector disappear. Thus, the expression of the error can be simplified as

$$E = \sum_{i=1}^N \|\mathbf{p}_i - \bar{\mathbf{p}}\|^2 + \sum_{i=1}^N \|\mathbf{y}_i - \bar{\mathbf{y}}\|^2 + N\|\mathbf{u}\|^2 - 2N(\text{Trace}(\mathbf{R}\mathbf{C})) \quad (5.3)$$

where  $\mathbf{C}$  is defined as

$$\mathbf{C} = \frac{1}{N} \sum_{i=1}^N (\mathbf{p}_i \mathbf{y}_i^T) - \bar{\mathbf{p}} \bar{\mathbf{y}}^T \quad (5.4)$$

This error  $E$  is minimized when  $\mathbf{u} = \mathbf{0}$  (positive term in (5.3)) and by finding  $\mathbf{R}$  that maximizes  $\text{Trace}(\mathbf{R}\mathbf{C})$  (negative term in (5.3)). Below we prove that the optimal  $\mathbf{R}$ ,  $\mathbf{R}^*$ , is found from the singular value decomposition of  $\mathbf{C}$ ,  $\mathbf{C} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ . That is,  $\mathbf{R}^* = \mathbf{V}\mathbf{U}^T$ . Then, from  $\mathbf{R}^*$  and  $\mathbf{u} = \mathbf{0}$ , we obtain the optimal translation vector  $\mathbf{t}^*$ ,  $\mathbf{t}^* = \bar{\mathbf{y}} - \mathbf{R}^* \bar{\mathbf{p}}$ .

**Proof  $\mathbf{R}^* = \mathbf{V}\mathbf{U}^T$  [2]:**

*Lemma:* For any positive definite matrix  $\mathbf{A}\mathbf{A}^T$ , and any orthonormal matrix  $\mathbf{B}$ ,

$$\text{Trace}(\mathbf{A}\mathbf{A}^T) \geq \text{Trace}(\mathbf{B}\mathbf{A}\mathbf{A}^T)$$

*Proof of Lemma:* Let define  $\mathbf{a}_i$  as the  $i$ -th column of  $\mathbf{A}$ . We therefore can state that

$$\text{Trace}(\mathbf{B}\mathbf{A}\mathbf{A}^T) = \text{Trace}(\mathbf{A}^T \mathbf{B} \mathbf{A}) = \sum_i \mathbf{a}_i^T (\mathbf{B} \mathbf{a}_i)$$

$$\text{Using the Schwarz inequality: } \mathbf{a}_i^T (\mathbf{B} \mathbf{a}_i) \leq \sqrt{(\mathbf{a}_i^T \mathbf{a}_i)(\mathbf{a}_i^T \mathbf{B}^T \mathbf{B} \mathbf{a}_i)} = \mathbf{a}_i^T \mathbf{a}_i$$

$$\text{Consequently, } \sum_i \mathbf{a}_i^T \mathbf{a}_i = \text{Trace}(\mathbf{A}\mathbf{A}^T) \geq \text{Trace}(\mathbf{B}\mathbf{A}\mathbf{A}^T)$$

Let the SVD of  $\mathbf{H}$  be:  $\mathbf{H} = \mathbf{U}\mathbf{\Lambda}\mathbf{V}^T$ , where  $\mathbf{\Lambda}$  is diagonal matrix with positive elements and  $\mathbf{U}$  and  $\mathbf{V}$  are orthonormal matrices. Then, if we define  $\mathbf{R} = \mathbf{V}\mathbf{U}^T$  (which is orthonormal), we can say that:

$$\mathbf{R}\mathbf{H} = \mathbf{V}\mathbf{U}^T \mathbf{U}\mathbf{\Lambda}\mathbf{V}^T = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$$

which is positive definite ( $\mathbf{z}^T \mathbf{V}\mathbf{V}^T \mathbf{z} = \mathbf{z}^T \mathbf{V} (\mathbf{z}^T \mathbf{V})^T > 0$ ). Consequently, using the *Lemma* stated above, for any orthonormal matrix  $\mathbf{B}$ :

$$\text{Trace}(\mathbf{R}\mathbf{H}) \geq \text{Trace}(\mathbf{B}\mathbf{R}\mathbf{H})$$

Therefore, since  $\mathbf{B}\mathbf{R}$  is any other rotation matrix different to  $\mathbf{R}$ , if  $\mathbf{R} = \mathbf{V}\mathbf{U}^T$ ,  $\text{Trace}(\mathbf{R}\mathbf{H})$  is maximized.



## Chapter 6

# Experimental Results

To evaluate the proposed graph matching algorithm we have focused on three different cases: graphs generated by random coordinates on the space, graphs generated by 3-dimensional point-clouds of rigid objects and graphs generated by 3-dimensional point-clouds of a video.

### 6.1 Random graphs

The first set of experiments consists of applying the graph matching algorithm to graphs such that one of them is generated by random two-dimensional coordinates over the space and the other is a noisy version of the original one, by applying a Gaussian noise over the two-dimensional coordinates. The variance of the Gaussian noise determines the similarity between the two graphs. Once we have defined the x-y coordinates of the vertices for both graphs, each vertex is connected to its  $k$  nearest nodes. The weights assigned to the edges are computed using (4.11).

To visually measure the performance of our algorithm <sup>1</sup>, each pair of aligned vertices are painted in the same color. That is, if  $v_i$  is colored in blue and the graph matching algorithm decides that  $(v_i \in V_1) \rightarrow (v'_j \in V_2)$ , then  $v'_j$  will be colored in blue as well. We have also numbered the colors to facilitate the location of the corresponding nodes. Furthermore, the final graph matching error  $J(\mathbf{P})$  is computed and displayed on top of the graphs.

Figs. 6.1.a, 6.1.b and 6.1.c show the performance of our graph matching algorithm for different levels of noise (from low to high). This final alignment computed by our scheme combines the information provided by all the resolution levels. Black arrows pinpoint the vertices that have not been correctly aligned. Fig. 6.2 displays the details for each resolution level that the algorithm obtains when matching two random graphs. In this case, we use the *DSC* downsampling method since it outperforms the *CSE* downsampling method when the graphs being matched do not have many vertices. If we focus on the different resolution levels in our algorithm, it can be seen that the vertices in each lower resolution are the ones that should be correctly matched, proving again the effectiveness of the graph downsampling method.

Matching random graphs is useful in applications such as indoor localization.

<sup>1</sup>Code available in <https://github.com/victor-gonzalez-navarro/TFG>

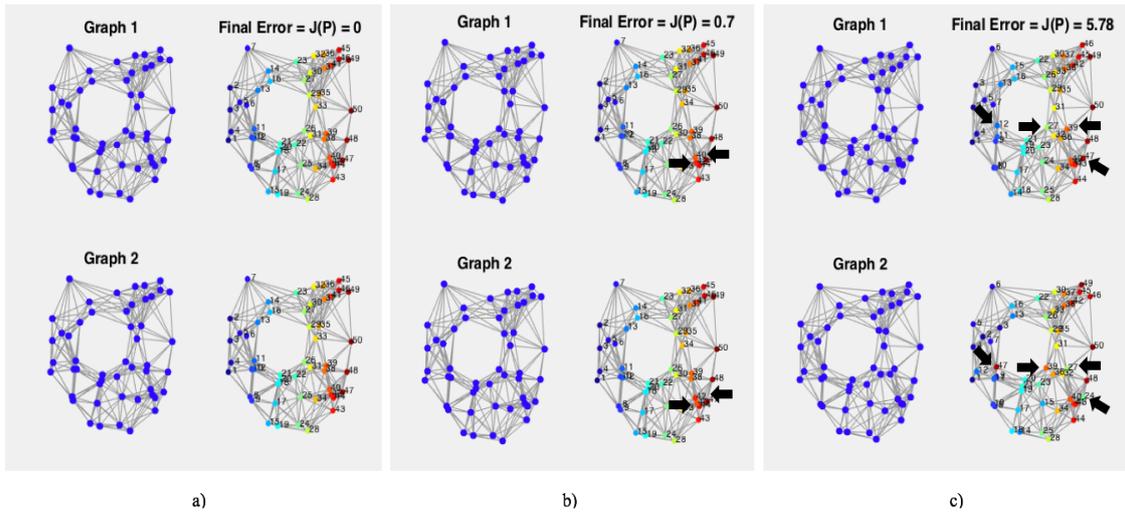


Figure 6.1: Performance of our graph matching algorithm for a) identical graphs; b) very similar graphs; c) similar graphs.

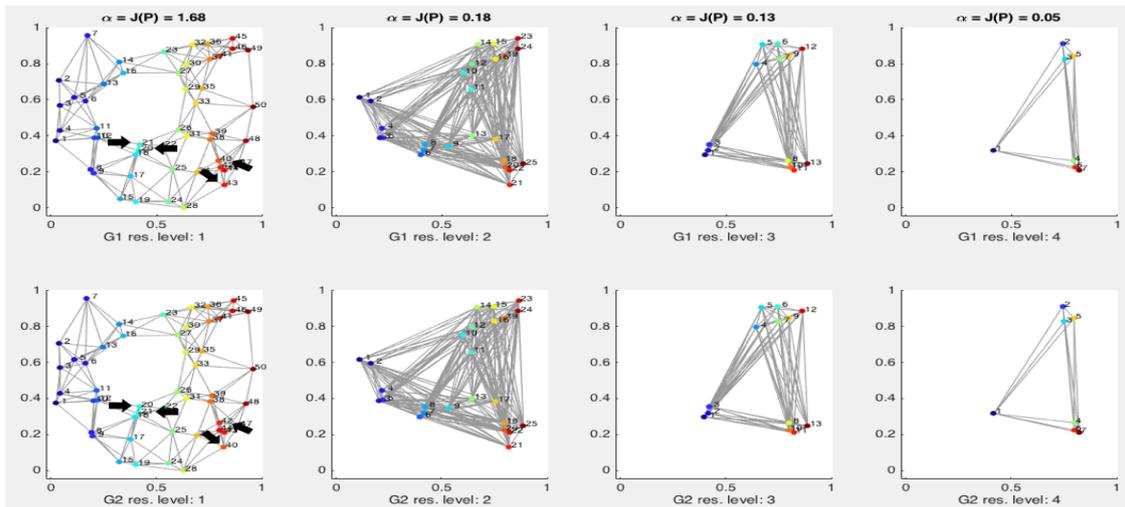


Figure 6.2: Details of the resolution levels using the multi-resolution approach for spectral graph matching.

Previous works generate a semantic map of passable areas and locate the user within that map. This semantic map is scale and direction free, and matching this semantic map to the actual floor plan is done manually by finding a direction and a scale [44]. However, to translate topological indoor localization to geographical localization we can use a graph matching algorithm, since the semantic map and the floor plan can be both modeled as graph. In [44], to evaluate the graph matching algorithm that they propose, one of the graphs is generated randomly while the other one is a noisy version of the previous graph, as it is done in this section. We compared our algorithm against theirs for the same graphs, and we found that our graph matching outperforms considerably their results.

## 6.2 3D point-clouds

Another very powerful application in the computer vision field is 3D point-cloud matching of rigid objects [27], [35]. For example, Yuan *et al.* [51] proposed a 3D point-cloud matching based on principal component analysis and iterative closest point. They generate the second graph by adding random noise to the x,y,z coordinates of one of the point-clouds. Since our graph matching algorithm can handle 3D coordinates, it is also capable of matching point-clouds.

We have evaluated our proposed algorithm using this type of graphs and the *CSE* downsampling method. Fig. 6.3, 6.4 and 6.5 illustrate the results of our graph matching algorithm applied to a rigid object represented by a point-cloud (the two graphs being matched are less similar for each of the experiments). Figure 6.6 illustrates the results of our graph matching algorithm applied to a different object. In this case, to facilitate the visualization of the results, we paint the nodes of the first graph with different colors but using similar colors for nodes that are close. In this case, we do not show the intermediate resolution levels which have been obtained using the *CSE* downsampling method, instead we plot different perspectives of the same 3D object to better visualize the results.

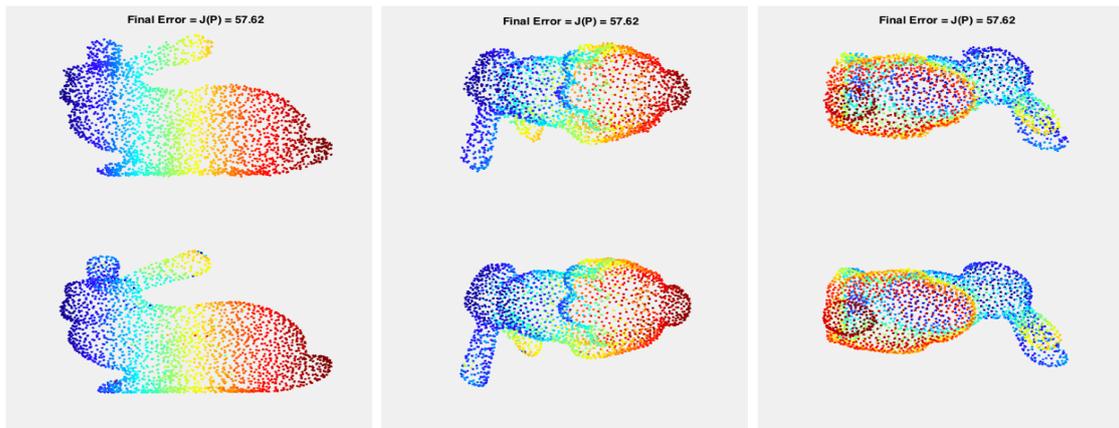


Figure 6.3: Performance of the graph matching algorithm with very similar 3D point-clouds (*3 different perspectives of bunny*).

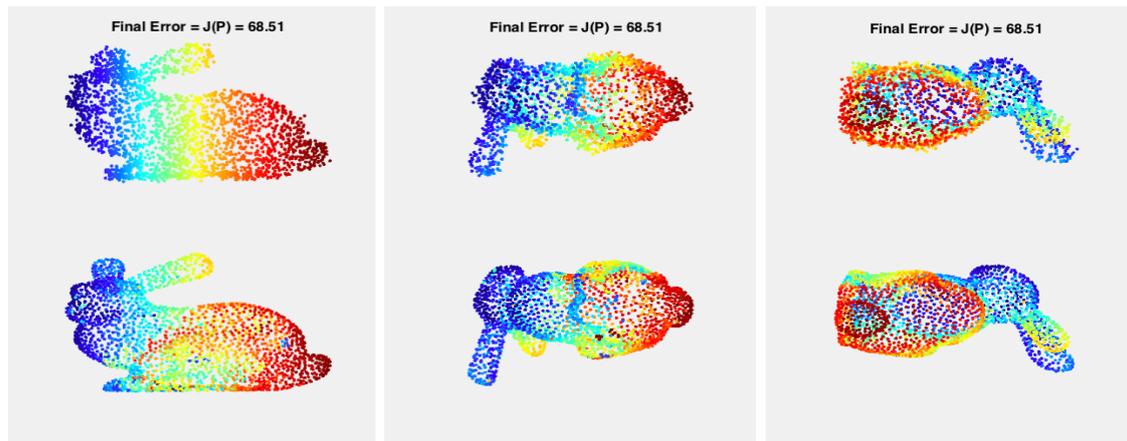


Figure 6.4: Performance of the graph matching algorithm with similar 3D point-clouds (*3 different perspectives of bunny*).

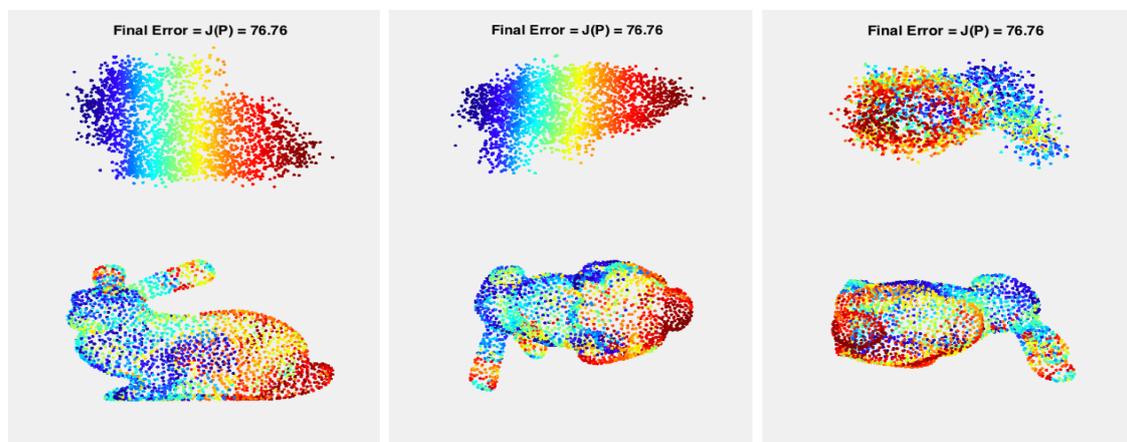


Figure 6.5: Performance of the graph matching algorithm with not similar 3D point-clouds (*3 different perspectives of bunny*).

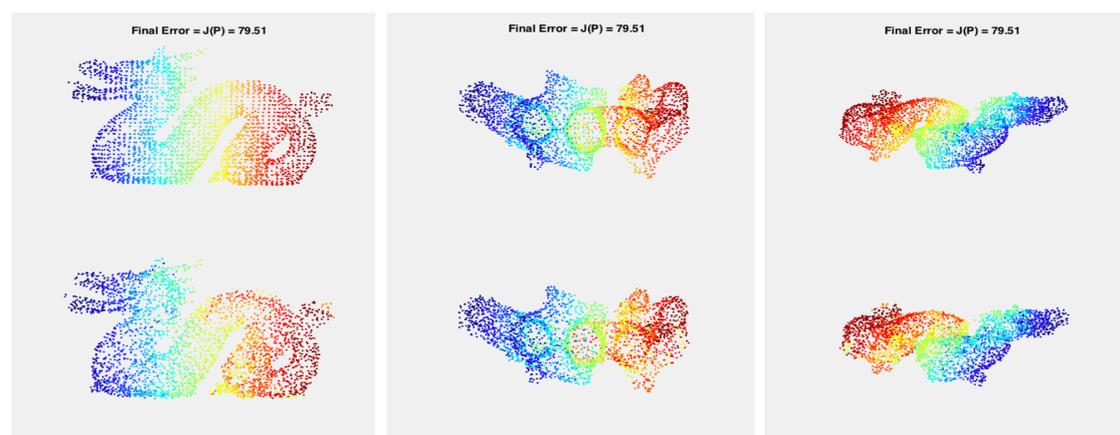


Figure 6.6: Performance of the graph matching algorithm with 3D point-clouds (*3 different perspectives of dragon*).

### 6.3 Video compression of point-clouds

With the advent of virtual and augmented reality, the captured information needs to be compressed for transmission and storage [36], [43]. For this reason, the last application used to test the performance of the graph matching algorithm consists of applying it to 3D point-clouds obtained from two consecutive frames of a video<sup>2</sup>. That is, one of the graphs is created from the 3D coordinates of the frame  $t$  while the other graph is created from the 3D coordinates of the frame  $t + 1$ . In this case, the problem is more challenging since the object in the video is an articulated object (a human) and his position changes between frames. Consequently, not only we are working with the presence of noise but also with graph deformations. Between the two proposed graph downsampling methods, we have seen that the *Corrected Second Eigenvector* outperforms the *Maximum Degree Gap* when the number of vertices is high.

As in the previous example for 3D rigid objects, to visually evaluate the performance of the algorithm we paint closer vertices with a similar color in one of the two graphs being matched. This facilitates the visualization of the vertices that are not correctly aligned by the matching algorithm. Figures 6.7 and 6.8 show the result of our graph matching algorithm applied to 3D point-clouds from a video sequence using the *CSE* downsampling method and the *MDG* downsampling method respectively. Three different perspectives are displayed for a better visualization. The graph matching error is increased in the second case, since the *MDG* downsampling algorithm is not as effective as the *CSE* downsampling algorithm.

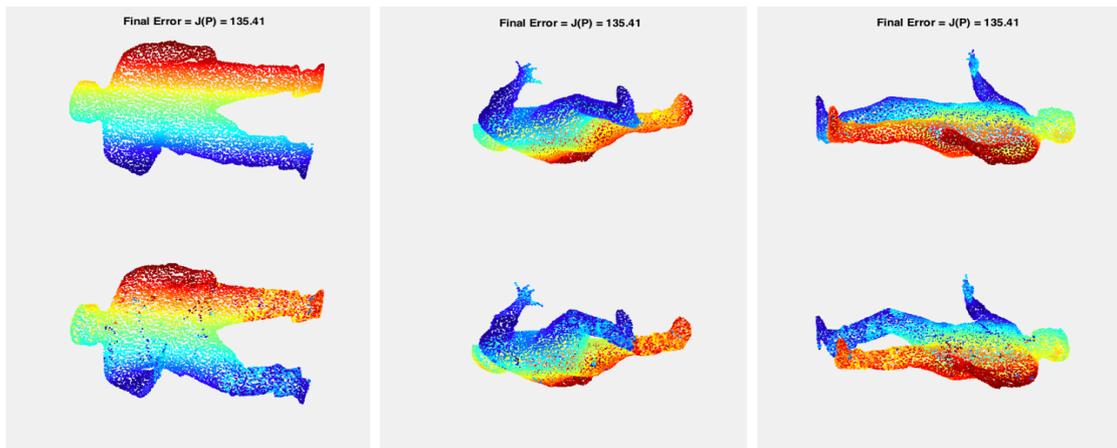


Figure 6.7: Performance of the graph matching algorithm using the *CSE* downsampling method with 3D point-clouds taken from a video (*3 different perspectives of human*).

<sup>2</sup>Point-cloud dataset available in <https://jpeg.org/plenodb/pc/8ilabs/>

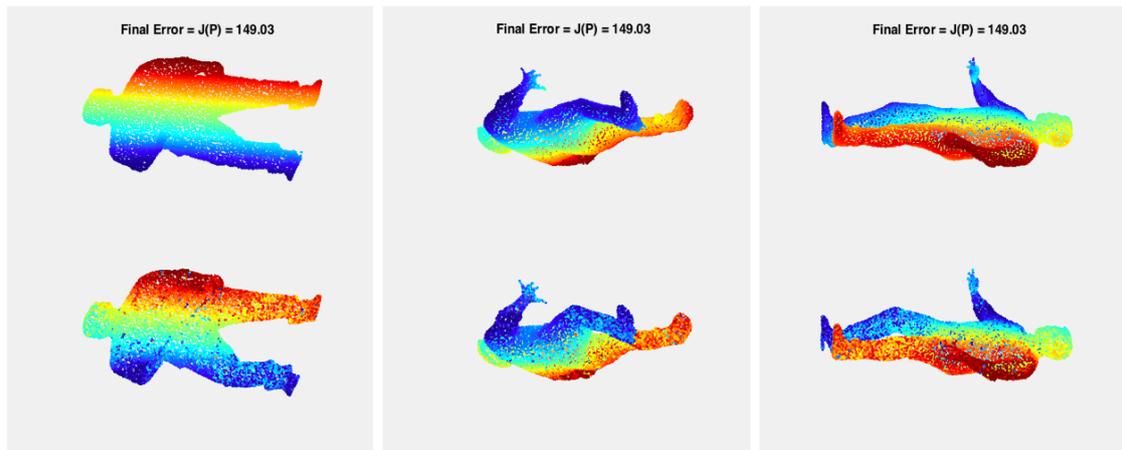


Figure 6.8: Performance of the graph matching algorithm using the *MDG* down-sampling method with 3D point-clouds taken from a video (*3 different perspectives of human*).

## Chapter 7

# Comparison With Previous Approaches

### 7.1 Comparison with other spectral graph matching algorithms

In order to have a global perspective of the quality of our graph matching algorithm, we compare the performance using the four graphs generated in Fig. 6.2, 6.3, 6.6 and 6.7 (random graphs, bunny, dragon and human) with other popular spectral graph matching algorithms. To measure objectively the performance of each of them, we compute  $J(\mathbf{P})$ . The numerical results are presented in Table 7.1, with one row per graph and one column per method, as follows:

- A: Shinji Umeyama [49]
- B: David Knossow et al. [30]
- C: Caelli & Kosinov [7]
- D: Our algorithm

Table 7.1: Comparison of the graph matching error  $J(\mathbf{P})$  for different spectral graph matching algorithms

Graph	A	B	C	D
<b>Random</b>	5.98	0.43	5.03	<b>0.26</b>
<b>Bunny</b>	85.98	79.40	59.39	<b>57.62</b>
<b>Dragon</b>	97.55	91.75	79.65	<b>79.51</b>
<b>Human</b>	170.49	166.11	136.63	<b>135.41</b>

Table 7.1 shows that our algorithm obtains the best results in all the evaluated graphs, outperforming the other methods.

Umeyama's method consist of finding the permutation matrix as the one that maximizes  $tr(\mathbf{P}|\mathbf{U}_1||\mathbf{U}_2^T|)$ , where  $|\mathbf{U}|$  is the absolute value of all the entries. This way of approaching the graph matching problem is penalized by not solving the sign ambiguity of the eigenvectors. For isomorphic graphs, this method is proved to find the permutation matrix  $\mathbf{P}$  that minimizes  $J(\mathbf{P})$ , which is equal to zero. Nevertheless, for graphs that are different, the result is far from optimal.

Knossow’s method is based on keeping the first eigenfunctions and using histograms to correct the sign ambiguity of all the eigenvectors. The histogram method is very sensitive to the number of chosen bins, resulting often in unsatisfactory results.

Caelli and Kosinov normalize the first eigenvectors and correct their sign with the dominant sign correction method, which does not perform well since most eigenvectors have about the same number of positive and negative entries, and a small difference between them may just be due to noise.

On the other hand, the algorithm we propose benefits from leveraging three main techniques:

- Having multiple resolutions of the same graph to perform the matching.
- Combining a sign ambiguity detector with the Fiedler eigenvector to correctly downsample the two graphs being matched.
- Weighting more the frequencies less affected by the dissimilarity between the two graphs being aligned.

## 7.2 Sensitivity Analysis

In this section we evaluate how different components of our scheme contribute to its final performance.

First, we evaluate the contribution of using one of our two downsampling methods (*MDG* and *SCE*) versus using other existing graph sampling algorithms: spectral graph downsampling [47] and the downsampling method proposed in [19]. Point registration is not applied in any experiment.

The numerical results are presented in Table 7.2, with one row per graph and one column per method, as follows:

- a) Our algorithm with the MDG downsampling method
- b) Our algorithm with the CSE downsampling method
- c) Our algorithm with the spectral downsampling method [47]
- d) Our algorithm with an alternative downsampling method proposed in [19]

These results demonstrate the benefit of using one of our graph downsampling method. Indeed, the other two downsampling methods are not designed for graph matching, and using them results in lower performance since vertices selected for a lower resolution are not the ones that are likely to be correctly matched. This loss

Table 7.2: Comparison of the graph matching error  $J(\mathbf{P})$  for different graph down-sampling methods

Graph	a	b	c	d
<b>Random</b>	<b>0.73</b>	2.7	6.66	5.03
<b>Bunny</b>	64.58	<b>53.22</b>	72.74	66.29
<b>Dragon</b>	88.12	<b>75.88</b>	90.63	91.19

in accuracy is significant in spite of the fact that our algorithm tries to minimize the weight of the information provided by the resolutions that have really bad performance.

Second, we evaluate the contribution of three key steps of our proposal: weighting each frequency differently (step 3), allowing to improve against graph deformations (step 6), and using the multi-resolution approach (step 5). For this purpose, in table 7.3 we report the performance of the matching in three different scenarios: a) our algorithm, b) our algorithm without steps 3 and 6 and finally c) our algorithm without step 5. Point registration is not applied in any experiment and we use *CSE* downsampling in all cases.

Table 7.3: Comparison of the graph matching error  $J(\mathbf{P})$  for different settings

Graph	a	b	c
<b>Random</b>	<b>2.7</b>	3.01	3.52
<b>Bunny</b>	<b>53.22</b>	56.79	57.45
<b>Dragon</b>	<b>75.88</b>	79.02	80.02

These results demonstrate the benefit of using the full scheme for the graph matching problem, since the best results are achieved when all the steps are performed. These three steps have a significant contribution to the final performance.

Lastly, we evaluate the contribution of the point set registration algorithm. For this, we report the performance of the matching in two different scenarios: a) our algorithm with point set registration and b) our algorithm without point set registration. *CSE* downsampling is used in the two cases. The results are presented in Table 7.4, with one row per graph and one column per method.

Table 7.4: Comparison of the graph matching error  $J(\mathbf{P})$  with and without point set registration

Graph	a	b
<b>Random</b>	<b>0.26</b>	2.7
<b>Bunny</b>	57.62	<b>53.22</b>
<b>Dragon</b>	79.51	<b>75.88</b>

These results show that point set registration is useful in some particular graphs

whereas it is detrimental for others. The reason for not improving the accuracy when performing point set registration is due to the fact that the two graphs being matched, although not being identical, could already be in the same rotation and translation. Therefore, point set registration is not needed.

## Chapter 8

# Conclusions

This project describes an unsupervised method for inexact matching of large and sparse graphs. The proposed method employs an analytic approach based on the eigen-decomposition of the Laplacian matrix of two similar graphs. The algorithm differs significantly from previously proposed methods since it finds an alignment for multiple resolutions of both graphs in order to achieve a better performance. We have shown that the algorithm is highly effective for both small and large graphs.

The first contribution of this thesis is to combine the information given by the matching of different resolution levels of the two graphs so as to decide the best alignment. The information is weighted proportionally to the performance of the alignment in each resolution. Consequently, the quality of the graph matching algorithm is improved, since each resolution level contributes according to its performance.

The second contribution of this thesis are two graph downsampling methods in order to get a lower resolution level. These downsampling techniques are designed so as to maximize the number of nodes that can be correctly matched in the lower resolution.

The third contribution of this thesis is a new interpretation of the graph matching problem, which is the basis for the proposed spectral graph matching algorithm. This new interpretation is related to the frequency information given by the two graphs being aligned. The idea is that not all the frequencies provide useful information for the graph matching problem. Therefore, removing some frequencies and weighting differently the remaining ones improves significantly the final alignment.

Experimentally, we have compared our method with some of the most popular spectral graph matching algorithms using two different types of graphs: graphs generated randomly and graphs generated through 3D point-clouds. The results show that the proposed graph matching algorithm has a better accuracy than the other methods being compared.

As future work, we believe that using the multi-resolution approach in other fields of Graph Signal Processing can be an alternative way of addressing the problem. Furthermore, for the specific case of graph matching, combining the information of 3D coordinates (graph structure) and RGB color from a video (*graph signal*) can further improve the performance of the graph matching algorithm since more information would be taken into account for the final alignment.



# Bibliography

- [1] Aamir Anis, Akshay Gadde, and Antonio Ortega. Towards a sampling theorem for signals on arbitrary graphs. In *ICASSP*, pages 3864–3868, 2014.
- [2] K Somani Arun, Thomas S Huang, and Steven D Blostein. Least-squares fitting of two 3-d point sets. *IEEE Transactions on pattern analysis and machine intelligence*, (5):698–700, 1987.
- [3] Christian Bauckhage, Sven Wachsmuth, and Gerhard Sagerer. 3d assembly recognition by matching functional subparts. In *Proc. 3rd IAPR-TC15 Workshop on Graph-Based Representations in Pattern Recognition*, 2001.
- [4] Per Bergström and Ove Edlund. Robust registration of point sets using iteratively reweighted least squares. *Computational Optimization and Applications*, 58(3):543–561, 2014.
- [5] Sofien Bouaziz, Andrea Tagliasacchi, and Mark Pauly. Sparse iterative closest point. In *Computer graphics forum*, volume 32, pages 113–123. Wiley Online Library, 2013.
- [6] Horst Bunke and Gudrun Allermann. Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters*, 1(4):245–253, 1983.
- [7] Terry Caelli and Serhiy Kosinov. An eigenspace projection clustering method for inexact graph matching. *IEEE transactions on pattern analysis and machine intelligence*, 26(4):515–519, 2004.
- [8] Sina Y Caliskan and Paulo Tabuada. Kron reduction of power networks with lossy and dynamic transmission lines. In *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*, pages 5554–5559. IEEE, 2012.
- [9] Hwann-Tzong Chen, Horng-Horng Lin, and Tyng-Luh Liu. Multi-object tracking using dynamical graph matching. In *null*, page 210. IEEE, 2001.
- [10] Siheng Chen, Rohan Varma, Aliaksei Sandryhaila, and Jelena Kovačević. Discrete signal processing on graphs: Sampling theory? *IEEE transactions on signal processing*, 63(24):6510–6523, 2015.
- [11] 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(03):265–298, 2004.
- [12] Florian Dorfler and Francesco Bullo. Kron reduction of graphs with applications to electrical networks. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 60(1):150–163, 2013.

- [13] Benoit Duc, Stefan Fischer, and Josef Bigun. Face authentication with gabor information on deformable graphs. *IEEE Transactions on Image Processing*, 8(4):504–516, 1999.
- [14] Frank Emmert-Streib, Matthias Dehmer, and Yongtang Shi. Fifty years of graph matching, network alignment and network comparison. *Information Sciences*, 346:180–197, 2016.
- [15] Rosario Feghali. An elastic graph matching approach for motion estimation on feature points in image sequences. In *Image Processing: Algorithms and Systems V*, volume 6497, page 64970Y. International Society for Optics and Photonics, 2007.
- [16] Andreas Fischer, Ching Y Suen, Volkmar Frinken, Kaspar Riesen, and Horst Bunke. A fast matching algorithm for graph-based handwriting recognition. In *International Workshop on Graph-Based Representations in Pattern Recognition*, pages 194–203. Springer, 2013.
- [17] Stefan Fischer, Kaspar Gilomen, and Horst Bunke. Identification of diatoms by grid graph matching. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 94–103. Springer, 2002.
- [18] F Fuchs and H Le Men. Building reconstruction on aerial images through multi-primitive graph matching. In *Proc. 2nd IAPR-TC15 Workshop Graph-Based Representations in Pattern Recognition*, pages 21–30, 1999.
- [19] Akshay Gadde, Aamir Anis, and Antonio Ortega. Active semi-supervised learning using sampling theory for graph signals. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 492–501. ACM, 2014.
- [20] Steven Gold and Anand Rangarajan. A graduated assignment algorithm for graph matching. *IEEE Transactions on pattern analysis and machine intelligence*, 18(4):377–388, 1996.
- [21] Marco Gori, Marco Maggini, and Lorenzo Sarti. Exact and approximate graph matching using random walks. *IEEE transactions on pattern analysis and machine intelligence*, 27(7):1100–1111, 2005.
- [22] Lee Gregory and Josef Kittler. Using graph search techniques for contextual colour retrieval. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 186–194. Springer, 2002.
- [23] Willem H Haemers. Interlacing eigenvalues and graphs. *Linear Algebra and its applications*, 226:593–616, 1995.
- [24] Adel Hlaoui and Shengrui Wang. A new algorithm for inexact graph matching. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, volume 4, pages 180–183. IEEE, 2002.

- [25] Alan J Hoffman and Helmut W Wielandt. The variation of the spectrum of a normal matrix. In *Selected Papers Of Alan J Hoffman: With Commentary*, pages 118–120. World Scientific, 2003.
- [26] Roger A Horn, Roger A Horn, and Charles R Johnson. *Matrix analysis*. Cambridge university press, 1990.
- [27] Jing Huang and Suya You. Point cloud matching based on 3d self-similarity. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on*, pages 41–48. IEEE, 2012.
- [28] Varun Jain and Hao Zhang. Robust 3d shape correspondence in the spectral domain. In *Shape Modeling and Applications, 2006. SMI 2006. IEEE International Conference on*, pages 19–19. IEEE, 2006.
- [29] Derek Justice and Alfred Hero. A linear formulation of the graph edit distance for graph recognition. *Ann Arbor*, 1001:48109, 2005.
- [30] David Knossow, Avinash Sharma, Diana Mateus, and Radu Horaud. Inexact matching of large and sparse graphs using laplacian eigenvectors. In *International workshop on graph-based representations in pattern recognition*, pages 144–153. Springer, 2009.
- [31] Wen-Jing Li and Tong Lee. Object recognition by sub-scene graph matching. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 2, pages 1459–1464. IEEE, 2000.
- [32] Bin Luo and Edwin R. Hancock. Structural graph matching using the em algorithm and singular value decomposition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(10):1120–1136, 2001.
- [33] Bin Luo, Richard C Wilson, and Edwin R Hancock. Spectral embedding of graphs. *Pattern recognition*, 36(10):2213–2230, 2003.
- [34] Diana Mateus, Radu Horaud, David Knossow, Fabio Cuzzolin, and Edmond Boyer. Articulated shape matching using laplacian eigenfunctions and unsupervised point registration. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.
- [35] N Meierhold, M Spehr, A Schilling, S Gumhold, and HG Maas. Automatic feature matching between digital images and 2d representations of a 3d laser scanner point cloud. *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci*, 38:446–451, 2010.
- [36] Bruce Merry, Patrick Marais, and James Gain. Compression of dense and regular point clouds. In *Computer Graphics Forum*, volume 25, pages 709–716. Wiley Online Library, 2006.
- [37] Paul Morrison and Ju Jia Zou. Inexact graph matching using a hierarchy of matching processes. *Computational Visual Media*, 1(4):291–307, 2015.

- [38] Sunil K Narang and Antonio Ortega. Downsampling graphs using spectral theory. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 4208–4211. IEEE, 2011.
- [39] Marcello Pelillo. Replicator equations, maximal cliques, and graph isomorphism. In *Advances in Neural Information Processing Systems*, pages 550–556, 1999.
- [40] Euripides G. M. Petrakis and A Faloutsos. Similarity searching in medical image databases. *IEEE Transactions on Knowledge and Data Engineering*, 9(3):435–447, 1997.
- [41] Aliaksei Sandryhaila and Jose MF Moura. Discrete signal processing on graphs: Frequency analysis. *IEEE Trans. Signal Processing*, 62(12):3042–3054, 2014.
- [42] Venu Satuluri, Srinivasan Parthasarathy, and Yiye Ruan. Local graph sparsification for scalable clustering. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 721–732. ACM, 2011.
- [43] Ruwen Schnabel and Reinhard Klein. Octree-based point-cloud compression. *Spbg*, 6:111–120, 2006.
- [44] Shervin Shahidi and Shahrokh Valaee. Graph matching for crowdsourced data in mobile sensor networks. In *Signal Processing Advances in Wireless Communications (SPAWC), 2014 IEEE 15th International Workshop on*, pages 414–418. IEEE, 2014.
- [45] Larry S Shapiro and J Michael Brady. Feature-based correspondence: an eigenvector approach. *Image and vision computing*, 10(5):283–288, 1992.
- [46] 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.
- [47] David I Shuman, Mohammad Javad Faraji, and Pierre Vandergheynst. A multiscale pyramid transform for graph signals. *IEEE Transactions on Signal Processing*, 64(8):2119–2134, 2016.
- [48] David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30(3):83–98, 2013.
- [49] Shinji Umeyama. An eigendecomposition approach to weighted graph matching problems. *IEEE transactions on pattern analysis and machine intelligence*, 10(5):695–703, 1988.
- [50] Mei Wang, Yoshio Iwai, and Masahiko Yachida. Expression recognition from time-sequential facial images by use of expression change model. In *Automatic*

*Face and Gesture Recognition, 1998. Proceedings. Third IEEE International Conference on*, pages 324–329. IEEE, 1998.

- [51] Chi Yuan, Xiaoqing Yu, and Ziyue Luo. 3d point cloud matching based on principal component analysis and iterative closest point algorithm. In *Audio, Language and Image Processing (ICALIP), 2016 International Conference on*, pages 404–408. IEEE, 2016.



# Glossary

<b>GSP</b>	Graph Signal Processing
<b>MDG</b>	Maximum Degree Gap
<b>CSE</b>	Corrected Second Eigenvector
<b>ICP</b>	Iterative Closest Point