School of Computing
FACULTY OF ENGINEERING AND
PHYSICAL SCIENCES

UNIVERSITY OF LEEDS

# Final Report

## A Study of Graph Cutting Algorithms Using the Graph Laplacian

**Michael Wright**

**Submitted in accordance with the requirements for the degree of**
**Computer Science with Digital Technology Solutions**

2022/23

COMP3932 Synoptic Project (EPA)

The candidate confirms that the following have been submitted.

| Items | Format | Recipient(s) and Date |
|---|---|---|
| Final Report | PDF file | Uploaded to Minerva (02/05/23) |
| Link to online code repository | URL | Sent to supervisor and assessor (02/05/23) |

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of Student) Michael Wright

## Summary

This project aims to analyze and evaluate three separate graph-cutting algorithms using the graph Laplacian. Two separate use cases have been chosen to showcase the functionality of these algorithms, a spatial data set and image segmentations. The idea of a graph cut is separate a graph into different sections, in this report only binary segmentations are considered. This is when the graph is separated into only 2 sections, hence the name. This report also explores how graphs are constructed from images and spatial data. One of the integral parts of this process is to select and implement a weight function, that describes the similarity between nodes. These functions are also evaluated alongside the segmentation algorithms. Other than the evaluation of the algorithms and weight functions, this report also covers all of the appropriate literature and knowledge needed to understand these algorithms. Topics such as the graph Laplacian and its associated eigenvalues and eigenvectors are covered in detail for readers who aren't aware of the topics. Conclusions are drawn about the effectiveness of the three algorithms as well as the different weight functions.

**Acknowledgements**

I would like to say thank you to my family and friends for being supportive throughout my time at university and for this project. Also, I would like to thank Thomas Ranner, my supervisor, who guided me in the right direction throughout this project.

# Contents

# Chapter 1

# Introduction and Background Research

## 1.1 Introduction and Motivations

Graph-cutting algorithms are powerful tools used to split graphs into two disjoint sets. Many problems in computer vision [27, 4] and network optimization [12] can be solved using these algorithms. The graph Laplacian as well as its associated eigenvalues and eigenvectors prove very useful for these problems as they inherently contain information about the structure of the graph they represent.

Computer vision can be summarised as building algorithms that attempt to imitate human vision [17]. Autonomous driving, assembly line production, and face recognition are all technologies that have spawned from computer vision, and there is no sign of a decline in their relevance in the future. One task of computer vision is the ability to classify images and segment them into sections that differ from one another, this will be a focal point in this report.

Graph cutting isn't the only approach to image segmentation, as there are also widely popular machine-learning models. These approaches provide very good solutions to these problems, however, they rely on large quantities of training data and computational power to train the models. More standard approaches, such as the ones outlined in this report, require no prior information to perform segmentations. Furthermore, implementations of graph cuts and the techniques used by graph cuts, such as the graph Laplacian, have been built upon to arrive at the newer machine learning techniques.

Thus, in this report, three separate algorithms that use the graph Laplacian are analysed to evaluate their performance on two separate tasks. These tasks are the segmentation of the Two Moons data set, as well as the binary segmentation of images, both of which are unsupervised tasks and require no additional training data. The algorithms in question are the Fielder Method, the Perona Freeman Method, and the convex splitting scheme for the Ginzberg Landau Functional.

## 1.2 Background Research

This section will review all of the necessary information needed to understand, implement and evaluate the algorithms in this paper. Some of the history, motivations, and critical analysis of other solutions are also covered in this section.

### 1.2.1 History of Graph Cuts

Ideas of cutting graphs can be traced back to the 1950s when the Americans were studying the Soviet rail network across Europe and devising ways of stopping the movement of resources[25]. This led them to formally define the minimum cut and maximum flow problems. The Americans managed to find an algorithmic method to approximate solutions to these problems and then later optimal solutions.

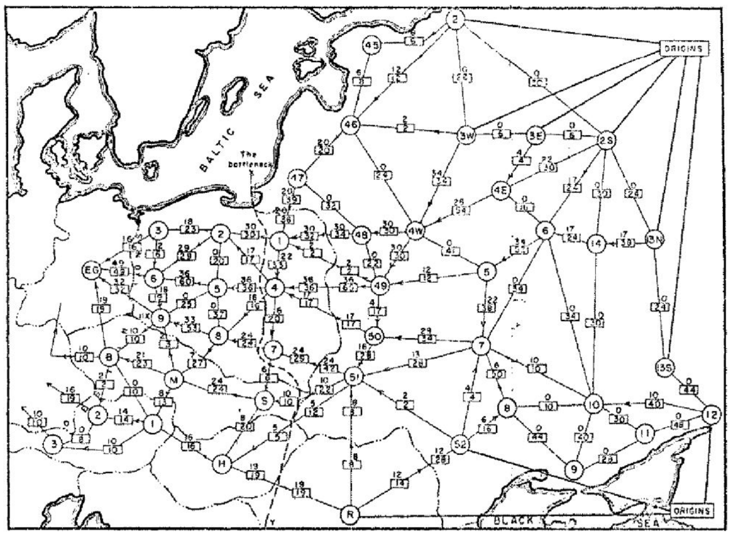


Figure 1.1: Map of the Soviet Union rail network as a graph from the declassified RAND report published by America [25]

Whilst these problems originated during the cold war for military and geopolitical purposes, see Figure 1.1, they have more recently been utilized in the field of computer vision and image segmentation. Graph cutting techniques began to be used in the field of computer vision and image segmentation [5, 27] around the 1990s. These papers showed that graph cuts can segment images as foreground and background components with a high degree of success. Since this breakthrough in the field of computer vision, these techniques have become standard practice and have been iterated upon and improved ever since. Not only have graph cuts been used for image segmentation problems, but they have also been utilized for other computer vision problems such as image restoration, and image smoothing.

### 1.2.2 Existing Solutions

The idea behind a graph cut is to achieve two basic goals, 1) group nodes that are similar together, 2) separate nodes that are different from each other. The outcome of these goals in binary segmentation is to label these nodes either 1 or 0 depending on what class they belong to. Any algorithm that achieves this can be considered to be a graph-cutting algorithm. There have been many different techniques devised that seek to achieve this aim of a graph cut. There are more standard solutions that lie in graph theory [27, 22], solutions that minimize energy functions across a graph [4, 13] and also the newer machine learning and neural network approaches [24]. Here these algorithms are covered and explained for the reader's benefit.

### Machine Learning and Neural Networks

It must be noted that graph cuts are significantly less effective at these tasks than neural networks have become and may become, especially for image classification. One of the reasons for this is the ability neural networks have to train a model on very large data sets, which allows them to then determine and classify images with a high degree of accuracy. Convolutional Neural Networks are fine examples of models that perform very well at this task and are considered to be industry-leading within the field of neural networks for image classification. Due to their large training time and inability to work with being trained, alternative methods aren't obsolete and are still worth consideration.

### Energy Minimization

Energy minimization is one of the more common and successful methods to cut a graph algorithmically. Very specific and more complex systems for the graph have to be made, but the idea is to treat the graph as a physical system that contains energy. Then the goal is to minimize the energy in that system, and if the graph is constructed correctly this can be done by using graph-cutting algorithms. These methods have been used many many times throughout the history of computer vision and they have achieved much success with the methods [19]. This paper from Boykov, Veksler and Zabih [6] has been cited over 10,000 times to date which speaks of the impact that it has had on this field of research. One of the methods analysed in this paper involves energy minimization of the Ginzburg Landau Functional [4].

### Spectral Graph Theory

Firstly the Fielder method represents a simple algorithm that can be used to demonstrate the effectiveness of the graph Laplacian along with its corresponding eigenvalues and eigenvectors. It's one of the fundamental algorithms within spectral graph theory and provides a nice stepping stone into the rest of the field. It is a fast algorithm with very low complexity and performs relatively well in terms of accuracy.

One of the seminal papers in the field of spectral graph theory, graph cuts and their use with image segmentation was Shi and Malek's paper that introduced their normalized cuts algorithm [27]. They showed results for image segmentation that were amazing for the time and the paper paved the way for many other researchers to build upon their work. This paper has received much recognition other the years and many people have reviewed its performance, therefore it will not be included in this project.

Two of those researchers who followed Shi and Malek were Perona and Freeman, who proposed their own method to perform a graph cut [22]. It is a method that takes advantage of multiple eigenvectors to produce its solution which is where it differs from that of the Fielder method. This is one of the algorithms that will be analysed in this paper, and with over 400 citations this paper clearly had an impact in its respective area.

Overall these algorithms mentioned here represent a class of graph-cutting algorithms that use spectral graph theory to segment images and graphs. Obviously, there are more advanced and complex algorithms that are used in this field that provide very strong and fast results for these problems. However many of these algorithms require a high amount of prior knowledge in spectral graph theory and linear algebra and therefore were not chosen to be a part of this report.

### 1.2.3 Evaluation Metrics

Evaluating the accuracy or the competency of an algorithm's performance is very important and there are many different ways in which this can be done. As shown in [30] a wide variety of metrics can be used to determine segmentation performance and it also points out that certain types of segmentation are more suited to certain metrics.

#### Speed

One of the main and most important metrics used in the evaluation of these algorithms and any algorithms for that matter is the speed of performance. This metric is very uncomplicated and implementation of this within a project is trivial in any coding language. This makes a comparison between existing solutions and the algorithm in question very easy, the unit of measurement (time) stays the same across all papers and therefore allows for quick evaluation of performance in relative terms. Of course, the hardware on which the algorithm is run plays an important part in the speed of execution, as well as the language in which the code is written. However, these speed humps can be ironed out and priced into the evaluation of the algorithm.

#### Robustness

Another important metric and testing point for algorithms is their robustness or reliability. The errors produced by an algorithm can come in many shapes and sizes, but in the context of this project, they will most likely arise from numerical divergence, timeouts, or high variance in speed or accuracy. One of the algorithms that are being tested is a convex splitting scheme that relies on the convergence of the variables around 0, if this doesn't happen these variables can diverge and spiral out of control causing the algorithm to fail. Timeouts may occur from very large graphs that need to be processed since the algorithm that computes the eigenvectors has a complexity of $O(n^3)$. A high variance in score or accuracy can come from issues with both of the previous faults but also if a specific algorithm is highly sensitive to initial conditions.

All of these errors that can arise ultimately affect the robustness of the algorithm. Often this metric is boiled down to a simple percentage that can represent the failure rate of a certain algorithm.

#### Accuracy

Arguably the most important metric for graph cuts and image segmentation lies in the accuracy of a specific segmentation. Correctly identifying the correct measure for accuracy is very important for a good and robust evaluation of the algorithm against other existing solutions as well as more generally. As previously discussed there are many ways of determining accuracy [30, 1], the pros and cons of some of the approaches are discussed here.

Firstly there is the obvious and simple metric of measuring the number of data points that are correctly classified, in this case, it would be nodes of the graph that would correlate to pixels in an image. Then with this number divide it by the total number of nodes in the graph resulting in a percentage accuracy, see Equation 1.1. This is a practical measure of accuracy and will prove useful as a lot of sources use this measure, which makes it easier to evaluate the accuracy of the algorithms in question.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1.1}$$

However, there is a drawback to this method which can be easily illustrated with a simple diagram, Figure 1.2. As you can see, high accuracy scores can be achieved using this method - whilst, in reality, it is obvious that it is a poor segmentation. This makes the algorithm less effective for a more detailed and comprehensive analysis of algorithmic accuracy, but it can still be used as a comparative measure when looking at results from other papers and research in the specified area. Situations in which the desired segmentation is much less than half of the entire graph are where this total accuracy begins to become weak.

An Alternative method for measuring the accuracy of segmentation results is to use the Jaccard index, first used by Paul Jaccard in 1901. This index measures the similarity between two sets, this can be used to evaluate the accuracy by using the expected segmentation and the actual segmentation as the two sets.

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{1.2}$$

Where $A$ and $B$ are two sets, $|A|$ and $|B|$ denote the number of elements in each set. This measure of similarity between desired segmentation and true segmentation can be used as a more robust measure of accuracy. This index is also used in an extensive amount of image segmentation research [3]. This will allow for a wide range of previous results to compare and evaluate.

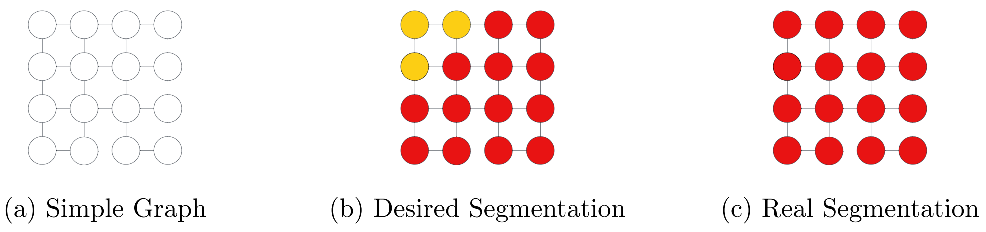

(a) Simple Graph (b) Desired Segmentation (c) Real Segmentation

Figure 1.2: This figure shows a graph, with its desired segmentation and an example of a poor real segmentation, subfigure c) has a Jaccard index of 0 and an accuracy of 0.8125

Figure 1.2 shows a poor example of segmentation, this is obvious to see with the human eye. But it is also a good representation of the differences between the Jaccard index and the standard accuracy. The high pixel accuracy score is a misrepresentation of the true accuracy of the segmentation, whereas the Jaccard index correctly identifies that the segmentation is poor with a score of 0.

### 1.2.4 Basic Definitions

This mini section will review and outline some of the basic definitions that are useful to know proceeding with this report. Let's start by defining what a graph is.

$$G = (V, E) \tag{1.3}$$

Where $V$ is the set of nodes and $E$ is the set of edges in a weighted graph $G$. The weighted graph $G$ has an associated weight function $w : V \times V \rightarrow R$ [8].

$$w(x, y) = w(y, x) \tag{1.4}$$

$$w(x, y) \geq 0 \tag{1.5}$$

Where $\{x, y\} \in E$ and if $\{x, y\} \notin E$ then $w(x, y) = 0$ and $x, y$ are vertexes of the set $V$.

Functions across a graph are an important part of these algorithms and how they work. Consider $f(x)$ where x is a node in a graph, the function will map values to each node in that graph. The following diagram is a good representation of a function across a graph.

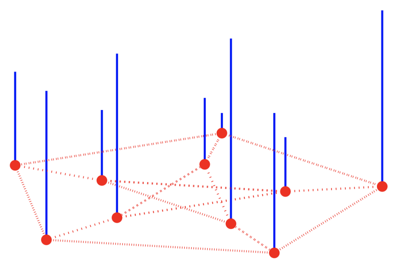


Figure 1.3: A graph with a graph function shown as the blue lines [28]

### 1.2.5 Graph Laplacian

The algorithms and methods in this paper revolve around the graph Laplacian matrix and its associated eigenvalue and vector pairs. The formulation for the graph Laplacian goes as follows:

$$L = D - A \tag{1.6}$$

Where D is the degree matrix and A is the adjacency matrix, both corresponding to the graph. The degree matrix is a diagonal square matrix where each element on the diagonal corresponds to a node, the value of that element in the matrix is equal to the sum of the weights of the edges coming in and out of that node. All other elements in the matrix are 0. Please note, D is often interchangeably used with W, which stands for the weight matrix, in many other papers surrounding these topics. The adjacency matrix is another square matrix that the weights of the edges between nodes. Terminology is often a little fuzzy here and can be used interchangeably with weight matrix or weighted adjacent matrix.

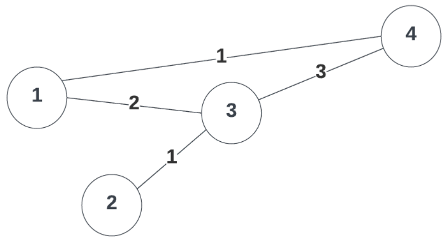


Figure 1.4: Topology of an undirected weighted graph

$$\begin{pmatrix} 3 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & 4 \end{pmatrix}$$

(a) degree Matrix $D$

$$\begin{pmatrix} 0 & 0 & 2 & 1 \\ 0 & 0 & 1 & 0 \\ 2 & 1 & 0 & 3 \\ 1 & 0 & 3 & 0 \end{pmatrix}$$

(b) Affinity Matrix $A$

$$\begin{pmatrix} 3 & 0 & -2 & -1 \\ 0 & 1 & -1 & 0 \\ -2 & -1 & 6 & -3 \\ -1 & 0 & -3 & 4 \end{pmatrix}$$

(c) Laplacian Matrix $L$

$$\begin{pmatrix} 1 & 0 & -\frac{2}{\sqrt{18}} & -\frac{1}{\sqrt{12}} \\ 0 & 1 & -\frac{1}{\sqrt{6}} & 0 \\ -\frac{2}{\sqrt{18}} & -\frac{1}{\sqrt{6}} & 1 & -\frac{3}{\sqrt{24}} \\ -\frac{1}{\sqrt{12}} & 0 & -\frac{3}{\sqrt{24}} & 1 \end{pmatrix}$$

(d) Normalized Laplacian Matrix $L_{norm}$

Figure 1.5: Above are the three matrices related to the graph depicted in 1.4

For all of the experiments and for all of the algorithms in this report, the graph Laplacian that will be used is the normalized graph Laplacian. Normalizing the Laplacian doesn't follow the same construction as the normalization of a matrix. Instead, the idea is to make the diagonal elements of the matrix equal to 1 and adjust the corresponding rows and columns accordingly. It has been repeatedly shown that the normalization of the base matrix will lead to eigenvectors that retain more of the structural properties relating to the desired graph cut [33, 4, 27].

$$L_{norm} = D^{-1/2} L D^{-1/2} \tag{1.7}$$

$$L^{i,j}_{norm} = \begin{cases} 1, & \text{if } i = j \\ -\frac{w(v_i,v_j)}{\sqrt{deg(v_i)deg(v_j)}}, & \text{if } i \neq j \text{ and } v_i \text{ and } v_j \text{ are adjacent} \\ 0, & \text{otherwise} \end{cases} \tag{1.8}$$

Where $L$ is the graph Laplacian, $D$ is the degree matrix, and $deg(v)$ signaling the degree of a certain vertex [7]. To compute the graph Laplacian matrix in this project, the NetworkX [15] library has been utilized. Within this library, there are 2 functions, one which returns the standard graph Laplacian and the other which returns the normalized graph Laplacian matrix.

### 1.2.6 Eigenvalues and Eigenvectors

Eigenvalue and eigenvector pairs are integral parts of the algorithms used in this paper and the speed at which they can be generated is important to the overall speed of these algorithms. But first, we need to define what eigenvalues and vectors are. If we define an NxN matrix as A, an eigenvalue as $\lambda$, and an eigenvector as $\phi$. The eigenvalue eigenvector pairs must satisfy the following equation:

$$A\phi = \lambda\phi \tag{1.9}$$

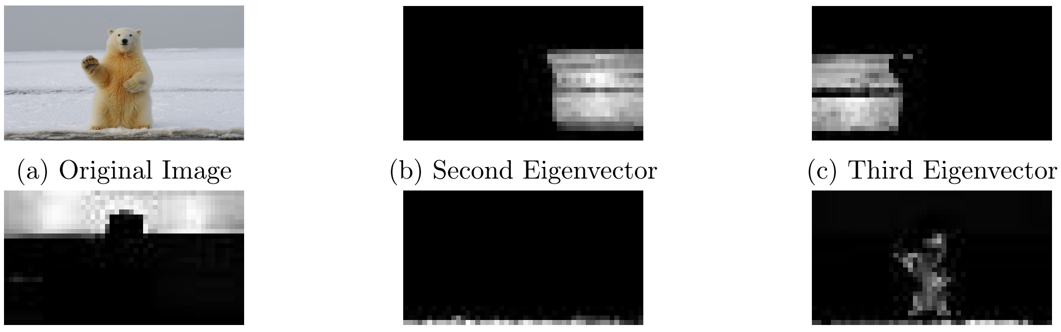
In this project, we care only about the eigenvalues and eigenvector pairs of the graph Laplacian matrix. This effectively means we are looking for the solutions to the previous equation but with the matrix being swapped for $L$:

$$L\phi = \lambda\phi \tag{1.10}$$

A graph Laplacian will have $N$ eigenvalue-vector pairs where $N$ is the number of nodes in the graph it corresponds to. All of the eigenvalues are real and non-negative and for the following algorithms they must be ordered in the following fashion:

$$0 = \lambda_1 < \lambda_2 < \lambda_3 ... \lambda_N \tag{1.11}$$

This gives a set of ordered eigenvalue-vector pairs $(\lambda_k, \phi_k)$. Each of the eigenvectors holds information about the graph's structural properties. For example, $\phi_2$ often referred to as the Fiedler vector [11] contains information that can approximate a relaxed cut for a graph. This idea can be seen in figure 1.6 which demonstrates how different eigenvectors relate to different structural information of an image. Other eigenvectors contain similar but slightly differing information, this fact is very important to the algorithms used and explained in this project.



(a) Original Image (b) Second Eigenvector (c) Third Eigenvector

(d) Fourth Eigenvector (e) Fifth Eigenvector (f) Sixth Eigenvector

Figure 1.6: The above images show the different areas of an image that eigenvectors capture

Another important point to touch on about the eigenvectors is that they can be effectively used as functions. All of the eigenvectors will be 1xN column vectors where N is the number of nodes in the graph it relates to, this allows each value within the vector to be 'mapped' to a node in the graph. Meaning $\phi_k(x)$ is a function that returns values from the eigenvector, where $x$ represents the nodes in the graph, this will be used later on in the paper.

For the computation and calculation of the eigenvalues and eigenvectors, the NumPy [16] module is used. This library has been used due to its extensive use throughout the community which means it is well-tested and the documentation as well as support is excellent in comparison to other modules that produce the same results. The complexity of this algorithm is $O(n^3)$, making it a slow algorithm that will most likely be the bottleneck component of the final solution.

# Chapter 2

# Methods

## 2.1 Similarity Functions

Weight functions or similarity functions are functions that are used to determine the 'similarity' between 2 mathematical objects. In this case, we are referring to nodes in a graph and the similarity function is used to construct the edges of that graph and therefore will have a great impact on the creation of the graph Laplacian and then the corresponding eigenvalue-vector pairs. There are many different types of similarity functions that can be used, for the graphs constructed in this project both the Euclidian norm and an angle norm have been used. The Euclidian norm, often referred to as the L2 norm, refers to finding the distance between two nodes by finding the root of the square of the components of the vectors corresponding to the nodes in the graph. Oppositely the angle norm represents the angle between two vectors and assigns a similarity accordingly. Both of these norms have their advantages and disadvantages and can be suited to different tasks, this is explored in the results section. Here, the three similarity functions will be explained in detail.

### Gaussian Function

One of the more simple similarity functions is the Gaussian kernel function [20]:

$$w(x, y) = \exp(-\frac{d(x, y)^2}{2\sigma^2}) \tag{2.1}$$

$d(x, y)$ simply refers to the Euclidean distance between two vector points $x$ and $y$ Given these two vectors are 2 dimensional you can compute this distance using the following formula:

$$d(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2} \tag{2.2}$$

This kernel function is a very popular and well-performing function for a few reasons. Mainly it is very fast to compute due to its low complexity, despite this, it performs very well across our large variety of different problems. As will most likely become apparent in the results section.

Figure 2.1 depicts the effect that the $\sigma$ value has on the output of the Gaussian function. It highlights the importance of choosing the correct value for the scale that you are working with. For example, if the scale of the problem is small then it may be beneficial to use a smaller value of $\sigma$ and visa versa.

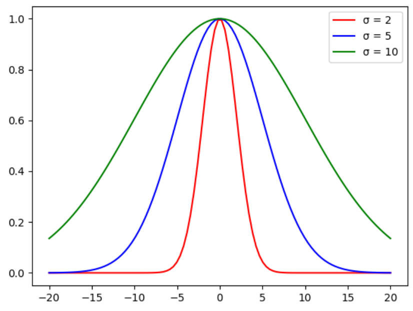


Figure 2.1: Gaussian Function plotted with different values of $\sigma$.

### Local Scaling Weight Function

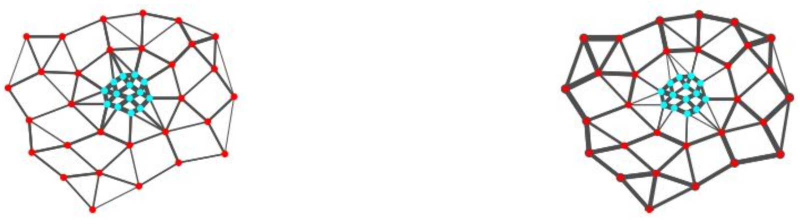
Another similarity method that has been used in the paper is the self-tuning similarity function which is laid out in [36]. For each weight that you assign to the nodes in the graph, you factor in the Nth nearest node into the function:

$$w(x, y) = \exp(-\frac{d(x, y)^2}{\sigma_x \sigma_y}) \tag{2.3}$$

This function is very similar to the Gaussian function but differs when looking at how we calculate $\sigma_x$ and $\sigma_y$:

$$\sigma_x = d(x, N) \tag{2.4}$$

Where N is the Nth nearest node to x. The value of N can vary depending on the needs of the problem at hand, but the motivation behind this local scaling is to evaluate how well-grouped or tightly packed different sets of data are.



(a) Gaussian Similarity Function (b) Locally Scaled Similarity Function

Figure 2.2: Example of the effect of local Scaling, the similarity between nodes encoded in the thickness of the line [36].

### Cosine Similarity Function

This similarity function is the angle norm that shall be investigated in this report. Values returned from this function will be in the range [-1, 1]. Where a value of 1 represents two vectors that are very similar to one another and a value of -1 represents two very dissimilar vectors. One of the main benefits of using this as a similarity function is the fact that the size and scale of the vectors in question are irrelevant to the measure which can provide better results for particular tasks. Usually, this measure is used in the field of information retrieval and for text analysis [29].

$$w(X,Y) = \frac{X \cdot Y}{|X| \cdot |Y|} = \frac{\sum_{i=1}^{n} X_i Y_i}{\sqrt{\sum_{i=1}^{n} X_i}\sqrt{\sum_{i=1}^{n} Y_i}} \tag{2.5}$$

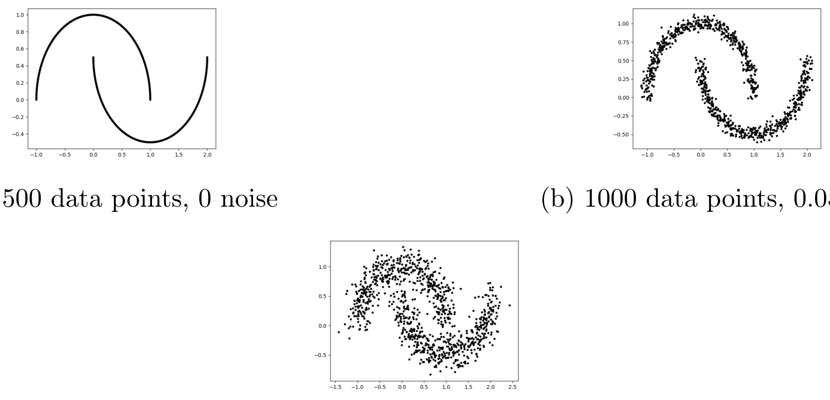
Where $X$ and $Y$ are vectors associated with nodes in the graph, $i$ represents the components of the vectors and $n$ represents the dimensionality of the vectors. Since the function returns values in the range [-1, 1], the output of the cosine function will become the exponent of $e$ so that negative weights can be avoided from being computed.

## 2.2 Creating Graphs for Different Situations

NetworkX [15] is an open-source Python package that is used to create and manipulate graphs. This package has extensive documentation, making it easy to learn and debug. The igraph [10] python package was also considered for development but despite its inherent speed advantage over NetworkX, it doesn't integrate as well with other packages being used such as Numpy. This and its ease of use are why NetworkX was chosen for this project.

### 2.2.1 Two Moons Data Set

The Two Moons data set is commonly used for evaluating algorithms that intend to perform binary classifications. It consists of 2 crescent shapes which overlap with one another in 2-dimensional space, the 2 clusters are easily identifiable to the human eye, which makes it an intuitive data set to use for the segmentation algorithms. The data set produces a number of data points which are represented by x and y coordinate values, the data points can also be generated with a degree of random noise which can increase the level of difficulty of the segmentation problem.



(a) 500 data points, 0 noise

(b) 1000 data points, 0.05 noise

(c) 1000 data points, 0.15 noise

Figure 2.3: Three examples of the two moons data set, with varying amounts of noise and data points

Turning the data set of coordinate positions into a graph that can be segmented is a relatively simple task. Each node of the graph had a correlated feature vector which is the position that it represents (coordinates), from this a similarity function can be applied to compute a value for the edge between 2 nodes. A fully connected graph is made which just means every node on the graph has a unique connection to every other node. Any similarity function can be chosen and the effectiveness of this choice is outlined in the results section.

To generate this data the sci-kit library [26] was chosen. This function has been well tested, it's optimized for efficiency and allows for reproducible data sets with the aid of a seed which proves very useful for testing.

### 2.2.2 Images

Constructing appropriate graphs for images requires a slightly different process from that of the Two Moons data set. Colour, intensity and position data are all embedded within images, whereas only position data is encoded in the Two Moons' data set. This means that the similarity function that is used to construct these graphs has to be modified to accommodate this change in feature vectors.

Let's start by thinking of each pixel in an image as a node in a graph, each of these pixels has associated data that accompanies it. This data is used to construct the edge set $E$ using modified similarity functions.

$$w(x,y) = exp(-\frac{|f(x) - f(y)|^2}{2\sigma^2}) \times N(x,y) \tag{2.6}$$

$$N(x,y) = \begin{cases} d(x,y), & \text{if } d(x,y) \leq r \\ 0, & \text{otherwise} \end{cases} \tag{2.7}$$

$f(x)$ and $f(y)$ denote functions that can return either the intensity of the pixel or the colour of the pixel. The difference between the values for the $x$ and $y$ pixels can be calculated using the Euclidean distance. Assuming we have RGB values for image pixels, calculating colour is trivial but below shows how to calculate intensity:

$$I(x_{RGB}) = \frac{x_R + x_G + x_B}{3} \tag{2.8}$$

This technique has been used in both [22, 27], the idea is to group sections of the image into neighbourhoods of similarity. You could imagine without $N(x,y)$ parts of the image that are the same colour or intensity but are at opposite ends of the graph would have a high similarity score, which wouldn't be desirable. However with the implementation of $N(x,y)$ these types of pixels would be correlated highly at all unless they were part of the same neighbourhood.

All of the images used in this project as processing using Pillow, a Python image processing library - PIL [9]. It's very thorough documentation and the healthy community that keeps it up to date make it an easy-to-learn library to easily get up to speed with image manipulation.

Moreover, other libraries that are used in this project such as numpy, integrate well with this package.

## 2.3 The Algorithms

All three algorithms that are being analysed in this report will be covered in this section. The steps for each algorithm will be covered as well as any prior knowledge that may need to be known to execute and understand the algorithms.

### 2.3.1 The Fiedler Method

This is a simplistic but relatively effective algorithm for partitioning a graph into two separate segments. The algorithm used in this report follows a method outlined in [35].

As touched upon in 1.2.6 the second eigenvector $\phi_2$ of the graph Laplacian is also known as the Fiedler vector [11]. This means that it can be used to approximate a relaxed cut of a graph.

Once we have computed the second eigenvector we can use it as a eigenfunction which maps each node on the graph to a value within the vector. Given this knowledge, we can then just apply k-means clustering with k, the number of clusters, set to two. This will then segment the nodes of the graph into 2 separate clusters achieving a binary segmentation.

Here is a simplified list of the steps needed to complete this algorithm:

```
Algorithm 1 Fielder Method
Input: ($\phi_k$) the eigenvectors of $L$.
Output: Binary labels for each node of the graph.
  1. Take $\phi_2$ and map the nodes of the graph to the values of the vector.
  2. Using k-means clustering with k=2 split the values of the eigenvector into two distinct sets
```

Despite this algorithm not yielding great accuracy scores in a variety of segmentation problems, it is a good benchmark for other algorithms to compare speed and accuracy.

### 2.3.2 Perona and Freeman Segmentation

Perona and Freeman [22] saw the work of Shi and Malik [27] a year before and consequently came up with a simpler technique to segment graphs.

Here are the basic steps of the algorithm laid out [34]:

It must be noted that this is a slight variation on the original algorithm. Perona and Freeman used the affinity matrix as the base matrix to find the eigenvectors from, however in this report, the Laplacian has been used. This makes it easier to compare and evaluate this approach to the others highlighted in this report. It also yielded better results when testing. Please note that this is the standard Laplacian and not the normalized Laplacian, as this

```
Algorithm 2 Perona Freeman Method
Input: ($\phi_k$) the eigenvectors of $L$.
Output: Binary labels for each node of the graph.
  1. Create a matrix $M = [\phi_1, \phi_2 ... \phi_k]$
  2. Treat each row of $M$ as a vertex in $V$ and cluster them using k-means clustering with k=2
```

algorithm produces poor results when using the normalized Laplacian.

### 2.3.3 Convex Splitting Scheme for the GL Functional

This algorithm proposed by Bertozzi and Flenner [4] brings together work from imaging methods with Partial Differential Equations (PDEs) and the methods from the spectral graph theory community. Their algorithm, in simplistic terms, revolves around minimizing the Ginzburg-Landau (GL) functional using the eigenvalues and eigenvectors of the normalized graph Laplacian.

In the paper, they present a numerical scheme that can be used to minimize the GL functional, the idea is to run this scheme for a set number of iterations then you will be left with a function $u(x)$ where $x$ represents a node in the graph. This function can then be split into 2 clusters which will be the graph cut.

```
Algorithm 3 GL Method
Input: ($\lambda_k, \phi_k$) the eigenvectors and eigenvalues of $L_{norm}$.
Output: Binary labels for each node of the graph.
  1. Initialize $u_0(x)$ with a function that relates to the problem being solved
  2. Create an $k \times N$ matrix $V$ where every row corresponds to an eigenvector $\phi_k$
  3. Initialise $a_k^0, b_k^0, d_k^0, D_k$ according to the following:
      (a) $a_k^{(0)} = u_0(x) \cdot V^T$
      (b) $b_k^{(0)} = [u_0(x)]^3 \cdot V^T$
      (c) $d_k^{(0)} = 0$
      (d) $D_k = 1 + dt(\epsilon\lambda_k + c)$
  4. For a set number of iterations $M$ update $a_k^{(n+1)}, b_k^{(n+1)}, d_k^{(n+1)}, u^{(n+1)}(x)$ according to the following
      (a) $a_k^{n+1} = D_k^{-1}[(1 + \frac{dt}{\epsilon} + cdt)a_k^n - \frac{dt}{\epsilon}b_k^n - dtd_k^n]$
      (b) $u^{n+1}(x) = a_k^{n+1} \cdot V$
      (c) $u^{(n)}(x) = u^n(x) - \overline{u^n(x)}$
      (d) $b_k^{n+1} = [u^{n+1}(x)]^3 \cdot V^T$
      (e) $d_k^{n+1} = (u^{n+1}(x) - u_0(x)) \cdot V^T$
  5. Segment $u^M(x)$ into two groups using 0 as the splitting point
```

This algorithm uses $k$ number of eigenvalues-vectors from the graph Laplacian, however, due to the size of many graphs this can become too computationally expensive. This is due to the fact that the number of eigenvalues and eigenvectors of the graph Laplacian is equal to the number of nodes in its related graph, as mentioned in section 1.2.6. Instead, just the first $n$ eigenvalues can be included in the algorithm, this number is arbitrary and its values will have an effect on the efficiency and complete accuracy of the solution.

This algorithm is the unsupervised version of the algorithm shown in the paper, this means that the fidelity function which contains known information about nodes on the graph isn't included. The supervised version of this algorithm allows labelled data from similar images and graphs to be used to segment future graphs. As this algorithm isn't using this method, instead a mean constraint is applied to $u(x)$, this mean constraint will centre the values of this function around 0 and therefore maintain a binary class representation for the segmentation. To apply this constraint $u(x)$ must be updated each iteration by subtracting the mean value of the function from all values of the function. This step is shown in item c, step 4 of algorithm 3.

## 2.4 Methodology of Development

Throughout the course of this project, an agile methodology was followed for the development of the code and the written report. This meant that the code base was improved through iterative design over the course of the project and as the requirements and goals of the project changed. The following subsections will cover some of the technologies used as well as the functionality of the code.

### 2.4.1 Python

This project is entirely developed in Python 3.8.10 [23]. One of the main reasons for this is the strong infrastructure and packages related to graphs and graph theory. It makes it simpler to learn and use these tools than other languages. C++ was considered due to its superior speed advantages but was ultimately let down because the documentation for many of these libraries was not as good as it was for Python. A Python virtual environment was used throughout the project development.

### 2.4.2 Using GIT

Throughout this project, a remote private GitLab repository has been used for version control of the code base. Commits were made to this repository whenever a goal for development had been hit or the functionality of the code had been changed. The repository contains a README.md file that documents how to use the software and should allow any future collaborators to pick up the software and continue further development.

### 2.4.3 Code

All of the code can be found in the git repository provided at the top of this project. There is no GUI or general interface with the code, however, there is a main.py file that can be run after installing the appropriate requirements from the requirements.txt file. All of the requirements are Python libraries so they can all be installed via pip. This main file will perform an image segmentation using all three algorithms and save the results to the appropriate folder.

The core design consists of two separate classes, one to deal with the creation of the graphs and the other to segment and return a set of binary labels for each node on the graph. Within the final code base, there are separate helper functions. They display the segmentations back to the user as a Two Moons plot or image segmentation depending on the use case.

# Chapter 3

# Results

This chapter will present the findings from experiments performed using the segmentation algorithms and similarity function to split the two moons' data sets as well as images. All of the performance tests were completed on a Windows 10 home PC with an Intel Core i3 and 8GB Ram for complete consistency between results. All graphs are plotted using the matplotlib [18], and recorded times for the speed of the algorithms were conducted using the time module for Python.

## 3.1 Two Moons Data Set

For this Section, all three of the algorithms are evaluated separately for the different similarity functions. First, the similarity functions will be compared by using the fielder method as a segmentation tool. Then all three of the algorithms will be compared using just the local scaling similarity function to reduce biases and accurately evaluate the algorithm. Both the accuracy and the speed of the algorithm will be analysed, and the robustness of the algorithms will be mentioned if necessary. The Jaccard index will not be used in this section as it is not appropriate for this data set. The class representation of the data points is 50/50, making this accuracy score no different from the absolute accuracy.

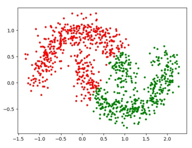
The parameters that are used for the creation of the Two Moons data set are as follows. 1000 data points are generated and the noise applied to these data points will be 0.15, 0.175 and 0.2, these represent an easy, medium and hard problem. From testing it became apparent that noise values below 0.15 were too easy for the algorithms to solve and therefore didn't give an appropriate representation of the algorithm's effectiveness. Also, noise values above 0.2 make the data set too difficult to solve and turn the 2 clusters into one bigger cluster.

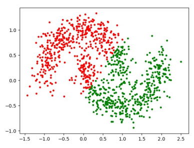
### 3.1.1 Algorithms

To evaluate these algorithms against each other whilst they segment this data set they will all use the same similarity function with the same parameters. This will be the local scaling function with $N = 10$, this value will provide a difficult problem for any of the algorithms to solve well. The three noise levels will be evaluated for each algorithm to show how effective the algorithms are at different noise levels.

#### Fielder Method

Figure 3.1 shows the Fielder method being applied at varying levels of noise, it is apparent that this method isn't the best at classifying the two moons, however, it is consistent and very fast.


(a) N = 0.15, A = 0.807


(b) N = 0.175, A = 0.796

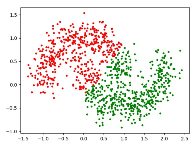

(c) N = 0.2, A = 0.787

Figure 3.1: The Fielder Method at differing noise levels for the data set, along with the average accuracy produced after 20 runs for each level. N represents the noise and A represents the accuracy

The split it picks out can be seen as the diagonal split that continues throughout the different noise levels. This means that the accuracy between the different data sets isn't vastly different as the same proportion of data points lies on either side. There is a slight decrease in accuracy as the noise increase shows that the problem gets harder. These results are as expected and are seen in other papers such as [4] where they too use the local scaling similarity function. This behaviour is also reflected across other similarity functions. It seems to suggest that this pattern is just the structural data that is picked out by the

## Perona Freeman Method

Table 3.1 shows that as the number of eigenvectors used to cluster the points goes up, the accuracy goes down. This trend is consistent across all three values for the noise and it is a noticeable difference from k=2 to k=10. The average fall in accuracy across the three levels of noise was 11% from k=2 to k=10. Notice that a value of 2 is the strongest for this algorithm, and with this value, it is possible to get some really good results. Unfortunately, good results are just as frequent as poor results, this algorithm is therefore very unstable and produces a wide range of accuracy values, hence giving a high variance score.

| **Perona Freeman Method** | | | | | |
|---|---|---|---|---|---|
| **NOISE** | **k = 2** | **k = 4** | **k = 6** | **k = 8** | **k = 10** |
| **0.15** | 0.806 | 0.780 | 0.747 | 0.732 | 0.719 |
| **0.175** | 0.793 | 0.771 | 0.744 | 0.729 | 0.721 |
| **0.2** | 0.788 | 0.759 | 0.743 | 0.729 | 0.711 |
| **Average** | **0.795** | **0.77** | **0.744** | **0.73** | **0.717** |

Table 3.1: Perona Freeman Algorithm with a varying number of eigenvectors used (k)

Figure 3.2 identifies that the variance score increases as more eigenvectors are used. This suggests that the algorithm becomes more unstable in terms of the accuracy scores that it produces. This means that one run may yield a high accuracy score and then the next run could return a very poor accuracy score.

Both the accuracy scores and the variance scores show that the optimal parameter setup for this algorithm is k = 2. This, therefore, makes the algorithm very similar both in performance and how the fielder algorithm works. Using this algorithm, however, can produce very strong accuracy scores well into the 90s which can't be said for the fielder method which has very little variance in the accuracy scores that it produces, making it a more stable algorithm. Figure 3.3

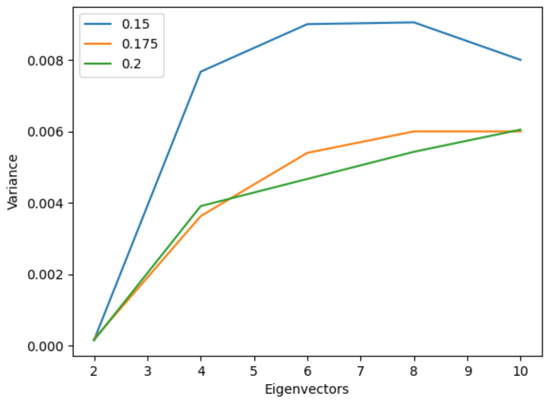


Figure 3.2: Variance for different numbers of eigenvectors per noise level

shows a visual representation of the variance in the accuracy results, it displays two segmentations with the same parameter setup and with vastly different outcomes.

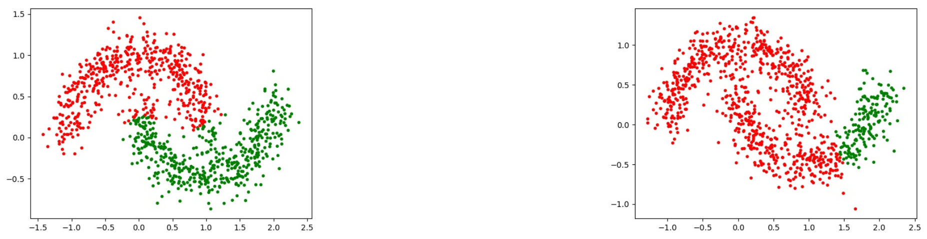


Figure 3.3: Two examples of the Perona Freeman Method with 4 eigenvectors selected, the left shows a successful split and the right shows a poor split

## GL Convex Splitting Method

For the following tests, the algorithm was set up with the following parameters, $u_0 = sgn(\phi_2(x) - mean(\phi_2))$ as described in [4], 250 iterations, time increment of 0.1 and $c$ set to 1. The following results will present the effect of the $\epsilon$ on the algorithm, both its accuracy and stability effects are shown.

| **GL Method** | | | | | |
|---|---|---|---|---|---|
| **NOISE** | $\epsilon = 2$ | $\epsilon = 4$ | $\epsilon = 6$ | $\epsilon = 8$ | $\epsilon = 10$ |
| **0.15** | 0.814 | 0.821 | 0.811 | 0.818 | 0.819 |
| **0.175** | 0.804 | 0.811 | 0.816 | 0.818 | 0.818 |
| **0.2** | 0.805 | 0.805 | 0.810 | 0.811 | 0.810 |
| **Average** | **0.808** | **0.812** | **0.812** | **0.816** | **0.816** |

Table 3.2: Accuracy values of the GL convex splitting algorithm with different parameters

Table 3.2 shows that the GL functional is a good-performing algorithm in terms of accuracy. All of the accuracy values are higher than the previous algorithms for all of the different noise levels as well as the algorithm's internal parameters. There does appear to be a trend that points to a higher value of $\epsilon$ and a lower value of $c$ being the more optimal values.

Furthermore, as Figure 3.4 identifies the average number of iterations required to reach the convergence tolerance increases with the value of the interface scale (with a fixed value of $c$). It appears to be that the smaller values of $\epsilon$ try to yield a sharper difference between the two classes it tries to identify and therefore requires fewer iterations.

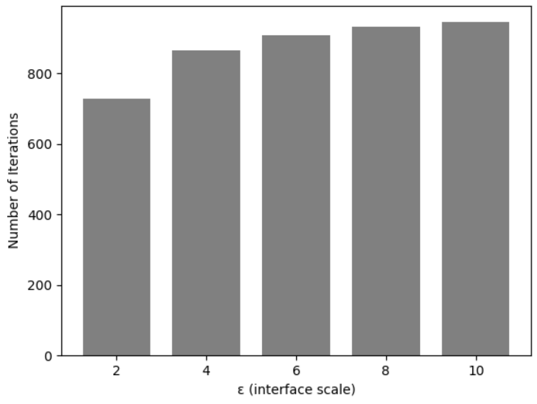


Figure 3.4: Number of Iterations for differing values of $\epsilon$

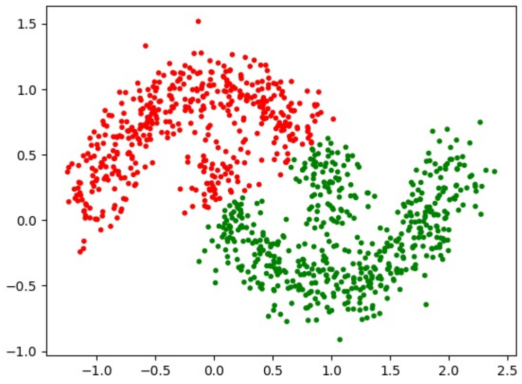


Figure 3.5: Example of a segmentation using GL method $\epsilon = 2$, $c = 1$

Figure 3.5 shows a segmentation of the two moons using the GL method, the accuracy isn't as high as shown in [4], however, the setup for the there graphs was not the same as this. They created a sparse graph whereas in these results a fully connected graph using the local scaling function is used, this may not account for all of the differences, but the algorithm displayed here still performs very fast and also better than the others in this report which is as expected.

## Overall

As can be seen above there was a difference in accuracy between the algorithms albeit minimal, however, this can be explained by the tough setup conditions presented by the similarity function and the noise levels of the Two Moons data set. It was clear that the GL algorithm had the highest accuracy levels, then the Fielder Method and finally the more inconsistent Perona Freeman algorithm. Another important part of the evaluation is the speed of these algorithms which can be seen in Figure 3.6b, this shows the times of the segmentation algorithms only not the graph setup and eigenvalue-vector computation. It identifies that the GL convex splitting algorithm is by far the slowest algorithm out of the 3 due to its increased complexity. Both the Fielder method and Perona Freeman method are extremely quick with the segmentation time being almost negligible. Despite the fact that the GL algorithm is an order of magnitude slower it is still relatively fast, with a speed of 0.78 seconds which is 65% faster than the quickest similarity function.

### 3.1.2 Similarity Functions

This section is a review of the different similarity functions applied to the Two Moons data set. Just the Fielder Method is used to segment the different similarity functions, as this function approximates a relaxed cut. Also, this algorithm doesn't accept any parameters, therefore only the similarity functions' effectiveness is being tested. All of the segmentations were performed 20 times and then the accuracy values were averaged.

Table 3.3 shows the accuracy values for varying levels of $\sigma$ when using the Gaussian function in combination with the Fielder method and different noise levels used to generate the two moons data set. A clear trend shows that values of $\sigma$ around 10 are most accurate, with values on either side of this yielding less accurate results. This table shows that it is possible to get high accuracy scores, with a simple algorithm such as the Fielder method if the parameters for the similarity function are set up correctly.

| **Gaussian Function** | | | | | |
|---|---|---|---|---|---|
| **NOISE** | $\sigma = 0.5$ | $\sigma = 2$ | $\sigma = 5$ | $\sigma = 10$ | $\sigma = 20$ |
| **0.15** | 0.737 | 0.765 | 0.802 | 0.84 | 0.859 |
| **0.175** | 0.739 | 0.763 | 0.789 | 0.81 | 0.807 |
| **0.2** | 0.739 | 0.762 | 0.782 | 0.796 | 0.772 |
| **Average** | **0.738** | **0.763** | **0.791** | **0.815** | **0.812** |

Table 3.3: Gaussian similarity function with a range of values for $\sigma$

Furthermore, table 3.4 presents a similar set of data but for a change of the value for N, which refers to the Nth nearest neighbour. Varying this data point from 2 all the way through to 20 shows a positive trend in the accuracy scores, albeit only a 6% increase. This table also shows slightly poorer results compared to that of the Gaussian Function, here the local scaling can't produce accuracy scores above 0.8.

| **Local Scaling Function** | | | | | |
|---|---|---|---|---|---|
| **NOISE** | **N = 2** | **N = 5** | **N = 7** | **N = 10** | **N = 20** |
| **0.15** | 0.708 | 0.756 | 0.773 | 0.783 | 0.789 |
| **0.175** | 0.8 | 0.797 | 0.798 | 0.797 | 0.797 |
| **0.2** | 0.724 | 0.759 | 0.767 | 0.77 | 0.775 |
| **Average** | **0.744** | **0.77** | **0.779** | **0.783** | **0.787** |

Table 3.4: Local Scaling Similarity function with a range of values for N

Finally, table 3.5 represents the cosine similarity function, there are no parameters for this function so it only has different values for the varying noise levels. The only trend that can be spotted is the fall in accuracy as the noise level increases which is to be expected as a higher noise value corresponds to a harder problem. The cosine function doesn't perform that well in the easiest test where the noise = 0.15, this is where the Gaussian Function shows its worth and has much higher accuracy values in this area.

| **Cosine Function** | |
|---|---|
| **NOISE** | |
| **0.15** | 0.785 |
| **0.175** | 0.782 |
| **0.2** | 0.778 |
| **Average** | **0.781** |

Table 3.5: Cosine Similarity function

Overall the Gaussian Function performs the best, especially with the correct $\sigma$ value where very high accuracy values can be found with the noise level being lower. Whilst the values above reflect accuracy values from the Fielder Method it can be said that this function also performs the best in terms of its speed performance, which is 40% faster than the cosine function and more than double the speed of the local scaling function which can be seen in figure 3.6a. The worst performing similarity function in regards to the Two Moons data set is the cosine function which across the three algorithms struggles to pull out the correct features of the graph in order to segment it correctly. Clearly, this data set is not well suited to an angle norm similarity function.

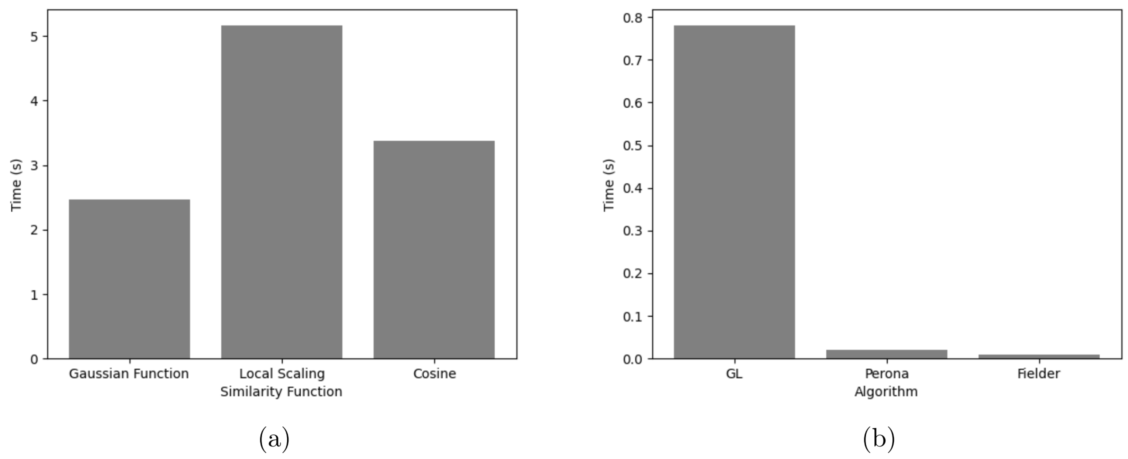


Figure 3.6: Speed charts for the computation time of a) the similarity functions and b) the algorithms

## 3.2 Images

This section will build on the last and show the different algorithms performing a binary segmentation on 3 separate images. These three separate images have all been sourced from NASA image library[21] or Unsplash [32], both copyright-free and allow for modification. All of the images hopefully will test the algorithms in a different way. The desired segmentation region has been hand-labelled using GIMP [31] which will allow for the Jaccard index to be used effectively by comparing the segmented image to the desired segmentation.

Due to the high fidelity of these images, the sub-sampling algorithm has been introduced, this will reduce the quality of the segmentations but allows for the computation of the results to be performed in a reasonable time frame. This will mean that the Jaccard accuracy score for the algorithms will inherently be reduced, but that is justified considering the implementation doesn't contain a fast method to solve the eigenvalues and vectors.

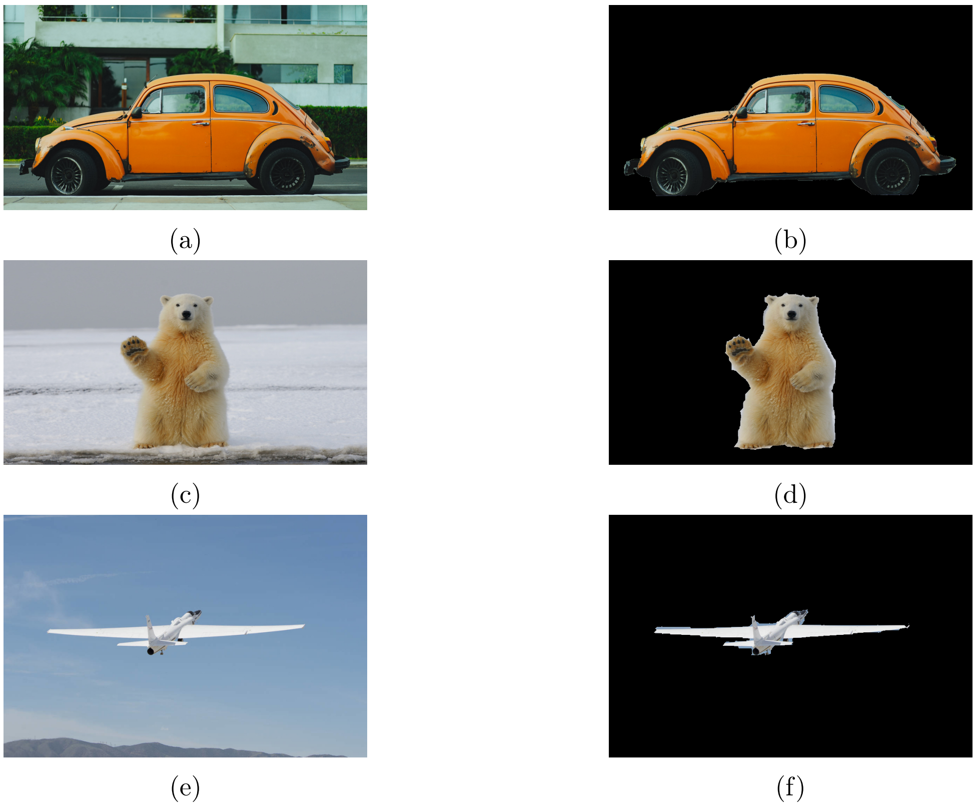


Figure 3.7: All of the sample images (left) and their respective desired segmentation (right). All three represent different challenges for binary image segmentation.

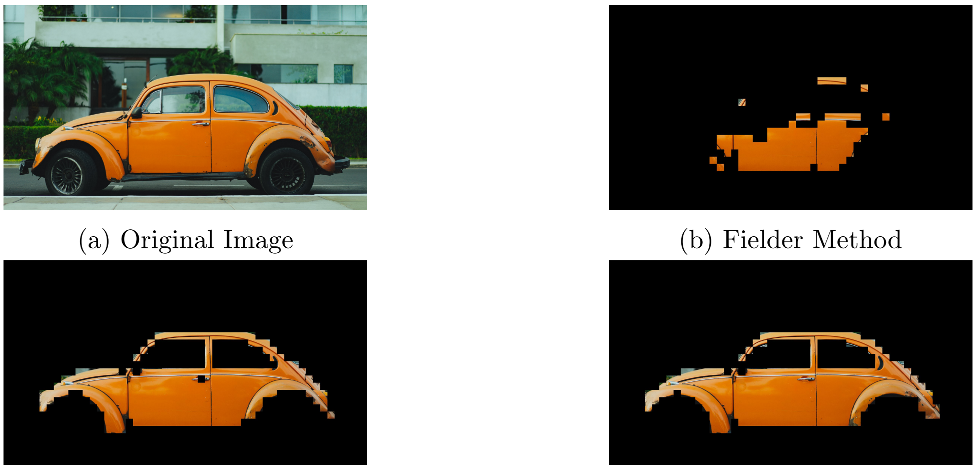
Looking at figure 3.7, the orange car represents a segmentation which should be simple to pick out the orange segments, however, it could be tricky for the algorithms to detect the tyres as they aren't similar in colour or intensity. The next image of the polar bear will be hard to segment because of its camouflaged look with its background, which is roughly the same colour. The plane image is similar to the car, with respect to the large colour separation from the foreground to the background, but due to its small tight edges, it could prove difficult due to the sub-sampling algorithm which is being used. One of the other challenges that the algorithms will face is the fact that the class representation on images 3.7c and 3.7e are much smaller than 50% which could prove a challenge for the algorithms, especially the GL function where the mean zero constraint is used.

For each image, the same similarity function will be used for all three segmentation algorithms. This will provide a setup that will hopefully not be biased toward any of the algorithms and should let us identify the best segmentation algorithm. The similarity function will differ from image to image and the parameter setup will be recorded.

### Image 1

For this image of the car, all three algorithms used the same similarity function, this was the modified Gaussian function using the RGB colour values to determine the Euclidean distance as discussed in section 2.2.2. The parameters used for this were, $r = 8$,$\sigma_g = 0.2$ and $\sigma_d = 4$. The Perona Freeman method was set up with 4 eigenvectors and the GL method used the parameters dt = 0.01, c = 21, $\epsilon = 0.1$ and 500 iterations. This setup was taken from [4] and from testing it produced some of the best results.

Figure 3.8 lays out the results that were obtained from this setup, as it clearly shows both the Perona Freeman and GL Methods produce strong results with a Jaccard score of 0.569 and 0.591 respectively. The wheels and windows proved to be an issue for both of the algorithms to correctly segment, these issues couldn't be solved well by using other similarity functions either. One of the only differences that can be seen between the Perona and GL methods is the handle of the car which wasn't correctly segmented by the Perona method. With a very poor Jaccard score of 0.273, this is a good example of the shortcomings of the Fielder Method, which fails to pick out any good segmentation at all.


(a) Original Image (b) Fielder Method

(c) Perona Method (d) GL Method

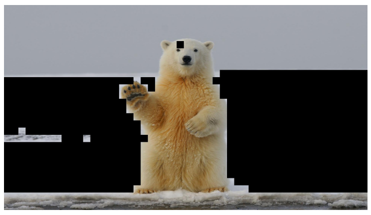
Figure 3.8: Shows the segmentations obtained from the different algorithms on the car image

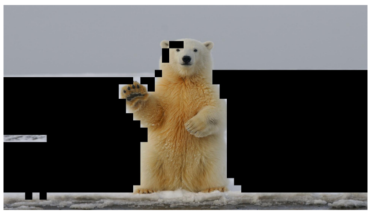
### Image 2

The Polar Bear image represents a tough problem due to the lack of change in colour throughout the image. For this problem, the modified Gaussian similarity function was chosen with the feature vector of the image pixels being the intensity (due to the lack of colour). The parameters used for this were, $r = 40$,$\sigma_g = 0.6$ and $\sigma_d = 2$. The Perona Freeman method was set up with 8 eigenvectors and also used the normalized laplacian instead of the standard. The GL method used the parameters dt = 0.1, c = 1, $\epsilon = 20$ and 500 iterations. From the testing that was executed, there were the parameter setups that yielded the best results.

Figure 3.9 clearly shows the GL method performs the best segmentation out of the 3 of them with a Jaccard score of 0.429. This was more than 50% better than both the Perona algorithm and the Feilder method which scored 0.255 and 0.281 respectively. Both of these algorithms

(a) Original Image


(b) Fielder Method - 0.281


(c) Perona Method - 0.255

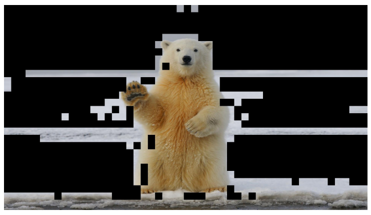

(d) GL Method - 0.429

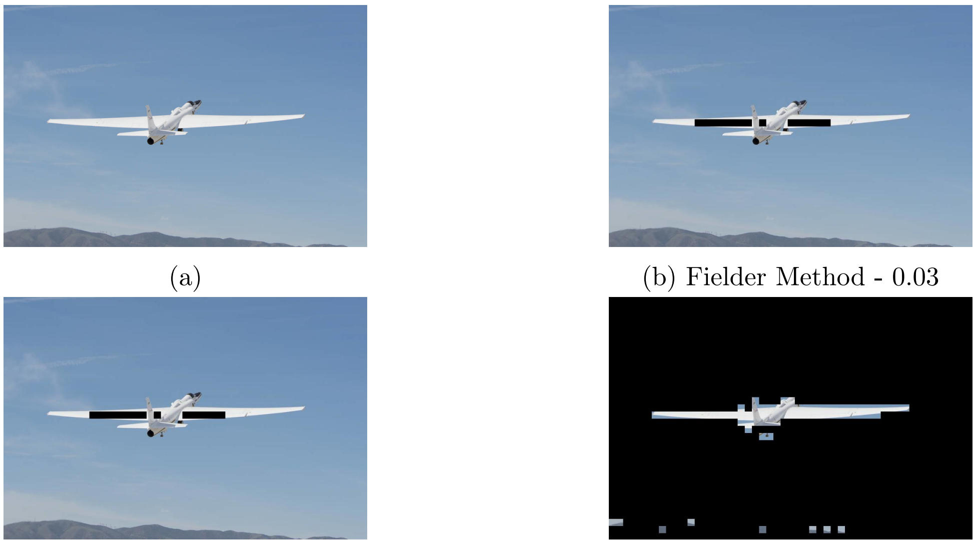
Figure 3.9: All of the sample images (left) and their respective desired segmentations (right). All three represent different challenges for image binary image segmentation

could identify the edges of the polar bear well but struggled to separate the sky and the darker foreground at the bottom from the bear. The GL algorithm identified the outline of the bear well but was patchy in areas and also really struggled with the darker foreground. That part of the image proved really tricky to segment with any parameters, most likely down to the fact that it is so vastly different in colour and intensity to the rest of the image, which can be hard to ignore when it isn't the segmentation target.

**Image 3**

For this problem, the modified Gaussian similarity function was chosen with the feature vector of the image pixels being the intensity (due to the lack of colour). The parameters used for this were, $r = 10$,$\sigma_g = 0.2$ and $\sigma_d = 4$. The Perona Freeman method was set up with 8 eigenvectors and also used the normalized laplacian instead of the standard. The GL method used the parameters dt = 0.01, c = 21, $\epsilon = 0.2$ and 500 iterations. This setup is similar to that of image 1 and the types of images reflect that similarity to the eye, making it easier to find good parameters for the segmentation.

Looking at figure 3.10, the GL method is once again the best, by some considerable margin with a Jaccard score more than 10 times that of the other two algorithms. However its Jaccard score isn't objectively amazing, this looks to be because of the sub-sampling of the image which makes the accuracy of the segmentation considerably smaller. Both the Fielder Method and the Perona Method yield poor results, once again confirming the fact that the Perona method struggles to maintain a high level of splitting accuracy and is extremely volatile to small changes in the weight function. Both of these algorithms manage to pick out the most obvious part of the image which may be where the second eigenvector is approximating the relaxed cut.


(a) (b) Fielder Method - 0.03

(c) Perona Method - 0.03 (d) GL Method - 0.43

Figure 3.10: All of the sample images (left) and their respective desired segmentations (right). All three represent different challenges for image binary image segmentation

**Overall**

From these three images and their corresponding segmentations, it is very clear that the GL method is by far the superior algorithm, confirming results shown in [4]. The flexibility of this algorithm allows it to achieve good results even with poor similarity functions. The effectiveness of the interface scale $\epsilon$ effectively sharpens the split of the two classes in the image. The Perona Freeman algorithm showed promise in certain tests but as can be seen, the variance in the results that it produces is too much for it to be a competent algorithm for this kind of image segmentation. The Fielder Method performed as expected due to the larger complexity of these images compared to the Two Moons data set. In regard to the similarity functions that were used to conduct these results for the images, only the Gaussian function (using intensity and colour as feature vectors) could pick out any meaningful segmentation. The cosine similarity function failed to pick out any useful segmentation from all of the testing that was conducted. Using the different feature vectors worked well for different types of images, images where colour depicted a lot of the segmentation allowed favour of the colour vectors and likewise with more monotone colour intensity vectors produced better results.

# Chapter 4

# Discussion

After exploring the origins of spectral graph methods and the graph cutting algorithms, then conducting the experiments in Chapter 3. This final Chapter aims to evaluate the effectiveness of the 3 algorithms and draw any conclusions that can be made.

## 4.1 Conclusions

In summary, Chapter 3 presents conclusive results detailing the effectiveness of the three algorithms used in the experiments. It became clear that the GL algorithm was the most stable and accurate across the three of them. Also, because of the range of parameter inputs for this algorithm, it became a much more versatile algorithm that could adjust its outputs depending on the desired segmentation. Testing across both the Two Moons' data set and the images has proved the effectiveness of the GL algorithm and has shown how well it performs on a variety of segmentation problems. Unfortunately, the algorithm and setup in this report failed to reproduce the results shown in [4] that presented the algorithm in regard to the Two Moons. However, the results shown when segmenting images were very good for the unsupervised version of the algorithm. As shown in Figure 1.6, different eigenvectors split different areas of an image. Therefore the algorithm's ability to combine and use multiple eigenvectors and eigenvalues together appears to give it a large advantage over the other algorithms.

The Perona Freeman method turned out to be a very volatile algorithm that was sensitive to the initial conditions and set-up of the graph weights. Chapter 3 highlights this fact with both the results from the Two Moons data set and the image segmentations, where both were subject to a range of good and poor cuts. One of the likely reasons for this is the fact that the algorithm relies on the standardised Laplacian instead of the normalized Laplacian, and the standard Laplacian may struggle to pick apart different segments of the graph if the weights are set up in its favour.

The Fielder method performed as expected, with good performing results in the Two Moons data set and poor results with the images. As discussed previously the second eigenvector of a graph Laplacian approximates a relaxed cut of that graph, this made it a good tool for segmentation problems but not a final solution. The reliance that these algorithms have on this eigenvector can be seen with the GL Method which, for all of the above tests, used an initial function, $u_0(x)$, based around the second eigenvector.

One of the biggest findings from Chapter 3 is the reliance these algorithms have on the setup of the graph and its weight function. For example, finding the correct parameter settings for specific image segmentation requires a good knowledge of the desired segmentation and the features of the graphs that one is dealing with. Good image segmentation took longer to achieve and required many different parameter changes. This issue is compounded by the lack of transfer ability with parameters onto other images, especially if the images differ in their features for example if one image is very similar in colour, but not in intensity and visa versa.

The graph setup time and the eigenvalue eigenvector computation heavily took up the computational time for the segmentations throughout this report. The Complexity of the Numpy algorithm is $O(n^3)$, which is the limiting complexity of the entire segmentation process. This made it very difficult to segment high-fidelity images, therefore a sub-sampling algorithm had to be implemented, which caused the accuracy of these segmentations to be less effective.

## 4.2 Ideas for future work

From this report, there are many directions a future researcher could take. Listed below are some interesting that come to mind.

- One of the main ways that someone could continue working on this project would be to speed up the computation of the entire segmentation process. This would mainly involve speeding up the eigenvalue eigenvector computation. There are many ways to do this and the implementation in the current project is very slow allowing for much improvement. There are very good libraries that provide fast ways of computing eigenvalues-vectors of matrices both sparse and dense, PETSc [2] is a good example of this. Moreover, The Nystrom extension is a strong method that computes eigenvalues-vectors of a dense matrix outlined in [4].
- Another interesting approach for further work from this report would be to implement the supervised version of the GL method with the fidelity function included. This would allow for the image labelling transfers and would also improve results significantly when segmenting images, as you would be able to include training data of images that looked similar and then feed that information in through the fidelity function.
- Finding the correct parameters to accurately segment images and graphs, proved to be hard from the experiments conducted in this project. Therefore, devising a more efficient way to find parameters that produce good segmentations would be a good extension of this project. This would allow for faster and more generic segmentations of images and other graphs.
- One final direction that could be taken with this project would be to take all of the implemented code and produce a teaching or learning tool. This could include a well-designed GUI that lets the user interactively change parameters to specific values and monitor the changes in real-time. Allowing users with little to no knowledge get to grasp these algorithms and the nuances of how they work. Python packages such as customTkinter would allow for this type of development. Alternatively, a web-based approach using a Flask app [14] and deploying to pythonAnywhere would also be a good approach.

# References

[1] S. Ahuja. Metrics to evaluate your semantic segmentation model. https://towardsdatascience.com/metrics-to-evaluate-your-semantic-segmentation-model-6bcb99639aa2, 2020. Accessed: 2023-04-09.

[2] S. Balay, S. Abhyankar, M. F. Adams, S. Benson, J. Brown, P. Brune, K. Buschelman, E. M. Constantinescu, L. Dalcin, A. Dener, V. Eijkhout, J. Faibussowitsch, W. D. Gropp, V. Hapla, T. Isaac, P. Jolivet, D. Karpeev, D. Kaushik, M. G. Knepley, F. Kong, S. Kruger, D. A. May, L. C. McInnes, R. T. Mills, L. Mitchell, T. Munson, J. E. Roman, K. Rupp, P. Sanan, J. Sarich, B. F. Smith, S. Zampini, H. Zhang, H. Zhang, and J. Zhang. PETSc Web page. https://petsc.org/, 2023.

[3] J. Bertels, T. Eelbode, M. Berman, D. Vandermeulen, F. Maes, R. Bisschops, and M. B. Blaschko. Optimizing the dice score and jaccard index for medical image segmentation: Theory and practice. In *Lecture Notes in Computer Science*, pages 92–100. Springer International Publishing, 2019.

[4] A. L. Bertozzi and A. Flenner. Diffuse interface models on graphs for classification of high dimensional data. *Multiscale Modeling & Simulation*, 8(4):1544–1569, 2010.

[5] Y. Boykov and M.-P. Jolly. Interactive graph cuts for optimal boundary & region segmentation of objects in n-d images. In *International Conference on Computer Vision*, volume 1, pages 105–112. IEEE, July 2001.

[6] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, 2001.

[7] F. Chung and R. L. Graham. *Spectral Graph Theory.* Number 92 in CBMS Regional Conference Series in Mathematics. American Mathematical Society, Providence, RI, 1997.

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

[9] A. Clark. Pillow (pil fork) documentation, 2015.

[10] G. Csardi and T. Nepusz. igraph: Network analysis and visualization. *Journal of Statistical Software*, 24(6):1–26, 2008.

[11] J. W. Demmel. Graph partitioning, part 2. Website, 2008. Accessed March, 2023.

[12] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[13] D. Freedman and P. Drineas. Energy minimization via graph cuts: Settling what is possible. volume 2, pages 939– 946 vol. 2, 07 2005.

[14] M. Grinberg. *Flask web development: developing web applications with python.* " O'Reilly Media, Inc.", 2018.

[15] A. A. Hagberg, D. A. Schult, and P. J. Swart. Networkx: High-productivity software for complex networks. https://networkx.github.io, 2021. Accessed on March 21, 2023.

[16] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, Sept. 2020.

[17] T. S. Huang. Computer vision: Evolution and promise. 1999.

[18] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[19] V. Kolmogorov and R. Zabin. What energy functions can be minimized via graph cuts? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2):147–159, 2004.

[20] C. McCormick. The gaussian kernel. https://mccormickml.com/2013/08/15/the-gaussian-kernel/, 2013. Accessed on: February 21, 2023.

[21] NASA Armstrong Flight Research Center. Guppy aircraft image. https://www.nasa.gov/centers/armstrong/multimedia/imagegallery/Guppy/AFRC2019-0062-003.html, 2019. Accessed: April 24, 2023.

[22] W. "Perona, P.and Freeman. "a factorization approach to grouping". In H. "Burkhardt and B. Neumann, editors, *"Computer Vision — ECCV'98"*, pages "655–670", "Berlin, Heidelberg", "1998". "Springer Berlin Heidelberg".

[23] Python Software Foundation. Python 3.8.10. https://www.python.org/downloads/release/python-3810/, 2021. Accessed November, 2023.

[24] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, jan 2015.

[25] A. Schrijver. On the history of the transportation and maximum flow problems. *Mathematical Programming*, 91(3):437–445, February 2002.

[26] Scikit-learn developers. Scikit-learn: Machine learning in Python. https://scikit-learn.org/stable/modules/classes.htmlmodule-sklearn.datasets, 2021. Accessed on March 21, 2023.

[27] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[28] D. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30, 10 2012.

[29] A. Singhal. Modern information retrieval: A brief overview. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 24(4):35–43, 2001.

[30] A. A. Taha and A. Hanbury. Metrics for evaluating 3d medical image segmentation: analysis, selection, and tool. *BMC Medical Imaging*, 15(1):29, 2015.

[31] The GIMP Development Team. Gimp.

[32] Unsplash. Unsplash. https://unsplash.com/. Accessed: April 24, 2023.

[33] Y. Weiss. Segmentation using eigenvectors: A unifying view. *Proc. of ICCV*, 1:975–982, 2001.

[34] Y. Weiss and G. Hamerly. Segmentation using eigenvectors: a unifying view. Lecture presented at University of California, Berkeley, 2002.

[35] J. Wyss-Gallifent. *Graph Theory.* Springer, 2021.

[36] L. Zelnik-Manor and P. Perona. Self-tuning spectral clustering. volume 17, 01 2004.

# Appendix A

# Self-appraisal

## A.1 Critical self-evaluation

This project was by far and away the most complex and largest project I have undertaken. It tested my patience and smarts to the greatest extent. However, I feel very proud of the amount of knowledge that I have managed to acquire in the field of spectral graph theory and graph-cutting algorithms considering I started with very limited knowledge on the topic. Whilst I am pleased with what I achieved, I am slightly frustrated I didn't reach some of the extended goals I initially established. With a clearer initial goal, I believe this project would have allowed me to go further and explore more.

The idea of self-teaching and learning with limited guidance was very intimidating, yet also very rewarding. While learning and researching many topics related to this project, I encountered many dead ends and followed many ideas that led to nothing and didn't progress my project. This was very infuriating and left me feeling like I was making little to no progress. However, looking back now it seems that all of this, what appeared unhelpful, reading gradually paid off and has positively contributed to the final piece of work. I spent a lot of time looking up definitions of new words when reading these papers, these words seem trivial now but I remember how confusing these terms were at the beginning.

Initially, I found the topic of the GL method very hard to comprehend and understand due to the lack of knowledge I had in these areas. However, after a lot of perseverance and a lot of further reading about the topic, the algorithm and how it worked in many aspects became more apparent. This gave me a great sense of achievement. Once I began to understand the algorithm, I still could get anywhere near the performance stated in the original paper, and after many iterations of this code, I finally got it to a point where the performance began to match that which was stated in the paper. I couldn't find any external resources explaining this algorithm, so with just the very hard-to-understand paper I believe I did a good job implementing the algorithm.

The experimental phase of my project was an area that I underestimated in terms of the amount of time that it would require. I knew the areas of the code and algorithm that I wanted to measure and present in the report, however building and making this a reality required unforeseen development. For example, I hadn't realised how slow the calculation of the eigenvalues and eigenvectors would be when applying them to a high-fidelity image. This meant I had to find a workaround, which ended up being the sub-sampling algorithm that made this problem more manageable. All in all, this amounted to time that ate into further development of the code to achieve some of the extended goals I had established.

## A.2 Personal reflection and lessons learned

One of the main things I would change if I had to repeat a similar project would be to start the report much sooner than I did. Even though I gave myself ample time to complete the report, there were many design changes and evaluation metrics that I had to adjust. These adjustments were made because I realized that the way that I had planned the evaluation and discussion would not work in practice. Starting my report earlier would have provided me with the necessary information to change these items earlier and make better progress on them.

Furthermore, I wish I had clarified my goals and requirements for the project earlier than I did. For the first months of my project, I found it difficult to establish exactly what I was aiming to do and what direction my project was heading in. This made it very difficult to really get started with large amounts of the coding and it made me feel like I was behind schedule the whole time.

My organisational skills have been tested to the fullest extent throughout this project and I am very happy with how they held up. From christmas onwards, I made sure I kept up consistent effort on this project, even if the quantity of work wasn't as high as I would like. In the end, this meant I had ample time to review and iterate upon the work that I had done, avoiding any last-minute cramming that may have plagued my work in previous years.

All in all, I feel that this project has given me a strong knowledge of the topics covered in this report as well as an overall understanding of how to approach a similar-sized task in the future. It has given me a strong sense of the practises that need to be implemented to help progress these projects easier and without as many headaches. Focusing on goals and requirements and making sure there is a consistent and dedicated effort goes a long way to improving the outcomes of these projects.

## A.3 Legal, social, ethical and professional issues

### A.3.1 Legal issues

All of the work conducted in this project is done so with software that is eligible for educational use and/or open source. All of the libraries that have been used are available for educational use and/or are open source. They have also all been cited correctly so that the source can be identified. Furthermore, the images that were used for the segmentation are copyright free, available for this use and cited.

### A.3.2 Social issues

No work in this report relates to any personal data or imagery. Images used in the paper are copyright free and cited correctly. No user testing was completed during this project, so there aren't any social issues to cover.

### A.3.3 Ethical issues

In relation to my work and the data used in this project, there aren't any ethical issues. However, looking at the field of image segmentation more widely, one may begin to find issues that arise in the use and data collection that can come from this field. Due to the sensitive image data used in an area such as medical imaging, this may raise ethical concerns relating to the mismanagement of this information. Furthermore, imagine a scenario where these image segmentation algorithms may be used to track and surveil specific individuals. This would be both a breach of privacy and civil liberties, potentially leading to a world where this algorithm could be used in a discriminatory manner. Despite these ethical issues not applying to specific work in this project, they are still important to address.

### A.3.4 Professional issues

Throughout the duration of this project I have taken steps to make sure I have avoided plagiarism in any elements of my report be it with the development of the code or the written report. All of my sources have been cited and the code written is all my own unless explicitly stated. My code repository and report are well structured and presented, hopefully speaking to my utilisation of best practices.

# Appendix B

## External Material

The Two Moons data set was sourced from the Python SciKit module which can be found here - https://scikit-learn.org/stable/

The computation of the graph Laplacian both normalised and standard relied on functions from the networkX module - https://networkx.org/

Computation of the eigenvectors and eigenvalues was done using functions from the Numpy module - https://numpy.org/doc/stable/index.html