0-1 Graph Partitioning and Image Segmentation

by

Chun Fan Goh

B. Eng, Nanyang Technological University, Singapore, 2007

Submitted to the School of Engineering
in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Computation for Design and Optimization

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2008

Signature of Author.....
Department of Computation for Design and Optimization
August 12, 2008

Certified by.....
Gilbert Strang
Professor of Mathematics
Thesis Supervisor

Accepted by.....
Jaime Peraire
Professor of Aeronautics and Astronautics
Co-Director, Computation for Design and Optimization Program

0-1 Graph Partitioning and Image Segmentation

by

Chun Fan Goh

Submitted to the School of Engineering
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computation for Design and Optimization

ABSTRACT

Graph partitioning is the grouping of all the nodes in a graph into two or more partitions based on certain criteria. Graph cut techniques are used to partition a graph. The Minimum Cut method gives imbalanced partitions. To overcome the imbalanced partitioning, the Normalized Cut method is used. However, it is computationally expensive. The Isoperimetric Partitioning is faster and more stable, and I aim to extend and develop the related ideas.

In this thesis, I propose a graph partitioning method – the *0-1 Graph Partitioning*. I treat a graph as an electrical circuit: a few nodes are fixed as the voltage inputs (sources), another few nodes are grounded (sinks), and the weight of each edge is seen as the conductance between the two ends (nodes) of the edge. With this setup, other nodes have voltages in between zero and input voltage. The method cuts the graph between the sinks and sources according to the nodes' voltages and in such a way that it minimizes the normalized cut value. The method leads to the Graph Laplacian System -- a linear system. As opposed to the Normalized Cut method, which solves an eigenvalue problem to partition a graph, solving a linear system is much faster and more stable. In addition to the speed, I have proven empirically that the quality of the bi-partitions is comparable to the Normalized Cut method. Based on the 0-1 method, I have also developed the Fiedler Quick Start algorithm, which can compute the Fiedler vector faster than solving the generalized eigensystem.

I have also applied the graph partitioning algorithm to image segmentation. In comparison to the Normalized Cut method, we show that the method not only gives good segmentation, but it is also much simpler and faster in terms of the construction of a graph from an image, and robust to any noise contained in an image. With the speed and simple graph construction advantage, the method can be applied to large images. The method is object-oriented. It focuses on the objects of images and it is able to segment out objects in the first bi-partition. For k-way image segmentation, the 0-1 method can be applied in both the simultaneous and recursive ways. Apart from the 0-1 image segmentation, I have also developed the Resized Image Segmentation Scheme and the Refinement Scheme (Fast and Thorough), which can speed up the image segmentation process and improve the segmentation. Both schemes can be used by any graph based image segmentation methods.

Thesis Supervisor: Gilbert Strang
Title: Professor of Mathematics, Department of Mathematics

ACKNOWLEDGEMENTS

First and foremost, I would like to express my sincere gratitude to my thesis advisor, Professor Gilbert Strang for his valuable advice and patient guidance. He has also imparted valuable knowledge and shared his experiences in numerical linear algebra, which benefit me greatly. Professor Strang, I truly enjoy your lessons in Computational Science and Engineering and your Linear Algebra video clips.

I would like to express my appreciation to Professor Jonathan A. Kelner for his valuable explanation of the normalized cut concept.

I would like to thank Dr. John Desforge and Miss Jocelyn Sales from the Singapore MIT Alliance (SMA) office for their support and advice throughout my stay at MIT. They make my MIT life wonderful.

A big 'Thank You' goes to the administrative staff from CDO (Mrs. Laura Koller) and SMA (Mr. Michael Lim) too for handling all the paper works, allowing me to focus on my study.

I also would like to thank my roommate, Mr. Kong Tian Fook for his useful discussions and suggestions on my research (and also his cooking). I owe thanks to Mr. Ooi Boon Hooi, Miss Fitriani, and Miss Zhang Yifeng for their constant encouragement and friendship.

Lastly, I would like to dedicate this thesis to my family, especially my new born niece, baby Xin Yee. Their love is my constant source of inspiration and aspiration.

CONTENTS

LIST OF FIGURES

Figure 3.13 Image Segmentation by the Spectral Rounding method. Column (a) shows the original images before segmentation and the pixel intensities of the square. Column (b) shows the partition vectors (blue curve) and the splitting points (red horizontal line). The *Ncut* values are given at the bottom of each plot in column (b). Comparing the partition vector plots with the Fiedler vector plot in Figure 3.12, we see that the vector plots by SR are discrete (binary). Column (c) shows the segmented part from the image (the square). The method fails to segments out the center square when the pixel intensity of the square is 40 (Row 3). Though the vector plot shows discrete values, but the segmentation is still incorrect. This is because the SR method starts with a continuous Fiedler vector (shown in the row 3 of Figure 3.12), an approximation which is far from the discrete solution of the Normalized Cut problem. For pixel intensity above 40, the method performs the segmentation correctly. Notice the *Ncut* value of the last row, it is higher than the *Ncut* value of the correct segmentation (0.0233). Incorrect segmentation gives higher *Ncut* value. 68

Figure 3.14 Image Segmentation by the Isoperimetric Partitioning. Column (a) shows the original images before segmentation and the pixel intensities of the square. Column (b) shows the partition vector (red curve) and the splitting point that gives the minimum *Ncut* value (blue horizontal dashed lines). The *Ncut* values are given at the bottom of each plot in column (b). Column (c) shows the images given by the isoperimetric solutions before discretization. Column (d) shows the segmented parts from the images (the square). The method fails to segment out the center square when the pixel intensity of the square is 10 (Row 3). For pixel intensity differences above 10, the method performs the segmentation correctly as the weak edges are weak enough to be detected. Notice that when the square is still observable in Column (c), the center square can be segmented out in Column (d). Also notice the *Ncut* value of the last row. It is interestingly smaller than the *Ncut* value of the correct segmentation (0.1998). The *Ncut* is not the absolute measurement for correct segmentation. 69

Figure 3.15 Image Segmentation by the Minimum Cut method for different image size. Column (a) shows the original images before segmentation with the image size stated at the bottom of the images. Column (b) shows the segmented parts from the images (the squares). The Normalized Cut values are stated at the bottom of each image in Column (b). The image size increases from 15 x 15 to 300 x 300 (Row 1 to 3). The method successfully segments out the

LIST OF TABLES

Chapter 1 Introduction

1.1 Background

1.1.1 Graphs

Graphs are the most important model in applied mathematics [13]. A graph $G(N, E)$ consists of n nodes (vertices) and m edges (links) that connect the nodes. A graph can be classified as a weighted graph or unweighted graph. For a weighted graph, an edge $e_{i,j}$ connecting node i and j is assigned a weight $w_{i,j}$. Apart from the edge weight, a graph can also be classified as a directed graph or undirected graph. If a graph contains ordered pairs of nodes, it is a directed graph [1]. In this thesis, I focus only on the undirected graphs. Figure 1.1 shows an example of a weighted undirected graph.

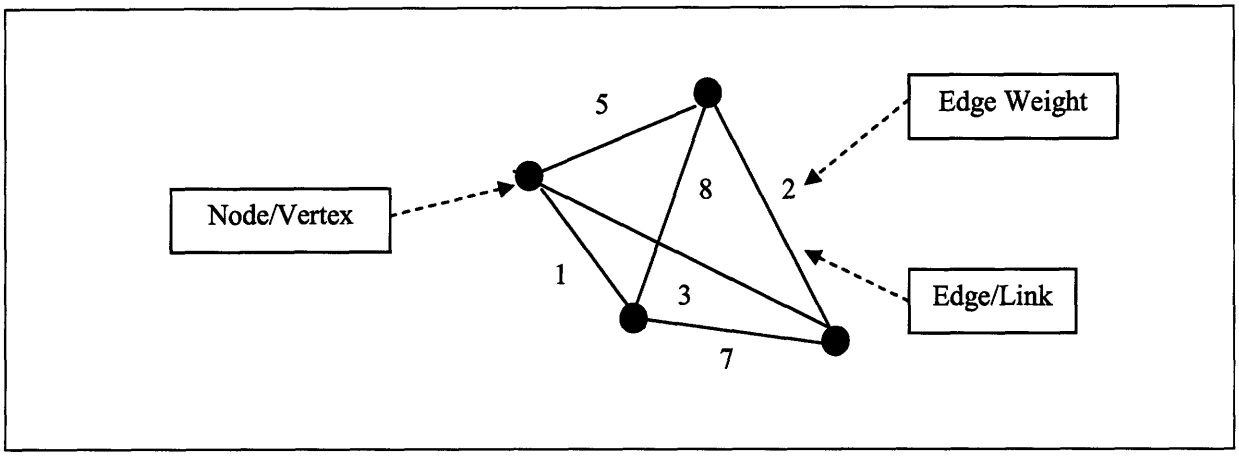


Figure 1.1 A weighted undirected graph.

Graphs can be represented in matrices: incidence, Laplacian, degree and adjacency matrices [13]. For the construction of the matrices from a graph, the graph nodes and edges are numbered. Incidence matrix $\boldsymbol{A}$ is a matrix of 0, 1 and -1 that records the nodes on its columns and the edges on its rows. On each row, the i^{th} entry and j^{th} entry with value of 1 and -1 represent an edge between i^{th} node and j^{th} node. The edge weight of the graphs can be stored in a diagonal matrix $\boldsymbol{C}$.

The Laplacian matrix $\boldsymbol{L}$ can be constructed by the multiplication of the incidence matrix $\boldsymbol{A}$ and diagonal weight matrix $\boldsymbol{C}$:

$$\boldsymbol{L} = \boldsymbol{A}^{\boldsymbol{T}} * \boldsymbol{C} * \boldsymbol{A}. \tag{1.1}$$

The diagonal degree matrix $\boldsymbol{D}$ is the diagonal of the Laplacian matrix $\boldsymbol{L}$. The diagonal entries give the sum of edge weights connected to the nodes. In the case of unweighted graphs, the diagonal entries of the matrix $\boldsymbol{D}$ give the number of edges connected to the nodes. The adjacency matrix $\boldsymbol{W}$ consists of the off diagonal entries of the Laplacian matrix $\boldsymbol{L}$. Similar to the incidence matrix $\boldsymbol{A}$, it gives the information of connectivity between the nodes. The relationship between the degree matrix $\boldsymbol{D}$, adjacency matrix $\boldsymbol{W}$ and Laplacian matrix $\boldsymbol{L}$ can be summarized as follows:

$$\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{W}. \tag{1.2}$$

The Laplacian matrix $\boldsymbol{L}$ is singular and symmetric positive semi definite. It has a zero eigenvalue.

1.1.2 Graph Partitioning

Graph partitioning is the grouping of all the nodes in a graph into two or more partitions based on certain criteria. The criteria can be the location of the nodes, the node value (pixel intensity, in the case of image segmentation), or the connectivity of the nodes. Graph cut techniques are used to partition a graph. They include the Minimum Cut method, Normalized Cut method and Isoperimetric Partitioning. The detail descriptions of the methods are reported in Chapter 2.

1.1.3 Applications

The graph is important because of its wide range of applications. It can be used to model many problems such as the transportation network problem, the modeling of electrical circuits, the Internet network, assignment problems, and scheduling problems [1].

Graph partitioning can be used in data clustering [11]. Currently, bioinformatics researchers are trying to use the graph cut to cluster the microarray data [8]. They tried to group the tissue samples according to the similarity in gene activity. At the same time, computer scientists use it in computer vision to segment images (image segmentation) [10, 12], helping the machines to see like humans.

Graph partitioning is also important in parallel computing. It helps to divide the data (node) into balanced groups and at the same time reduces the connection (edges) between the data groups [4]. In this way, we can divide the work load equally to each of the parallel computers for faster computation.

1.2 Motivation

Graph cut techniques are used to partition a graph. In 1993, Wu and Leahy [14] used the minimum cut (*min cut*) criterion to partition a graph. However, they also pointed out that the criterion often gives imbalanced partitions. To overcome the imbalanced partitioning, Shi and Malik [12] introduced a new criterion -- normalized cuts (*Ncut*). Unfortunately, the normalized cut problem is non-deterministic-polynomial-time-complete (NP-complete). The relaxed problem leads to a generalized eigenvalue problem, which is computationally expensive. The lack of speed limits the application of the method, especially in image segmentation. Grady and Schwartz's work on Isoperimetric Partitioning in 2005 [7] showed that we can partition a graph using a linear system, which is much faster and more stable, but the partitions' quality is

compromised due to the limitation of grounding. Hence, there is a need to find a graph partitioning algorithm that is fast and gives good partitions. To achieve this, the current methods need to be further studied to know their strengths and weaknesses. In doing this analysis, I hope to develop a new method that combines the advantages and excludes the disadvantages of the current methods.

Image segmentation is an important application of graph partitioning. A good graph partitioning method should be easily extended to image segmentation. Hence, image segmentation can be a good test of practicality of any newly developed graph partitioning method.

1.3 Objectives

The main goals of this thesis are to:

i. review the performance of the current graph partitioning techniques.
ii. study empirically the application of the graph cut techniques in image segmentation.
iii. develop a new graph partitioning algorithm that is faster, and gives more stable and better partitions compared to the current techniques.
iv. apply the new graph partitioning algorithm in image segmentation.

1.4 Thesis Outline

In this thesis, my focus is on graph partitioning and its application in image segmentation. In Chapter 2, I review the graph partitioning methods (Minimum Cut, Normalized Cut and Isoperimetric Partitioning methods) and analyze the strengths and weaknesses of the methods in partitioning unweighted and weighted graphs, and also their ability to extend from bi-partitioning to k-way partitioning. In Chapter 3, I study empirically the performance of the methods in image segmentation. This allows me to study further the strengths and weaknesses of the methods in partitioning weighted graphs.

After determining the weaknesses of each method in Chapter 2 and 3, I developed a new graph partitioning method – 0-1 Graph Partitioning by combining the electrical circuit concept of Isoperimetric Partitioning and the minimum normalized cut discretization of the Normalized Cut method. Chapter 4 describes the details of the algorithm and the derivation of the method. It also presents the *2*-way and *k*-way unweighted graph partitioning results using the new method.

I have also applied the new method to image segmentation to show the practicality of the method. In Chapter 5, the algorithms and results of the application of 0-1 Graph Partitioning in 2-way and *k*-way image segmentation are described. In addition, I also analyzed the advantages and disadvantages of the method. Along with the 0-1 image segmentation, I also developed two general schemes for graph-based image segmentation methods. The detail description and results of these two schemes are also reported in Chapter 5.

Finally, Chapter 6 concludes the thesis and gives recommendations for future works.

Chapter 2 Graph Cut Techniques

2.1 Minimum Cut

Given a graph $G(N, E)$ partitioned into two groups: S and $\bar{S}$, a minimum cut (*min cut*) is defined as:

$$\min_S \sum_{i \in S,\ j \in \bar{S}} w_{i,j}, where\ S \subseteq N. \tag{2.1}$$

It is initially used to solve the maximum flow problem. In 1993, Wu and Leahy [14] used the *min cut* criterion to partition a graph. However, this criterion often causes imbalanced partitions.

2.2 Normalized Cut Partitioning

Shi and Malik introduced a new criterion – Normalized Cuts (*Ncut*) in 2000 [12]. For a graph $G(N, E)$ partitioned into two groups (S and $\bar{S}$), the normalized cut is defined as:

$$Ncut(S, \bar{S}) = \frac{cut(S,\bar{S})}{assoc(S,N)} + \frac{cut(S,\bar{S})}{assoc(\bar{S},N)}, \tag{2.2}$$

where $cut(S, \bar{S})$ is the sum of the edge weights between S and $\bar{S}$, and $assoc(S, N)$ is the sum of edge weights over all the connections of nodes in S.

For a graph $G(N, E)$ with multiple partitions $S_1, S_2, \ldots, S_k$, the normalized cut is defined as:

$$Ncut(S_1, S_2, \ldots, S_k) = \sum_{i=1}^{k} \frac{cut(S_i, \bar{S}_i)}{assoc(S_i,N)}, \tag{2.3}$$

where $cut(S_i, \bar{S}_i)$ is the sum of the edge weights between S_i and its complement $\bar{S}_i$, and $assoc(S_i, N)$ is the sum of edge weights over all the connections of nodes in S_i.

Representing the graph in a degree matrix $\boldsymbol{D}$ and adjacency matrix $\boldsymbol{W}$; and the partitions in k binary partition vectors $\boldsymbol{p_{i,\ i=1,\ldots,k}}$; the *Ncut* value is computed as follows:

$$Ncut\left(\boldsymbol{D}, \boldsymbol{W}, \boldsymbol{p_{i,\ i=1,\ldots,k}}\right) = \sum_{i=1}^{k} \frac{\boldsymbol{p_i}^T * \boldsymbol{W} * \overline{\boldsymbol{p}}_i}{\boldsymbol{p_i}^T * \boldsymbol{D} * \boldsymbol{p_i}}, \tag{2.4}$$

where $\overline{\boldsymbol{p}}_i$ is the complement of $\boldsymbol{p_i}$.

Minimizing the *Ncut* value of a graph gives a balanced partition of the graph. Unfortunately, the normalized cut problem is non-deterministic-polynomial-time-complete (NP-complete). Thus, this leads to spectral clustering. The problem is relaxed, and Shi and Malik [12] showed that the relaxed problem is an eigenvalue problem with constraints:

$$\begin{aligned} &(\boldsymbol{D} - \boldsymbol{W}) * \boldsymbol{x} = \lambda * \boldsymbol{D} * \boldsymbol{x}, \\ &s.t \quad \boldsymbol{x} * \boldsymbol{D} * \boldsymbol{1} = \boldsymbol{0}, \end{aligned} \tag{2.5}$$

where $\boldsymbol{D}$ is the degree matrix, and $\boldsymbol{W}$ is the adjacency matrix. The second eigenvector $\boldsymbol{x_2}$ is called the Fiedler vector. It can be discretized to give the partitions for *2*-way graph partitioning. Hence, for *2*-way graph partitioning, the Normalized Cut method is also called the Fiedler method. For *k*-way graph partitioning, the first *k* eigenvectors are used to give the *k* partitions.

Though the Normalized Cut method gives good partitions, the cost to compute the eigenvalues and eigenvectors is high. Many applications, especially in image processing, require fast computation and large graphs. The lack of speed limits the application of the method. In addition, since the solution to the eigenvalue problem is not unique, the partition may vary from time to time. The method is not stable. Apart from these two problems, the method is sensitive to noise (in the case of image segmentation, see Chapter 4).

2.3 Isoperimetric Partitioning

To address the issue of speed and stable partition, Grady and Schwartz created the Isoperimetric Partitioning [1], in which they partition a graph by optimizing the Isoperimetric constant of the graph. The Isoperimetric constant h of a graph $G(N, E)$ is defined as:

$$h = \inf_S \frac{cut(S,\bar{S})}{vol(S)}, \tag{2.6}$$

where $cut(S,\bar{S})$ is the sum of the edge weights between S and its complement $\bar{S}$; and $vol(S)$ can be defined as either the sum of edge weights over all the connections of nodes in S or the number of nodes in S [7].

Expressing the graph in a Laplacian matrix $\boldsymbol{L}$, we can calculate the Isoperimetric constant h of the graph by the following matrix equation:

$$h(\boldsymbol{p}) = \frac{\boldsymbol{p}^T * \boldsymbol{L} * \boldsymbol{p}}{\boldsymbol{p}^T * \boldsymbol{1}}, \tag{2.7}$$

where $\boldsymbol{p}$ is the binary vector that represents S and $\bar{S}$ (nodes with entry of 1 are the nodes in S; nodes with entry of 0 are the nodes in $\bar{S}$).

In [7], Grady and Schwartz have shown that this criterion can lead to a linear system:

$$\boldsymbol{L} * \boldsymbol{x} = \boldsymbol{1}, \tag{2.8}$$

where $\boldsymbol{L}$ is the Laplacian matrix of a graph; and $\boldsymbol{x}$ is the continuous vector solution that gives the partitions when it is discretized. Since the Laplacian matrix is singular, the system can only be solved by fixing an entry of $\boldsymbol{x}$ to zero (grounded). In this way, the equation can be interpreted as a problem of a grounded electrical circuit with current sources (Chapter 4). In this thesis, I refer to $\boldsymbol{x}$ as the potential vector.

Since the method only involves a linear system, it is much faster and more stable than the Normalized Cut method. However, the partitions' quality may not be as good as the Normalized Cut method, depending on its grounding strategy (which node to be grounded).

2.4 Discretization

The Normalized Cut method and the Isoperimetric Partitioning give a continuous partition vector (Fiedler vector and potential vector). In order to obtain the partitions, the vectors need to be discretized.

In [12], Shi and Malik used α splitting points (thresholds) that discretize the Fiedler vector into α different partitions. Next, they choose the partition that gives the minimum *Ncut* value. The greater the α value, the better the partitions will be (lower *Ncut* value). Throughout this thesis, I used $\alpha = 100$.

In addition to the above discretization criterion, Grady and Schwartz [7] suggested other discretization criteria: median cut, jump cut and ratio cut. The median cut uses the median of the potential vector's entries as the threshold. This ensures that the partitions have equal number of nodes. For jump cut, we sort the potential vector's entries and find the largest difference between two entries in sequence. The entries before the two entries are grouped as one partition; whereas the entries after the two entries are grouped into another partition. The ratio cut is similar to the discretization techniques used in [12]. The difference is the partition is selected based on the lowest Isoperimetric constant, instead of the minimum *Ncut* value.

For the reason explained in the next section later, I used the minimum *Ncut* as the discretization criteria for all the experiments in this thesis.

2.5 The Quality of Partition and Segmentation

How do we tell if the segmentation of an image or the partition of a graph is good? Generally, we can judge the quality qualitatively or quantitatively.

The qualitative way is to use the judgment of our eye. For example, in image segmentation, our eye can tell if an object in an image is segmented out correctly. However, this may not work well in graph partitioning unless the nodes' values or edges' values are shown visually with spatial representation. Though a graph is presented visually, when a graph is large (many nodes) and complicated (many edges), our eye often cannot tell what the good partition should be.

Finally we resort to the quantitative way. However, we face the problem of how to measure a good partition. Wu and Leahy [14] suggested the minimum cut (*min cut*) criterion. The cut value measures the difference between the groups. However, the disadvantage of the *min cut* is it tends to cut out a small portion of a graph [12]. The two pieces are not balanced in size. In the extreme case, one of the partitions can consists of only a single node. For example, in Figure 2.1, the Minimum Cut method only segment out the source.

To overcome the imbalanced partitioning caused by the *min cut*, Shi and Malik [12] introduced a new criterion – Normalized Cuts (*Ncut*). They claim this criterion measures not only the difference between groups, but also the similarity within each group. Minimizing the *Ncut* value of a graph gives a balanced partition of the graph. However, this criterion is still not the precise criterion for good partition or segmentation. Consider the case of image segmentation in Figure 2.2.

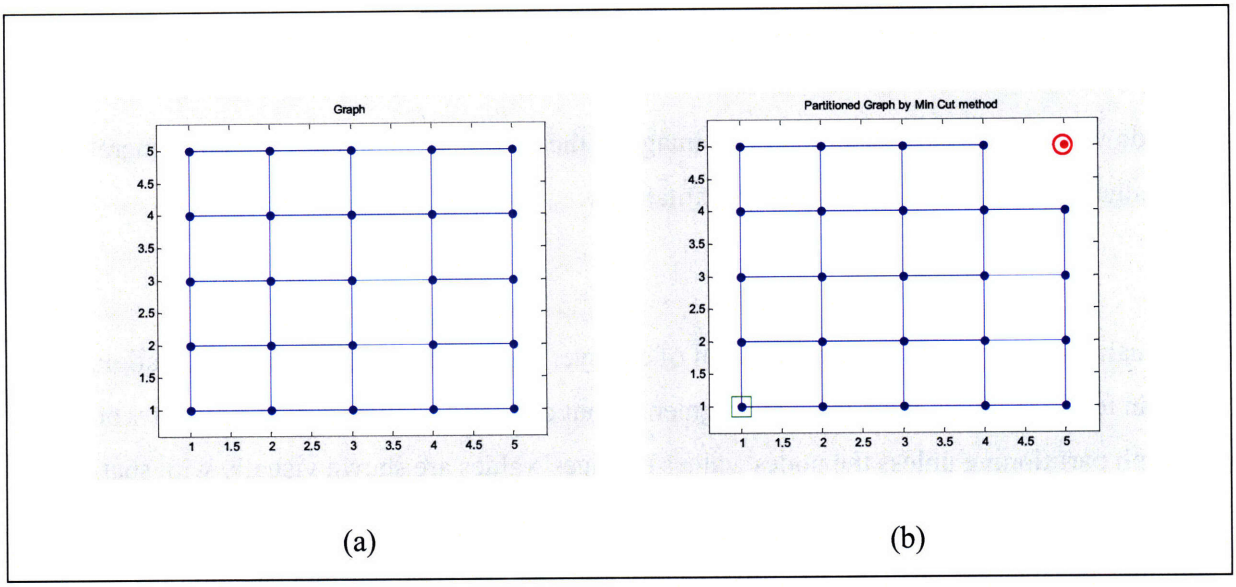


Figure 2.1 **Graph partitioning of a 25-node lattice unweighted graph using the Minimum Cut method. Figure (a) shows the graph before partitioning while Figure (b) shows the partitioned graph. The green square in (b) is the sink and the red circle in (b) is the source. Notice that the two partitions in (b) are not balanced. One of the partitions only contains a single node (source).**

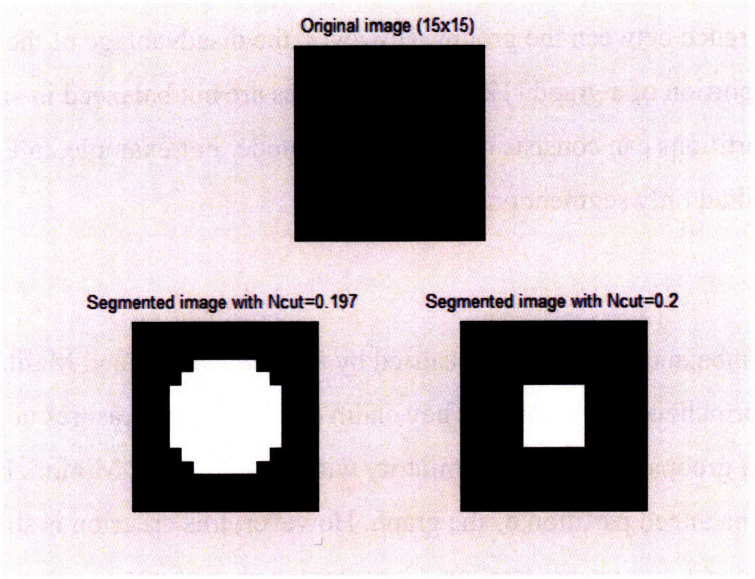


Figure 2.2 **The top image shows a dim solid square (5x5) in the middle of the image (15x15) with pixel intensity of 10, while the background pixel intensity is 0. The bottom left corner shows the segmented portion (white space) with the minimum *Ncut*, but the segmentation is incorrect; while the segmented image at bottom right has a larger *Ncut* value, but it is the correct segmentation. Obviously, this shows that minimizing the *Ncut* value does not necessarily give the correct segmentation.**

The first row of Figure 2.2 shows a 15x15 image with a 5x5 dim square in the center. The pixel intensity of the square is 10 while for the rest are 0. I constructed a 225-node lattice graph (similar to the graph in Figure 2.1) with each node representing a pixel in the image. I weighted the graph edges using the following function:

$$w = e^{-\left(\frac{|\Delta I|}{0.1*255}\right)^2}, \tag{2.9}$$

where w is the edge weight between two pixels and ΔI is the pixel intensity difference between the pixels. The second row of Figure 2.2 shows two different segmentations. The right segmentation is the correct segmentation. Though the left segmentation is incorrect, it has a smaller *Ncut* value. The results tell us that in certain cases, the *Ncut* criterion may not work. The criterion is not universal.

For the Isoperimetric Partitioning, Grady and Schwartz [7] propose the use of the combinatorial Isoperimetric constant. It is similar to the *Ncut*. Both *Ncut* and Isoperimetric constant not only measure the difference between partitions but also help to prevent imbalance between the partitions by dividing the cut value by the size of the partitions ($assoc(S_i, N)$ in the *Ncut*, Equation 2.3; $vol(S)$ in the Isoperimetric constant, Equation 2.6).

Comparing the *Ncut* and Isoperimetric constant, we prefer the *Ncut* because it can be extended to measure the partitions' quality of k-way graph partitioning (Equation 2.4) while the Isoperimetric constant can only measures the partition quality of bi-partitioning. Hence, throughout this thesis, I use the *Ncut* as the standard measurement of the partition's quality.

2.6 Unweighted Graph Partitioning

For unweighted graphs, the graph partitioning methods try to give partitions that cut through the least number of edges. Figure 2.3 shows an example. In the example, the graph partitioning method only cut through two links (two single links at the top left and bottom right of the graph).

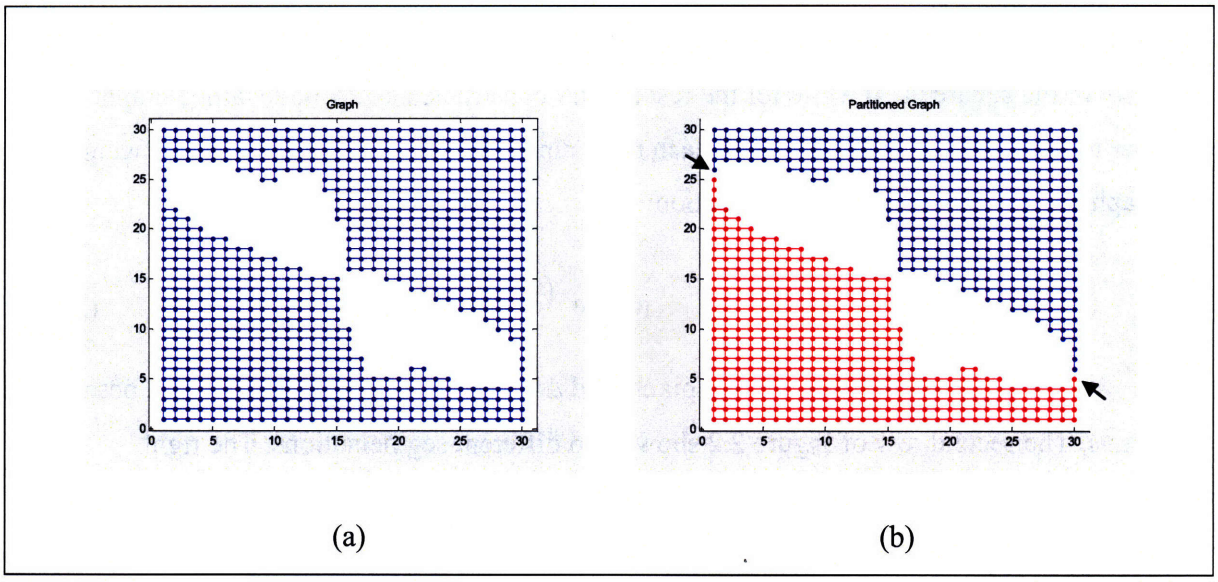


Figure 2.3 Graph Partitioning of a 'butterfly' graph generated by MATLAB function '*delsq*'. Figure (a) shows the original graph and Figure (b) shows the partitioned graph. Notice the single links at the top left and bottom right of the graphs. The graph partitioning methods (Normalized Cut method and Isoperimetric Partitioning) partitions the graph by cutting the single links (pointed by arrows).

The connectivity of the nodes in a graph is important in unweighted graph partitioning. Consider the example adapted from [13] in Figure 2.4. The graph is constructed such that there are more links connecting the nodes within the groups and fewer links between the groups. This means that the nodes have higher connectivity within the same groups than between the groups. As a result, the nodes at both ends of the bridging links between the two groups are not grouped together. They are grouped to the nodes that have higher connectivity with them (their initial group). The position of the nodes in the graph only reflects the connectivity. It does not affect the partitioning.

If there is no obvious single links or all the nodes have the same connectivity with each other, both the Normalized Cut method and Isoperimetric Partitioning method tries to partition the graph in a balanced way (The partitions are similar in size). An example of such a case is a 100-node square lattice graph as shown in Figure 2.5.

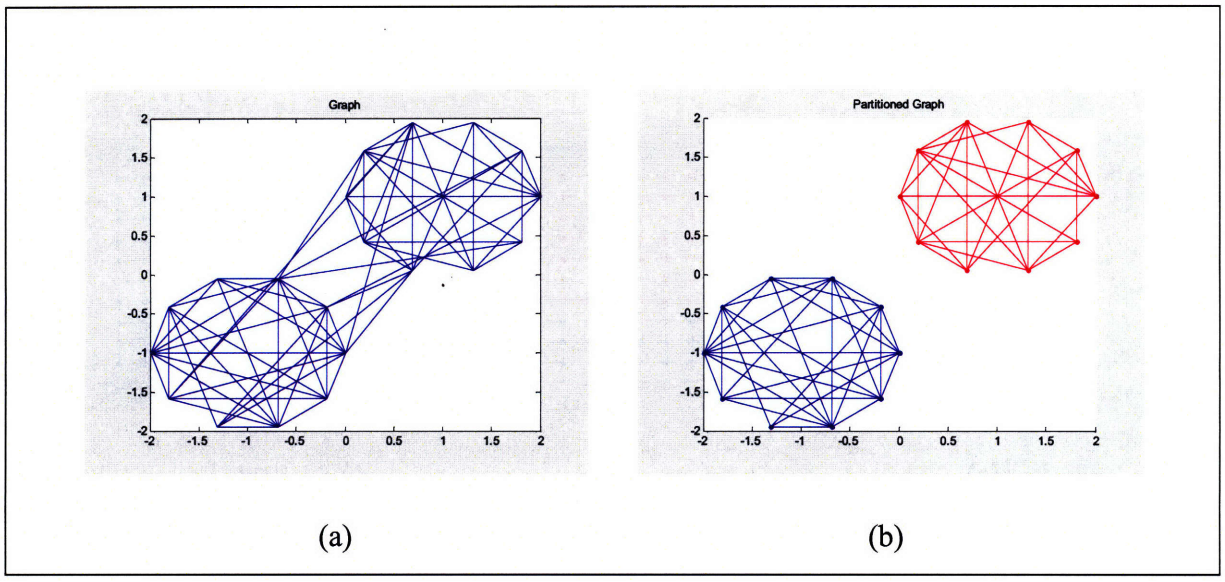


Figure 2.4 Graph partitioning of a graph according to the connectivity of nodes (adapted from [13]). Figure (a) shows the original graph and Figure (b) shows the partitioned graph. Notice that the nodes are more connected within the two groups than between the two groups. Hence, the two groups are separated by the graph partitioning methods. The position of the nodes in the graph only reflects the connectivity. It does not affect the partitioning.

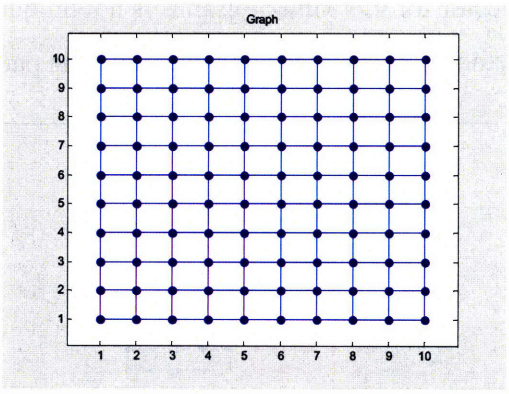


Figure 2.5 A 100-node square lattice unweighted graph. Notice that all the nodes (except the boundary nodes) have same connectivity.

The performance of the Isoperimetric Partitioning differs with different ground nodes. Figure 2.6 shows the partitions given by the method for of the 100-node square lattice graph shown in Figure 2.5. Using the same methods (Isoperimetric Partitioning) but different sink (ground) locations, the partitions vary with the sink (ground) location. Grady and Schwartz [7] suggested

two grounding strategies: random and maximum degree. However, these two strategies do not work well here because almost all the nodes in this graph have the same degree or connectivity.

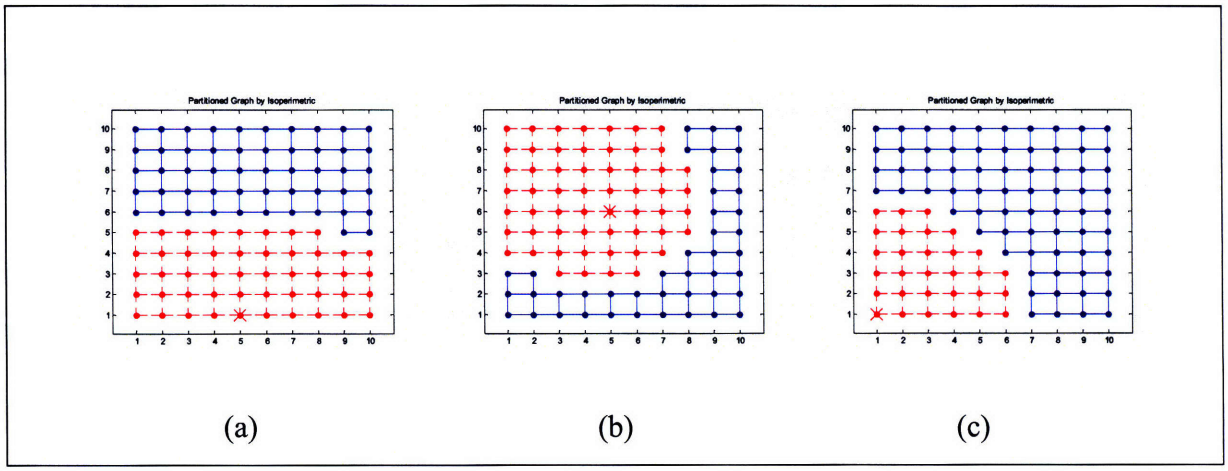


Figure 2.6 Unweighted graph partitioning by the Isoperimetric Partitioning. Figures (a), (b) and (c) shows the different partitions of the graph shown in Figure 2.5. The red 'X's in the three partitioned graphs are the sinks. Notice that the partitions vary, depending on the sink position.

The partitions given by the Normalized Cut can be unstable and differ from time to time. This happens when it tries to partition graphs with all the nodes having similar connectivity. The reason is because the eigenvector for this kind of graphs is not unique. Figure 2.7 shows an example. Using the same methods (Normalized Cut Method), the partitions vary at each time.

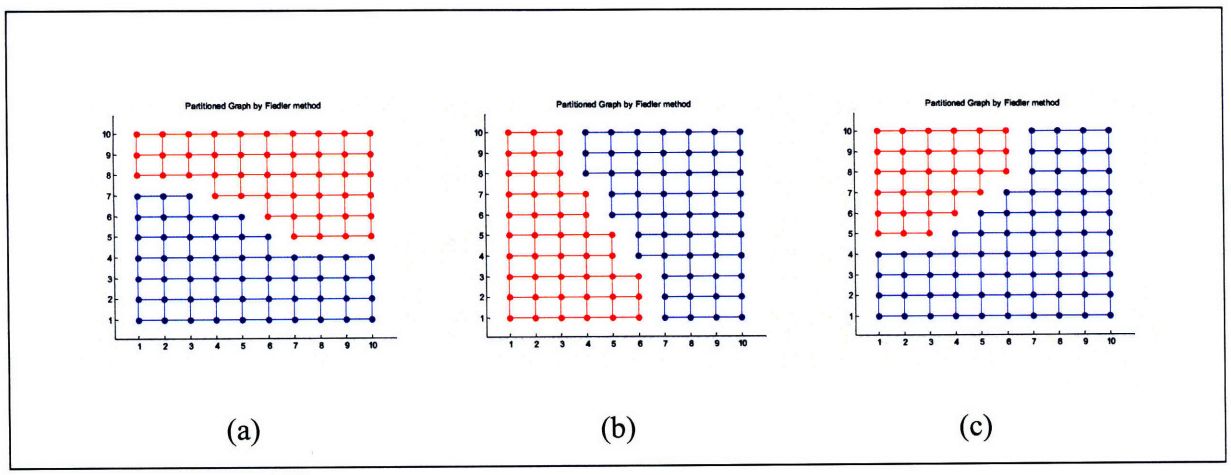


Figure 2.7 Unweighted graph partitioning by the Normalized Cut method. Figures (a), (b) and (c) shows the partitions of the graph shown in Figure 2.5. Notice that the partitions vary, even though the same graph partitioning (Normalized Cut method) is used.

2.7 Weighted Graph Partitioning

For weighted graph partitioning, apart from the connectivity, the graph partitioning methods also try to partition graphs along the links with lower weights (weak links). For illustration, I created a 2-weight graph shown in Figure 2.8. The graph edges' weights are either 0.5 or 1. The edges with 0.5 weights are the weak links. The graph partitioning methods cut through all the weak links as shown in (b) of Figure 2.8.

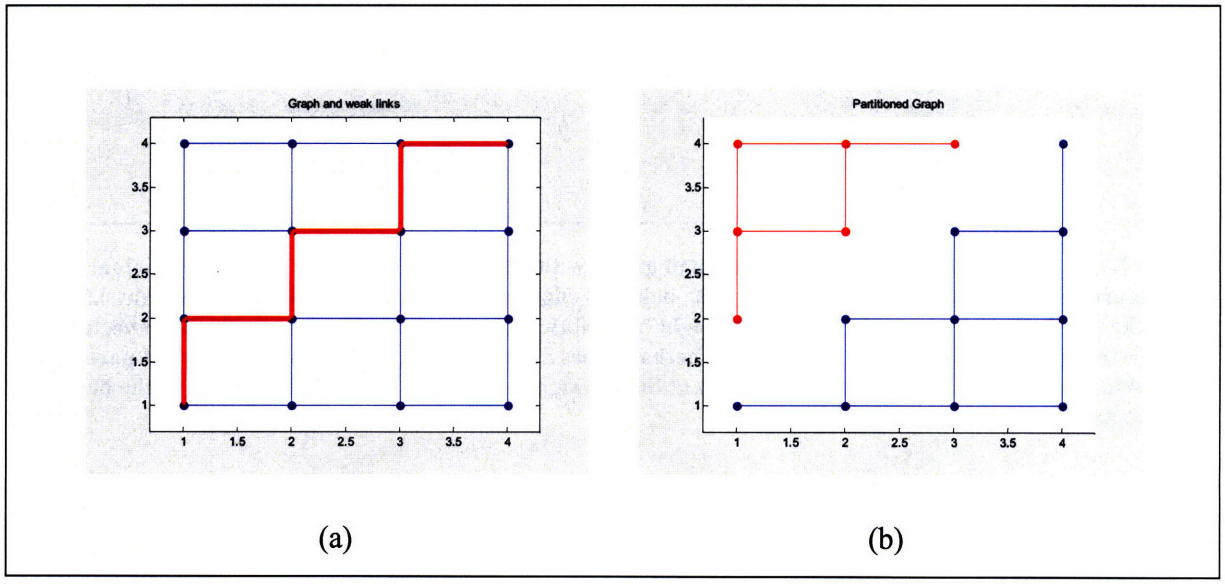


Figure 2.8 Graph partitioning of a weighted graph with weak links. Figure (a) shows the original graph and Figure (b) shows the partitioned graph. The bold red edges in (a) are the weak links with weight 0.5; while the thin blue edges in (a) has the edge weight of 1. The graph partitioning methods cut through all the weak links and partition the square graphs diagonally in (b).

The existence of weak links does not guarantee that the partitions will cut through all the weak links. If the weak links are scattered, some of the weak links will be ignored. Figure 2.9 shows an example. In the example, we see that the weak links are no longer well connected like the case in Figure 2.8. Hence, the three weak links at the bottom of the graph are ignored.

Generally, weighted graphs have more applications. One of the applications is in image segmentation. I shall explore more the weighted graph partitioning in Chapter 3, focusing only on image segmentation.

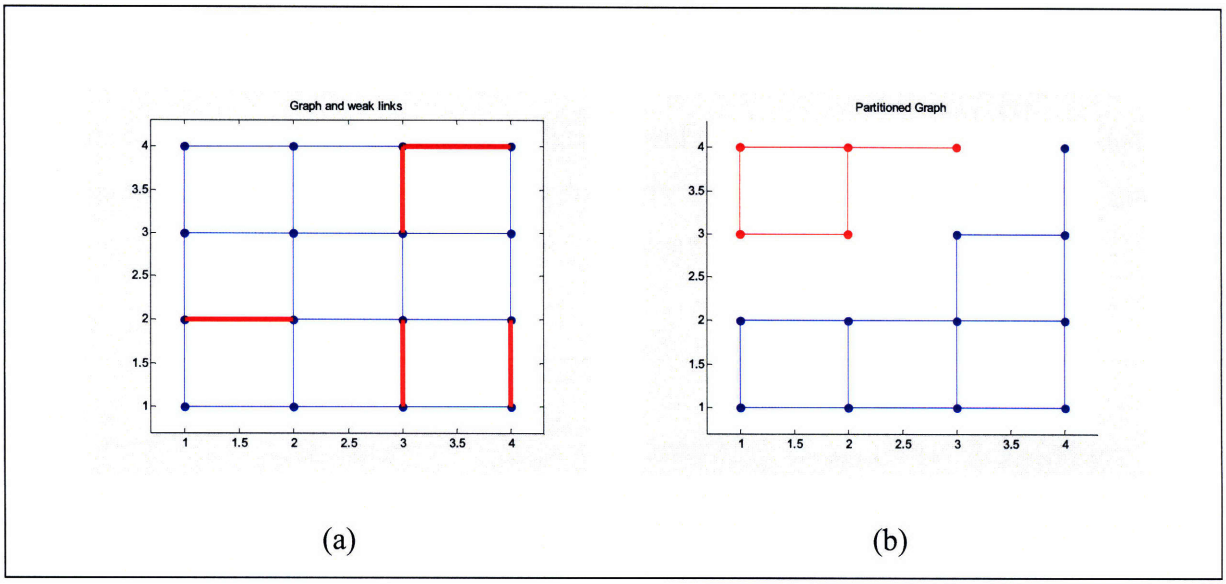


Figure 2.9 Graph partitioning of a weighted graph with weak links. Figure (a) shows the original graph and Figure (b) shows the partitioned graph. The bold red edges in (a) are the weak links with weight 0.5; while the thin blue edges in (a) has the edge weight of 1. Notice that the weak links are not well connected like the case in Figure 2.8. The graph partitioning methods does not cut through all the weak links and partition. The partitions only cut through two weak links at the top right of the graph. Three weak links at the bottom of the graph are ignored.

2.8 *k*-way Graph Partitioning

k-way graph partitioning partitions a graph into *k* partitions. However, for the Minimum Cut method and Isoperimetric Partitioning, they can only partition a graph into two partitions. To obtain *k* partitions, we can recursively apply the graph partitioning methods on the partitioned graph.

For the Normalized Cut method, the *k*-way graph partitioning can be achieved by both the recursive and simultaneous method. In the recursive method, we can recursively apply the Normalized Cut method on the partitioned graph. On the other hand, the simultaneous achieves the *k*-way graph partitioning by using the first *k* eigenvectors with the *k* smallest eigenvalues.

Depending on the type of graphs, the recursive and simultaneous Normalized cut method can gives different partitions. Figure 2.10 shows an example, in which both the recursive and simultaneous Normalized Cut method give same partitions. Figure 2.11 shows another example, in which both methods give slightly different partitions.

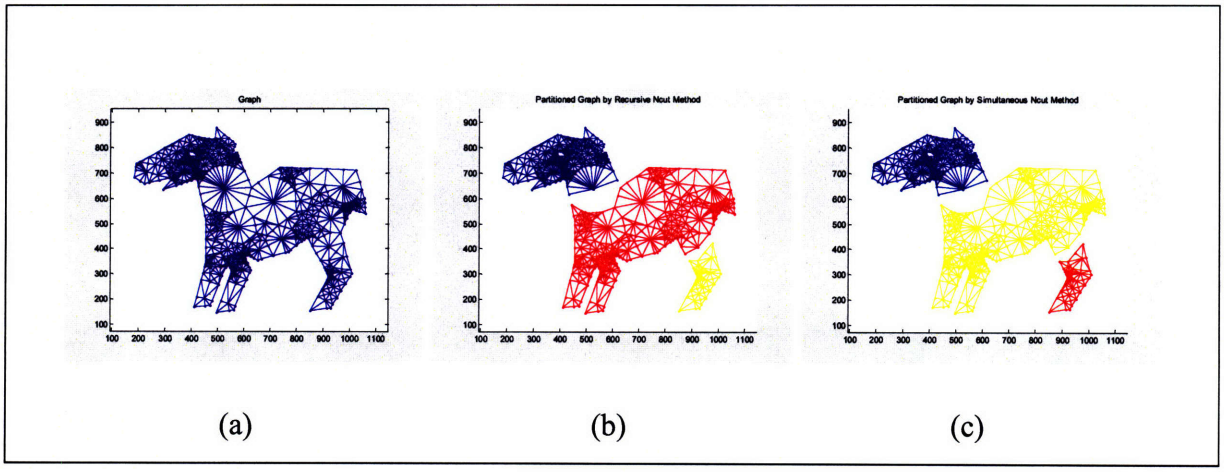


Figure 2.10 ***3*-way graph partitioning of an unweighted graph ('tapir' mesh[1]). Figure (a) shows the original graph. Figures (b) and (c) show the partitions given by the simultaneous and recursive methods. Both methods give same partitions.**

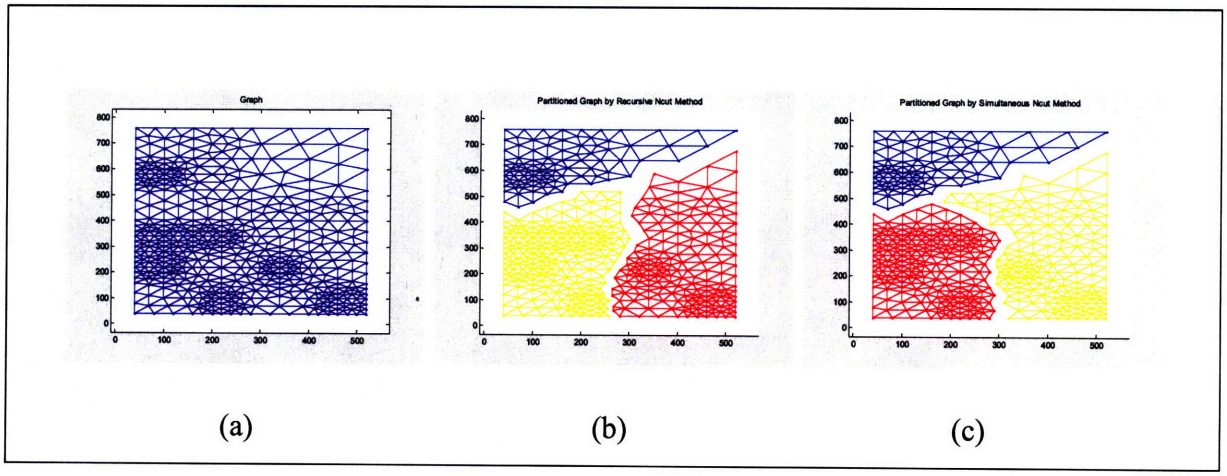


Figure 2.11 ***3*-way graph partitioning of an unweighted graph ('eppstein' mesh[1]). Figure (a) shows the original graph. Figures (b) and (c) show the partitions given by the simultaneous and recursive methods. The partitions in (b) and (c) differ slightly at the two bottom partitions.**

[1] Obtained from FTP site of John Gilbert and the Xerox Corporation at

ftp://ftp.parc.xerox.com/pub/gilbert/meshes.tar.Z

Chapter 3 Application of Graph Cut Techniques in Image Segmentation

An image segmentation problem can be formulated into a graph partitioning problem. Hence, we can use the graph partitioning algorithms described in Chapter 2 to solve image segmentation problems.

3.1 Graph Construction from an Image

An image is converted into a graph by treating the pixels of the image as nodes in a graph. Any two nodes (pixels) in the graph that are adjacent to each other are connected by an edge. The edges are weighted by weighting functions according to the pixel intensity difference between the two ends (nodes) of the edges. Figure 3.1 shows a simple example of the conversion of an image into a lattice-like graph.

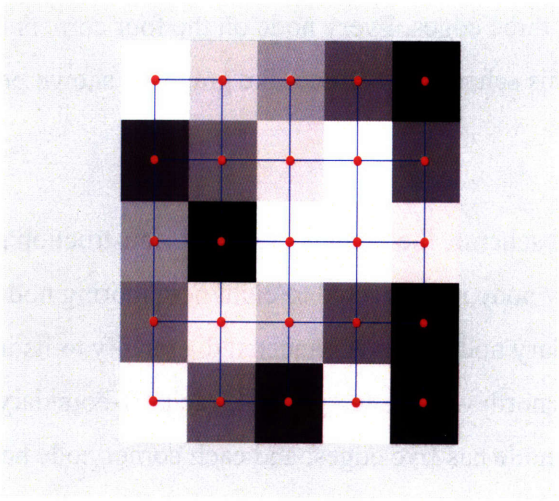


Figure 3.1 Construction of a graph with 25 nodes (red dots) and 40 edges (blue lines) from a 5x5 image. The image pixels' intensities are randomly generated. Each node of the graph represents a pixel of the image. The nodes are connected to their immediate neighbors only by horizontal and vertical edges.

3.1.1 Graph Edge Construction Schemes

Every pixel in an image can be represented by a node in a graph. In other words, the number of pixels in an image gives the number of nodes in the graph that represents the image. For example, an m x n image results in a graph with mn nodes. In the previous example in Figure 3.1, the 5x5 image is represented by a 25 nodes graph. Throughout this thesis, the numbering of the nodes of the graph constructed form images starts at the upper left corner and ascends horizontally. Though the number of nodes is fixed, the number of edges can vary. The number of edges in a graph representing an image depends on how the nodes in the graph are connected. According to [5], there are three ways to connect the nodes: the 4-connected, 8-connected and r-radially connected edge construction schemes.

For the 4-connected edge construction scheme, each node, except the nodes on the boundary of the graph, are connected to its immediate north, east, south and west nodes respectively by an edge. In other words, each off-boundary node has four edges. Each node on the boundary, except the four corner nodes, has three edges. Every node on the four corners has only two edges. The constructed graph under this scheme is a lattice-like graph, as shown previously in Figure 3.1.

Similar to the 4-connected scheme, the 8-connected edge construction scheme is named as such because each off-boundary node is connected to eight neighboring nodes. In addition to the four directions, each off-boundary node is also connected diagonally to its immediate north east, south east, south west and north west nodes. In total, each off-boundary node has eight edges; each off-corner boundary node has five edges; and each corner node has three edges. The constructed graph under this scheme is similar to the graph in the previous scheme, but with extra diagonal edges. Figure 3.2 shows an example of a graph constructed under this scheme.

Shi and Malik [12] introduced the r-radially connected edge construction scheme. Under this scheme, each node (as the center node) is connected to all its neighboring nodes located within a

diameter of r nodes from the center node. For example, if $r = 2$, then each node is not only connected to its immediate adjacent nodes (like the graph under the 8-node scheme), but is also connected to the nodes that are two nodes away from the center node. Under this definition, the 8-connected scheme is, in fact, a special case of the r- radially connected scheme with $r = 1$. Hence, from this point onwards, we shall group the 8-node scheme under the r-node edge construction scheme. Figure 3.3 shows an example of this scheme with $r = 2$.

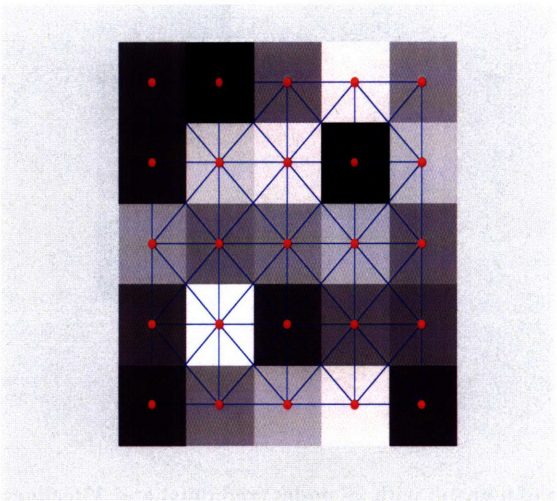


Figure 3.2 Construction of a graph with 25 nodes (red dots) and 72 edges (blue lines) from a 5x5 image using the 8-node edge construction scheme. Notice that each node is not only connected horizontally and vertically, but also diagonally to the neighboring nodes.

Comparing the 4-connected and r-radially connected edge construction schemes, the former is simpler than the latter in terms of graph construction because fewer edges are constructed in the graph. Hence, the coding effort for the 4-connected scheme is less. Furthermore, we can also expect a shorter run time for the graph construction process under the 4-connected scheme. In addition, the simple schemes also lead to more sparse Laplacian matrices. They can be more easily solved (linear system or eigensystem). For convenience, I refer the scheme as $r = 0$, though it is not under the r-radially connected scheme.

The advantage of the r-radially connected edge construction scheme is it allows the use of distance weighting functions besides the intensity weighting functions (Section 3.12). It also

makes the segmentation of non-connected objects possible. The graph partitioning performance of the Normalized Cut method increases with increased r value. Generally, it requires larger r value (typically $r > 5$). On the other hand, the Isoperimetric Partitioning only needs graphs with fewer edges (smaller r). Hence, we use different graph edge construction schemes for image segmentation based on different graph partitioning methods.

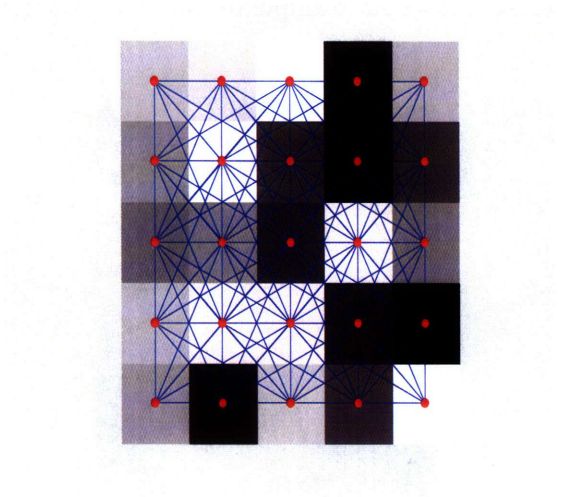


Figure 3.3 Construction of a graph with 25 nodes (red dots) and 336 edges (blue lines) from a 5x5 image using the *r*-node edge construction scheme with $r = 2$. In addition to the immediate neighboring nodes, each node (center node) is also connected to the nodes that are two nodes away from the center node.

Apart from the type of graph partitioning method, the type of image also affects the number of r we should use. For images with simple object, for example a simple white square in a black background, a simple $r = 0$ or $r = 1$ scheme is sufficient. For images with complicated objects, a larger r value may be needed.

3.1.2 Graph Edge Weighting Functions

Each edge in a graph that represents an image is weighted by weighting functions. The weighting functions used by Shi and Malik [12] for grayscale images (using only pixel intensity) are:

$$w_{ij} = \frac{255 - |\Delta I_{ij}|}{255}, \tag{3.1}$$

$$w_{ij} = e^{-\frac{|\Delta I_{ij}|}{\sigma_I * 255}}, \tag{3.2}$$

$$w_{ij} = e^{-\left(\frac{|\Delta I_{ij}|}{\sigma_I * 255}\right)^2}, \tag{3.3}$$

where w_{ij} is the edge weight between pixel (node) i and pixel (node) j; ΔI_{ij} is the pixel intensity difference between pixel (node) i and pixel (node) j; and σ_I in Equation (3.2) and (3.3) is an adjustable parameter. In [6], Grady and Jolly proposed another weighting function using a reciprocal function:

$$w_{ij} = \frac{1}{1 + \frac{|\Delta I_{ij}|}{\sigma_I * 255}}. \tag{3.4}$$

Among the four functions, the third function decays at the fastest rate and is followed by the second and forth functions. The first function has a very slow and linear decay rate (See Figure 3.4).

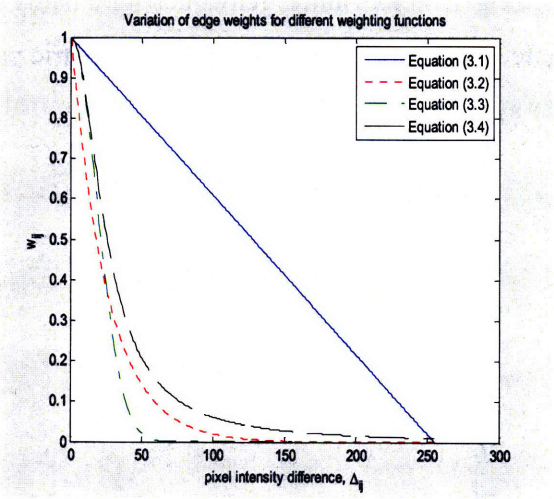


Figure 3.4 Variation of the edge weights with pixel intensity difference for different weighting functions. The four weighting functions are: Equation (3.1); Equation (3.2) with $\sigma_I = 0.1$; Equation (3.3) with $\sigma_I = 0.1$; and Equation (3.4) with $\sigma_I = 0.1$. The third function (green curve) has the fastest decay rate. It is followed by the second function (red curve) and forth function (black curve). The first function (blue curve) decays very slowly and the decay is linear.

The four weighting functions are all functions of the pixel intensity difference between two connecting pixels (nodes). This is because the objects in an image, which we want to segment out, often have a large difference in pixel intensity with the background of the image. Moreover, from Chapter 2, we know that the graph partitioning methods always try to partition a graph along weak edges, which have smaller edge weights. The existence of weak edges in a graph helps the graph partitioning methods to give better partitions. Hence, we weight the edges of the graph in such a way that the edges between the nodes in the objects and the nodes in the background are weak edges. The four functions serve this purpose well. However, the third function gives the best result as it is able to give smaller weights for relatively small pixel intensity differences (faster decay rate). As a result, the third function is used as the weighting function in the following sections.

Theoretically, decreasing the value of σ increases the decay rate (See Figure 3.5). However, there is a limit to the value of σ_I. Exceeding the limit will cause the resulting Laplacian matrix to be badly conditioned. This makes the graph partitioning methods such as the Normalized Cut method and Isoperimetric partitioning to fail. The ARPACK package used in the Normalized Cut method fails to give accurate eigenvectors and eigenvalues for a badly conditioned matrix. On the other hand, the Conjugate Gradient method used in Isoperimetric partitioning cannot solve the badly conditioned linear system accurately.

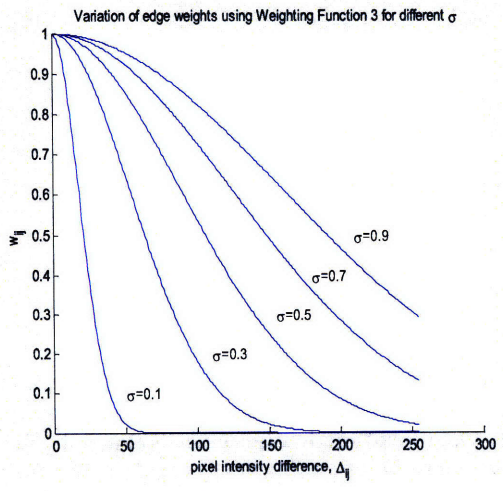


Figure 3.5 **The decay rate of the third weighting function increases with the decreasing σ.**

For graph constructed under *r*-radially connected edge construction scheme, we may want to include the information of distance in the weighting functions [12]. We want the adjacent nodes to have larger edge weight with the center node compared to the more remote nodes. Hence the weighting function should be as such: the edge weights between the center node and other connected nodes decrease when the connected nodes are further away from the center node. Shi and Malik [12] suggested that we can multiply the intensity weighting functions (Equation 3.1 to 3.4) with distance weighting functions. The distance weighting functions are similar to the intensity weighting functions except that the variables are changed from intensity difference to radial distance from the center node; and σ_I to σ_D. The combined weighting functions are:

$$w_{ij} = \frac{255-|\Delta I_{ij}|}{255} * (1 - \frac{d}{r}), \tag{3.5}$$

$$w_{ij} = e^{-\frac{|\Delta I_{ij}|}{\sigma_I * 255}} * e^{-\frac{d}{\sigma_D * r}}, \tag{3.6}$$

$$w_{ij} = e^{-\left(\frac{|\Delta I_{ij}|}{\sigma_I * 255}\right)^2} * e^{-\left(\frac{d}{\sigma_D * r}\right)^2}, \tag{3.7}$$

$$w_{ij} = \frac{1}{1+\frac{|\Delta I_{ij}|}{\sigma_I * 255}} * \frac{1}{1+\frac{d}{\sigma_D * r}}, \tag{3.8}$$

where d is the distance between the center nodes and connected nodes (measured in number of nodes); r is the radius defined previously in the *r*-radially connected edge construction scheme; and σ_D is the parameter similar to σ_I. The σ_D value cannot be set to too low or the edge weights between the center node and the remote connected node approach zero. This defeats the purpose of using the *r*-radially connected edge construction scheme.

3.2 Image Segmentation by Graph Partitioning Methods

After constructing a graph to represent an image, we can now solve the image segmentation problem as a graph partitioning problem by using the graph partitioning algorithms described in Chapter 2. They are the Minimum Cut method, Normalized Cut method, and Isoperimetric

partitioning. I also included the Spectral Rounding method, which is a variant of the Normalized Cut method.

3.2.1 Minimum Cut

For the Minimum Cut method, a pair of source and sink is needed. One of them should be placed in the to-be-segmented object and another should be placed at the area outside of the object. To illustrate this, we consider an example: a simple 15 x 15 image as shown in Figure 3.6. I use 4-connected edge construction scheme and weight the edges using Equation (3.4). The resulting graph is shown in Figure 3.6.

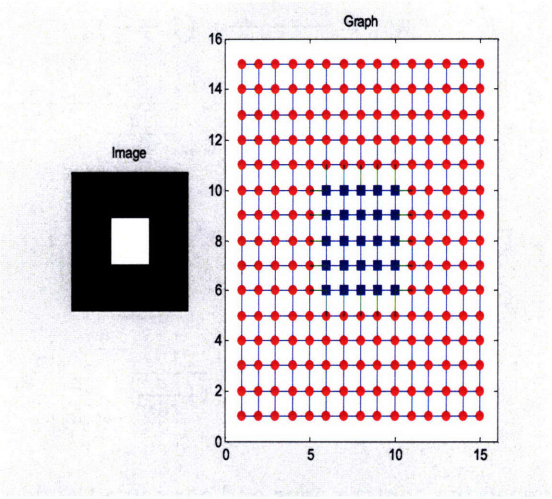


Figure 3.6 A 15x15 image (left) and its graph (right) constructed using the 4-connected scheme and weighted using the third weighting function. The blue nodes represent the pixels in the white (255) square while the red nodes represent the pixels in the black (0) background. The green edges are the weak edges (smaller edge weights due to the weighting function) connecting the nodes in the square to the nodes in the background.

To segment out the center white square, the source is located at the first node in the background while the sink is located in the square (See Figure 3.7). Since the minimum cut method tries to find a cut that minimizes the cut value, the existence of weak edges along the border of the image with the background prompt the method to cut through all the weak edges and subsequently segment out the object. I used the maximum flow solver developed by Boykov and Kolmogorov [2] to obtain the minimum cut.

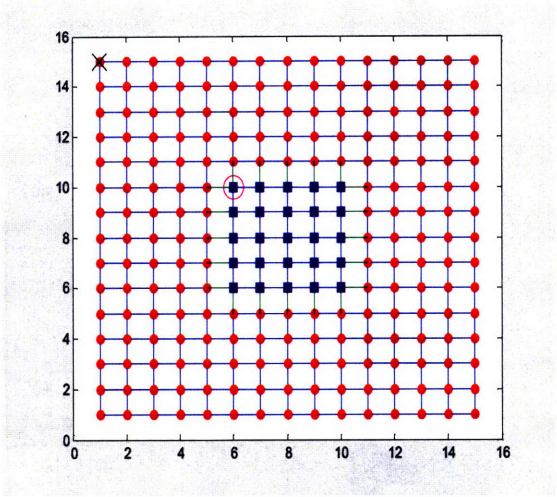


Figure 3.7 For the Minimum Cut method, the source (black 'X') is located in the background (red dots) while the sink (pink circle) is located in the square (blue dots).

3.2.2 Normalized Cut

In contrast with the Minimum Cut method, the Normalized Cut method does not require any source or sink. Hence, we can solve the problem directly by using the constructed graph to compute the Fiedler vector of the graph. The Fiedler vector gives an approximated solution to the normalized cut problem. In the discretization stage, I use 100 equally-spaced splitting points to search for the segmentation that gives the minimum Normalized Cut value.

3.2.3 Spectral Rounding

Extended from Shi and Malik's idea of Normalized Cut [12], Tolliver and Miller [3] introduce Spectral Rounding (SR). The main advantage of the method is that it directly produces discrete solutions (Figure 3.8). They also claimed that it produces smaller *Ncut* values and is less lightly to split an image in homogeneous regions.

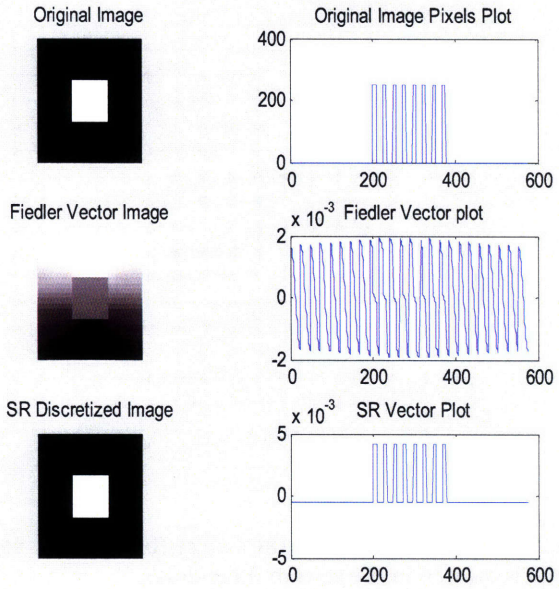


Figure 3.8 Row 1 shows the original image and its pixel intensity plot. Row 2 shows the image given by the Fiedler vector and the Fiedler vector plot. Row 3 shows the image given by the partition vector of SR and the vector plot. SR provides a discretized partition vector directly (Row 3) while Fiedler method gives a continuous partition vector (Row 2) which needs to be discretized in the discretization stage.

The idea of spectral rounding is based on the fact that a partitioned graph can be reflected on its decoupled Laplacian matrix. When a graph is cut into k partitions, the broken edges of the graph have zero weight. The resulting Laplacian matrix can be decoupled into k Laplacian sub matrices. Also, the first k eigenvalue of the matrix becomes zero and the k eigenvectors are discrete. Figure 3.9 shows an example of a connected graph and a disconnected graph and their Laplacian matrices, eigenvalues and second eigenvector. This observation sparks the idea of decreasing the weights of certain edges and the first k eigenvalues to split a graph into k pieces. We can achieve this by Spectral Rounding.

Spectral Rounding is an iterative method. To segment a graph into k pieces, the edge of the graph is reweighted. The new weights are given by reweighting functions based on the eigenvectors and eigenvalues of the current weighted graph. New eigenvectors and eigenvalues are computed based on the reweighted graph and will be used in the reweighting process of the next iteration. The new eigenvectors and eigenvalues can be calculated quickly using simple power method

because the new eigenvectors are close to the previous eigenvectors. The reweighting is done in such a way that it reduces the weights of certain edges (usually the edges with small weights) and the first k eigenvalues decrease after reweighting. In this study, I used a simple reweighting function named Inverse Fractional Reweighting suggested by Tolliver and Miller [3]. The iteration continues until certain edges are broken (having zero weights at the broken edges and zero first k eigenvalues). The graph is now cut into k pieces.

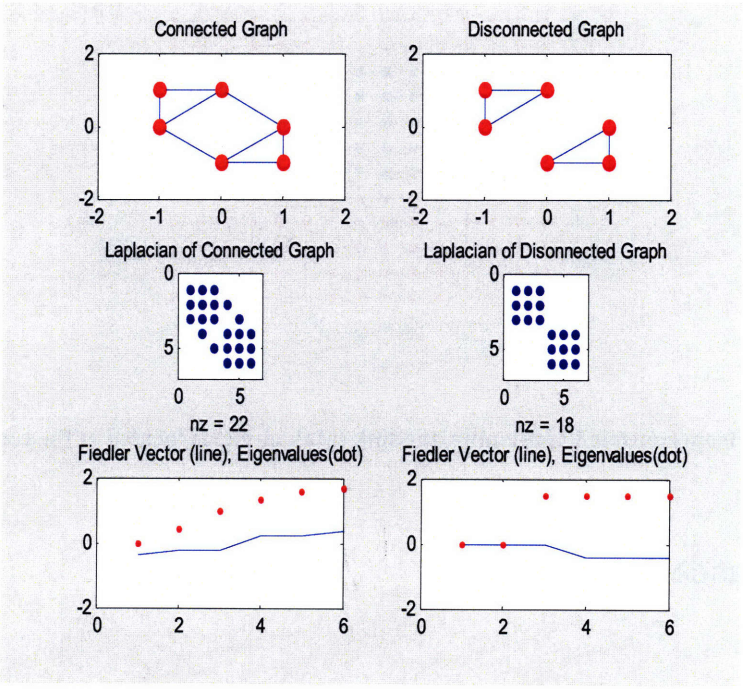


Figure 3.9 A connected graph (Row 1, Column 1) results in a coupled Laplacian matrix (Row 2, Column 1). Only the first eigenvalue is zero and the Fiedler Vector is continuous (Row 3, Column 1). A graph partitioned into two (Row 1, Column 2) results in two decoupled block Laplacian matrices (Row 2, Column 2). The first and second eigenvalues are zero and the Fiedler Vector is discrete (Row 3, Column 2).

3.2.4 Isoperimetric Partitioning

For Isoperimetric Partitioning, a sink (ground) is needed. According to [7], it should be placed at the node with maximum degree. In terms of image segmentation, the node should be placed at the homogeneous area in the image. In the following performance comparison study (Section 3.3), I choose to place the node inside the to-be-segmented object. For example, for the image shown in Figure 3.6, I place the node at the center of the image (See Figure 3.10). In the

discretization stage, I use 100 equally-spaced splitting points to search for the segmentation that gives the minimum Normalized Cut value. I use the minimum Normalized cut as the discretization criterion instead of the Isoperimetric constant for regularity and ease of comparison with other methods.

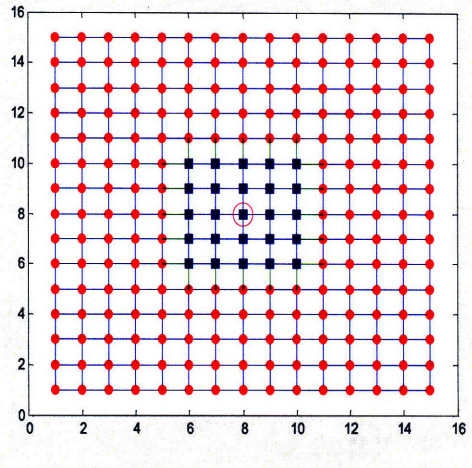


Figure 3.10 For Isoperimetric Partitioning, the sink (pink circle) is located at the center of the image or graph.

3.3 Performance Comparison

3.3.1 Setup

In order to test the image segmentation ability of the methods, I use the simple image shown in Figure 3.6 as the test image. In the 15 x 15 image, a 5 x 5 solid square is located at the center of the image. The image segmentation ability of each method will be tested by varying the intensity difference between the object and background in the image and the size of image. Apart from these factors, I also introduce different kind of noise into the image to test the methods' robustness towards noise.

For all the tests, unless specified, the image size is 15 x 15. The intensity of the white square (object) is 100 while the background is 0. This makes the intensity difference to be 100. For

graph construction, the 4-connected edge construction scheme is used for all the methods. This scheme is used because of its simplicity. Furthermore, I would like to compare the performance of the methods and thus regularity is needed. For edge weighting, the third weighting function is used, with σ_I set to 0.1.

I will measure the segmentation performance qualitatively by checking if the square in the test image is segmented out correctly. The quantitative way is to calculate the Normalized Cut value of the partitioned graph (image). Usually, correct segmentation has the minimum Normalized Cut value. However, I have shown that this criterion is not universal in Chapter 2. Nevertheless, we shall assume that the Normalized Cut measures the partitions' quality precisely. Apart from the partitions' quality, the speed performance of each method is measured by recording the run time for each method.

3.3.2 Pixel Intensity

In this test, I would like to test the sensitivity of the method to the pixel intensity difference. I varied the pixel intensity of the solid square and fixed the pixel intensity of the background to zero (See Figure 3.6, page 58). Smaller pixel intensity in the square increases the pixel intensity difference between the square and the background, and subsequently makes the image segmentation more difficult. Therefore, to test the limitation of the methods, I decrease the pixel intensity in the square until the segmentation fails.

The segmentation results of the Minimum Cut method are shown in Figure 3.11 (page 66). The pixel intensity of the square in the image decreases from 20 to 1, and the square becomes less distinguishable from its background (Row 1 to 3). The center square (Row 1, (a)) is still distinguishable for the pixel intensity of 20. When the pixel intensity drops to 10 (Row 2, (a)), the center square is hardly distinguished from its background. The center square disappears

(Row 3, (a)) when the difference in intensity is just 1. However, the method still segments out the center square successfully as shown in Column (b).

Quantitatively, the Normalized Cut value increases when the pixel intensity in the square decreases. This is because decreasing the intensity in the square increases the weights of the weak edges along the border of the square. The image segmentation results are surprising. In Chapter 2, I have shown that the disadvantage of the Minimum Cut is that it tends to cut out a small portion of a graph. I expected the Minimum Cut method to perform poorly. The reason for this good performance (ability to segment the square with just one pixel intensity difference) is the weighting function used. The weighting function results in weak edges with very small edge weights and thus, the Minimum Cut method is able to cut out the square along the weak edges.

The segmentation results of the Normalized Cut method is shown in Figure 3.12 (page 67). The critical pixel intensity difference for the Normalized Cut method is 40. With this pixel intensity, the method splits the image diagonally into two (See Row 3, Column (d)). The reason for this failure is that a Fiedler vector is just an approximated solution to the Normalized Cut problem. The approximation becomes less accurate when the intensity difference decreases. This effect can be seen clearly in the Fiedler vector plots in Column (b): the Fiedler vector plot changes from a discrete plot (Row 1) to a more continuous plot (Row 2), and finally the plot shows few discrete parts (Row 3) when the method fails.

The segmentation results of the Spectral Rounding method is shown in Figure 3.13 (page 68). The critical pixel intensity difference for the Spectral Rounding method is 40 (similar to the Normalized Cut method). With this pixel intensity, the method splits the image into half (See Row 3, Column (d)). The reason for this failure is that though the Spectral Rounding method gives discrete solution directly, it is still based on the Normalized Cut method and uses the Fiedler vector as the starting vector. From previous discussion, we know that the Fiedler vector is just an approximated solution to the Normalized Cut problem. Hence, if the Fiedler vector is far from the discrete solution to the Normalized Cut problem, the Spectral Rounding will fail as well. The approximation becomes less accurate when the intensity difference decreases. This

affects the Fielder vector and subsequently affects the segmentation given by the Spectral Rounding method.

The segmentation results of Isoperimetric Partitioning are shown in Figure 3.14 (page 69). The pixel intensity difference for the method to fail is 10. The image in Row 3, Column (d) shows that the method segments out a circle with the source at the center when the pixel intensity difference is 10. The circular segmentation is observed because the entry of the potential vector is minimum (0) at the sink and decreases evenly in all directions away from the source. The entry of the potential vector drops sharply across the weak edges. However, when the difference in pixel intensity between the square and the background becomes smaller, the weak edges gain more weights. Hence, the drop in the potential vector entries across the weak edges becomes insignificant and cannot be detected during the discretization stage. We can observe this decrease in the potential vector plots in Column (b): the first plot has a large and distinct sudden drop; the second plot has a reduced sudden drop; and the last plot has no sudden drop. The sudden drop corresponds to the border of the square. Hence, when the sudden drop vanishes, the border of the square also becomes undistinguishable as shown in the last row of Column (c).

In this test, I have tested the image segmentation performance of the four methods by varying the intensity difference between the object and the background in the test image. Generally, the performance of the last three methods decreases when we decrease the pixel intensity difference. Table 3.1 gives the critical pixel intensity differences, that is, the difference at which the image segmentation fails, for the four image segmentation methods.

Table 3.1 **Critical pixel intensity difference for different image segmentation methods in segmenting the 15 x 15 test image.**

Methods	Critical pixel intensity difference
Minimum Cut	0
Normalized Cut	40
Spectral Rounding	40
Isoperimetric Partitioning	10

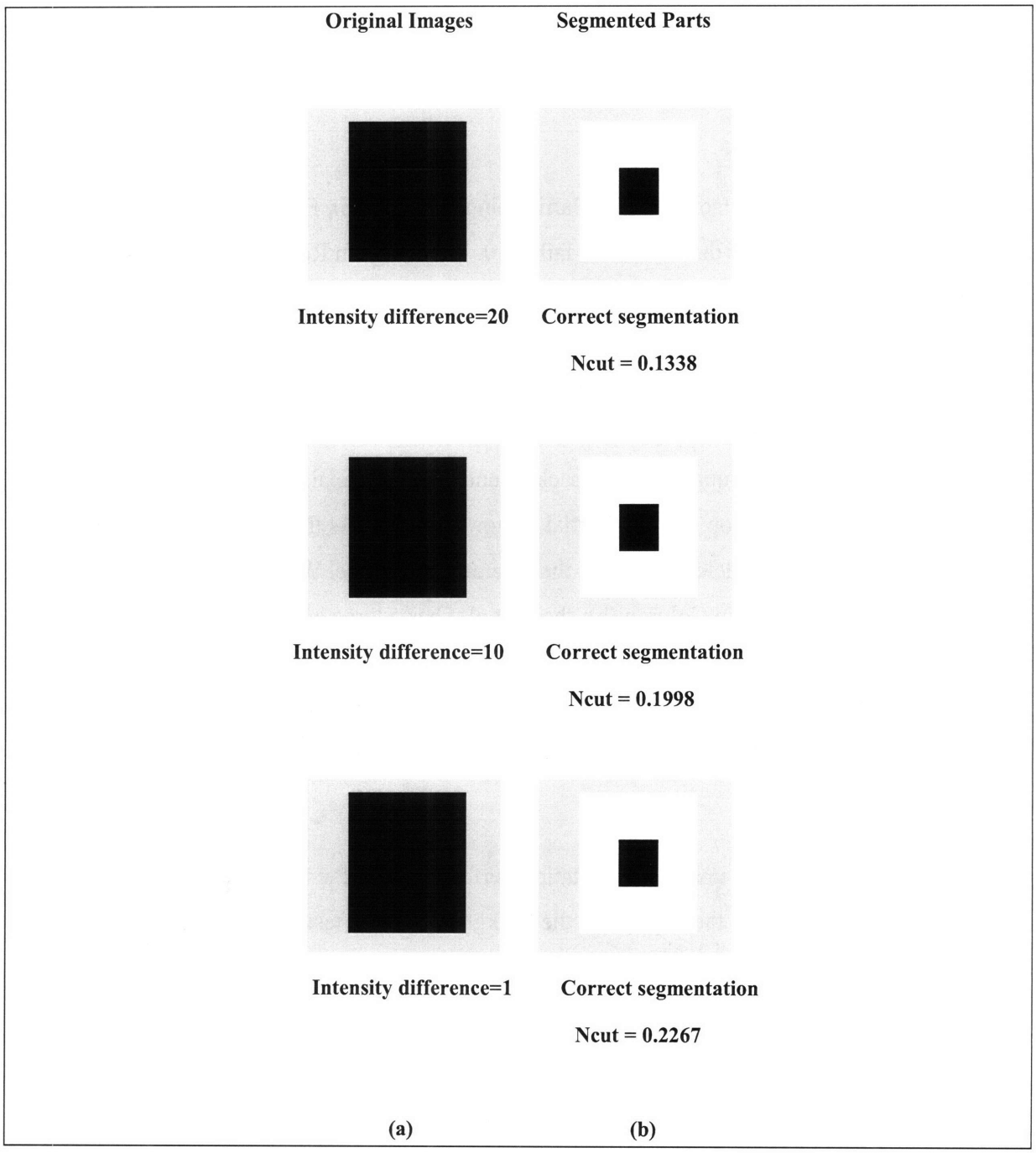


Figure 3.11 Image Segmentation by the Minimum Cut method for different pixel intensities. Column (a) shows the original images before segmentation with the pixel intensity differences between the square and the background stated at the bottom of the images. Column (b) shows the segmented parts from the images (the squares). The *Ncut* values are stated at the bottom of each image in Column (b). The pixel intensity of the square decreases from 20 to 1 (Row 1 to 3). The center square (Row 1, (a)) is distinguishable for the pixel intensity of 20. When the pixel intensity drops to 10 (Row 2, (a)), the center square is hardly distinguished from its background. The center square disappears (Row 3, (a)) when the difference in intensity is just 1. However, the method still segments out the center square successfully as shown in Column (b). Notice also that the *Ncut* value increases with decreasing pixel intensity difference.

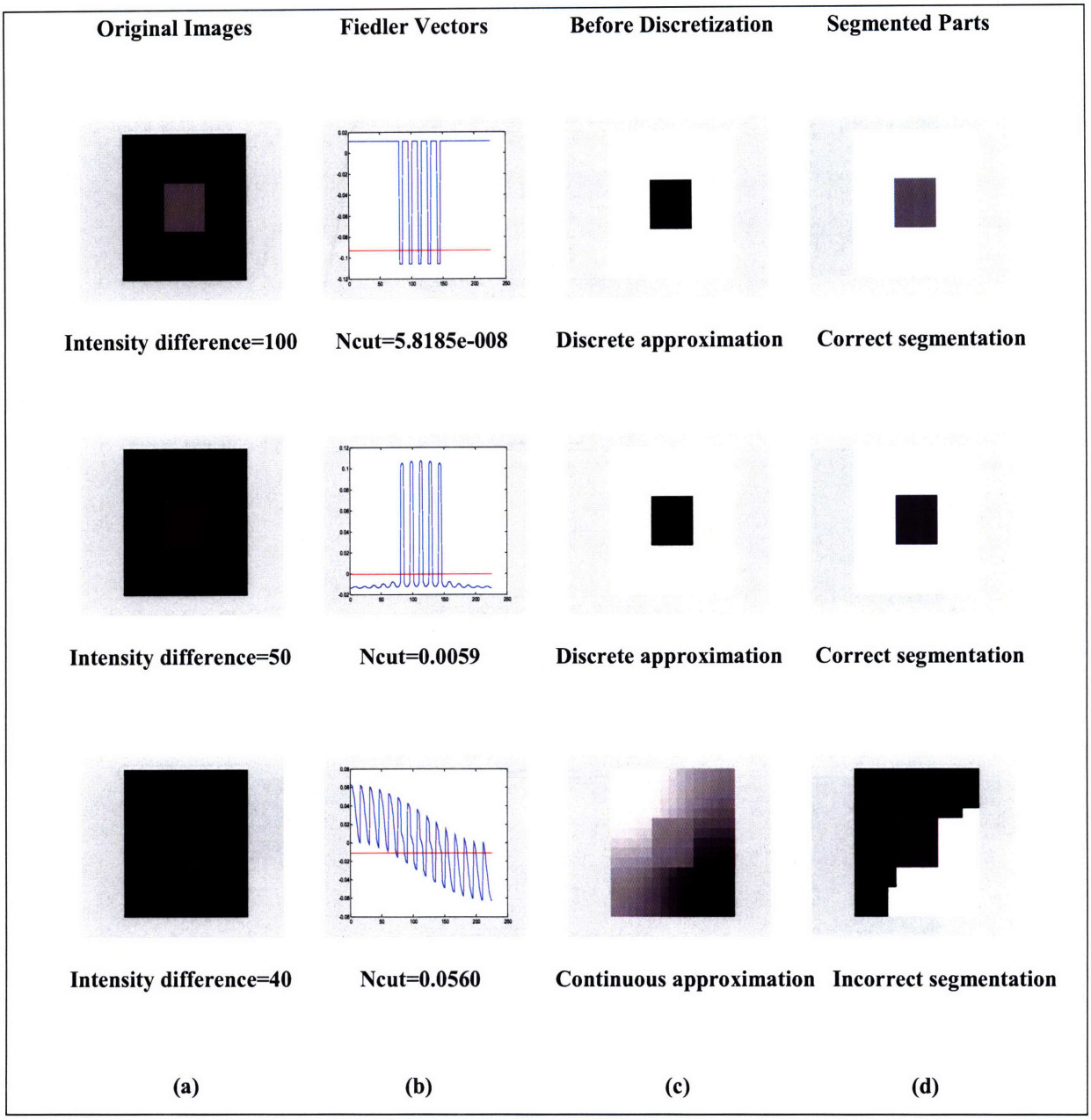


Figure 3.12 **Image Segmentation by the Normalized Cut method. Column (a) shows the original images before segmentation and the pixel intensities of the square. Column (b) shows the Fiedler vectors (blue curve) and the splitting points that give the minimum *Ncut* value (red horizontal line). The *Ncut* values are given at the bottom of each plot in column (b). Column (c) shows the images given by the Fiedler vector before discretization. Column (d) shows the segmented part from the image (the square). The method fails to segments out the center square when the pixel intensity of the square is 40 (Row 3). For pixel intensity above 40, the method performs the segmentation correctly. The continuous state of the Fiedler vector is also reflected in its image plot before discretization. From the image in the last row of Column (c), we can observe that the pixel intensity varies continuously from the upper left corner to the lower right corner (from bright to dark). Notice the *Ncut* value of the last row, it is higher than the *Ncut* value of the correct segmentation (0.0233). Incorrect segmentation gives higher *Ncut* value.**

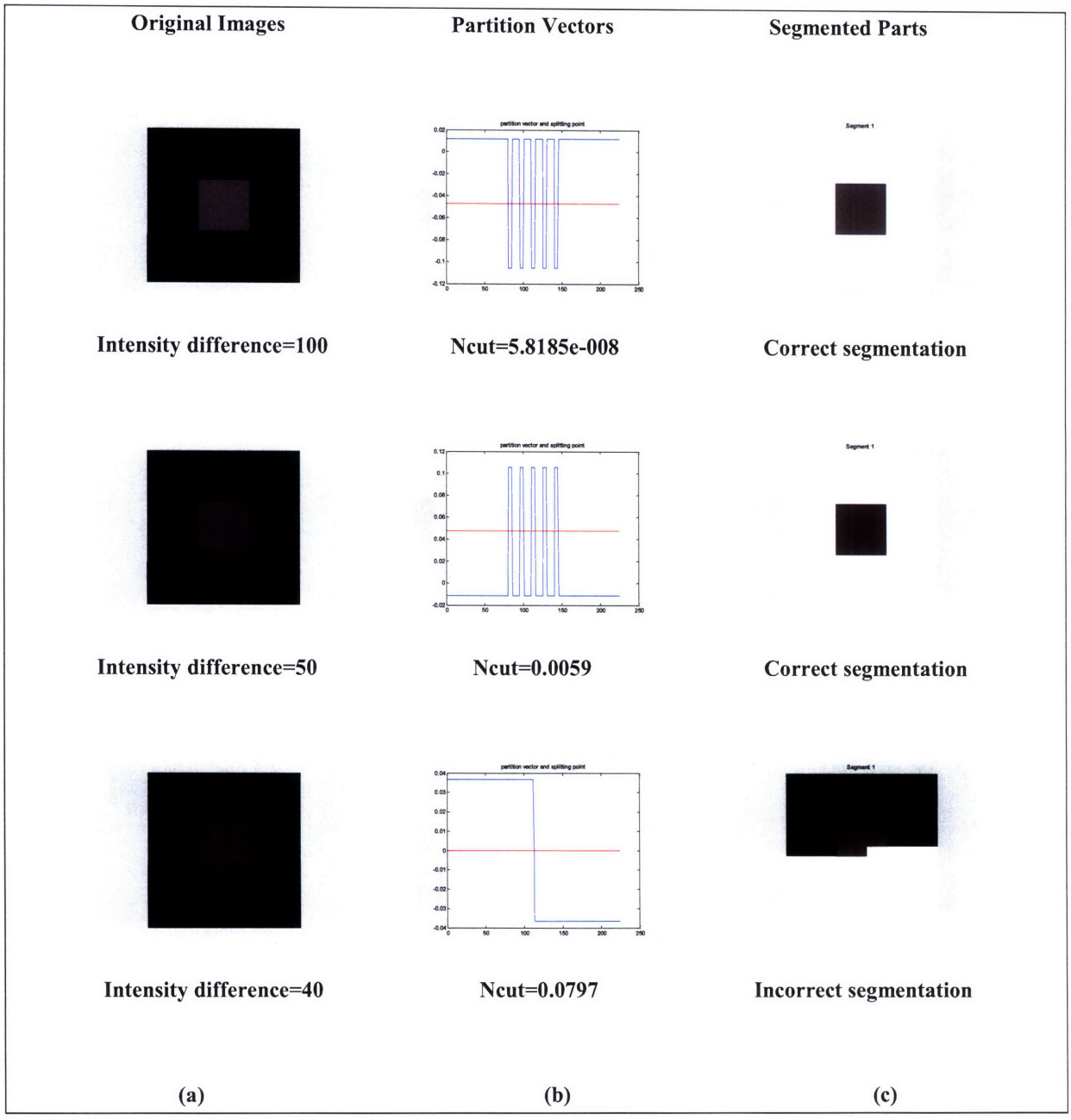


Figure 3.13 **Image Segmentation by the Spectral Rounding method. Column (a) shows the original images before segmentation and the pixel intensities of the square. Column (b) shows the partition vectors (blue curve) and the splitting points (red horizontal line). The *Ncut* values are given at the bottom of each plot in column (b). Comparing the partition vector plots with the Fiedler vector plot in Figure 3.12, we see that the vector plots by SR are discrete (binary). Column (c) shows the segmented part from the image (the square). The method fails to segments out the center square when the pixel intensity of the square is 40 (Row 3). Though the vector plot shows discrete values, but the segmentation is still incorrect. This is because the SR method starts with a continuous Fiedler vector (shown in the row 3 of Figure 3.12), an approximation which is far from the discrete solution of the Normalized Cut problem. For pixel intensity above 40, the method performs the segmentation correctly. Notice the *Ncut* value of the last row, it is higher than the *Ncut* value of the correct segmentation (0.0233). Incorrect segmentation gives higher *Ncut* value.**

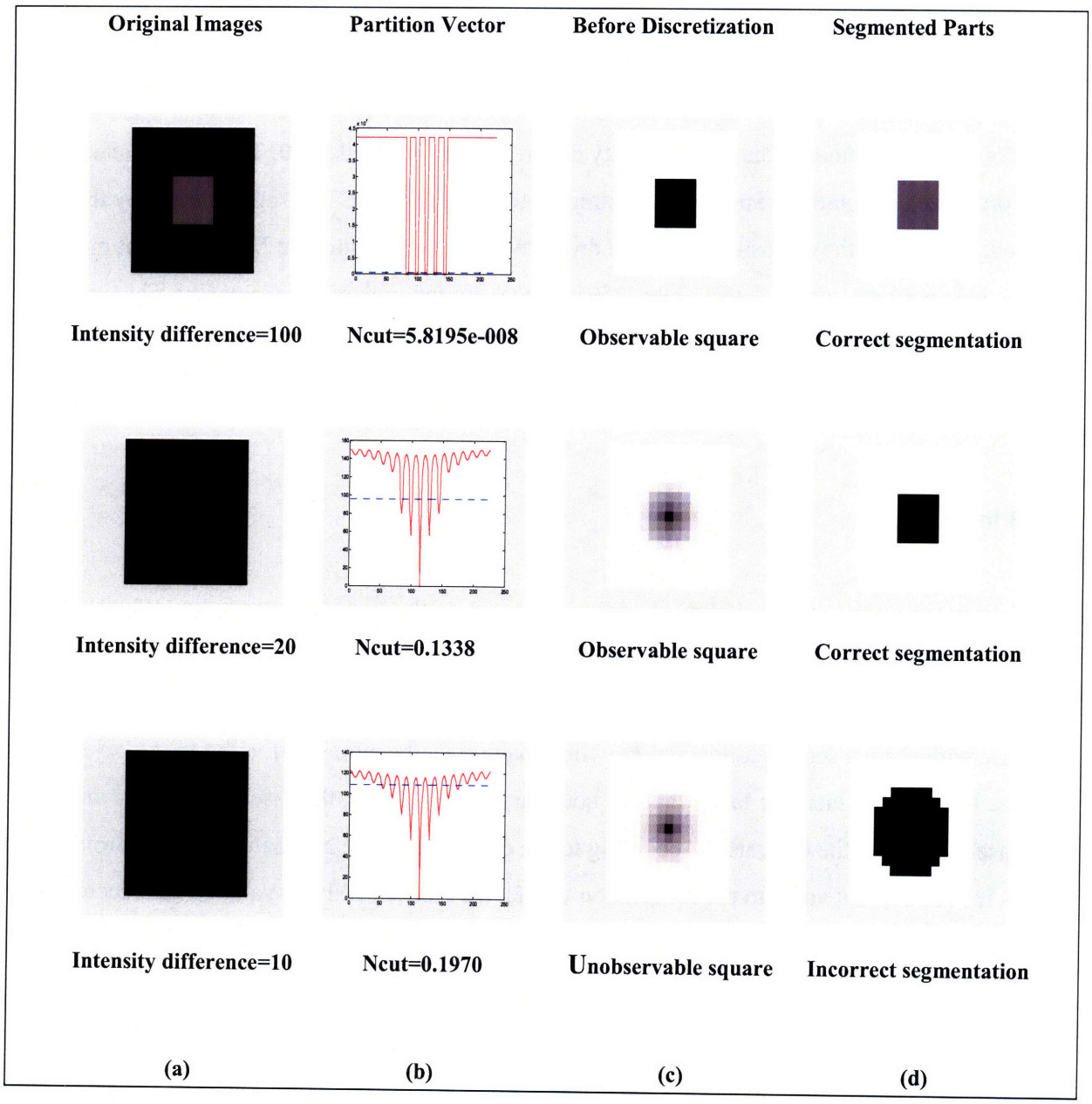


Figure 3.14 Image Segmentation by the Isoperimetric Partitioning. Column (a) shows the original images before segmentation and the pixel intensities of the square. Column (b) shows the partition vector (red curve) and the splitting point that gives the minimum *Ncut* value (blue horizontal dashed lines). The *Ncut* values are given at the bottom of each plot in column (b). Column (c) shows the images given by the isoperimetric solutions before discretization. Column (d) shows the segmented parts from the images (the square). The method fails to segment out the center square when the pixel intensity of the square is 10 (Row 3). For pixel intensity differences above 10, the method performs the segmentation correctly as the weak edges are weak enough to be detected. Notice that when the square is still observable in Column (c), the center square can be segmented out in Column (d). Also notice the *Ncut* value of the last row. It is interestingly smaller than the *Ncut* value of the correct segmentation (0.1998). The *Ncut* is not the absolute measurement for correct segmentation.

As can be seen in Table 3.1 (page 65), the image segmentation performance of the Minimum Cut method is the best. The method is able to segment out the square even when the pixel intensity difference between the square and the background is only one. Isoperimetric Partitioning fails to give correct segmentation when the intensity difference has dropped to 10. The performance of the Normalized Cut and the Spectral Rounding method is the worst. The segmentation by these methods fails when the intensity difference drops to 40. In conclusion, the Minimum Cut method has the best performance to detect objects form a background with similar intensity. This is followed by Isoperimetric partitioning. The Normalized Cut and Spectral Rounding methods perform poorly compared to others.

3.3.3 Image Size

In this test, we want to know the effect of the image size to the image segmentation. We changed the size of the test image but keep the size proportion of the square and the background. We expect the *Ncut* values to decrease with the image size for correct segmentation. This is so because with correct segmentation and growing image size, the number of nodes in the object and the background increase faster that the boundary nodes. Hence, the association value will increase faster than the cut value. According to the definition of the Normalized Cut (Normalized Cut is the ratio of cut value to the association value), the Normalized Cut value should decreases with the image size.

The segmentation results of the Minimum Cut method for various image sizes are shown in Figure 3.15 (page 73). The image size increases from 15 x15 to 300 x300. The method segments out the center square successfully for all the cases.

From the test, the Minimum Cut method seems not affected by the size of the image. This is again because of the chosen weighting function and parameter; and the intensity difference. These factors caused the edges weights between the object and background small enough to be easily detected by the method.

For illustration purpose, we calculate the edge weight between a node in the object and a node in the background. Given the pixel intensity difference between the object and the background is 100, the edge weight calculated using the third weighting function is 2.094e-7. The edges between the nodes in the background or the objects have weights of 1. The usual failure that the Minimum Cut faces is it singles out the sink or source alone during the segmentation (imbalanced segmentation). Consider that we placed the sink and source at the positions shown in Figure 3.7 (page 59), the degree (sum of weights around a node) of sink and source are 2+2*2.094e-7 and 2, respectively. In the case of 300 x 300 image shown in Figure 3.15 (page 73), the sum of weights of all the edges along the boundary is 8.379e-5, which is far smaller than the degree of the sink and source. In order to exceed the value of 2, the size of the object should have the size of 2/2.094e-7/4 = 2.388e6, which is a large number. In other words, the method will never fail to segment out the square from the test image, regardless of the size and the pixel intensity difference.

The segmentation results of the Normalized Cut method for different image size are shown in Figure 3.16 (page 74). The critical image size for the Normalized Cut method is 255 x 255. With this image size, the method splits the image diagonally into two (See Row 3, Column (c) of Figure 3.16, page 74). The reason for this failure is again due to the fact that the Fiedler vector is just an approximated solution to the Normalized Cut problem. The approximation becomes less accurate when the image size increases.

The segmentation results of the Spectral Rounding method is shown in Figure 3.17 (page75). Unlike the Normalized Cut method, for all the image size up to 900 x 900, Spectral Rounding method performs the segmentation correctly. This shows that the Spectral Rounding (modified from the Normalized Cut method) is able to improve the Normalized Cut method by reweighting the edge weights.

The segmentation results of Isoperimetric Partitioning are shown in Figure 3.18 (page 76). We observed that the method is successful in segmenting the test image even if the image size reaches 900 x 900. However, the pixel intensity difference for this case is 100. Based on the observation in Section 3.3.2, we should expect the image size limit to decrease when the pixel intensity decreases.

Table 3.2 summarizes the image segmentation performance of the different methods in image size test. With the pixel intensity difference of 100, the Minimum Cut, Spectral Rounding and Isoperimetric partitioning does not have any image size limit. The methods are said to have no image size limit based on the fact that they can segment the test image with size up to 900 x 900. The Normalized Cut method fails at the image size of 255 x 255. In conclusion, the Normalized Cut method has the worst performance in segmenting large images compared to others. The other methods are not affected by the image size.

Table 3.2 **Critical image size for different image segmentation methods in segmenting the test image with pixel intensity difference of 100.**

Methods	Critical Image Size
Minimum Cut	No Limit
Normalized Cut	255 x 255
Spectral Rounding	No Limit
Isoperimetric Partitioning	No Limit

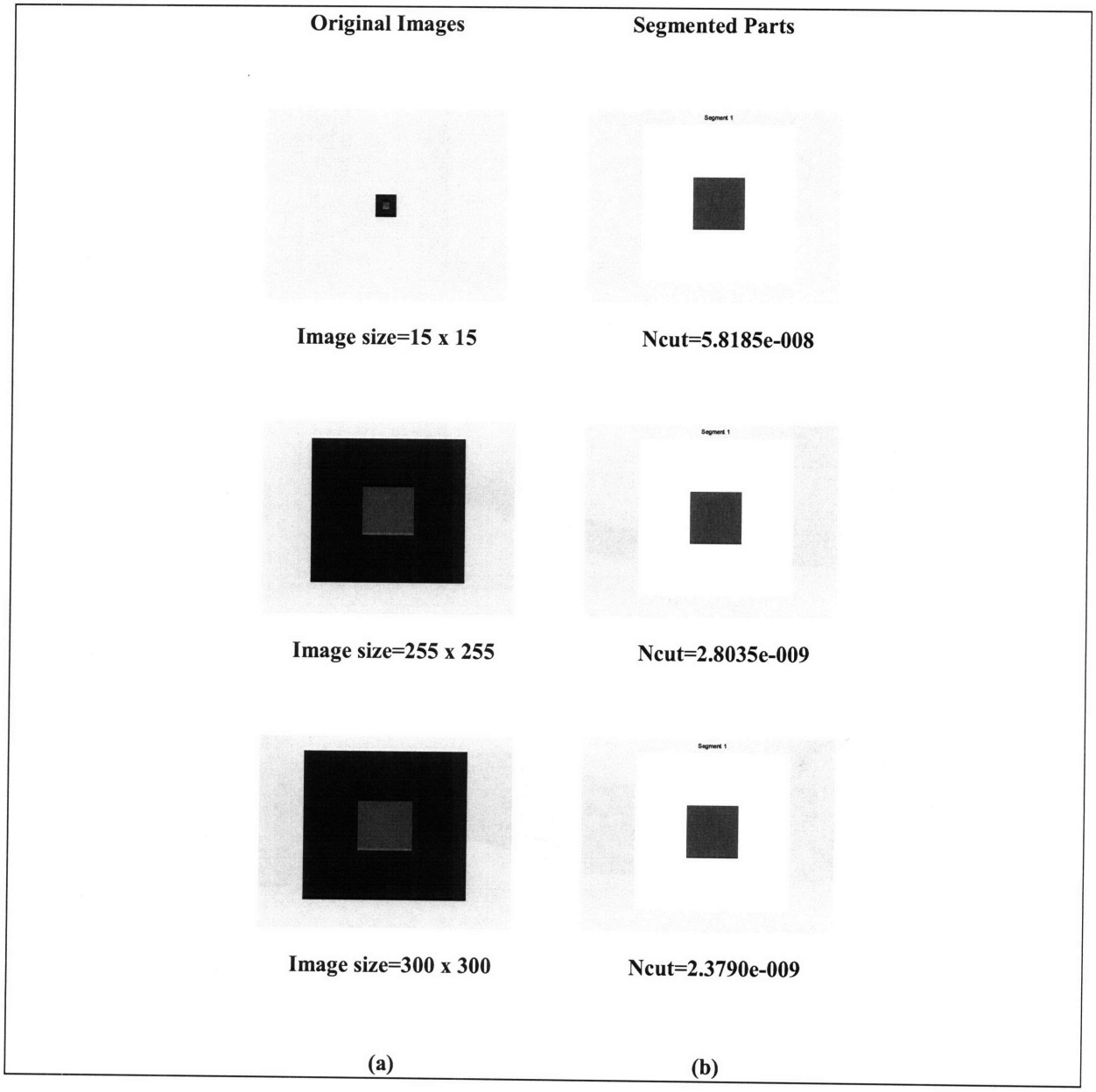


Figure 3.15 **Image Segmentation by the Minimum Cut method for different image size. Column (a) shows the original images before segmentation with the image size stated at the bottom of the images. Column (b) shows the segmented parts from the images (the squares). The Normalized Cut values are stated at the bottom of each image in Column (b). The image size increases from 15 x 15 to 300 x 300 (Row 1 to 3). The method successfully segments out the center square for all the sizes tested up to 300 x 300 as shown in Column (b). The *Ncut* value decreases with the increasing image size when the segmentation is correct (Column (b)).**

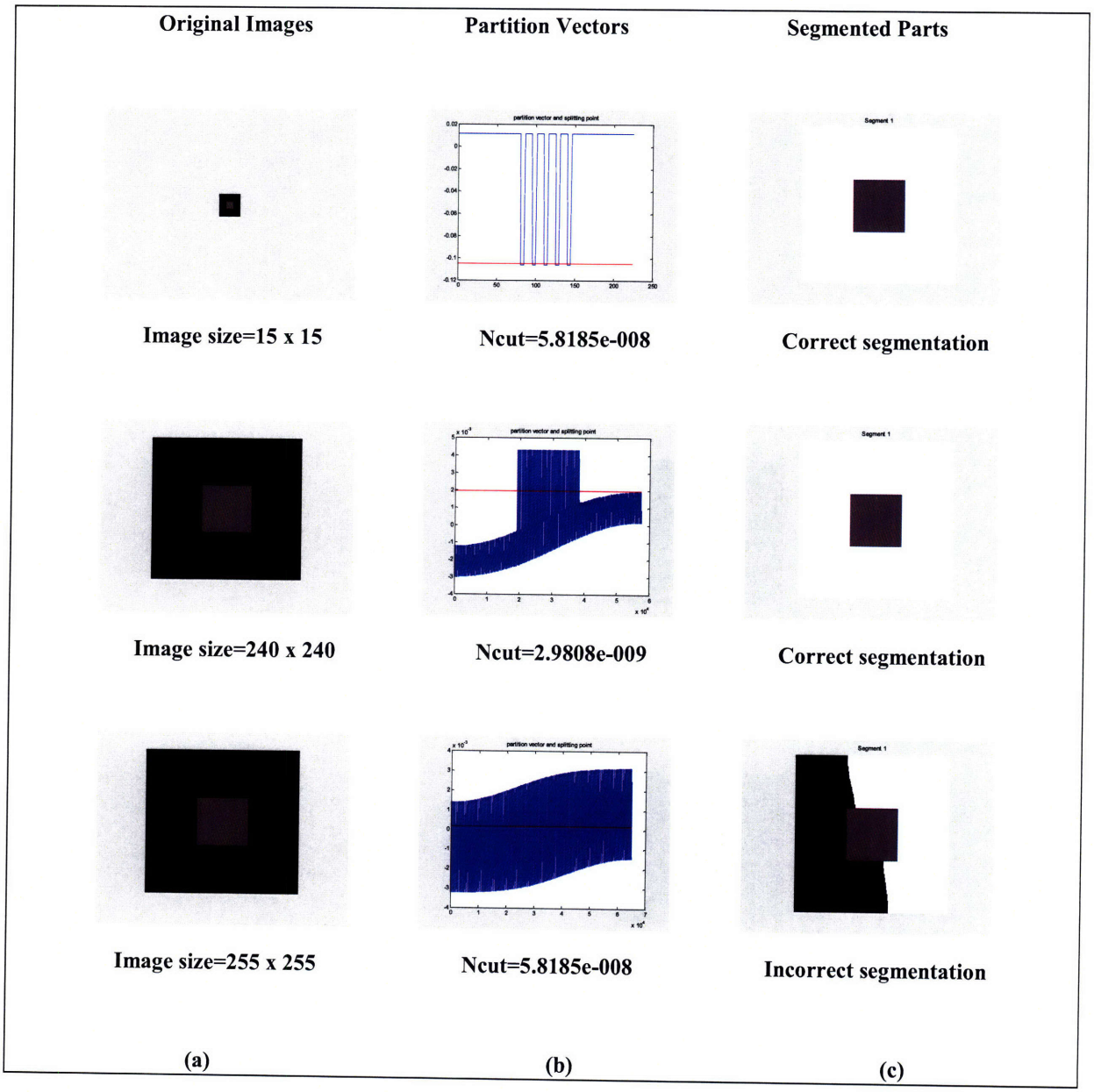


Figure 3.16 Image Segmentation by the Normalized Cut method for different image sizes. Column (a) shows the original images before segmentation and the pixel intensities of the square. Column (b) shows the Fiedler vectors (blue curve) and the splitting points that give the minimum *Ncut* value (red horizontal line). The *Ncut* values are given at the bottom of each plot in column (b). Column (c) shows the segmented part from the image (the square). The method fails to segments out the center square when the image size is 255 x 255 (Row 3). For image size below 255 x 255, the method performs the segmentation correctly. From the Fiedler vector plot, we can see why the method fails. The Fiedler becomes more continuous when the image size increases. This means the Fiedler vector as the approximation to the discrete Normalized Cut problem becomes less accurate. The failure is also shown in the increase of *Ncut* value from 2.9808e-009 (Row 2) to 5.8185e-008 (Row 3). The *Ncut* value should decreases with the increasing image size.

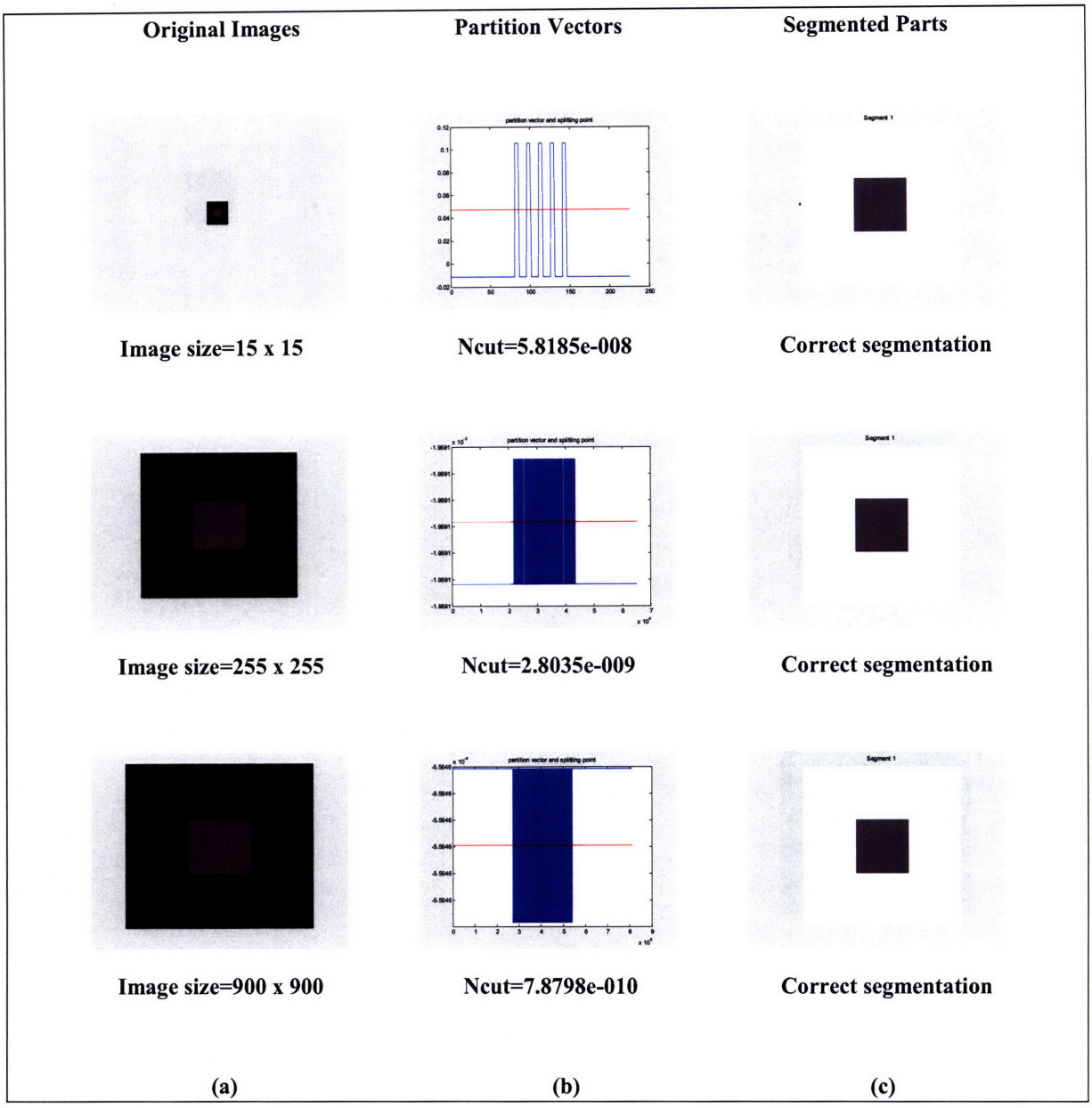


Figure 3.17 **Image Segmentation by the Spectral Rounding method for different image sizes. Column (a) shows the original images before segmentation and the pixel intensities of the square. Column (b) shows the partition vectors (blue curve) and the splitting points (red horizontal line). The *Ncut* values are given at the bottom of each plot in column (b). Comparing the partition vector plots with the Fiedler vector plot in Figure 3.12, we see that the vector plots by SR are discrete (binary). Column (c) shows the segmented part from the image (the square). Unlike the Normalized Cut method, for all the image size up to 900 x 900, Spectral Rounding performs the segmentation correctly. Since all the segmentations are correct, the *Ncut* value decreases with the increasing image size (Column (b)).**

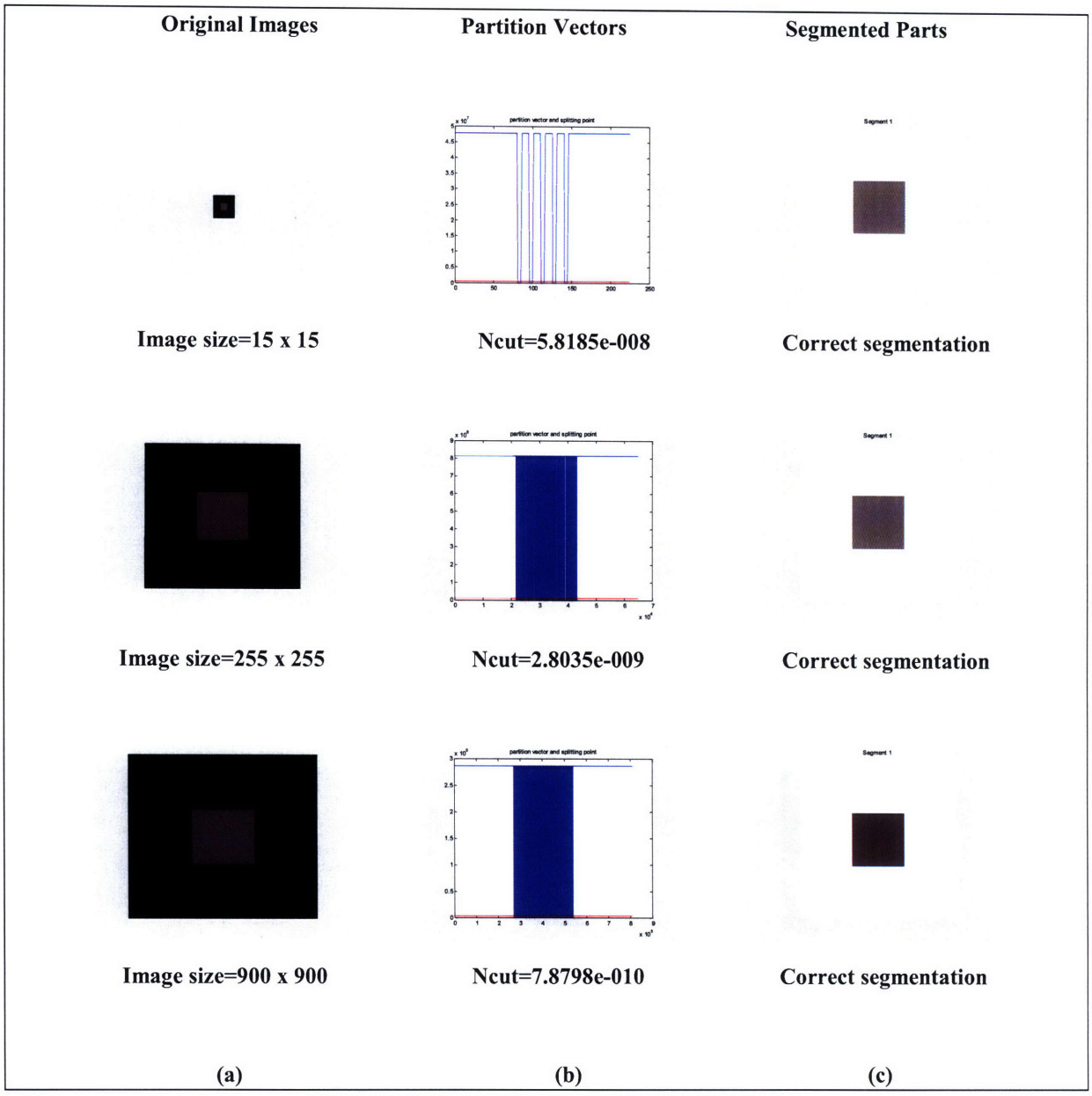


Figure 3.18 Image Segmentation by the Isoperimetric Partitioning for different image sizes. Column (a) shows the original images before segmentation and the pixel intensities of the square. Column (b) shows the partition vector (blue curve) and the splitting point that gives the minimum Normalized Cut value (red horizontal line). The *Ncut* values are given at the bottom of each plot in column (b). Column (c) shows the segmented parts from the images (the square). For all the image size up to 900 x 900, the Isoperimetric Partitioning performs the segmentation correctly, as shown in Column (c). With the correct segmentation, the *Ncut* value decreases with the increasing of image size.

3.3.4 Noise

In this test, I introduced noise to the 75 x 75 test image and test the resistance of the image segmentation methods towards noise. The noise types I used in the test are: *'Gaussian'*, *'Poisson'*, *'Salt and Pepper'* and *'Speckle'*.

Figure 3.19 (page 79) shows the image results of the noisy images using the Minimum Cut method. The method fails to segment all the noisy images except the image with *'Salt and Pepper'* noise. In all the failed cases, the method only segments out the single sink. This is the typical failure faced by the method because the degree of the sink and source is always the bottleneck of the method. If the degree is low, one of the sink and source with lower degree will be segmented out. In the previous pixel intensity difference and image size test, this does not happen because the sink and sources' degree is always larger than the sum of the edges weights along the objects boundary. With the introduction of noise around the sink and source, this is no longer true. Hence the method fails. In the case of *'Salt & Pepper'* noise (Row 3 of Figure 3.19, page 79), the method succeeds because the noise does not affect the degree of the sink and source. Observing the noisy image in Row 3, we can see that there is no noise element around the sink and source. (Upper left of the image and the object in the image).

In Figure 3.20 (page 80), we see that the Normalized Cut method is able to detect the square under *'Poisson'* and *'Speckle'* noise, but fails under *'Gaussian'* and *'Salt & Pepper'* noise. In segmenting the image affected by *'Gaussian'* noise, the method splits the image in the center. The cause to this failure can be seen in the Fiedler vector plots (Row 1, Column 2 of Figure 3.20, page 80). The continuous Fiedler vector fails to approximate the discrete solution of the Normalized Cut problem. The Normalized Cut method can segment the images with *'Poisson'* and *'Speckle'* noise because the noise is only distributed inside the object. Hence the edge weights between the object and the background are still small enough to be detected by the method. In the case of *'Salt and Pepper'* noise, the method failed because the resulting Laplacian matrix from the image is badly conditioned.

Similar to the Normalized Cut method, Spectral Rounding failed in segmenting the image with *'Gaussian'*, and *'Salt& Pepper '* noise, but succeeds in segmenting the image with *'Poisson'* and *'Speckle'* noise (Figure 3.21, page 81). The explanations to the failure and success are similar to those for the Normalized Cut method.

The Isoperimetric partitioning performs well in segmenting the noisy images. It is able to segment the images with *'Gaussian'*, *'Poisson'* and *'Speckle'* noise (Figure 3.22, page 82). The only failure is in segmenting the image with *'Salt & Pepper'* noise. The failure is caused by the badly conditioned Laplacian matrix constructed from the image.

Table 3.3 summarized the noise test results. Of all the methods, the Isoperimetric partitioning is the most robust towards noise because it survives the three noise tests. It is followed by the Normalized Cut method and Spectral Rounding method with two noise tests passed. The Minimum Cut method is the least robust towards noisy images. It only passes one noise test. However, it survives the *'Salt & Pepper'* noise test, which none of the other methods does. This is because this method does not involve the badly conditioned Laplacian matrix constructed from the image.

Table 3.3 The noise test results for the image segmentation methods.

Methods	**Noise**			
	Gaussian	Poisson	Salt & Pepper	Speckle
Min Cut	X	X	O	X
Ncut	X	O	X	O
SR	X	O	X	O
Iso	O	O	X	O

O – pass X – fail

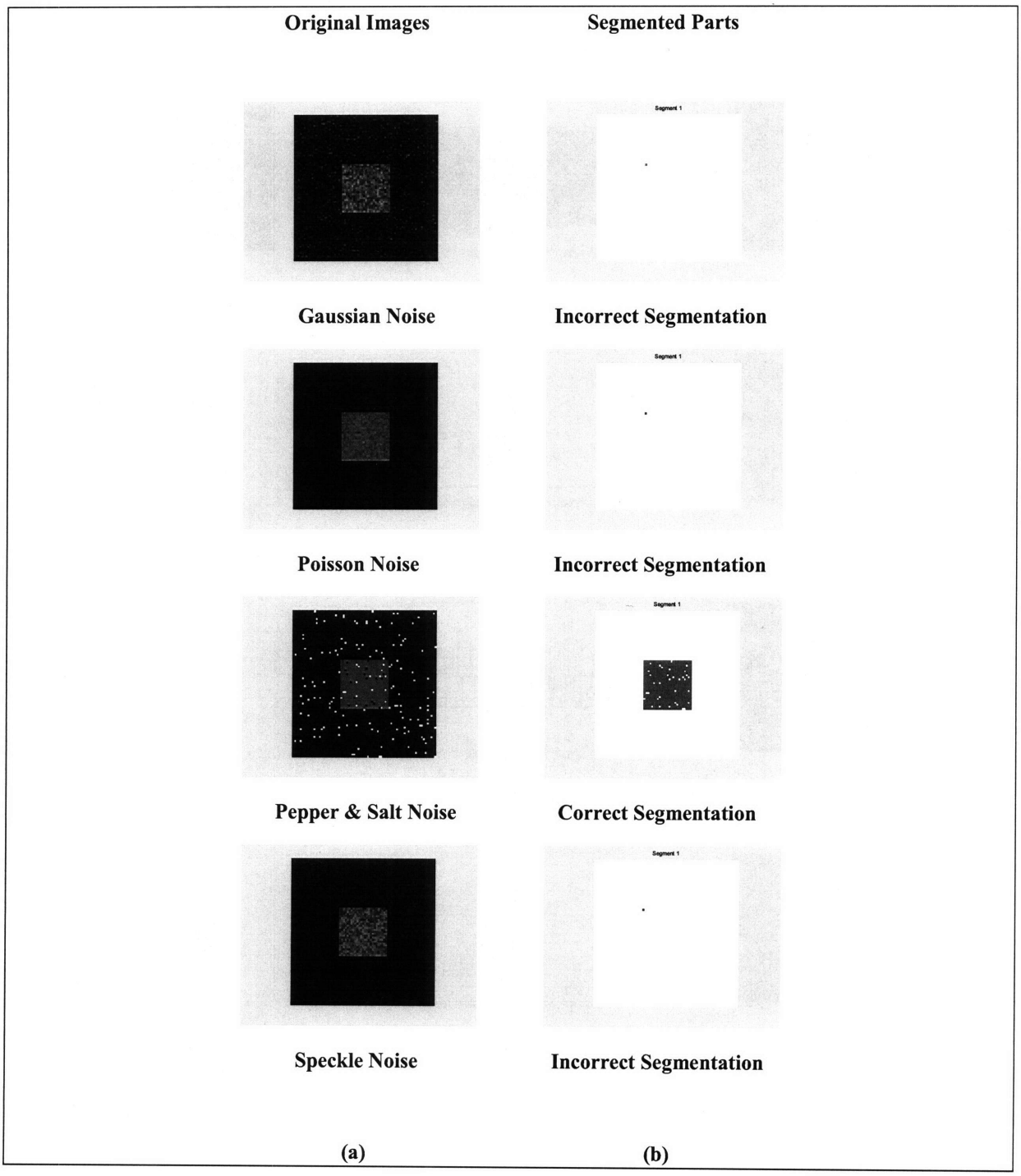


Figure 3.19 **Image Segmentation by the Minimum Cut method under four different noise types. Column (a) shows the original images affected by the noise before segmentation. The noise types are stated at the bottom of the images. The noise types used are: *'Gaussian', 'Poisson', 'Salt & Pepper'* and *'Speckle'* (Row 1 to 4). Column (b) shows the segmented parts from the images (the squares). The method fails to segments out the center square from the images affected by *'Gaussian'*, *'Poisson'*, and *'Speckle'* (Row 1, 2 and 4), but succeeds for the image affected by *'Salt & Pepper'* noise (Row 3).**

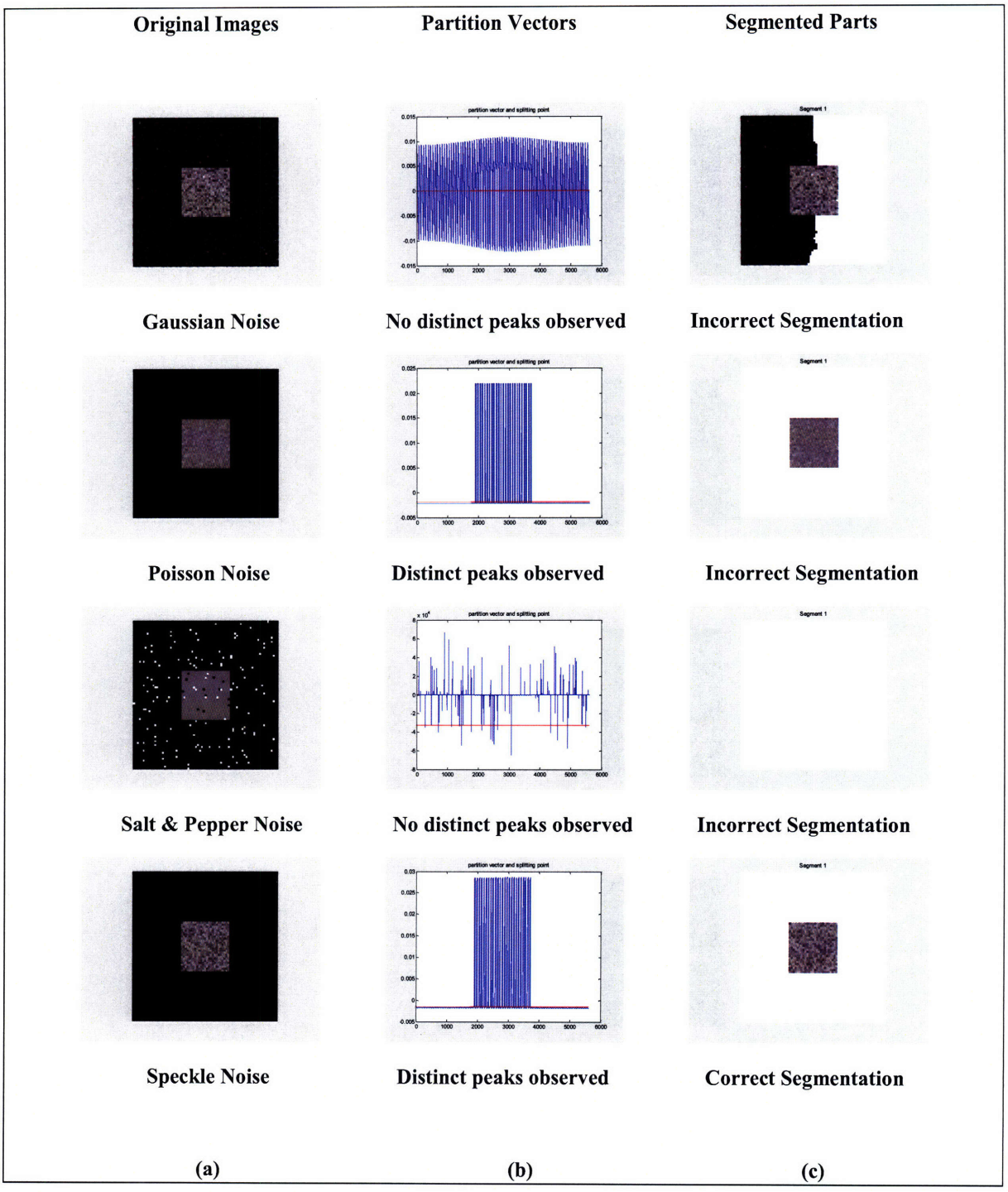


Figure 3.20 Image Segmentation by the Normalized Cut method under four different noise types. Column (a) shows the original images affected by the noise before segmentation. The noise types are stated at the bottom of the images. The noise types used are: *'Gaussian', 'Poisson', 'Salt & Pepper'* and *'Speckle'* (Row 1 to 4). Column (b) shoes the partition vector plots. Column (c) shows the segmented parts from the images (the squares). The method successfully segments out the center square from the images affected by *'Poisson'*, and *'Speckle'* (Row 2 and 4), but fails to segment the image affected by *'Gaussian;* and *'Salt & Pepper'* noise (Row 1 and 3). Notice that distinct peaks are observed in the vector plot (Row 2 and 4 of Column (b)) when the segmentation is successful. They allow the splitting point to cut through it easily.

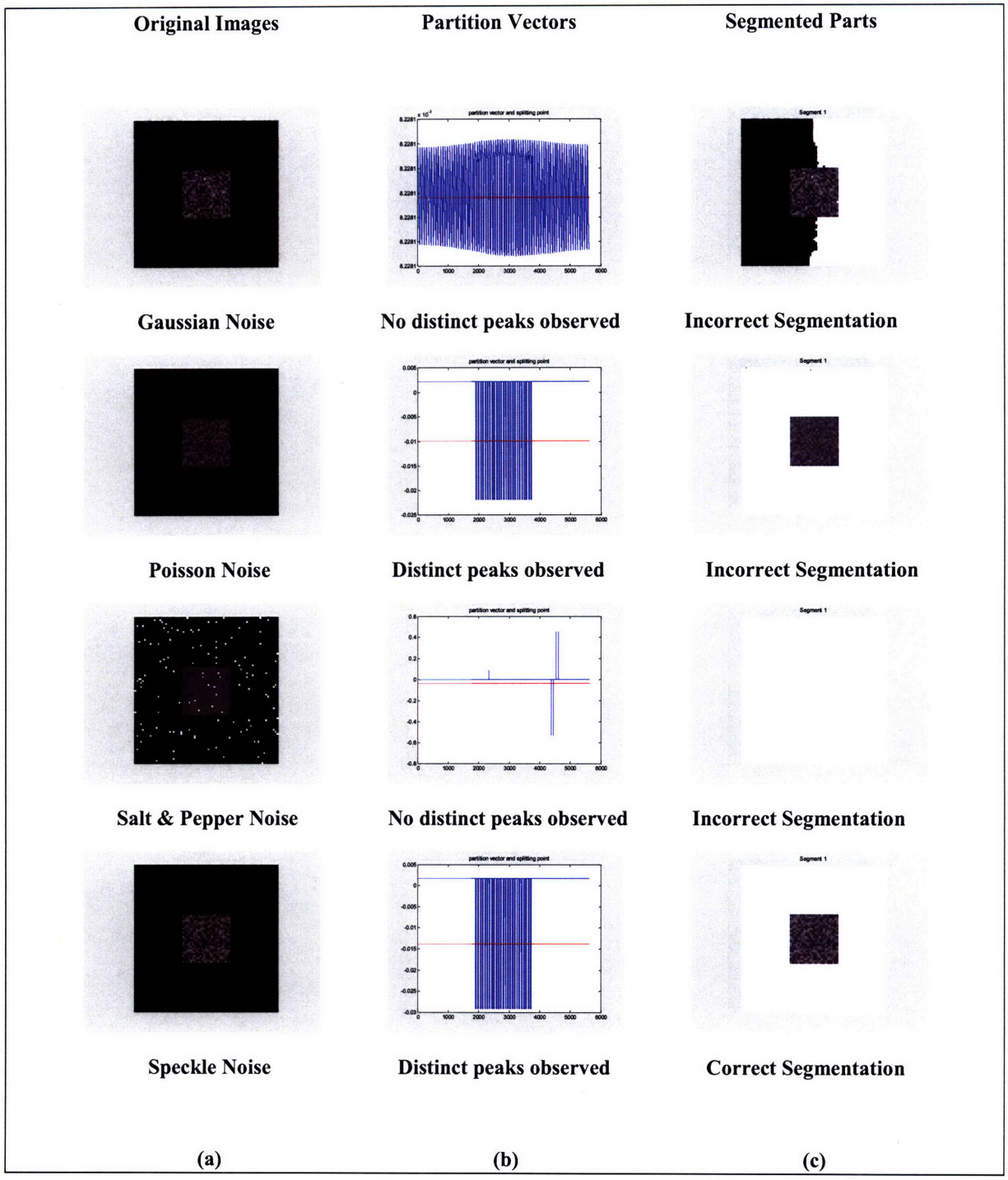


Figure 3.21 Image Segmentation by the Spectral Rounding method under four different noise types. Column (a) shows the original images affected by the noise before segmentation. The noise types are stated at the bottom of the images. The noise types used are: '*Gaussian*',' *Poisson*', *'Salt & Pepper'* and' *Speckle*' (Row 1 to 4). Column (c) shows the segmented parts from the images (the squares). The method successfully segments out the center square from the images affected by '*Poisson*' and '*Speckle*' (Row 2 and 4), but fails to segment the image affected by '*Gaussian'* and '*Salt & Pepper*' noise. Notice that distinct peaks are observed in the vector plot (Row 2 and 4 of Column (b)) when the segmentation is successful. They allow the splitting point to cut through it easily.

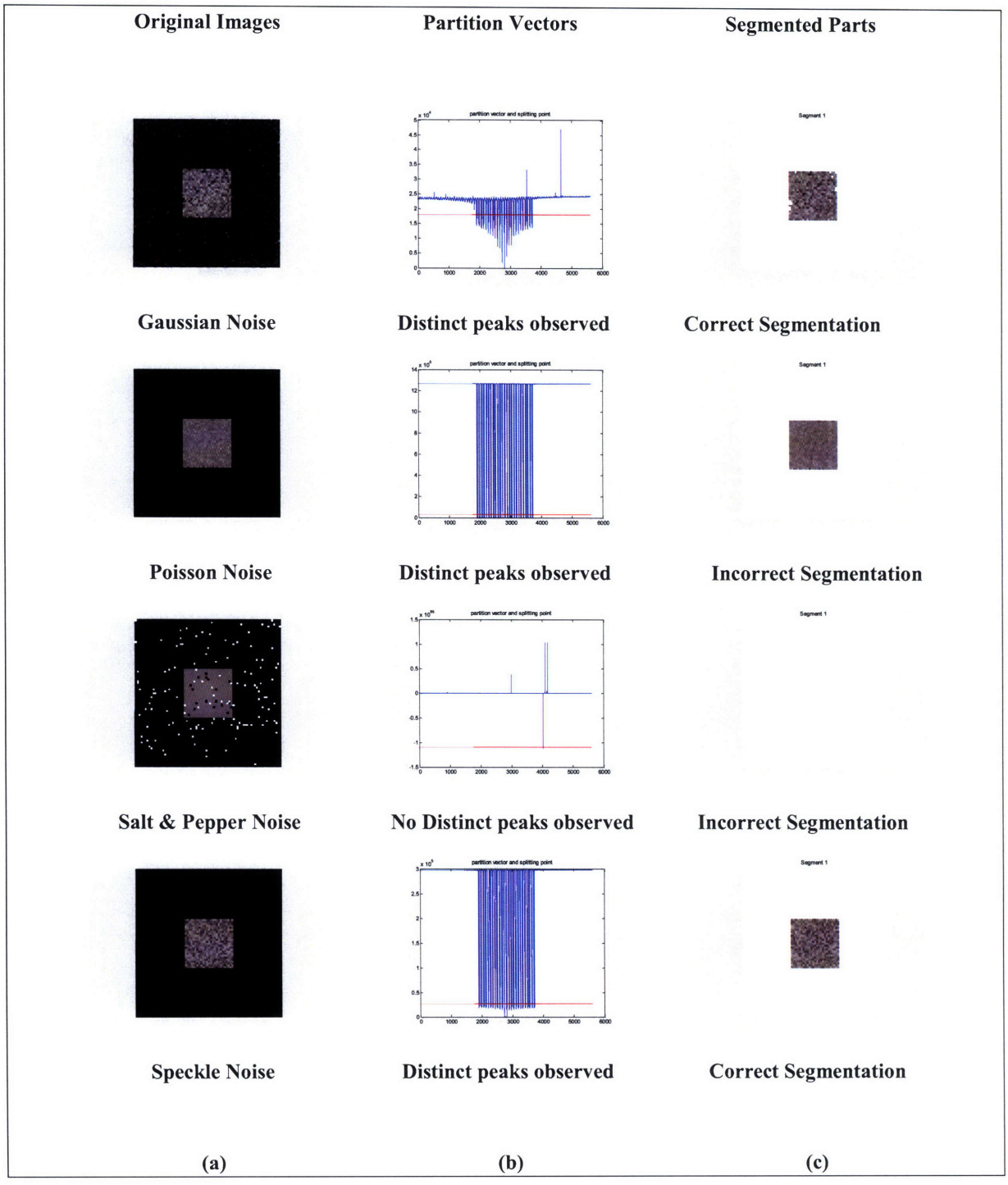


Figure 3.22 Image Segmentation by the Isoperimetric Partitioning under four different noise types. Column (a) shows the original images affected by the noise before segmentation. The noise types are stated at the bottom of the images. The noise types used are: *'Gaussian', 'Poisson', 'Salt & Pepper'* and *'Speckle'* (Row 1 to 4). Column (b) shows the segmented parts from the images (the squares). The method successfully segments out the center square from the images affected by *'Gaussian', 'Poisson'* and *'Speckle'* noise (Row 1, 2 and 4), but fails to segment the image affected by *'Salt & Pepper'* noise (Row 3). Notice that distinct peaks are observed in the vector plot (Row 1, 2 and 4 of Column (b)) when the segmentation is successful.

3.3.6 Run Time

Apart from the partitions' quality, the speed performance of each method is measured by recording the run time for each method. Since the construction of the graph for each method is the same, I omit the graph construction time from the run time. The run time measures the time for the image segmentation methods to take in a graph constructed from an image and give the discrete partitions. The run time variation with the image size for the four methods is shown in Figure 3.23. For better comparison between the Isoperimetric Partitioning and the Minimum Cut method, I re-plot the graph without the Normalized Cut and Spectral Rounding curves in Figure 3.24. All the results are generated on a 2.0 GHz Intel Core 2 Duo computer with 3 GB RAM.

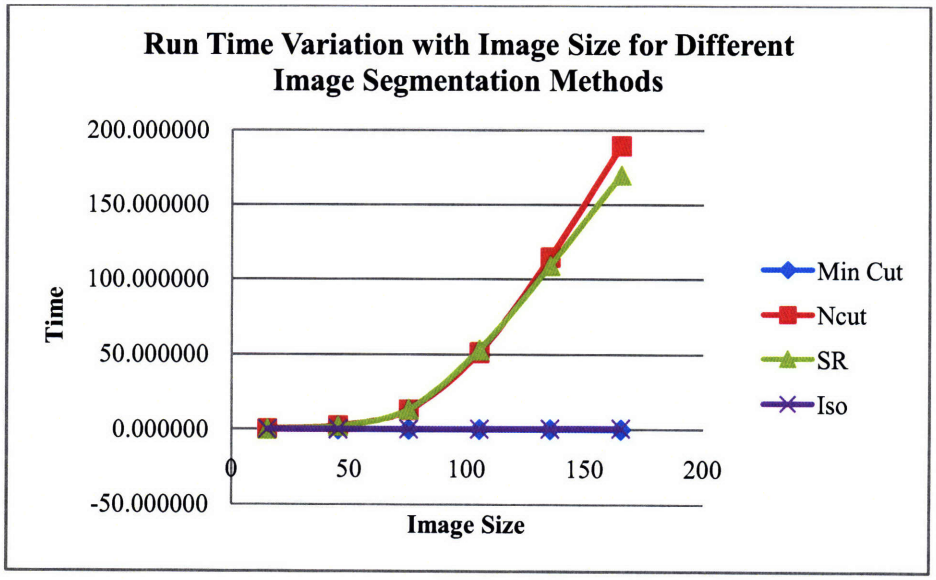


Figure 3.23 Run time variation with image size for different image segmentation methods. The run time increase with the image size. Among the four methods, the increase rates for both Normalized Cut and Spectral Rounding methods are the highest. It is followed by the Isoperimetric Partitioning and the Minimum Cut method.

From Figure 3.23, we see that the Minimum Cut method and Isoperimetric Partitioning are faster than the Normalized Cut and Spectral Rounding method. This is as expected, because The Minimum Cut problem is a polynomial-time-deterministic problem; and the Isoperimetric Partitioning only solves a linear system; while the Normalized Cut method solves an eigenvalue problem. From Figure 3.24 , we know that the Minimum method is indeed the fastest of all.

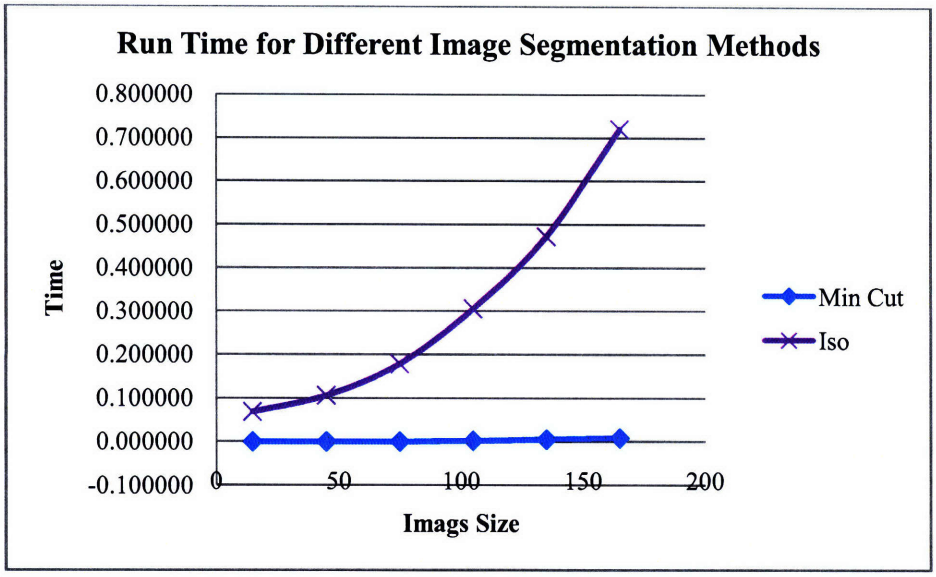


Figure 3.24 Run time variation with image size for the Minimum Cut method and Isoperimetric partitioning. The Minimum Cut method is faster than the Isoperimetric partitioning.

3.4 Summary

In this chapter, I have compared the image segmentation performance of the four methods in terms of their sensitivity to the change in pixel intensity difference between objects and background, their ability to segment large images, their robustness towards noise and their computation speed. Given a rating from 1 to 3, with 3 is the best, the performance of the four methods can be summarized in Figure 3.25.

From the figure, we see that the Minimum Cut method has the best score for three criteria: Intensity Difference, Image Size and Speed. However, it has a deadly shortage: its vulnerability to noisy image. The Normalized Cut method performs poorly in all the criteria. The Spectral Rounding method is good at segmenting large images but requires long computation time and is sensitive to the pixel intensity difference between objects and background. The overall performance of Isoperimetric Partitioning is good. It is not too sensitive to the pixel intensity

difference, and is good at segmenting large and noisy images. Furthermore, it is fast. To better understand the overall performance of the methods and to find the best method, I calculate the average performance score of the four methods (Figure 3.26).

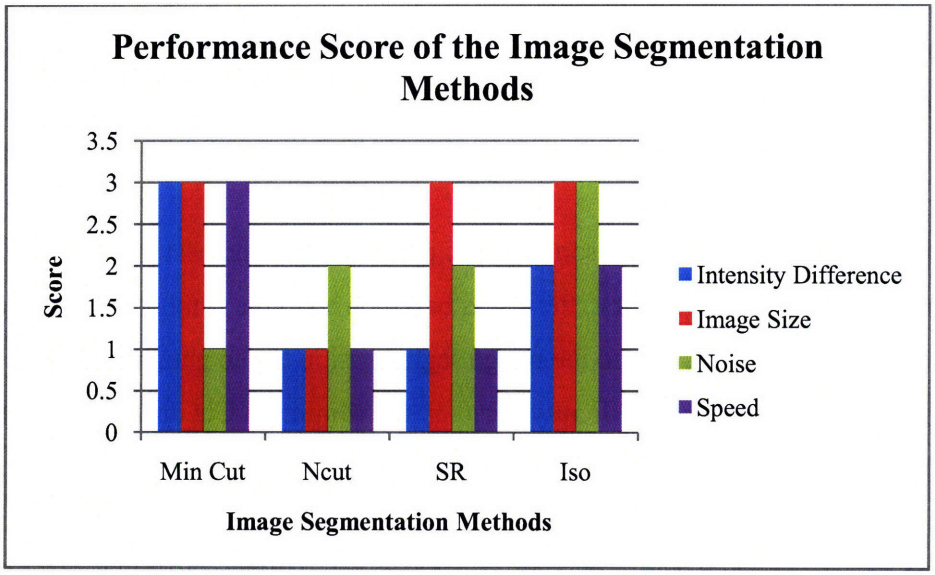


Figure 3.25 **The performance score of the four image segmentation methods according to the following criteria: the sensitivity to the pixel intensity difference between objects and background (Intensity Difference, blue bar), the ability to segment large image (Image Size, red bar), the robustness towards noise (Noise, green bar) and the computation speed (Speed, purple bar).**

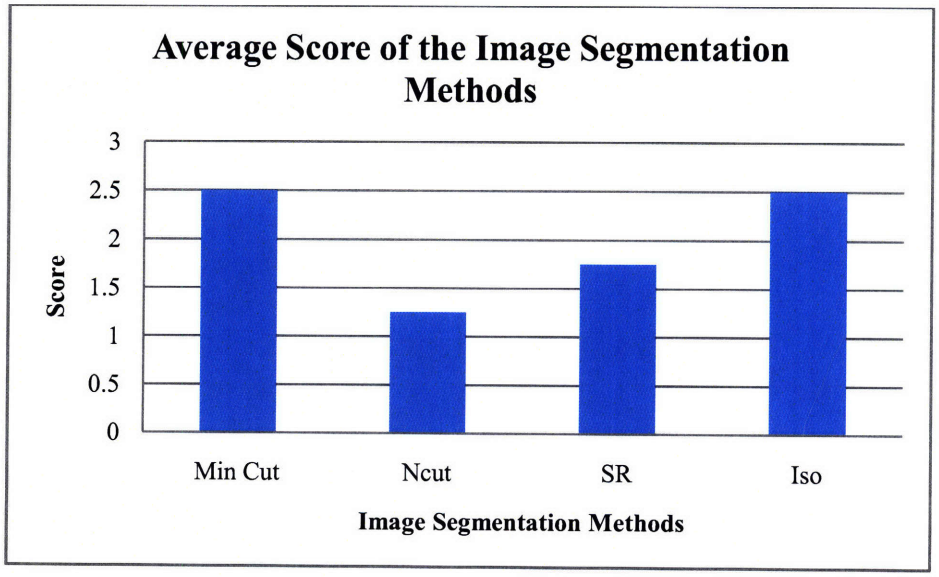


Figure 3.26 **Average performance score for the four images segmentation methods. The Minimum cut and the Isoperimetric partitioning has the highest score. They are followed by the Spectral Rounding method. The Normalized Cut method has the worst performance (lowest score).**

From Figure 3.26, we see that the Minimum Cut method and Isoperimetric partitioning have the highest average score. This is followed by the Spectral Rounding method. The Normalized Cut method has the lowest average score. Since the Minimum Cut method has a lowest score in one of the criteria, I concluded that Isoperimetric Partitioning has the best performance in image segmentation and is followed by the Minimum Cut method.

Though from the above analysis, the Isoperimetric partitioning is the best method, there are still a few issues we should take note. First, the test is based on a simple synthetic image, which is not conclusive enough. We should extend the image segmentation test to natural images. Furthermore, I have overlooked a difficulty faced by the Minimum Cut and Isoperimetric Partitioning. Both methods need the placement of sink and source and the location of the nodes places a crucial role in determining the success of the methods. Lastly, in the process of calculating the overall performance score, no weight is given to the criteria. The analysis will be more conclusive if weights are use based on the degree of influence of the factors towards the process of image segmentation.

Chapter 4 0-1 Graph Partitioning

4.1 The Idea

The idea of the 0-1 Graph Partitioning is developed from Grady's Isoperimetric Partitioning [7]. The Isoperimetric Partitioning has an electrical circuit analogy. The author treats graphs as electrical circuits: every node has a voltage (node value) and the edge weights are the conductances. By inputting a current source to each node, except ground node with zero voltage, we can solve for the nodes' voltages by solving a linear system (Equation 2.8). Then, the voltage vector can be further discretized to give the graph's partition. The electrical circuit analogy of the Isoperimetric Partitioning is as follows:

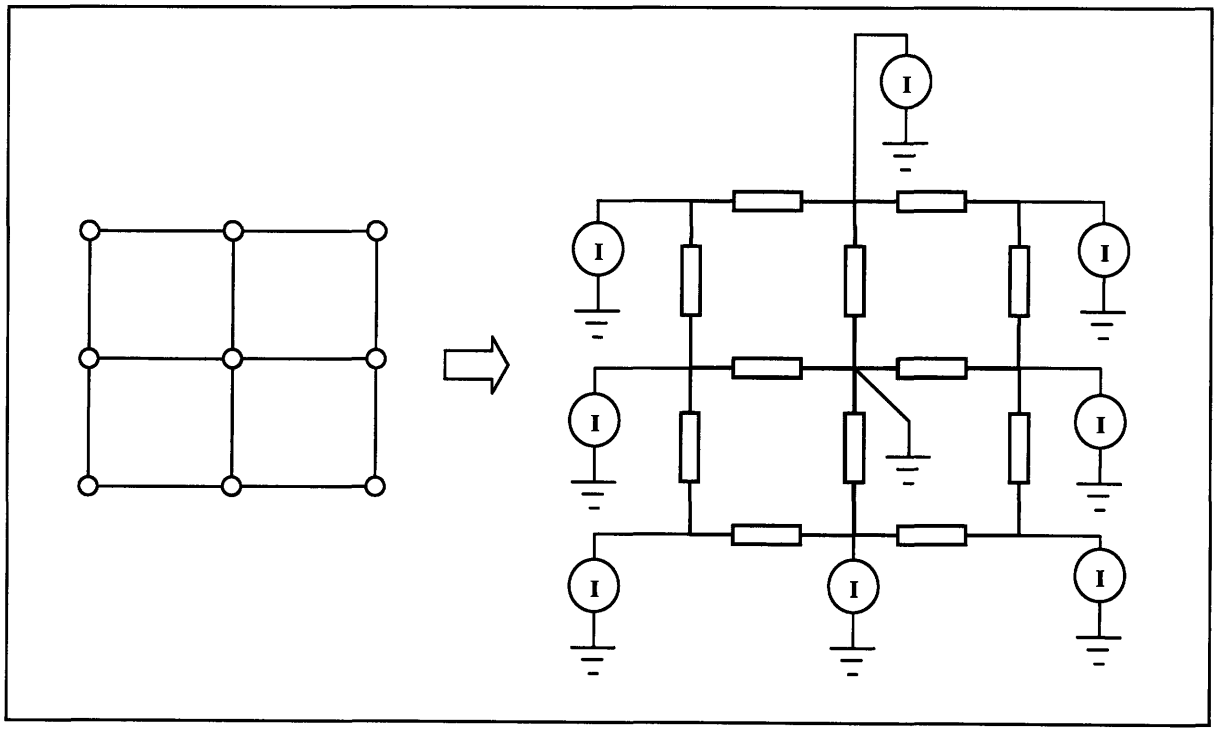


Figure 4.1 A 9-node graph (left) is represented by an electrical circuit with eight current sources and one ground (right) (Adapted from [7]). In the electric circuit, each node is connected to a current source except the ground node (middle node). The edge weights of the graph are represented by electrical conductors (rectangular boxes).

The 0-1 method is the modification of the electric circuit analogy of Isoperimetric Partitioning. Instead of a current source at every node, voltage sources at certain nodes are used as shown in Figure 4.2.

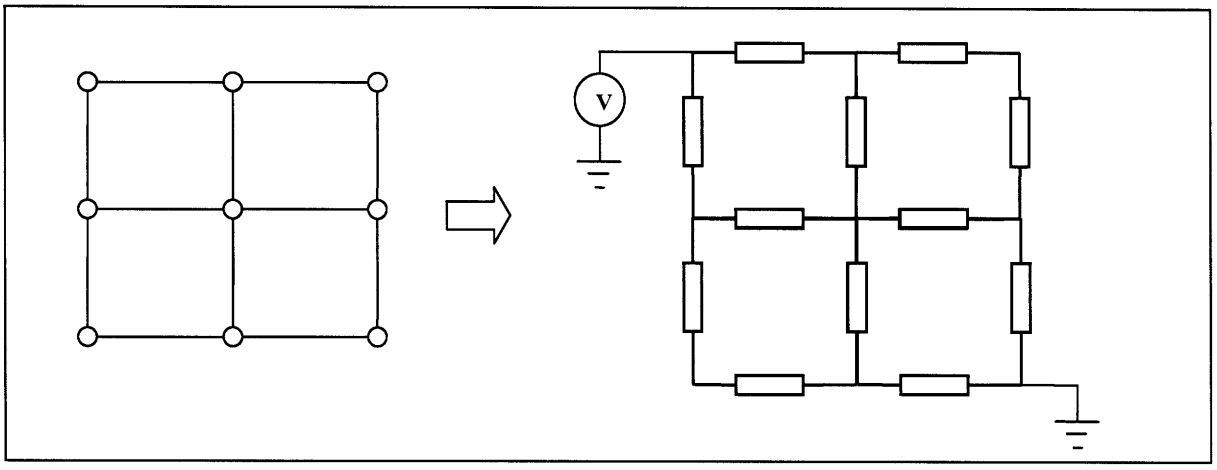


Figure 4.2 A 9-node graph (left) is represented by an electrical circuit with one voltage source and one ground (right). In the electric circuit, a node (source) is connected to a voltage source while another node (sink) is grounded. The two nodes cannot be the same node. The edge weights of the graph are represented by electrical conductors (rectangular boxes) .

The voltage vector can be used as the continuous partition vector because the voltage difference between two nodes is proportional to the edge weight (conductance) of the link that connects the two nodes. Apart from edge weights, the information of the connectivity or proximity between nodes is also considered in this electric circuit concept. Nodes that are close to each other are likely to be grouped together because connected nodes or adjacent nodes tend to have similar voltages.

4.2 Basic 0-1 Algorithm

The 0-1 method can be summarized as follows:

Algorithm 4.1

Given an n-node graph with its Laplacian matrix, $\boldsymbol{L}$ and its diagonal matrix, $\boldsymbol{D}$; p source, $sn_{i,\ i=1,\ldots,p}$; q sink, $gn_{i,\ i=1,\ldots,q}$; to obtain the 0-1 vector, $\boldsymbol{x}$, we:

i. Set $\boldsymbol{f} = \boldsymbol{0}$

ii. Add column $sn_{i\ ,\ i=1,\ldots,p}$ of $\boldsymbol{L}$, $\boldsymbol{l}_{sn_i}$ to $\boldsymbol{f}$:

$$\boldsymbol{f} = \boldsymbol{f} + \sum_{i=1}^{p} \boldsymbol{l}_{sn_i}$$

iii. Remove column and row $sn_{i,\ i=1,\ldots,p}$ of $\boldsymbol{L}$ to form a reduced matrix, $\boldsymbol{L'}$

iv. Remove column and row $gn_{i,\ i=1,\ldots,q}$ of $\boldsymbol{L'}$ to form a reduced matrix, $\boldsymbol{L''}$

v. Remove row $sn_{i,\ i=1,\ldots,p}$ and $gn_{i\ ,\ i=1,\ldots,q}$ of $\boldsymbol{f}$ to form a reduced vector, $\boldsymbol{f'}$

vi. Solve $\boldsymbol{L''} * \boldsymbol{y} = \boldsymbol{f'}$

vii. Set $\boldsymbol{x} = \boldsymbol{0}$

viii. Set $x_k = 1$ for $k = \{sn_{i\ ,i=1,\ldots,p}\}$

ix. Set $x_{k_j} = y_j$ for $k = \{1, \ldots, n\} \backslash \{sn_{i\ ,i=1,\ldots,p}\} \backslash \{gn_{i\ ,i=1,\ldots,q}\}$ and $j = 1, \ldots, n - p - q$

4.3 0-1 Vector

Similar to the Fiedler method that produces a Fiedler vector, the 0-1 method produces a vector that gives the voltages of the nodes. Since the vector values lie in between zero and one, hence the method is named as 0-1 method and the vector as 0-1 vector. The Fiedler vector and 0-1 vector are both continuous vector used to approximate the discrete partition vector. We can obtain the discrete partition vector by discretizing the vectors (Section 4.4). Both vectors serve as the continuous partition vector and under certain conditions, the 0-1 vector can be a good approximation to the Fiedler vector (Section 4.11). The 0-1 vector can be computed faster than the Fiedler vector because it only involves a linear system, while the Fiedler vector involves a generalized eigensystem.

4.4 Discretization

As mentioned earlier, the 0-1 vector is a continuous vector. For graph partitioning, our objective is to obtain the partitions. We can represent the partitions by binary discrete partition vectors. In order to convert the continuous 0-1 vector to the discrete partition vector, a discretization process is performed to the 0-1 vector. We want to discretize the 0-1 vector in such a way that the resulting partitions give the minimum *Ncut* value. We can achieve this by using α evenly spaced splitting points [3] to discretize the vector into α partition vectors. Then we compute the *Ncut* values based on the partition vectors (Equation 2.4). Finally, we choose the best partition vector that gives the minimum *Ncut* value. The Minimum *Ncut* Discretization algorithm is summarized below:

Algorithm 4.2:

Given a degree matrix, $\boldsymbol{D}$; adjacency matrix, $\boldsymbol{W}$; and continuous vector (Fiedler or 0-1 vector), $\boldsymbol{v}$, to obtain the discrete partition vector $\boldsymbol{p}$, we:

i. $v_R = \max(\boldsymbol{v}) - \min(\boldsymbol{v})$

ii. For $i = 1, \ldots, \alpha$

For $j = 1, \ldots, n$

If $v_j < \frac{i}{\alpha+1} * v_R$

$x_{p_{i,j}} = 1$

Else

$x_{p_{i,j}} = 0$

End if

End for

$Ncut_i = Ncut(\boldsymbol{D}, \boldsymbol{W}, \boldsymbol{x_{p_i}})$

End for

iii. $\beta = argmin_{i=1,\ldots,\alpha}(Ncut_i)$

iv. $min\, Ncut = Ncut_\beta$

v. $\boldsymbol{p} = \boldsymbol{x_{p_\beta}}$

Apart from the consideration of the minimum *Ncut*, we also want to discretize the vector in the shortest time possible. To achieve this, we may compromise the quality of the partition by using the half-cut criterion instead of the minimum *Ncut* criterion. The 0-1 method ensures that the half-cut still gives a reasonably low *Ncut* value. The Half-cut Discretization algorithm is summarized below:

Algorithm 4.3:

Given a degree matrix, $\boldsymbol{D}$; adjacency matrix, $\boldsymbol{W}$; and continuous vector (Fiedler or 0-1 vector), $\boldsymbol{v}$, to obtain the discrete partition vector $\boldsymbol{p}$, we:

i. $v_R = \max(\boldsymbol{v}) - \min(\boldsymbol{v})$

ii. For $i = 1, \ldots, n$

    If $v_j < 0.5 * v_R$

        $x_{p_i} = 1$

    Else

        $x_{p_i} = 0$

    End if

    End for

iii. $Ncut = Ncut(\boldsymbol{x_p})$

iv. $\boldsymbol{p} = \boldsymbol{x_{p_\beta}}$

4.5 Mathematical Interpretation of 0-1 Method

Thought 0-1 Method is derived from the electric circuit analogy used in Isoperimetric Partitioning [7], I try to interpret the 0-1 method using some mathematical concepts.

Given a graph $G(N, E)$ with n nodes and e edges, with its Laplacian matrix, $\boldsymbol{L}$, the 0-1 algorithm can be described mathematically by the following matrix equation:

$$\boldsymbol{L} * \boldsymbol{x} = \boldsymbol{0}. \tag{4.1}$$

Since the Laplacian matrix, $\boldsymbol{L}$ is singular, we solve Equation (4.1) for vector $\boldsymbol{x}$ (continuous 0-1 vector) by prefixing the value of certain elements of the vector to be zero (sink) or one (source).

Equation (4.1) can be derived from the concept of the minimum cut. For simplicity, we consider the case of bi-partitioning. A graph cut separates a graph into two (groups S and $\bar{S}$) and the cut value is defined as the sum of the broken edges' weights. The cut value can be calculated as follows:

$$cut = \boldsymbol{p}^T * \boldsymbol{L} * \boldsymbol{p}, \tag{4.2}$$

with $\boldsymbol{p}$ as a binary partition vector (1 representing group S) and (0 representing group $\bar{S}$). If we relax the discrete problem to allow a continuous solution, $\boldsymbol{x}$, we can minimize the cut value by minimizing the following function:

$$f(\boldsymbol{x}) = \boldsymbol{x}^T * \boldsymbol{L} * \boldsymbol{x}. \tag{4.3}$$

We minimize the function by setting the derivative to be zero:

$$\nabla f(\boldsymbol{x}) = 2 * \boldsymbol{L} * \boldsymbol{x} = \boldsymbol{0}. \tag{4.4}$$

Removing the constant 2 from Equation (4.4), we obtain Equation (4.1).

From the mathematical derivation, we know that 0-1 method is actually the relaxed problem of the minimum cut problem. The relaxation is partly redundant since we can solve the minimum cut problem exactly in polynomial time (Ford-Fulkerson algorithm). However, the relaxed solution can be a good and quick approximation to the normalized cut problem (Section 4.11). Hence, we use the 0-1 method to solve the normalized cut (*Ncut*) problem, instead of the minimum cut problem. We impose the minimum *Ncut* constraint in the discretization stage (Section 4.4). In a nutshell, the 0-1 method tries to find a cut in between the sinks and sources that minimize the *Ncut* value.

4.6 Location of Sinks and Sources

The sinks and sources will be located in different partitions -- they cannot coexist in the same subset. In other words, their location affects the partitioning. Hence, the location of the sink and sources is crucial. The criteria for good sink and sources differ for different types of graph. I will discuss the criteria for each application (Unweighted Graphs and Weighted Graphs).

4.7 *k*-way Partitioning

To partition a graph into k partitions, we apply the bi-partitioning recursively. The shortage for recursive bi-partitioning is that for odd number of partitions, the partitions are not balanced. Apart from this, the recursive bi-partitioning cannot assure that the k partitions minimize the *Ncut* value globally. For illustration, we consider the following case: a graph partitioned into two partitions minimizes the *Ncut* value, and gives balanced partitions. However, when one of the partitions is further partitioned into two partitions, the overall three partitions do not minimize the *Ncut* value because the three partitions are not balanced in size. To avoid this situation, we use the simultaneous partitioning.

A simple recursive bi-partitioning using 0-1 method has been developed. However, the simultaneous k-way partitioning using 0-1 method for general graphs is not developed yet. The current simultaneous k-way partitioning using 0-1 method can only be applied specifically to image segmentation (see Chapter 5).

4.8 Comparison with Isoperimetric Partitioning

The 0-1 method is developed from the Isoperimetric Partitioning. Both methods solve a linear system for a continuous partition vector. They both have an electrical circuit analogy. They also need ground nodes in graphs to convert the singular Laplacian matrix to a non-singular matrix.

Apart from the similarities, there are a few differences. First, apart from ground (sink) nodes, 0-1 method needs sources. In terms of electrical circuit analogy, 0-1 method has voltage sources (see Figure 4.2, page 88) while Isoperimetric Partitioning uses current sources (see Figure 4.1, page 87). Since 0-1 method applies voltage sources with magnitude of one, the resulting voltage at each node lies in between zero and one (sinks have zero voltage and sources have voltage of one). In contrast, for Isoperimetric Partitioning, the voltage at each node has a lower bound of zero with no upper bound (sinks have zero voltage).

The linear systems created from the two methods differ in the right hand side. Isoperimetric Partitioning creates a linear system with the right hand side as an all-1-vector. For 0-1 method, the right hand side is a zero vector. Isoperimetric Partitioning is derived from the isoperimetric problem. The method tries to minimize the Isoperimetric constant of a graph. Though 0-1 method is developed from Isoperimetric Partitioning, the method has a mathematical interpretation of minimum cut (Section 4.5).

4.9 Application in Unweighted Graph Partitioning

4.9.1 Location of the Sinks and Sources

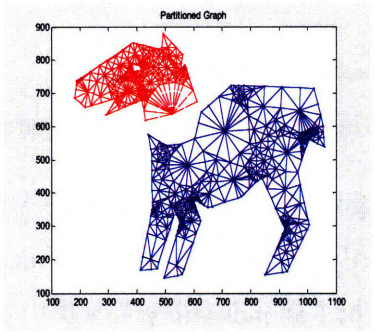
The 0-1-method's partitioning quality relies on the location of the sink and sources. The effect can be seen in Figure 4.3 (page 96). From there, we deduce some criteria for good sink-sources:

i. The sink and sources are located as far as possible from each other (for graph with coordinates: mesh).
ii. No link or short path is desirable between the sink and sources.
iii. The sink and sources must be located at the correct segmented parts separately.

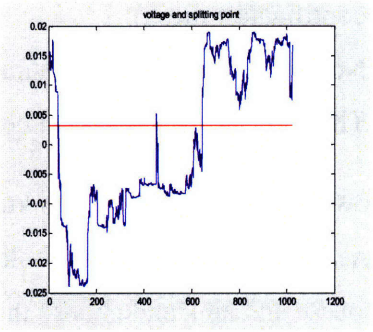
Given an unknown graph, it is difficult to know any good sinks and sources' locations that fulfill the above criteria. The solution to this problem is partly by trial and error. We randomly generate a few pairs of source and sink and choose the best. The best pair will give a 0-1 vector that minimizes the Ncut value when the vector is discretized. However, this method requires more trials for better result, and more trials will surely cost longer running time.

We can make an educated guess by considering the node numbering convention. One of the usual conventions aims to number the nodes from the nearest to the furthest or from linked to unlinked. If we assume all the graphs follow this convention, we can locate the source at the first node and sink at the last node. Nevertheless, this assumption is not valid all the time. For a better result, we should also consider to put the sink at fixed interval from the first node. For example, for the case of four intervals, we can put the sink at the first, second, third quartile and last nodes with the first node fixed as the source. With more intervals, we should expect better results at the expense of running time.

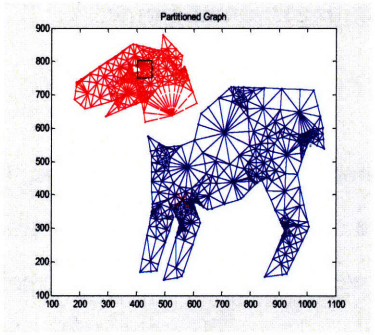
If we are given the coordinates of the graph nodes, we can make use of the information and put the source at the nearest node and the sink at the furthest node. This location will be another good guess, but may not be the best.



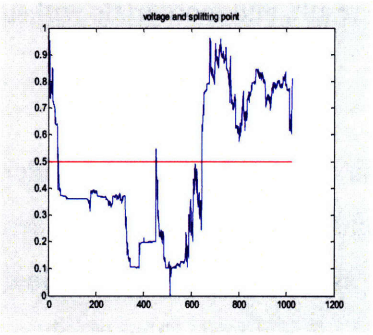
(a) Fiedler method



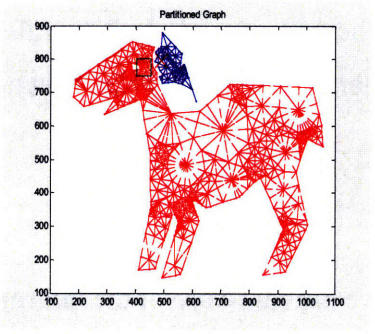
(b) Ncut = 0.0080



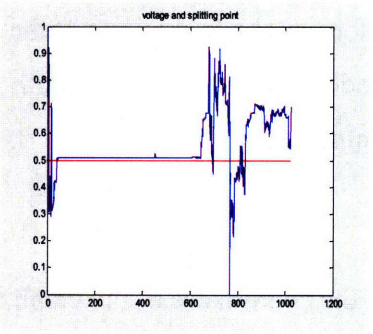
(c) 0-1 method: source: 1, sink: $n/2$



(d) Ncut = 0.008



(e) 0-1 method: source: 1, sink: $3n/4$



(f) Ncut = 0.0602

Figure 4.3 An n-node graph (horse-shaped mesh) is segmented using the Fiedler method and 0-1 method with 0.5 as splitting point (half cut). The results in first row are produced by using the former while the last two columns are produced by the latter. The first column shows the partitioned graph while the second column shows the Fiedler vector or 0-1 vector plots (blue curves) and the splitting point (red lines). Observing the last two rows, we see that the difference in sink-sources' locations produces very different results. Using the result in first row as the benchmark, the result in the second row is better with correct segmentation and lower *Ncut* value. Looking at (c), we can see that the sink and sources are located far apart and on the two separated segments. Another important observation is that the vector plot (d) resembles the Fiedler vector plot in (b). In contrast, for the last row, the sink and sources are located close to each other and the vector plot in (f) shows a very different vector from the Fiedler vector in (b).

Combining the two educated guesses, we can run the basic 0-1 algorithm for a few sinks and sources' locations. In order to shorten the computation time, the Half-cut Discretization scheme is used to give the discrete partition vectors and calculate the *Ncut* values. Then I choose the partition vector that minimizes the *Ncut* values. This give rise to the modified 0-1 method – Auto 0-1 Method as summarized below:

Algorithm 4.4:

Given an n-node graph with its adjacency matrix, $\boldsymbol{W}$; its diagonal degree matrix, $\boldsymbol{D}$; and coordinates (if available); and we decide to use α intervals to obtain the 0-1 vector, $\boldsymbol{v}$:

i. Set node 1 and node n as the sink and source
ii. Run the Basic 0-1 algorithm to give 0-1 vector, $\boldsymbol{v_i}$
iii. Run the Half-cut Discretization algorithm to give partition vector, $\boldsymbol{p_i}$
iv. $Ncut_i = Ncut(\boldsymbol{W}, \boldsymbol{D}, \boldsymbol{p_i})$
v. Repeat step (i) - (iv) for sink-source pairs:
$\left[1, \frac{n}{\alpha}\right], \left[1, \frac{2n}{\alpha}\right], \ldots, [1, \frac{(\alpha-1)n}{\alpha}]$ and $[nearest, furthest]$ (if coordinate is available)
iv. $\beta = argmin_{i=1,\ldots,\alpha,[nearest,furthest]}(Ncut_i)$
v. $min\, Ncut = Ncut_\beta$
vi. $\boldsymbol{v} = \boldsymbol{v_\beta}$

4.9.2 Experiment and Results

a. Setup

In this experiment, I compare the partitions obtained by the Fiedler method and 0-1 method. The partition quality is measured quantitatively by the *Ncut* value. Apart from the partition quality, the running times for the methods are compared as well. The graphs used in the experiment are

the meshes obtained from FTP site of John Gilbert and the Xerox Corporation[2]. For Fiedler method, the tolerance for convergence is set to be $10^{15} * \varepsilon$ (ε = 2.2204e-016) because the eigenvectors of most of the graphs used converge very fast. For the Auto 0-1 method, I use four intervals: first node is the source and the four quartile (1st, 2nd, 3rd and 4th quartile) nodes are the sinks. I also include the nearest-furthest sink-source pair to produce the 0-1 vectors. The input sinks and sources generated from the Auto 0-1 algorithm (bi-partitioning) are tabulated in Table 4..

Table 4.1 Sink-Source pairs used in the 0-1 method for different graphs (meshes).

Mesh	Number of nodes	Source	Sink
airfoil1	4253	1	2127
airfoil2	4720	1	3540
eppstein	547	1	410
tapir	1024	1	512
triangle	5050	1	5050
crack	5120	1	10
parc	1240	1	310
parcweb	1939	46	19
spiral	1200	1	600
smallmesh	136	1	34

For *k*-way partitioning, the Auto 0-1 algorithm is applied recursively. For comparison purpose, I use the recursive Fiedler method instead of the simultaneous Fiedler method (multiple eigenvectors).

b. 2-way Graph Partitioning

Table 4.2 shows the result of the numerical comparison between Fiedler method and 0-1 method for bi-partitioning. Measured in *Ncut* value, four out of ten meshes (highlighted red in Table 4.2)

[2] *ftp://ftp.parc.xerox.com/pub/gilbert/meshes.tar.Z*

have improvements by using the 0-1 method. The biggest improvement is 13.64% decrease in *Ncut* value for *'spiral'* mesh. Nevertheless, three meshes have no improvement in *Ncut* value: *'tapir'*, *'parc'* and *'smallmesh'* (highlighted green in Table 4.2). The *Ncut* values are probably the lowest values achievable and hence we cannot improve further. For *'airfoil2'*, *'eppstein'* and *'triangle'*, the *Ncut* values increase. Among them, *'airfoil2'* has the largest increase in *Ncut* value (13.33%).

Table 4.2 ***Ncut* values obtained by the Fiedler and 0-1 bi-partitioning for different graphs (meshes).**

Mesh	Number of nodes	Ncut		
		Fiedler	0-1	Change (%)
airfoil1	4253	0.0094	0.0092	-2.13
airfoil2	4720	0.0135	0.0153	13.33
eppstein	547	0.0407	0.0420	3.19
tapir	1024	0.0080	0.0080	0.00
triangle	5050	0.0184	0.0190	3.26
crack	5120	0.0194	0.0185	-4.64
parc	1240	0.0125	0.0125	0.00
parcweb	1939	0.0096	0.0090	-6.25
spiral	1200	0.0044	0.0038	-13.64
smallmesh	136	0.0679	0.0679	0.00

Figure 4.4 to Figure 4.13 (page 100 to 106) show the graphical comparison between the partitions by the Fiedler and 0-1 methods for ten different meshes. In Figure 4.4 (page 100), we see that both methods give balanced partitions. Both methods cut the mesh into two through the center region of the mesh, where the mesh is less dense. In this way, the partitions cut through least number of edges. Despite the similarity between the two partitions, there are slight differences in the partitions, especially in the lower region of the mesh (along the cut). This slight change in the partitions gives 2.13% improvement in the *Ncut* value (Table 4.2).

Figure 4.5 (page 101) shows the partitioned meshes by Fiedler and 0-1 method for *'airfoil2'* mesh. Comparing to *'airfoil1'*, *'airfoil2'* has denser nodes in the region around the airfoil. The effect is the segmentation no longer cut through the mesh. Instead, a small region of the rear part of the airfoil is segmented out in both cases (Fiedler and 0-1) as shown in Figure 4.5. Observing the

zoom-in partitions in (c) and (d), we can see the partitions are similar. However, the difference in *Ncut* value is large. The 0-1 method gives 13.33% increase in *Ncut* value compared to Fiedler method (Table 4.2, page 99).

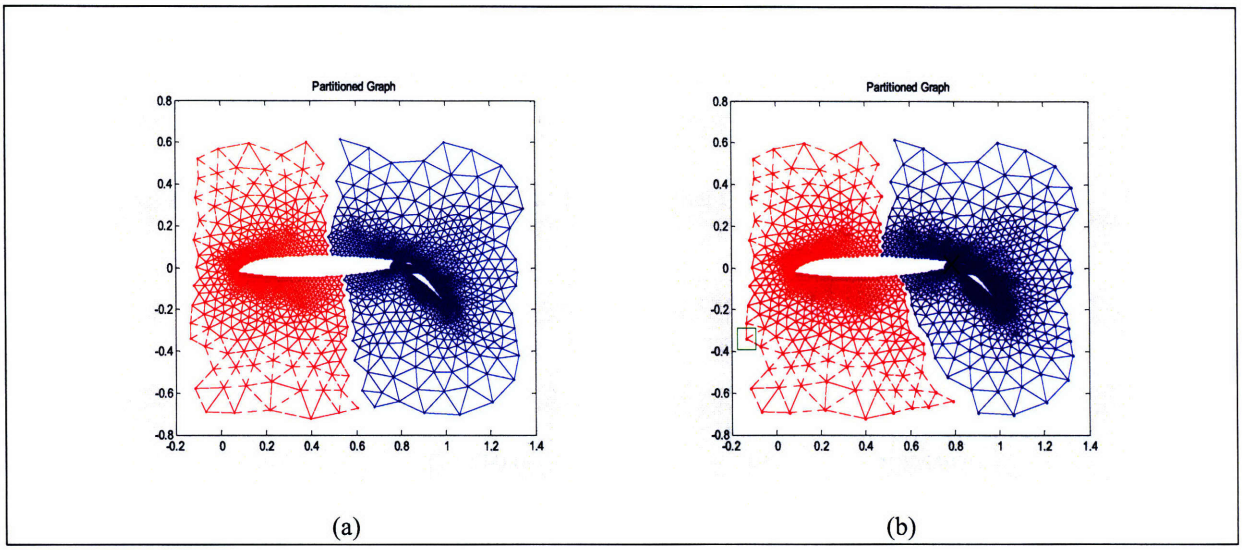


Figure 4.4 Graph partitioning of *'airfoil1'* mesh using: (a) Fiedler method and (b) 0-1 method. In (b), the green square is the source and the red 'X' is the sink. Both methods give similar and balanced partitions. The difference in the partitions is more obvious in the lower region of the cut.

Figure 4.6 shows the partitioned meshes by Fiedler and 0-1 method for *'eppstein'* mesh. The two methods give very different partitions. Both methods give unbalanced partitions. However, Fiedler method gives lower *Ncut* value as shown in Table 4.2 (page 99).

The partitioning of *'tapir'* mesh by the Fiedler and 0-1 method are shown in Figure 4.7 (page 102), (a) and (b) respectively. The two methods give the same partition. As a result, the *Ncut* values given by the two methods are the same (Table 4.2, page 99). The same *Ncut* value tells us that this value is probably the minimum *Ncut* value we can achieve.

Figure 4.8 (page 103) shows the partitioning of *'triangle'* mesh by the Fiedler method in (a) and 0-1 method in (b). The two partitions are different at first look. However, if we rotate the partitioned mesh in (b) anticlockwise by 60 degree, we will see that the partition is almost the

same. In terms of the *Ncut* value, the 0-1 method has a slightly higher *Ncut* value (3.26%) more than that given by the Fiedler method as shown in Table 4.2 (page 99).

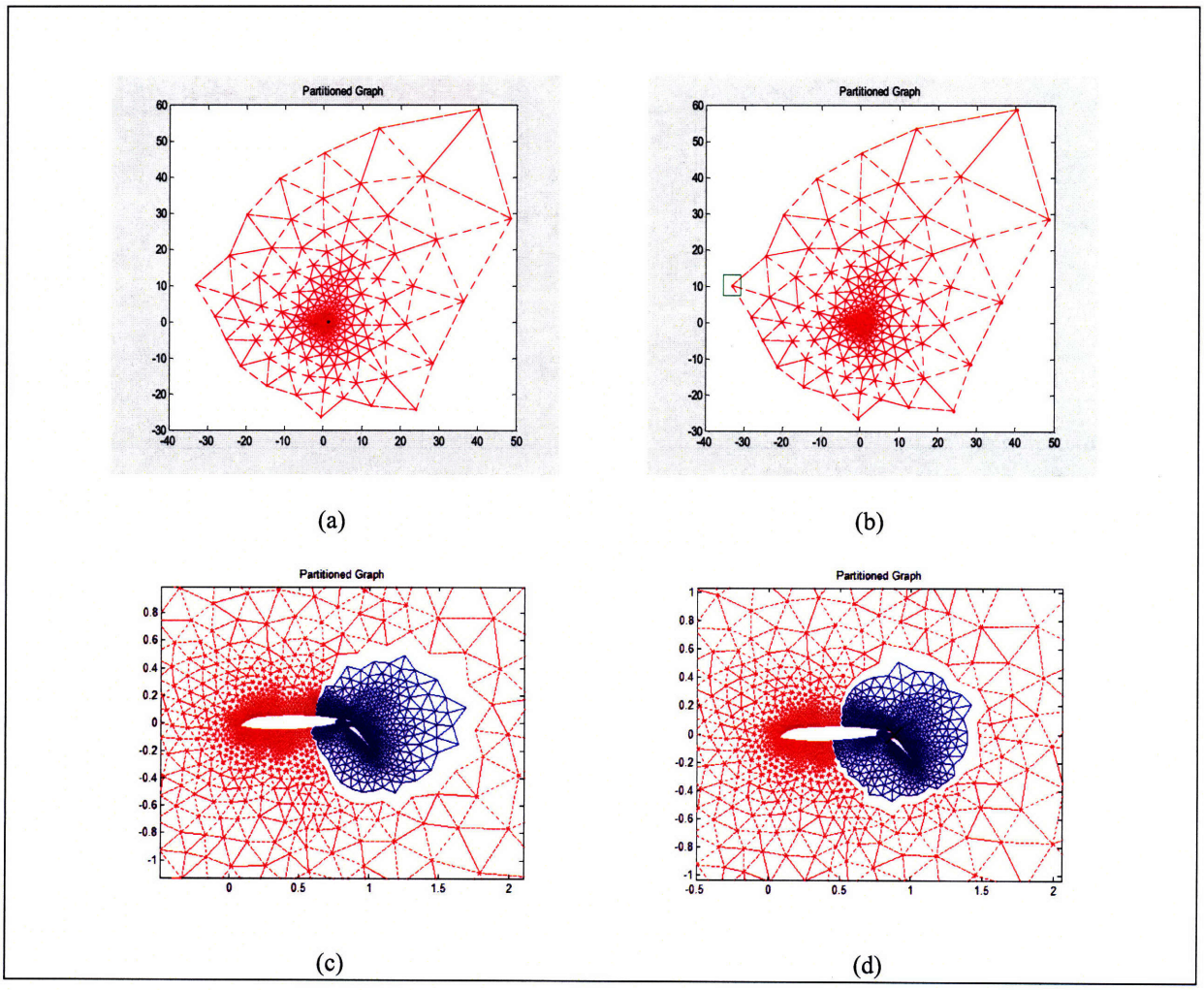


Figure 4.5 Graph partitioning of *'airfoil2'* mesh using: (a) Fiedler method and (b) 0-1 method. The partitions in row 2 is the zoom-in of the of the center region of the mesh. In (b) and (d), the green square is the source and the red 'X' is the sink. Both methods give similar partitions.

Figure 4.9 (page 103) shows the partitioned *'crack'* mesh by the Fiedler method (a) and 0-1 method (b). Both partitioned meshes are balanced and similar. The only difference is shown along the cut in the right region. This difference has improved the *Ncut* value from 0.0194 to 0.0185 (4.64%, Table 4.2, page 99).

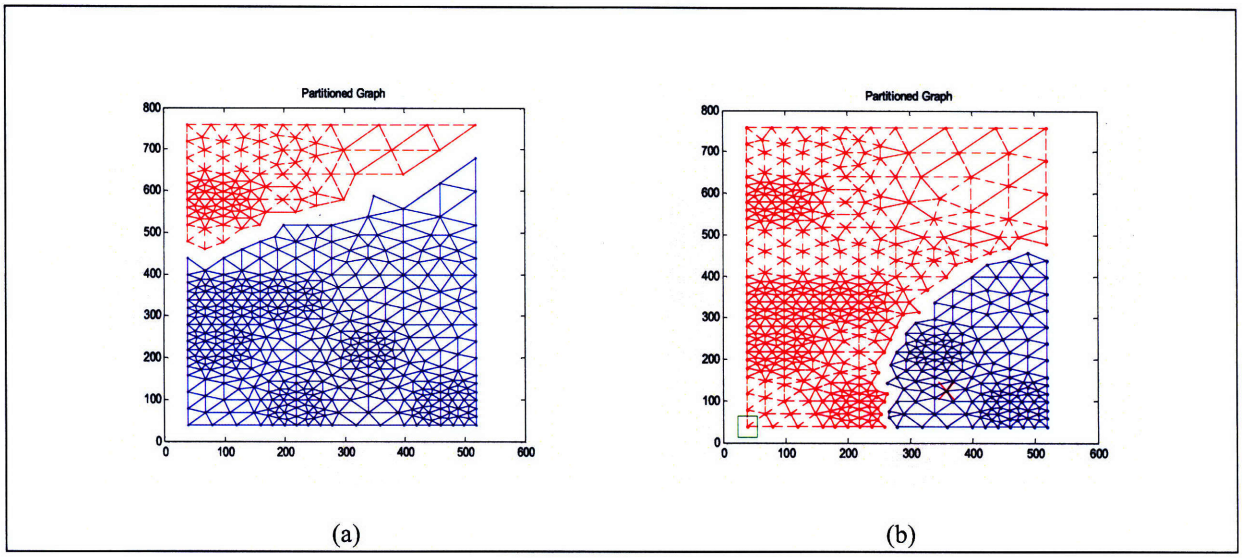


Figure 4.6 **Graph partitioning of *'eppstein'* mesh using: (a) Fiedler method and (b) 0-1 method. In (b), the green square is the source and the red 'X' is the sink. The two methods give different partitions. Both partitions by 0-1 method in (b) and by Fiedler method in (a) is not balanced.**

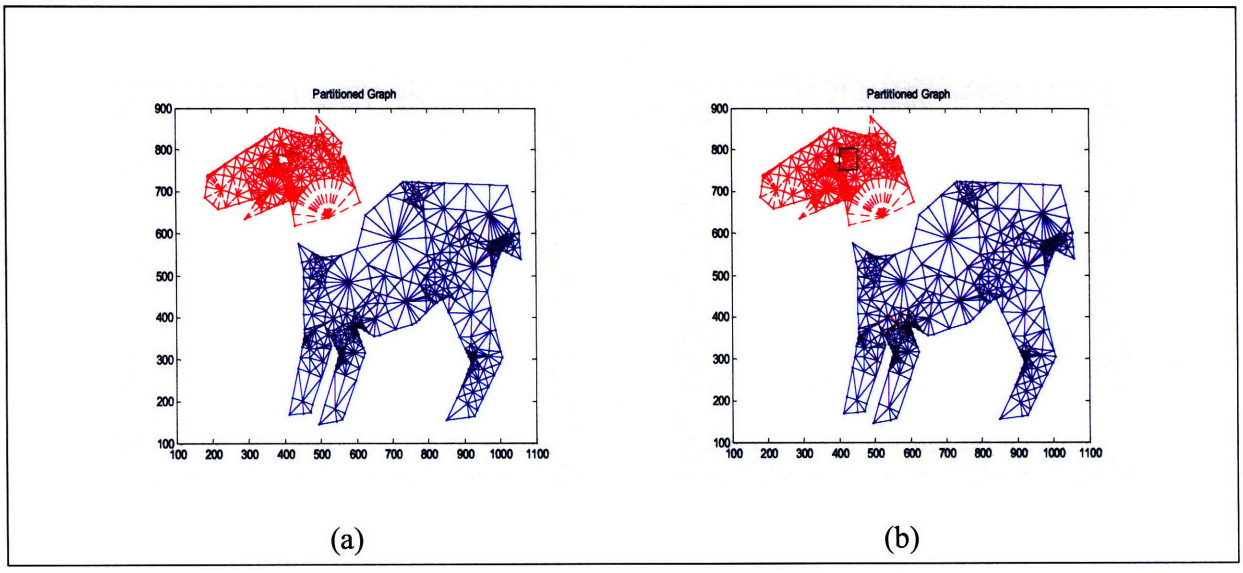


Figure 4.7 **Graph partitioning of *'tapir'* mesh using: (a) Fiedler method and (b) 0-1 method. In (b), the green square is the source and the red 'X' is the sink. The two methods give the same partition.**

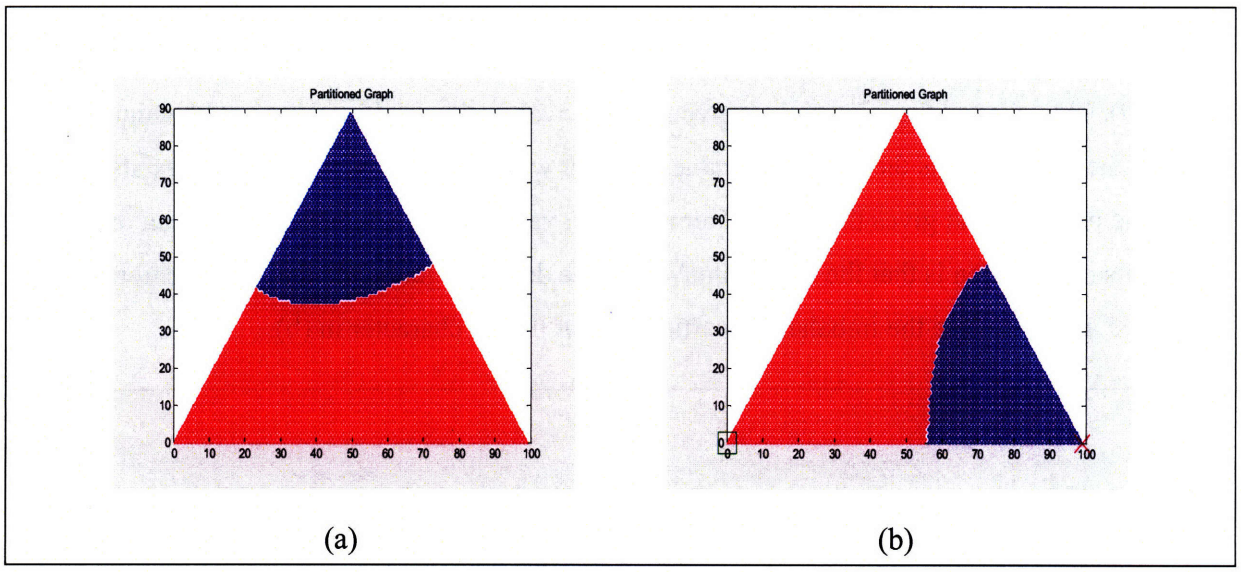


Figure 4.8 Graph partitioning of *'triangle'* mesh using: (a) Fiedler method and (b) 0-1 method. In (b), the green square is the source and the red 'X' is the sink. The two methods give different partitions. However, when we rotate the partitioned graph in (b) anticlockwise by 60 degree, we will see that the two partitions are actually similar.

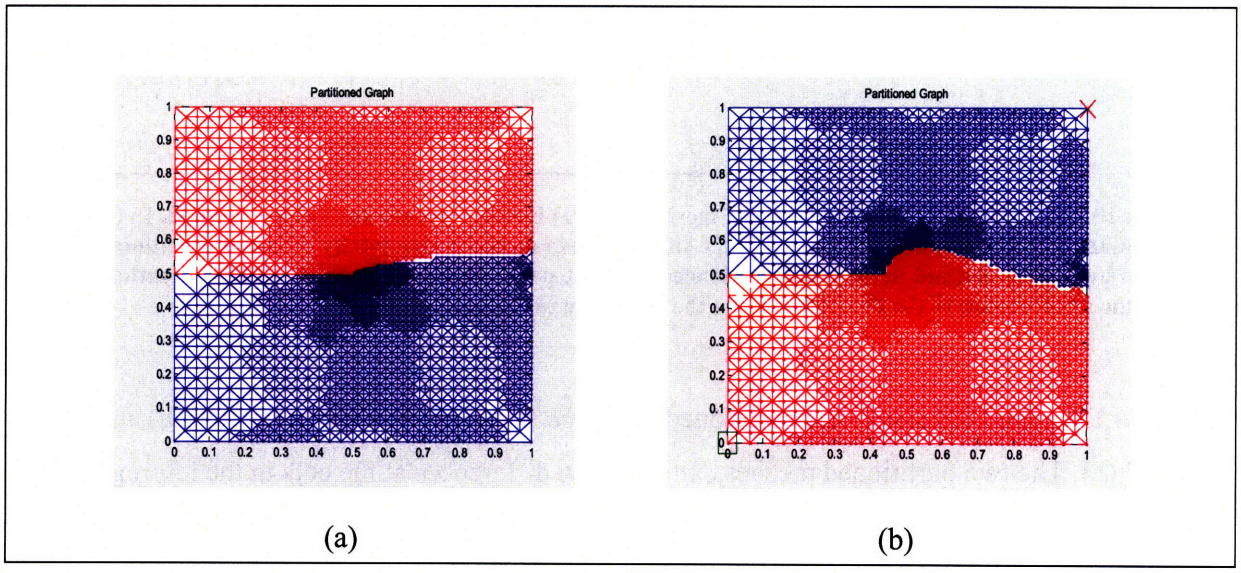


Figure 4.9 Graph partitioning of 'crack' mesh using: (a) Fiedler method and (b) 0-1 method. In (b), the green square is the source and the red 'X' is the sink. Both methods give similar and balanced partitions. The major difference is in the right region along the cut.

Figure 4.10 (page 104) shows the *'parc'* mesh being segmented by the Fiedler method in (a) and 0-1 method in (b). The partitions displayed by the two methods are totally different. The Fiedler method separates the mesh through the center (more balanced) while 0-1 method only segment

out a small part of the 'C' character in the mesh. Despite the great difference shown in the partitioned meshes, interestingly, both methods give the same *Ncut* value: 0.0125 (Table 4.2, page 99). This observation shows that, given an *Ncut* value, the partition may not be unique, even if the *Ncut* value is the minimum value. In other words, it is possible that there exists more than one partition that gives the same minimum *Ncut* value. Another conclusion we can draw from this observation is that the minimum *Ncut* value does not guarantee that the partition is balanced as we can see the imbalanced partitioning of the 'C' character in (b).

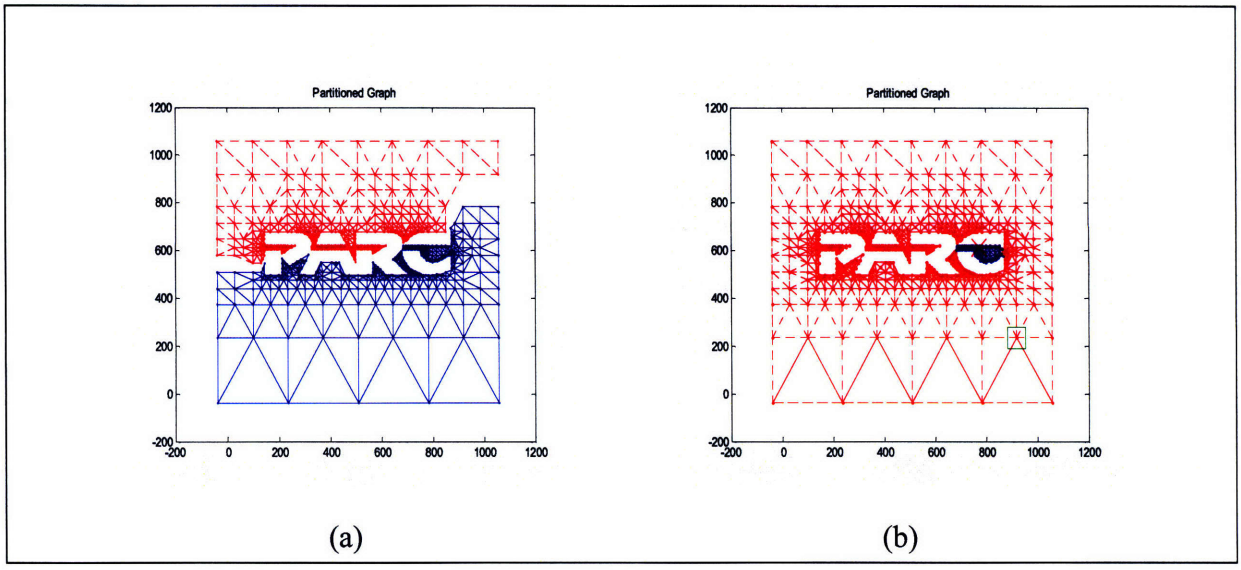


Figure 4.10 Graph partitioning of 'parc' mesh using: (a) Fiedler method and (b) 0-1 method. In (b), the green square is the source and the red 'X' is the sink. The two methods give totally different partitions. The partition by Fiedler method in (a) is more balanced than the partition by 0-1 method in (b). 0-1 method only cut out the small region in the character 'C' of the mesh due to higher density.

Figure 4.11 (page 105) shows the partitioned *'parcweb'* mesh by the Fiedler method (a) and 0-1 method (b). The two partitioned meshes exhibit a few differences at the cuts in the left region near the 'P' character and the right region near the 'C' character. Again, this difference in partition improves the *Ncut* value by 6.25% (Table 4.2, page 99).

Figure 4.12 (page 105) shows the partitioned *'spiral'* mesh by the Fiedler method in (a) and 0-1 method in (b). The two methods exhibit a subtle difference in the region around the cut. Though

the difference is small, 0-1 method gives an improved *Ncut* value of 0.0090 compared to 0.0096 by the Fiedler method (improved by 13.64%) as shown in Table 4.2 (page 99).

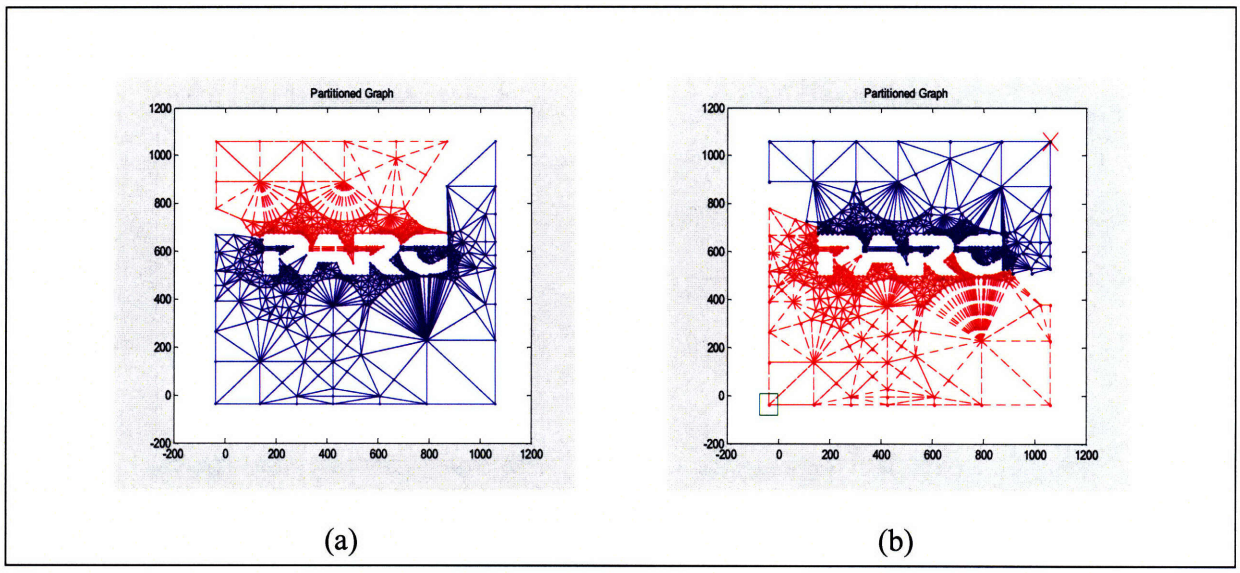


Figure 4.11 Graph partitioning of *'parcweb'* mesh using: (a) Fiedler method and (b) 0-1 method. In (b), the green square is the source and the red 'X' is the sink. The two methods give balanced but slightly different partitions. However, the partition by 0-1 method in (b) is more balanced than the partition by Fiedler method in (a).

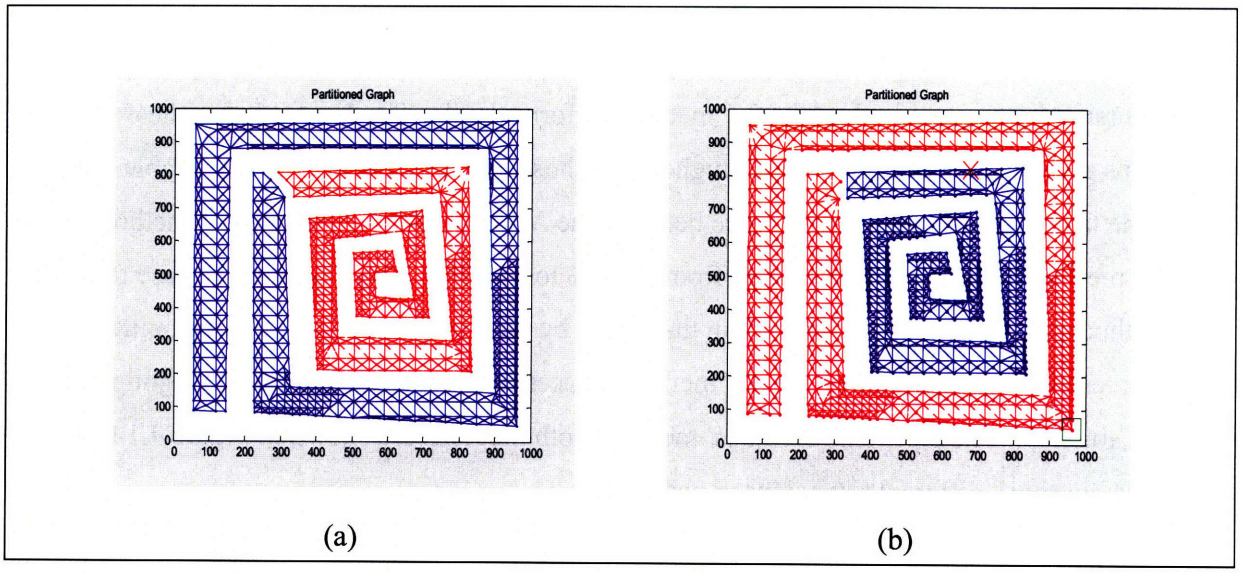


Figure 4.12 Graph partitioning of 'spiral' mesh using: (a) Fiedler method and (b) 0-1 method. In (b), the green square is the source and the red 'X' is the sink. The two methods give balanced and similar partitions. The difference between the two is subtle (a minor difference in the region around the cut).

The partitioning of *'smallmesh'* mesh by the Fiedler and 0-1 method are shown in Figure 4.13, (a) and (b) respectively. Similar to *'tapir'* mesh, the two methods give the same partition. As a result, the *Ncut* values given by the two methods are the same (Table 4.2, page 99). The same *Ncut* value tells us that this value is probably the minimum *Ncut* value we can achieve for this mesh.

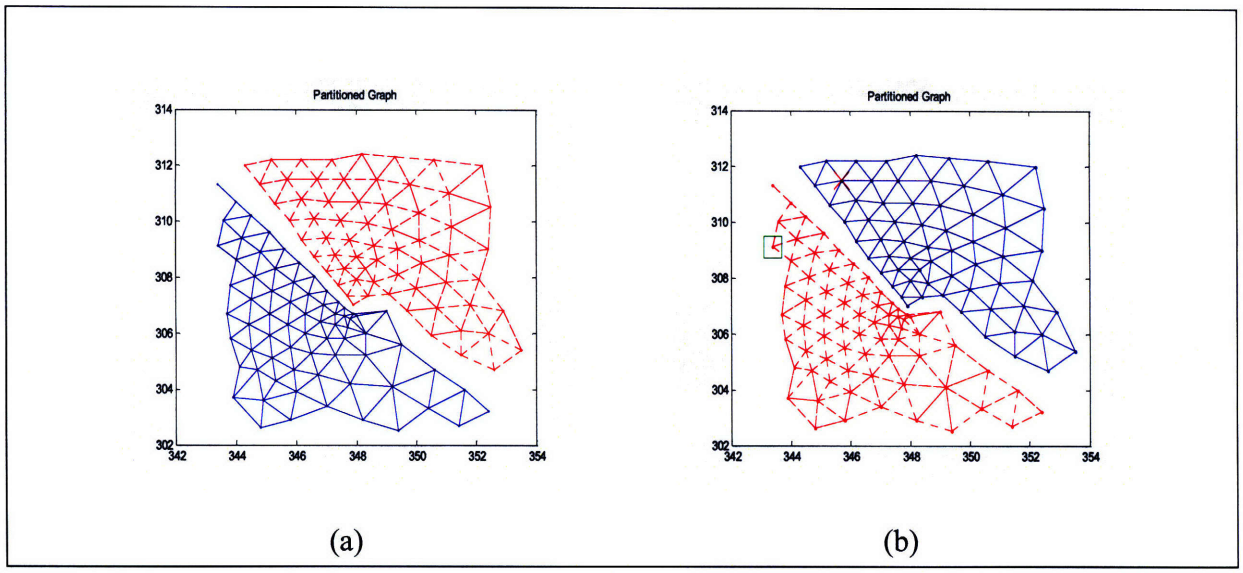


Figure 4.13 Graph partitioning of *'smallmesh'* mesh using: (a) Fiedler method and (b) 0-1 method. In (b), the green square is the source and the red 'X' is the sink. The two methods give the same partition.

b. k-way Graph Partitioning

For recursive *k*-way partitioning, the 0-1 method performs badly. The *Ncut* value of the 4-partitions given by 0-1 method is much higher than those given by Fiedler method (Table 4.3). From the table, we see that the difference between the *Ncut* values given by the 0-1 method and Fiedler method for four partitions range from -2.08% to 64.47%. There is only one case that the 0-1 method gives a *Ncut* value lower than that given by the Fiedler method. In partitioning *'crack'* mesh into four partitions, the 0-1 method reduces the *Ncut* value by 2.08% (highlighted in red in Table 4.3). The partitions of *'crack'* mesh by both methods are shown in Figure 4.14. Both methods give very different but balanced partitions.

Table 4.3 ***Ncut*** **values obtained by the Fiedler and 0-1** ***k*****-way partitioning for different graphs (meshes).**

Mesh	Number of nodes	Ncut		
		Fiedler	0-1	Change (%)
airfoil1	4253	0.0583	0.0881	51.11
airfoil2	4720	0.0727	0.0795	9.35
eppstein	547	0.2359	0.3391	43.75
tapir	1024	0.0833	0.1043	25.21
triangle	5050	0.0783	0.1054	34.61
crack	5120	0.0913	0.0894	-2.08
parc	1240	0.047	0.0773	64.47
parcweb	1939	0.0406	0.0614	51.23
spiral	1200	0.028	0.0352	25.71
smallmesh	136	0.4159	0.4182	0.55

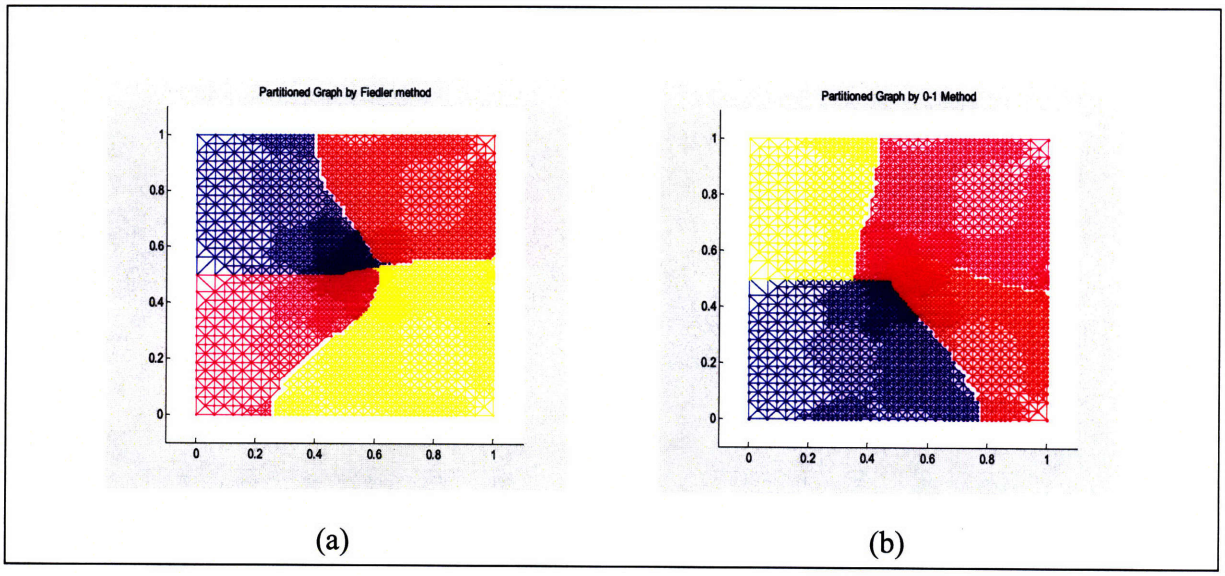


Figure 4.14 **The four partitions of 'crack' mesh given by (a) Fiedler method and (b) 0-1 method. Both methods give different but balanced partitions. Though the Fiedler method gives more balanced partitions, the 0-1 method has a lower** ***Ncut*** **value.**

The other meshes have increased *Ncut* values when they are partitioned by the 0-1 method. Among the meshes, *'airfoil1', 'eppstein', 'tapir', 'triangle', 'parc;, 'parcweb' and 'spiral'* have relatively large increase in *Ncut* value (over 10%) compared to those obtained using the Fiedler method (Table 4.3). The large differences are not acceptable. The reason for the poor performance is because the numbering of the node in the partitioned graphs are different. It may

not follow the initial convention of the initial graphs. Hence, the guess we use in the Auto 0-1 algorithm may not work well in finding good sinks and sources. The failure to locate good sinks and sources results in poor partitions (higher *Ncut* values) because the 0-1 method does not try to find a global minimum *Ncut* value. Instead, it finds a cut in between sinks and sources that gives the minimum *Ncut* value.

Apart from the *Ncut* values, the partitions given by the two methods differ. The differences are not solely because of the change in node numbering, but also because of the difference in partitions during the first bi-partitioning. If during the first bi-partitioning, the partitions given by both methods are different, the further bi-partitioning will give even more different partitions. This is the intrinsic characteristic of the recursive bi-partitioning. Figure 4.15 gives an example of this scenario.

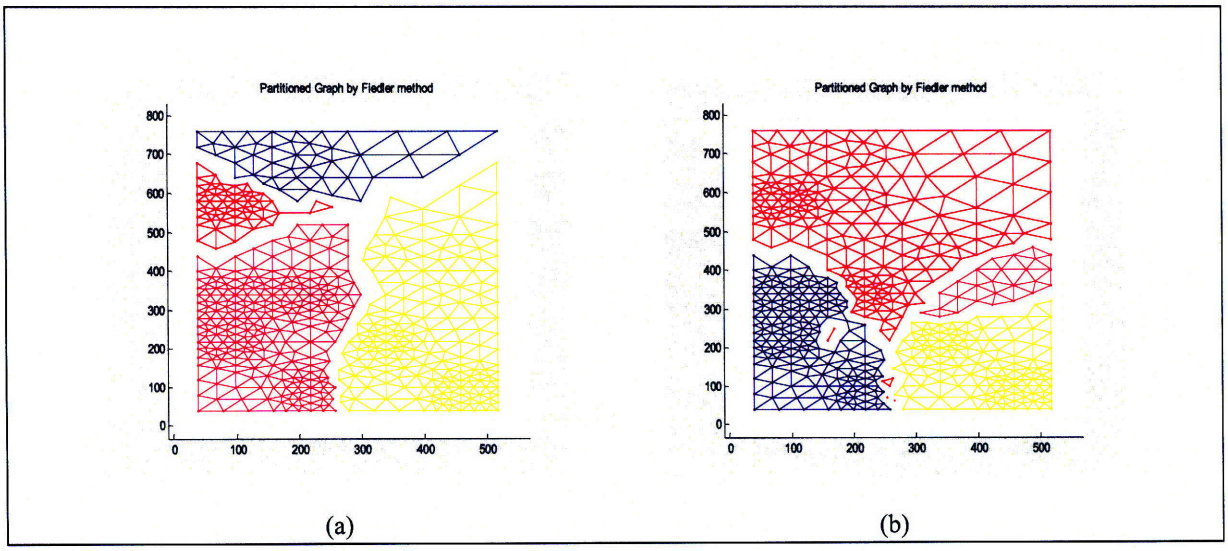


Figure 4.15 The four partitions of *'eppstein'* mesh given by (a) Fiedler method and (b) 0-1 method. Both methods give different partitions. The first bi-partitions by the two methods differ (Figure 4.6, page 102) and hence, the further bi-partitioning too gives different partitions.

Even though the first bi-partitions by the two methods are the same, there is no guarantee that the further bi-partitions will be the same as shown in the case of *'tapir'* mesh (Figure 4.16). However, there is a chance that the partitions will be similar as shown in the case of *'smallmesh'* (Figure 4.17).

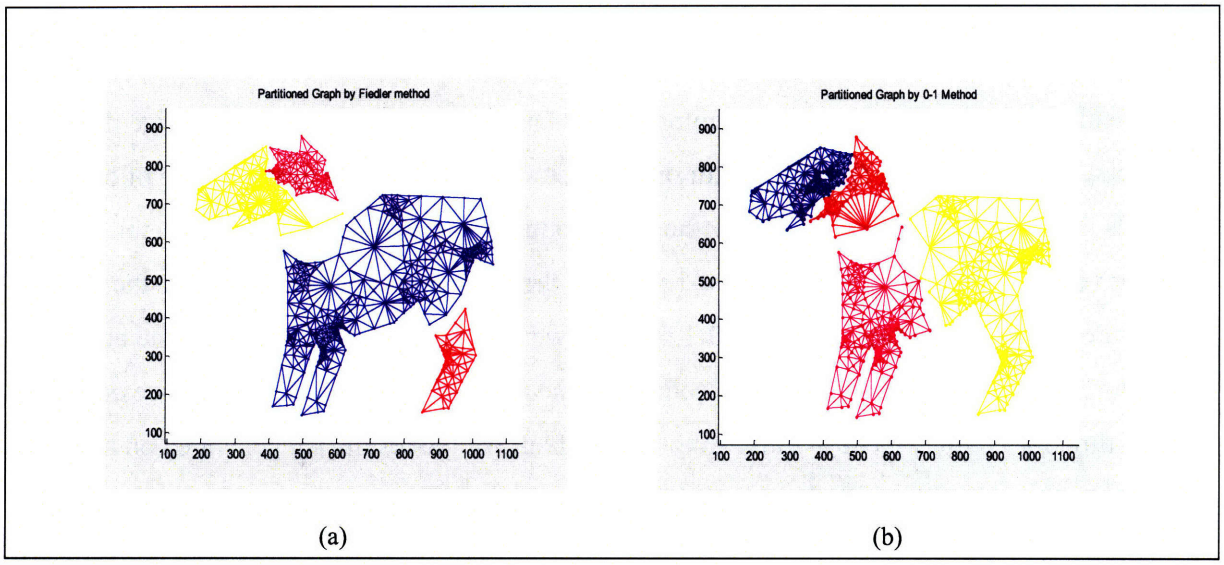


Figure 4.16 The four partitions of *'tapir'* mesh given by (a) Fiedler method and (b) 0-1 method. Both methods give different partitions. Though the first bi-partitions by the two methods are the same (Figure 4.6), the further bi-partitioning gives different partitions.

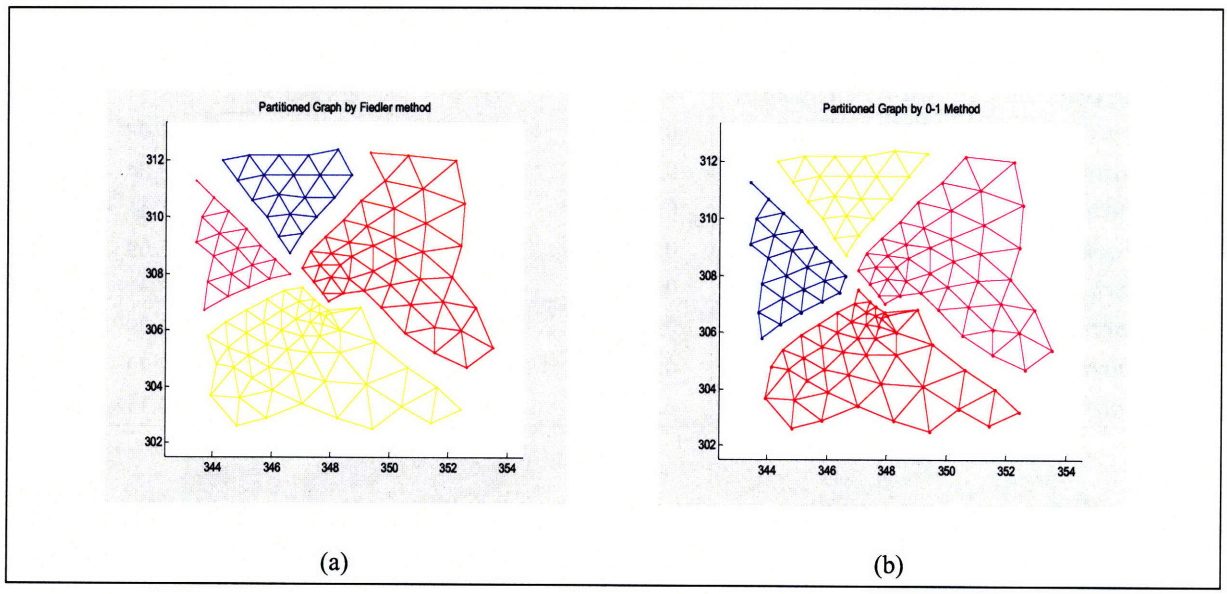


Figure 4.17 The four partitions of *'smallmesh'* mesh given by (a) Fiedler method and (b) 0-1 method. Both methods give similar partitions. The first bi-partitions by the two methods are the same (Figure 4.13, page 106), the further bi-partitioning gives similar partitions.

c. Run Time Comparison

Both Fiedler and 0-1 method use the Minimum *Ncut* Discretization scheme to discretize the continuous partition vector (Fiedler vector or 0-1 vector). Therefore, the running time of both methods is defined as the time for the methods to produce the continuous partition vector (Fiedler vector or 0-1 vector). For Auto 0-1 method, the algorithm needs to determine the best sink-source pair. In contrast, for Basic 0-1 method, we assume that we have known the best sink-source pair and hence skip the sink-source pair searching process. In Table 4.4, we compare the run time for Fiedler, Basic 0-1 and Auto 0-1 method. All the results are generated on a 2.2 GHz Pentium 4 computer with 768 MB RAM.

Table 4.4 Run time of Fiedler method, Basic and Auto 0-1 methods for different graphs (meshes). All the results are generated on a 2.2 GHz Pentium 4 computer with 768 MB RAM.

Mesh	Number of nodes	Running time				
		Fiedler	Basic 0-1	Change (%)	Auto 0-1	Change (%)
airfoil1	4253	0.9413	0.0195	-97.93	0.0997	-89.41
airfoil2	4720	1.2074	0.0241	-98.00	0.1215	-89.94
eppstein	547	0.0719	0.0026	-96.38	0.0122	-83.03
tapir	1024	0.1639	0.0036	-97.80	0.0196	-88.04
triangle	5050	0.7491	0.0292	-96.10	0.2891	-61.41
crack	5120	0.9874	0.0241	-97.56	0.1275	-87.09
parc	1240	0.3455	0.0042	-98.78	0.0231	-93.31
parcweb	1939	0.3949	0.0067	-98.30	0.0352	-91.09
spiral	1200	0.8934	0.0037	-99.59	0.0204	-97.72
smallmesh	136	0.0679	0.0009	-98.73	0.0040	-94.11

From Table 4.4, we see that generally the running times for Basic 0-1 method and Auto 0-1 method are shorter than that of the Fiedler method. Basic 0-1 method has over 90% improvement in running time while Auto 0-1 method has over 60% improvement in running time. The advantage in time for 0-1 method is because it only involves the solving of linear systems, whereas Fiedler method involves a generalized eigensystem. Auto 0-1 method takes longer than the Basic 0-1 method because extra time is needed to determine the best sink and source locations at the start of the Auto 0-1 algorithm.

Figure 4.18 shows the variation of the running time with the size of graphs for the Fiedler, Basic 0-1 and Auto 0-1 method. Generally the running time increases with the graph size. The Fiedler method shows the fastest increase rate with the graph size, followed by Auto 0-1 method and Basic 0-1 method. Judging from the higher increase rate in time for Fiedler method, we can see that the time required by Fiedler method for larger graphs will be prohibitive. Hence, we need a faster method like the 0-1 method to partition large graphs.

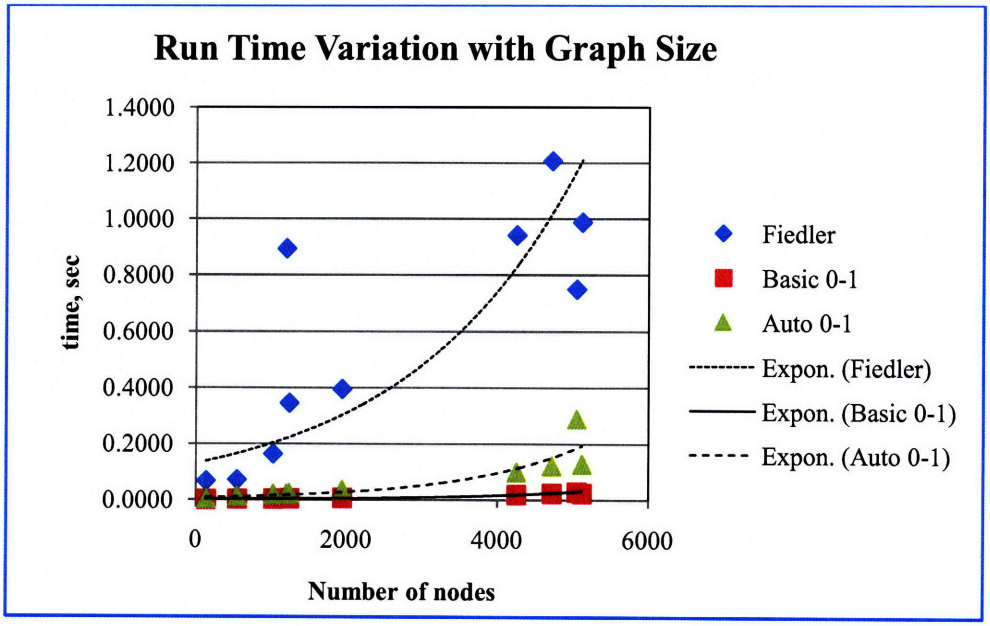


Figure 4.18 Variation of the run time with size of graphs for the Fiedler, Basic 0-1 and Auto 0-1 method. Exponential trend lines are added to show that the running time increase exponentially with the size of the graphs.

4.10 Application in Weighted Graph Partitioning

Weighted graphs have generally more applications than unweighted graphs. Some of the applications include data clustering, bioinformatics and image segmentation. For the application of the 0-1 method in weighted graph partitioning, I will focus specifically on its application in image segmentation in the next chapter.

4.11 Fiedler Quick Start using 0-1-method

From the previous example shown in Figure 4.3 (page 96), we observe that with a good sink and source, the 0-1 method can produce a 0-1 vector that is close to the Fiedler vector. Since we know that the 0-1 method has the time advantage, we can use the 0-1 method to find the Fiedler vector without solving an eigensystem. In this way, we hope to reduce the computation time. We can achieve this by performing inverse power and Rayleigh quotient iterations on the 0-1 vector so that it converges to the Fiedler vector. Since the 0-1 vector is a good approximation to the Fiedler vector (when good sink and sources are used), the convergence is fast. In the case of bad sink and sources, which is the worst case, the time advantage of this method vanishes (more iterations are needed). Combining the ideas of 0-1 method and inverse power iteration, I have developed the Fiedler Quick Start Algorithm. The algorithm is summarized below:

Algorithm 4.5

Given an n-node graph with Laplacian matrix, $\boldsymbol{L}$; its degree matrix, $\boldsymbol{D}$; and parameters $(\alpha, \beta, \gamma, \mu, \sigma)$:

i. Run the Auto 0-1 Algorithm to obtain the 0-1 vector, $\boldsymbol{v_o}$ and $Ncut$

ii. $\boldsymbol{v_o} = \boldsymbol{v_o} - \mathbf{0.5}$, $\boldsymbol{v_o} = \boldsymbol{v_o} / |\boldsymbol{v_o}|$

iii. $\lambda_0 = \alpha * Ncut$

iv. $\boldsymbol{L} = \boldsymbol{D}^{-1/2} * \boldsymbol{L} * \boldsymbol{D}^{-1/2}$, set $k = 1$

v. Inverse power method: $\boldsymbol{v} = (\boldsymbol{L} - \lambda_0 * \boldsymbol{I})^{-1} * \boldsymbol{v_0}$, $\boldsymbol{v} = \boldsymbol{v}/|\boldsymbol{v}|$

vi. Rayleigh quotient: $\lambda' = (\boldsymbol{v}^T * \boldsymbol{L} * \boldsymbol{v}) / (\boldsymbol{v}^T * \boldsymbol{v})$

vii. If $|\lambda_0 - \lambda'| / |\lambda_0| < \beta$, set $\lambda_k = \lambda'$

Else, set $\lambda_0 = \lambda'$, and go to (vi)

viii. Set $\lambda_0 = \lambda_k/\gamma$, $k = k + 1$

ix. Inverse power method: $\boldsymbol{v} = (\boldsymbol{L} - \lambda_0 * \boldsymbol{I})^{-1} * \boldsymbol{v_0}$, $\boldsymbol{v} = \boldsymbol{v}/|\boldsymbol{v}|$

x. Rayleigh quotient: $\lambda' = (\boldsymbol{v}^T * \boldsymbol{L} * \boldsymbol{v}) / (\boldsymbol{v}^T * \boldsymbol{v})$

xi. If $|\lambda_k - \lambda'|/|\lambda_k| > \mu$, set $\lambda_0 = \lambda'$, go to (viii)

xii. If $\lambda_k < 10^{-10}$, $\lambda_k = \lambda_{k-1}$

xiii. Inverse power method: $\boldsymbol{v} = (\boldsymbol{L} - \lambda_k * \boldsymbol{I})^{-1} * \boldsymbol{v_0}$, $\boldsymbol{v} = \boldsymbol{v}/|\boldsymbol{v}|$

xiv. If $(\|\boldsymbol{v}\| - \|\boldsymbol{v_0}\|)/\|\boldsymbol{v_0}\| > \sigma$, $\boldsymbol{v_0} = \boldsymbol{v}$, go to (xiii)

xv. $\boldsymbol{v} = \boldsymbol{D}^{-1/2} * \boldsymbol{v}$

4.11.1 Initial eigenvector and eigenvalue guess

The 0-1 vector has its elements distributed between 0 and 1. According to Shi and Malik [3], the Fiedler vector is a relaxed solution to the discrete partition vector of -1 and 1 for a segmented graph. Hence, the Fiedler vector has positive and negative elements. In order to approximate the Fiedler vector, the 0-1-method vector has to be shifted by 0.5 so that it has positive and negative elements too (step (ii)). Since the Fiedler vector is $\boldsymbol{D}$-orthogonal, the 0-1-method vector is normalized and should be multiplied by $\boldsymbol{D}^{-1/2}$ (step (ii)). However, in latter steps, the vector will need to be multiplied by $\boldsymbol{D}^{1/2}$ to convert the generalized eigensystem to a normal eigensystem (step (iv)). Hence, to avoid the redundant computation, the step of $\boldsymbol{D}^{-1/2}$ multiplication is skipped.

The eigenvalue corresponding to the Fiedler vector is the relaxed solution to the minimum *Ncut* value and also the lower bound to the minimum *Ncut* value [12]. Based on this fact, we can use the *Ncut* value obtained from the 0-1-method as the initial guess of the Fiedler eigenvalue, λ_0. Since the eigenvalue is often smaller than the *Ncut* value, we can use a fraction, α of the *Ncut* value as the initial guess for the eigenvalue λ_0 (step (iii)).

4.11.2 Inverse power and Rayleigh quotient method

a. Eigenvalue iteration

Using the initial eigenvector guess, $\boldsymbol{v_o}$ and eigenvalue guess, λ_0, step (iv) to (vii) are repeated until the eigenvalue is converged. If the initial eigenvector is close to the Fiedler vector, the converged eigenvalue should be close to the Fiedler eigenvalue. The stopping criterion is

controlled by a parameter β in step (vii). For speed consideration, the iteration is also capped at five iterations. This may affect the eigenvector convergence rate later.

b. Second lowest eigenvalue check and iteration

To ensure that the converged eigenvalue, λ_k is the second lowest eigenvalue, a check (step (viii) - (xi)) is performed. We reduce the converged eigenvalue, λ_k by dividing it by γ (step (viii)) and repeat the inverse power and Rayleigh quotient iteration using the reduced eigenvalue, λ_0 and the initial eigenvector, $\boldsymbol{v_o}$ to obtain a new eigenvalue, λ'. By doing this, we are shifting the eigenvalue to a smaller eigenvalue. If there is such a smaller eigenvalue, the shifting should cause a large change in the new eigenvalue. Based on this argument, if the eigenvalue changes by a magnitude larger than μ, the current eigenvalue, λ_k may not be the second lowest eigenvalue (step (xi)) and we need to search for a smaller eigenvalue. Inverse power and Rayleigh quotient iteration are performed again until the eigenvalue is converged again. If the change is smaller than μ, the current converged eigenvalue, λ_k is indeed the second lowest eigenvalue. At the end of the check, if the converged eigenvalue, λ_k is zero (numerically close to zero, less than 10^{-10}), the previous converged eigenvalue, λ_{k-1} is the second lowest eigenvalue (step (xii)).

c. Eigenvector iteration

With the eigenvalue, λ_k obtained in the eigenvalue iteration, the inverse power iteration is used to obtain the corresponding eigenvector—the Fiedler vector, $\boldsymbol{v}$ (step (xiii)). The convergence will be fast as the correct eigenvalue is used. The stopping criterion is controlled by parameter σ. For speed consideration, the iteration is capped at ten iterations. The effect is less accurate Fiedler vector. However, this is justifiable because in graph partitioning, only the shape (or sign) of the Fiedler vector plot matters [12].

4.11.3 Experiment and Results

a. Setup

In this experiment, I compare the Fiedler vector obtained by the usual generalized eigensystem and the Fiedler Quick Start algorithm by looking at the vector plot and the Fiedler eigenvalue. Apart from this, the running times for both methods are compared as well. The graph used in the experiment is the meshes obtained from FTP site of John Gilbert and the Xerox Corporation[3].

Since we only concern about the general shape of the Fiedler vector plot or sign of the vector elements in graph partitioning, the tolerance of convergence for both methods are reduced. Another purpose of this reduced tolerance is to boost the speed in computing the Fiedler vector. For the generalized eigensystem solved in MATLAB using ARPACK package, the tolerance is set to be $10^{15} * \varepsilon$ (ε = 2.2204e-016). The tolerance is low enough to maintain the general shape of the Fiedler vector plot. For the Fiedler Quick Start algorithm, eigenvalue iteration and eigenvector iteration are capped at five and ten iterations.

The parameters used in this experiment are: $\alpha = 0.1$, $\beta = 10^{-2}$, $\gamma = 1.5$, $\mu = 0.1$ and $\sigma = 10^{-2}$. These parameters are set to balance the accuracy and speed.

b. Fiedler Vector Comparison

Table 4.5 shows the result of the numerical comparison of Fiedler vector obtained by solving the generalized eigensystem (Fiedler method) and using the Fiedler Quick Start algorithm. I use the 2-norm of the eigenvectors difference as a measure of comparison. The closer the two eigenvectors, the smaller the 2-norm will be. The eigenvalue is also used to verify if the vector obtained from the Fiedler Quick Start is an eigenvector.

[3] *ftp://ftp.parc.xerox.com/pub/gilbert/meshes.tar.Z*

Table 4.5 **Comparison of the Fiedler vector and its eigenvalue obtained by the generalized eigensystem and the Fiedler Quick Start algorithm.**

Mesh	Number of Nodes	Eigenvalue		||Eigenvector Difference||
		Fiedler	0-1	
airfoil1	4253	3.2037E-04	3.2037E-04	1.6168E-05
airfoil2	4720	3.9569E-04	3.9568E-04	1.9215E-04
eppstein	547	3.7000E-03	3.7000E-03	2.5351E-06
tapir	1024	1.2000E-03	1.2000E-03	9.0790E-04
triangle	5050	4.4601E-04	4.4601E-04	2.4170E-01
crack	5120	5.1716E-04	5.1716E-04	2.5624E-05
parc	1240	6.2466E-04	8.7991E-04	6.1400E-01
parcweb	1939	5.5547E-04	5.5547E-04	2.3191E-04
spiral	1200	5.7496E-05	5.7496E-05	8.7741E-08
smallmesh	136	9.0000E-03	9.0000E-03	3.3281E-10

Figure 4.19 and Figure 4.20 show the graphical comparison between the Fiedler vectors obtained by the two methods for the meshes. From Table 4.5, Figure 4.19 and Figure 4.20, we can observe that only two out of ten meshes (highlighted) have different Fiedler vectors from that given by the Fiedler method. For mesh '*airfoli2*' and *'tapir'*, the sign is different because the shifting in the inverse power method causes negative eigenvalues. For the '*triangle'* mesh, we observe that both methods give same eigenvalue. Since the eigenvector for a matrix is not unique, we can conclude that the Fiedler vector obtained from the 0-1-method is correct. Hence, the only failure for the method is caused by the *'parc'* mesh. The reason for this failure is because the difference between the starting vector and the Fiedler vector is great (Figure 4.21, page 119). For other meshes, the norm of the Fiedler vector difference between the two methods varies from the order of 10^{-4} to 10^{-10}.

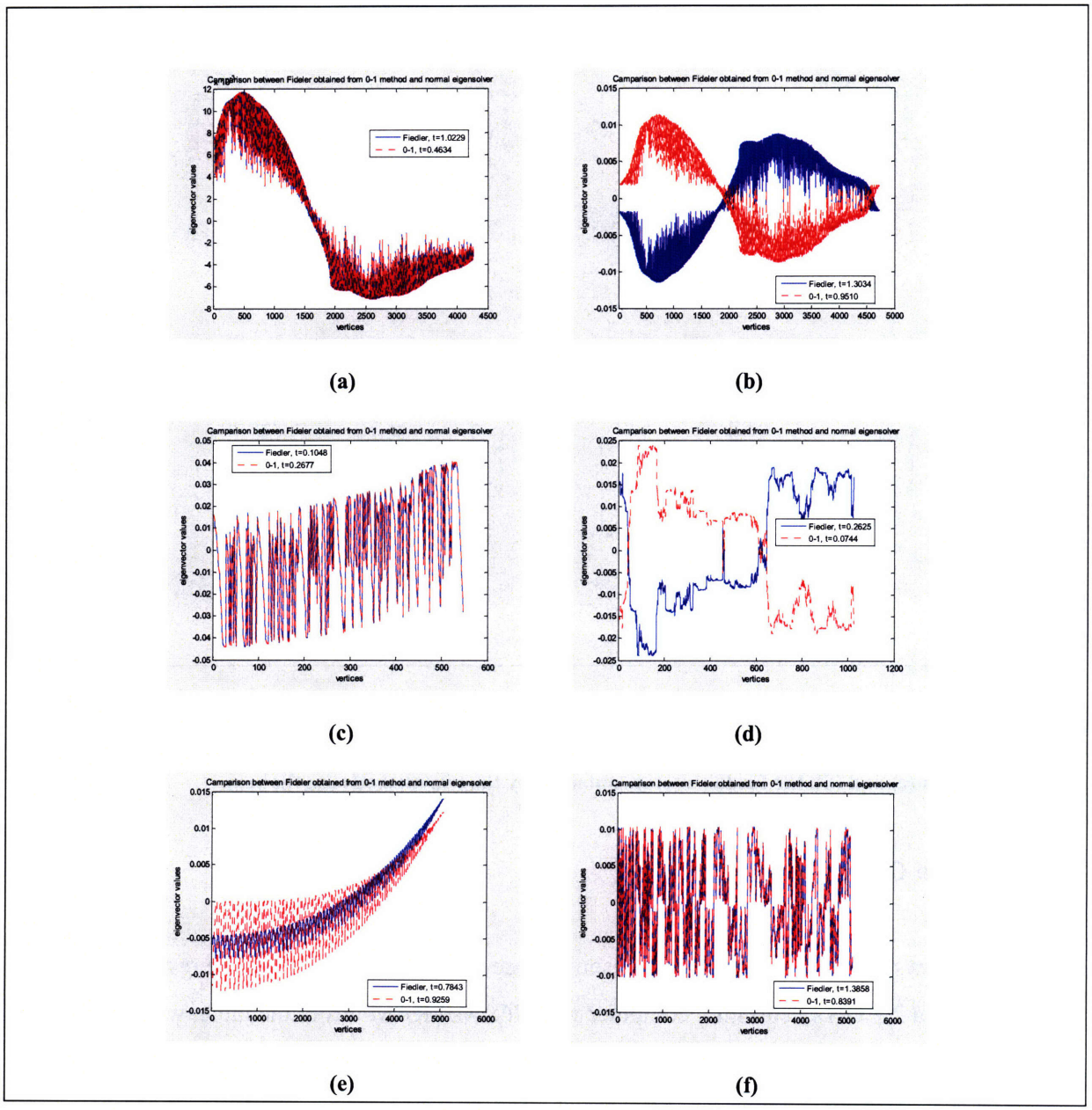


Figure 4.19 **Fiedler vector plots obtained using the Fiedler method and the Fiedler Quick Start method for different meshes: (a) *'airfoil1'*, (b) *'airfoil2'*, (c) *'eppstein'*, (d) *'tapir'*, (e) *'triangle'*, and (f) *'crack'*. The blue curves are the Fiedler vector given by the Fiedler method while the red curves are the Fiedler vector given by the Fiedler Quick Start method. For mesh (a), (c) and (f), the Fiedler vectors obtained by the two methods are the same. For mesh (b) and (d), the vectors has same magnitudes but opposite signs. For mesh (e), the vectors are different but share the same trend.**

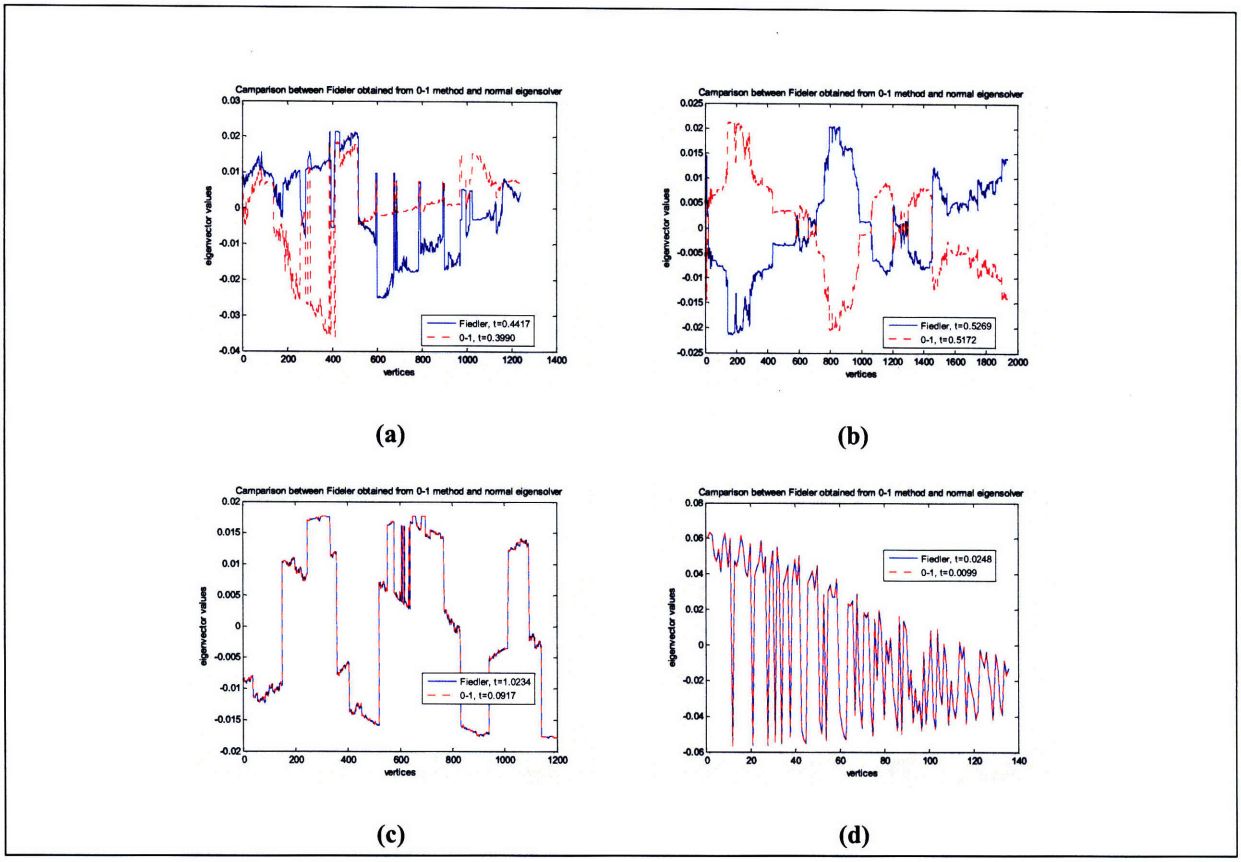


Figure 4.20 Fiedler vector plots obtained using the Fiedler method and the Fiedler Quick Start method for different meshes: (a) *'parc'*, (b) *'parcweb'*, (c) *'spiral'*, and (d) *'smallmesh'*. For mesh (a), the vectors are different. For mesh (a), the vectors are different. For mesh (b), the vectors has same magnitudes but opposite signs. For mesh (c) and (d), the Fiedler vectors obtained by the two methods are the same.

c. Run Time Comparison

Table 4.6 shows the running time comparison between the two methods. All the results are generated on a 2.2 GHz Pentium 4 computer with 768 MB RAM. From the table, we can observe that the Fiedler vectors of almost all the meshes, except two meshes (highlighted), can be obtained in shorter time using the Fiedler Quick Start method compared to those obtained using the usual Fiedler method. The time advantage varies from 1% to 90% improvement.

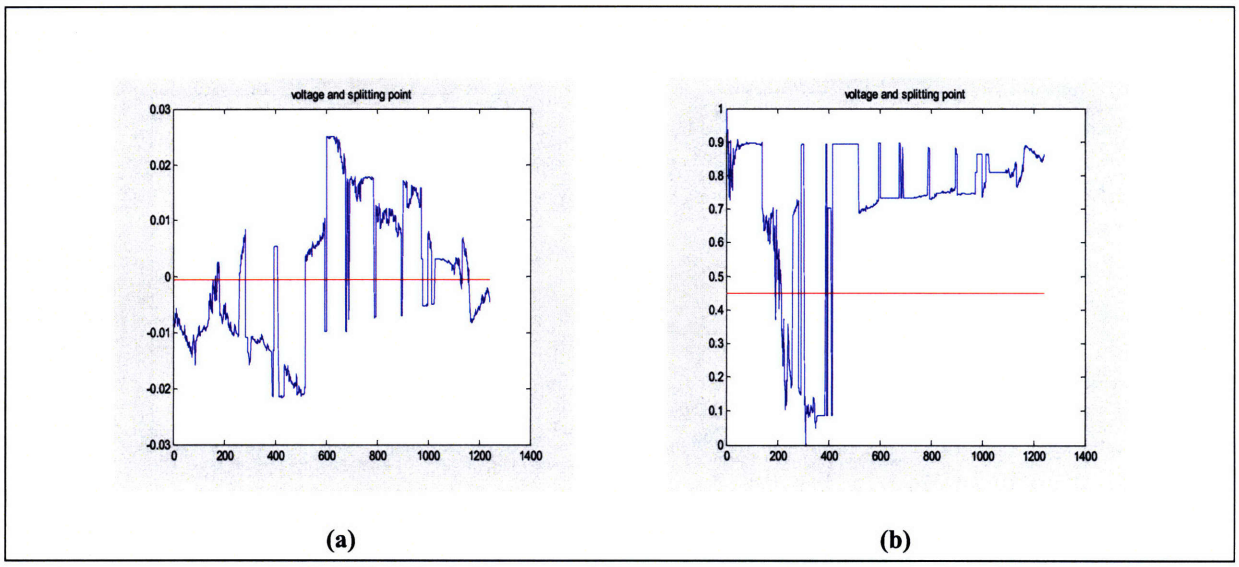


Figure 4.21 **(a) The Fiedler vector plot (blue curve) and its splitting point (red line) for *'parc'* mesh. (b) The 0-1-method vector plot (blue curve) and its splitting point (red line) for *'parc'* mesh. The two vector plots bear little resemblance.**

Table 4.6 **Running time comparison between the Fiedler method and Fiedler Quick Start method. All the results are generated on a 2.2 GHz Pentium 4 computer with 768 MB RAM.**

Mesh	Number of Nodes	Time		Time Changes (%)
		Fiedler	0-1	
airfoil1	4253	1.0229	0.4634	-54.70
airfoil2	4720	1.1046	0.9510	-13.91
eppstein	547	0.1048	0.2677	155.44
tapir	1024	0.2625	0.0744	-71.66
triangle	5050	0.7843	0.9259	18.05
crack	5120	1.3858	0.8391	-39.45
parc	1240	0.4417	0.3990	-9.67
parcweb	1939	0.5269	0.5172	-1.84
spiral	1200	1.0234	0.0917	-91.04
smallmesh	136	0.0248	0.0099	-60.08

The time advantage depends on the 0-1 vector. The closer the vector to the Fiedler vector, the faster the Fiedler Quick Start method obtains the Fiedler vector. For example, the 'spiral' mesh has the best time advantage because of the resemblance of its 0-1-method vector (starting vector) to the Fiedler vector (Figure 4.22).

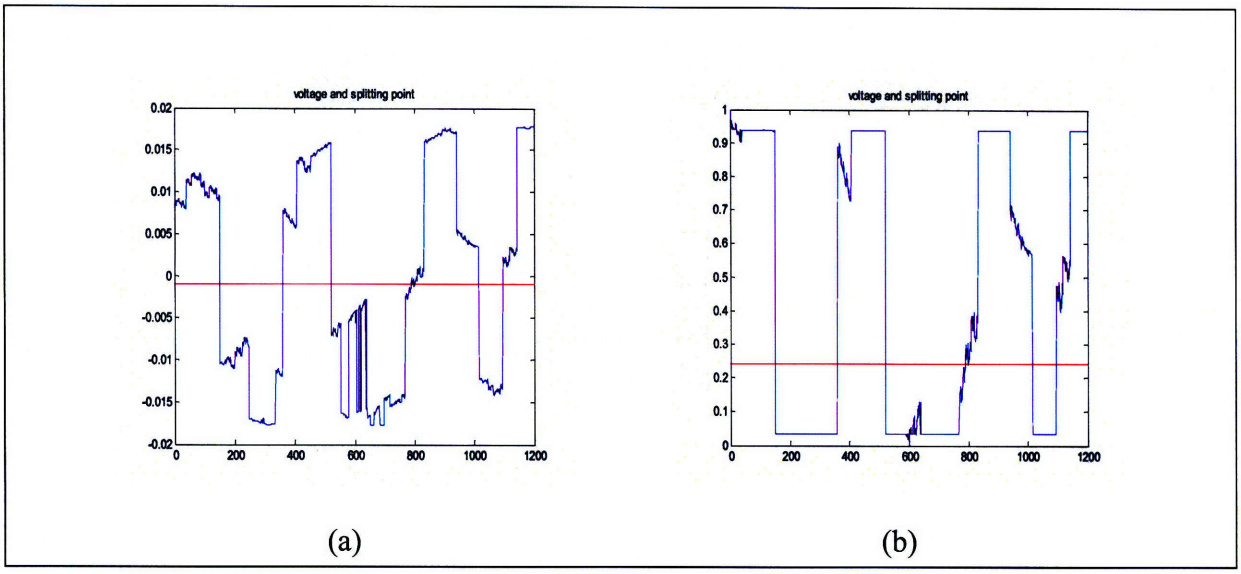


Figure 4.22 (a) The Fiedler vector plot (blue curve) and its splitting point (red line) for *'spiral'* mesh. (b) The 0-1-method vector plot (blue curve) and its splitting point (red line) for 'spiral' mesh. The two vector plots resemble each other.

For *'triangle'* and *'eppstein'* meshes, the Fiedler Quick Start method takes a longer time to obtain the Fiedler vectors. Again, the reason is because of the difference between the 0-1 vector and the Fiedler vector (Figure 4.23 and Figure 4.24).

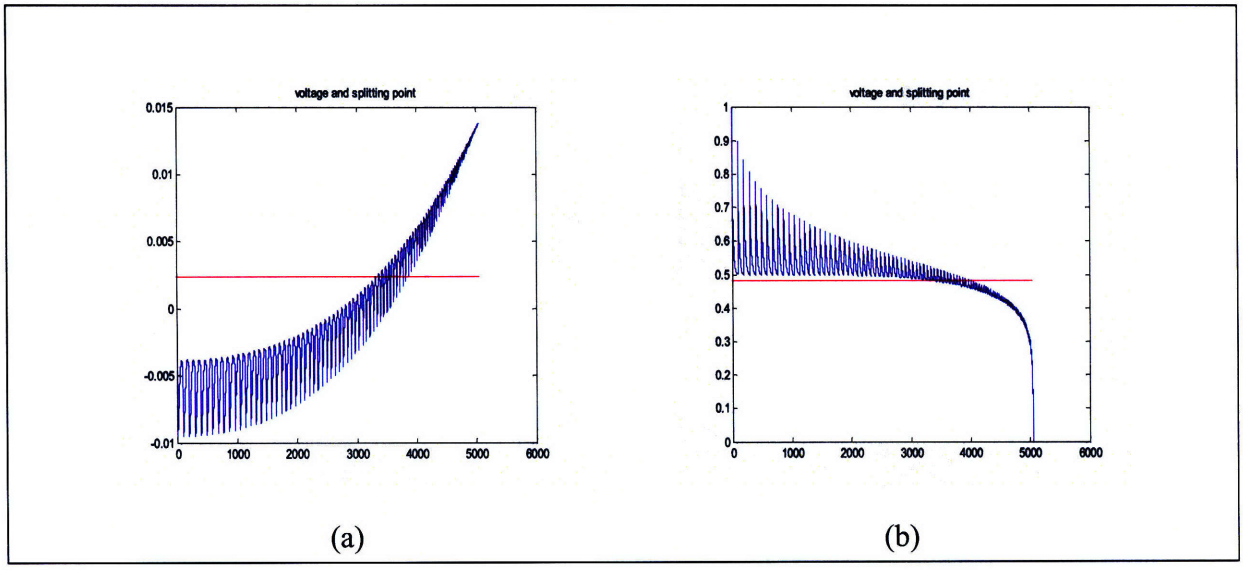


Figure 4.23 (a) The Fiedler vector plot (blue curve) and its splitting point (red line) for *'triangle'* mesh. (b) The 0-1-method vector plot (blue curve) and its splitting point (red line) for 'triangle' mesh. The two vector plots have a big difference (two different trends: increasing and decreasing).

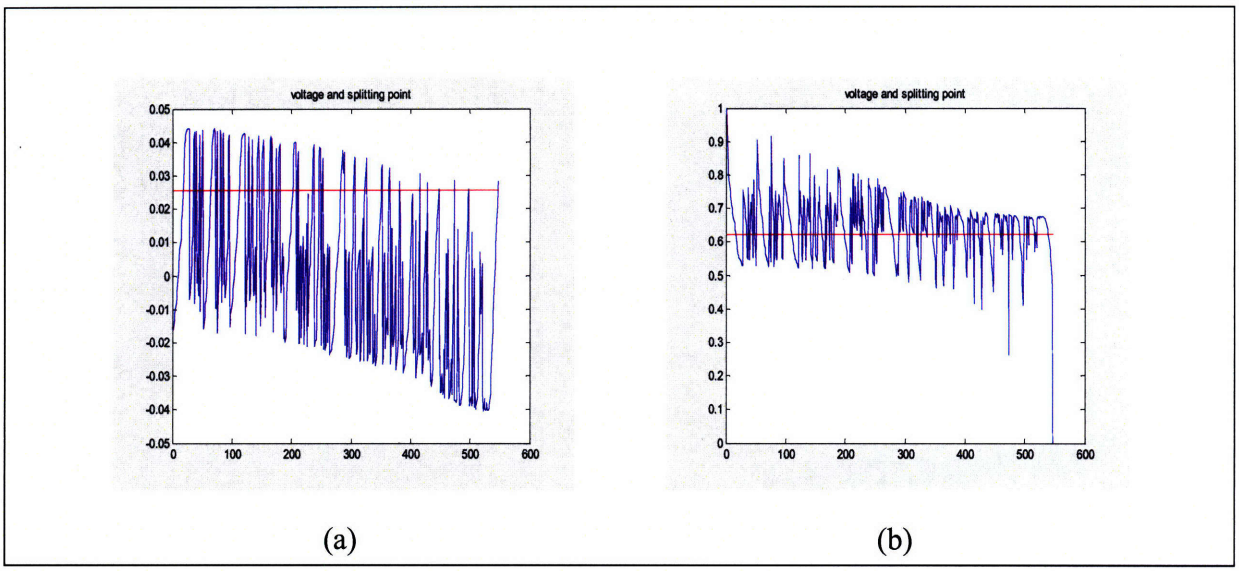


Figure 4.24 (a) The Fiedler vector plot (blue curve) and its splitting point (red line) for *'eppstein'* mesh. (b) The 0-1-method vector plot (blue curve) and its splitting point (red line) for *'eppstein'* mesh. Though the two vector plots looks similar in the global trend, there exist a crucial difference. Looking at the first and last few elements of each plot, we can observe that the local trend is opposite. For plot (c), the trend is increasing while for plot (d), the trend is decreasing.

4.12 Summary

For unweighted graph partitioning, the 0-1 method is able to give bi-partitions of comparable quality to that of the Fiedler method. Though the partition quality is not guaranteed to be better than the Fiedler method, the 0-1 method is quicker than the Fiedler method.

In the case of recursive k-way partitioning, the 0-1 method performs badly. This is because the assumption that the usual node numbering convention is applied in the graph is no longer true in the partitioned graph. Furthermore, the existing difference in the first bi-partitions between Fiedler and 0-1 methods aggravates the difference when the partitioned graphs are further bi-partitioned by the two methods.

The Fiedler Quick Start has the potential to compute the Fiedler vector faster than solving the generalized eigensystem. However, the method is still not robust enough as a few parameters are involved. Apart from this, the success of this method relies on the starting 0-1 vector. In order to obtain a good starting 0-1 vector, a good pair of sink and source is needed. To improve the

overall algorithm, more improvements should be done on the determination of a good sink and source pair.

Chapter 5 Image Segmentation using 0-1 Graph Partitioning

In this chapter, I try to apply the developed 0-1 Graph Partitioning to image segmentation. To ensure that the method is up to the standard, I repeat the tests in Chapter 3 using the 0-1 method.

5.1 Performance Tests

To apply the 0-1 method in image segmentation, one of the sink or source must be placed inside the object. For the test image used in Chapter 2, I placed four sinks at the four corners of the constructed graph and a source at the center of the graph (center of the square). The rationale behind putting four sinks at the four corners of the constructed graph is that it helps to reduce the variation of voltage across the homogeneous background. Hence, the four sinks are located at the four corners of the background as shown in Figure 5.1.

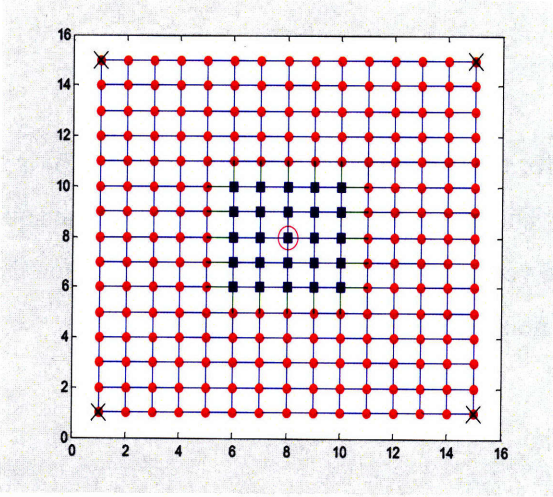


Figure 5.1 For 0-1 method, the source (pink circle) is located at the center of the graph while the four sinks are located at the four corners of the graph.

5.1.1 Pixel Intensity

The segmentation results of the 0-1 method for different pixel intensity differences are shown in Figure 5.2. The minimum pixel intensity required by this method is above 10. The failure mode of the 0-1 method is similar to that of the Isoperimetric Partitioning. The observed circular segmentation in the last row of Column (d) is because of the even decrease of voltage in all directions away from the source.

From Figure 5.2, we see that the voltage drop across the border of the square decreases when the intensity difference decreases. This can be seen in the partition vector plots in Column (b). There is a sudden jump in voltage in the first plot. The sudden jump is barely noticeable in the second plot, and in the last plot, the jump vanishes. When the sudden jump disappears, the square in the image before discretization also becomes unobservable (Figure 5.2).

5.1.2 Image Size

The image size test result for the 0-1 method (Figure 5.3, page 126) is similar to that of the Isoperimetric Partitioning. The method successfully segmented the square for image size up to 900 x 900 and the partition vector is still discrete. Hence, I conclude that, for the pixel intensity difference of 100, there is no size limit for the 0-1 method.

5.1.3 Noise

Figure 5.4 (page 127) shows the results of the noise test undergone by the 0-1 method. Basically the result is similar to the result by the Isoperimetric partitioning. The 0-1 method has survived the 'Gaussian', 'Poisson' and 'Speckle' noise tests. The method fails at the 'Salt & Pepper' noise test. This is again due to the badly conditioned Laplacian matrix constructed from the noisy image.

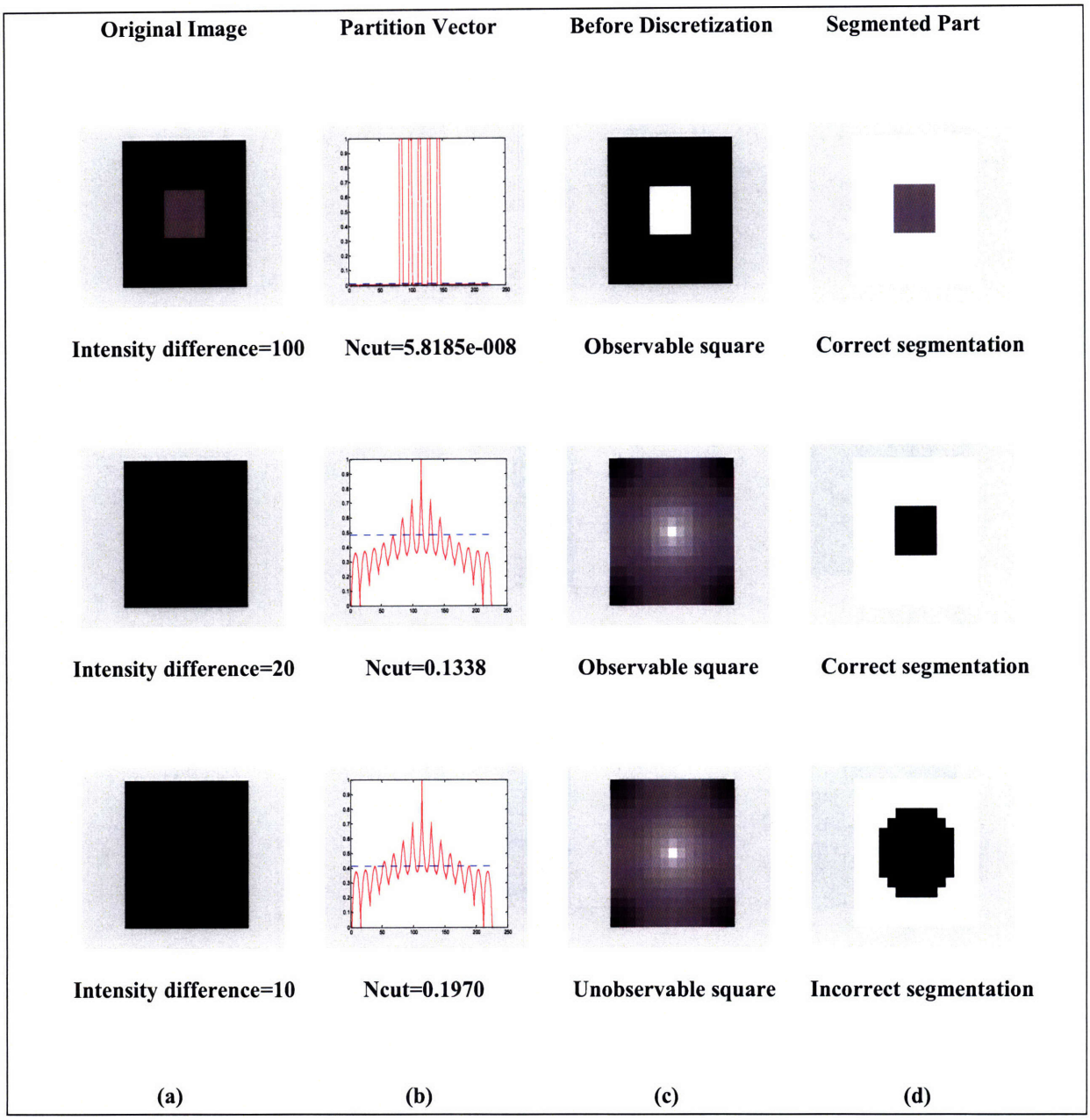


Figure 5.2 Image Segmentation by the 0-1 method. Column (a) shows the original images before segmentation and the pixel intensities of the square. Column (b) shows the partition vector plots (red plots) and the splitting points that give the minimum Normalized Cut values (blue horizontal lines). The Normalized Cut values are given at the bottom of each plot in column (b). Column (c) shows the images given by the continuous partition vector before discretization. Column (d) shows the segmented parts from the image (the square). The method fails to segment out the center square when the pixel intensity of the square is 10 (Row 3). Notice that when the square is still observable in Column (c), the center square can be segmented out in Column (d). Also notice the *Ncut* value of the last row. It is interestingly smaller than the *Ncut* value of the correct segmentation (0.1998). The *Ncut* is not the absolute measurement for correct segmentation.

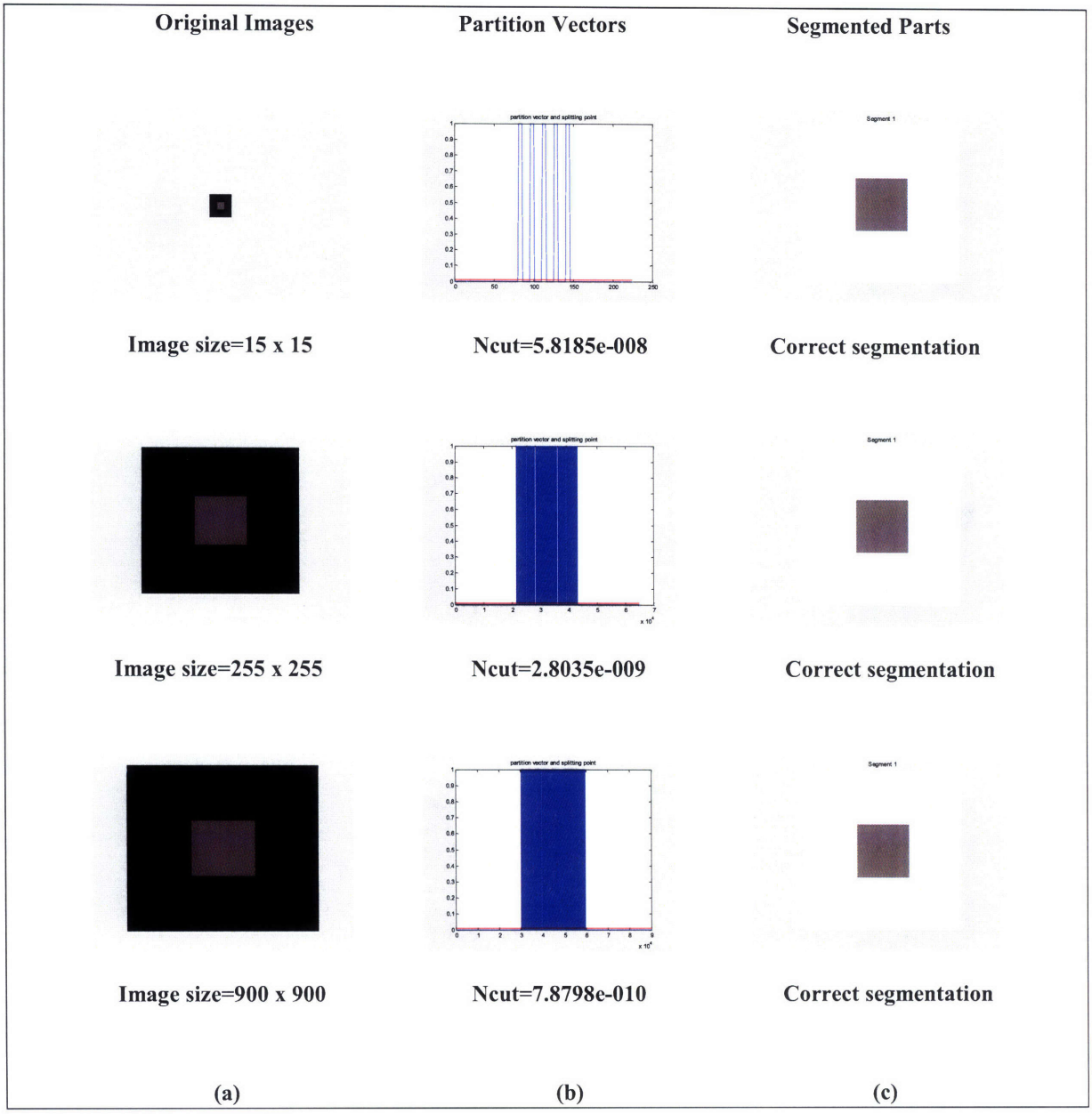


Figure 5.3 Image Segmentation by the 0-1 method for different image sizes. Column (a) shows the original images before segmentation and the pixel intensities of the square. Column (b) shows the partition vector (blue plots) and the splitting point that gives the minimum Normalized Cut value (red horizontal lines). The Normalized Cut values are given at the bottom of each plot in column (b). Column (c) shows the segmented parts from the images (the square). For all the image size up to 900 x 900, the 0-1 method performs the segmentation correctly, as shown in Column (c). With the correct segmentation, the *Ncut* value decreases with the increasing image size. Notice that distinct peaks are observed in the vector plot (Column (b)). They allow the splitting point to cut through it easily.

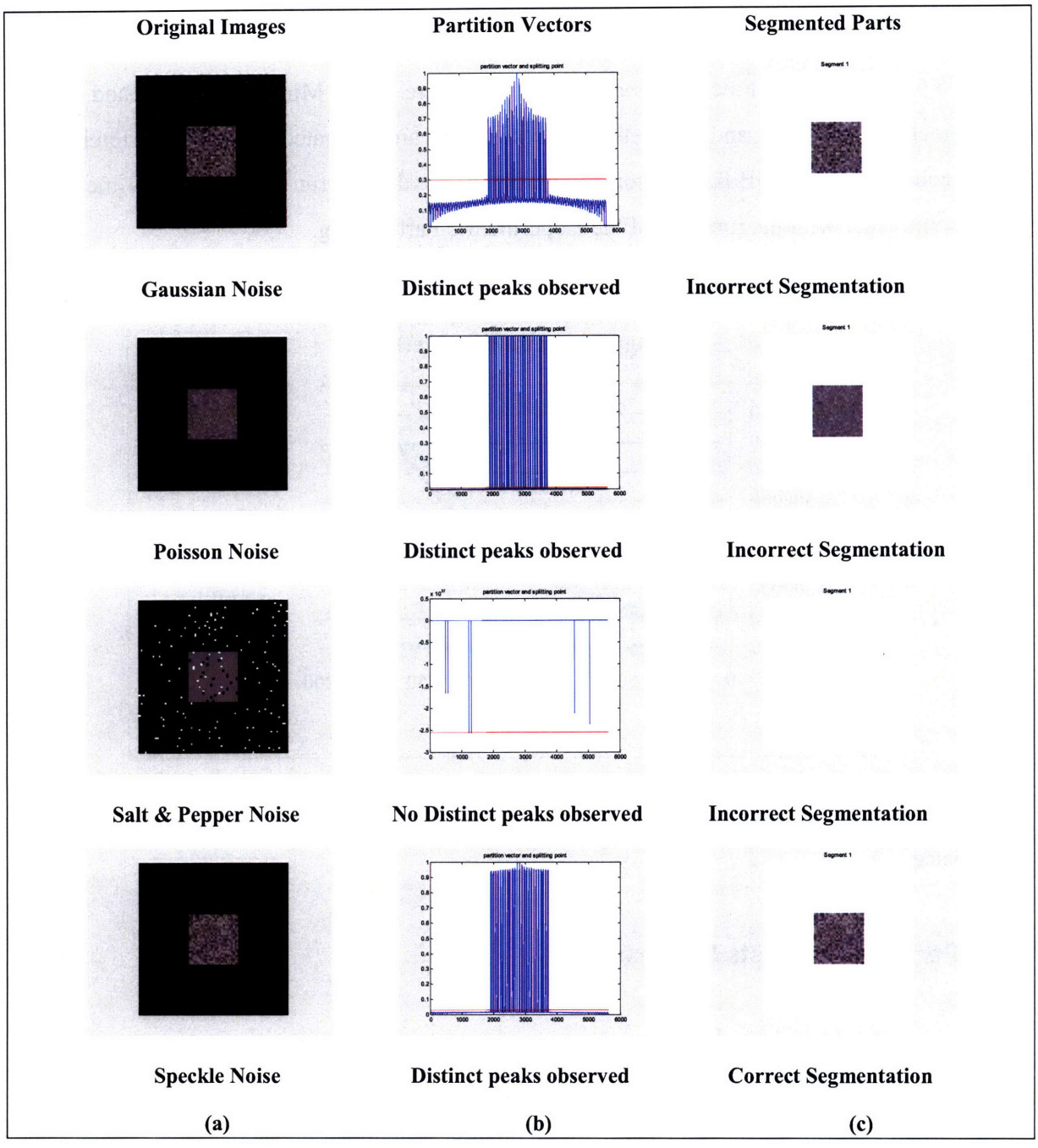


Figure 5.4 Image Segmentation by the 0-1 method under four different noise types. Column (a) shows the original images affected by the noise before segmentation. The noise types are stated at the bottom of the images. The noise types used are: *'Gaussian', 'Poisson', 'Salt & Pepper'* and *'Speckle'* (Row 1 to 4). Column (b) shows the segmented parts from the images (the squares). The method successfully segments out the center square from the images affected by *'Gaussian', 'Poisson'* and *'Speckle'* noise (Row 1, 2 and 4), but fails to segment the image affected by *'Salt & Pepper'* noise (Row 3). Notice that distinct peaks are observed in the vector plot (Row 1, 2 and 4 of Column (b)) when the segmentation is successful. They allow the splitting point to cut through it easily.

5.1.4 Speed

Figure 5.5 shows the run time variation with the image size for the Minimum Cut method, Isoperimetric Partitioning and 0-1 method. All the results are generated on a 2.0 GHz Intel Core 2 Duo computer with 3 GB RAM. From the graph, we see that the run time for the 0-1 method is in the same order with the run time of the Isoperimetric Partitioning.

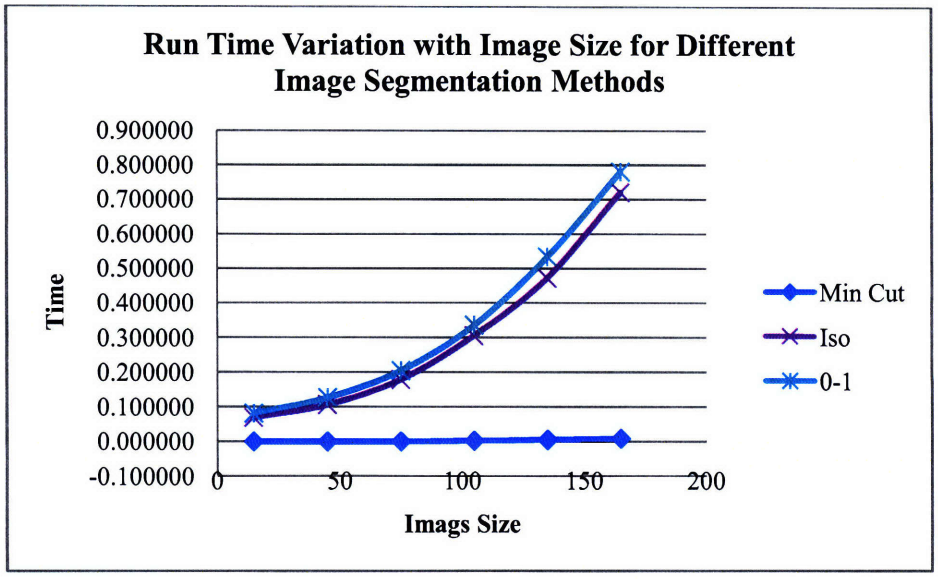


Figure 5.5 Run Time Variation with Image Size for the Minimum Cut method, Isoperimetric Partitioning and 0-1 method. The run time of the 0-1 method is similar to that of the Isoperimetric Partitioning.

5.1.5 Performance Tests Summary

The similarity between the results by the 0-1 method and Isoperimetric Partitioning is not surprising as the 0-1 method is developed from the Isoperimetric Partitioning. Not only they share the same advantage in speed and partition quality, they both too face the same problem: the placement of sink and sources. In conclusion, the image segmentation capability of the 0-1 method is on par with the Isoperimetric Partitioning.

5.2 Sinks and Sources' Locations

From previous discussion, we know that the sinks and sources' locations are important for good partitioning. In image segmentation, we want to extract objects in an image. To achieve this purpose, one of the sink or sources must be placed inside the objects. In the previous section, I manually placed a source inside the to-be-extracted objects and four sinks at the four corners. However, for an unknown image, we do not know the objects' locations in the image beforehand. To overcome this problem, we must have a simple algorithm to approximate the locations of the objects in the image so that we can place the sources in the objects and perform the image segmentation using the 0-1 method.

5.2.1 Source Candidates

By observation, I found that the 0-1 method itself is able to locate the objects' location. Apart from this, I also found this idea can be used in the k-way image segmentation (Section 5.3). To illustrate the mechanism, we consider the test image used in the previous section. We placed two sinks at two corners and two sources at the other two corners, as shown in the first column of Figure 5.6. Then we apply the 0-1 method, the resulting continuous partition vectors is shown in the second column of Figure 5.6.

From the plots in Figure 5.6, we notice that the regions corresponding to the nodes in the objects are flat. Expressed in the electric circuit analogy, the voltage drops in the objects are small compared to those in the background. This is because the voltage drop across the boundary between the object and the background is large due to the small edge weights (small conductance, large resistance) between them. This great drop in voltage decrease the voltage drop between the nodes inside of the objects. This observation hints that if we can find the nodes in the 'flat' region, we can locate the object.

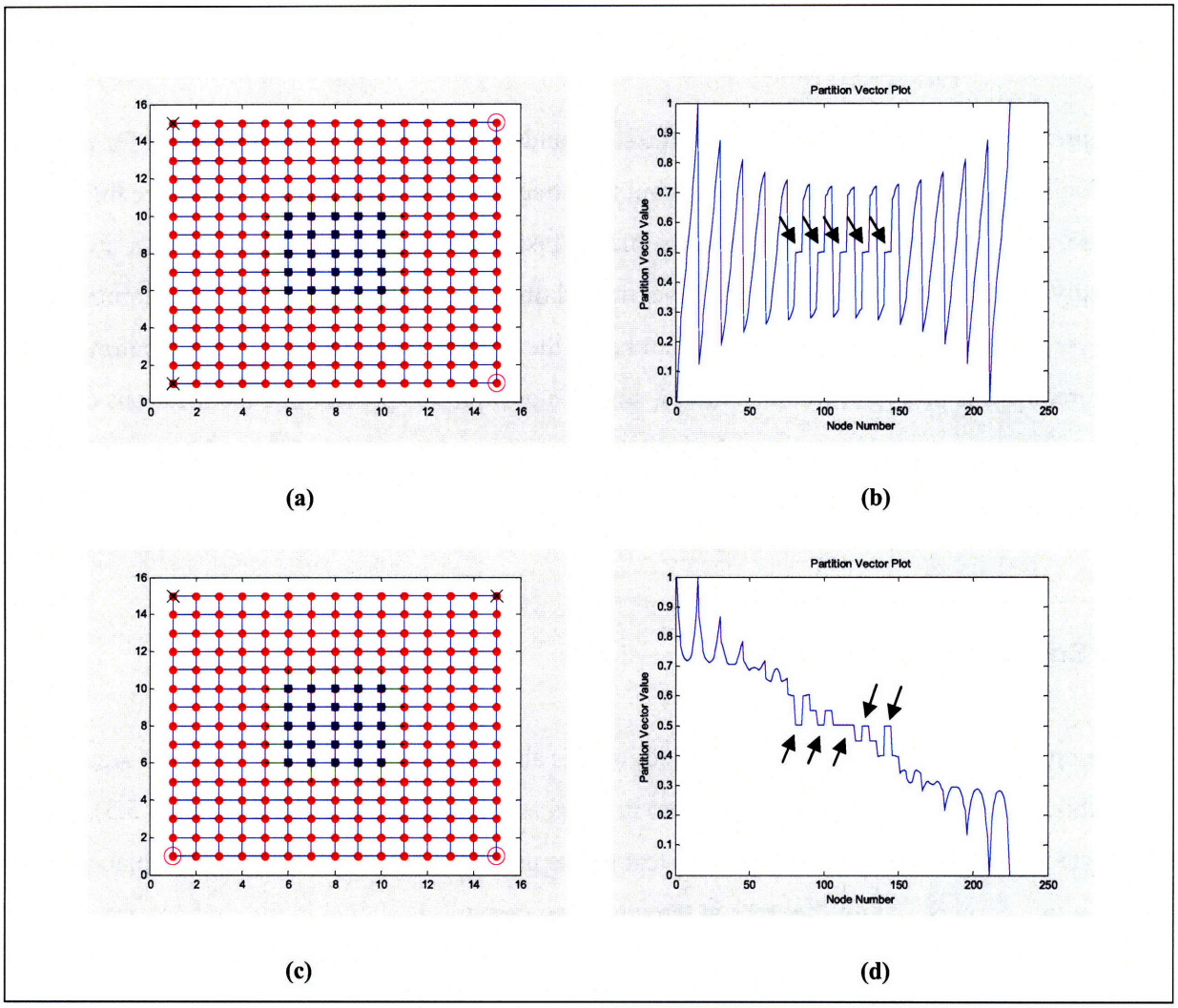


Figure 5.6 The first column shows the sinks and sources' locations on the graph constructed from the 15 x 15 test image. The two black 'X' at the left corners in (a) and at the upper corners in (c) represent the sinks while the two red circles at the right corners in (a) and at the bottom corners in (c) represent the sources. The green edges are the weak links. The second column ((b) and (d)) shows the continuous partition vector plots given by the 0-1 method. Notice the flat portion of the plots (pointed by arrows). They correspond to the object in the image.

Based on the observation, I have developed a simple algorithm using the 0-1 method to generate the source candidates that fall inside the objects – 0-1 Source Candidates Algorithm. The detailed algorithm is as shown below:

Algorithm 5.1

Given a graph with its Laplacian matrix, $\boldsymbol{L}$ constructed from an $m\ x\ n$ image and a parameter n_{min}; to locate the sources sn, we:

i. Set node 1 and node n as the sinks
ii. Set node $mn - n + 1$ and node mn as the sources
iii. Run the Basic 0-1 algorithm to give 0-1 vector, $\boldsymbol{v}$
iv. Reshape $\boldsymbol{v}$ to $m\ x\ n$ matrix $\boldsymbol{V}$
v. Find the horizontal difference, $|\Delta \boldsymbol{V_h}|$
vi. Add dummy column vector to $|\Delta \boldsymbol{V_h}|$
vii. Reshape matrix $|\Delta \boldsymbol{V_h}|$ to vector $|\Delta \boldsymbol{v_h}|$
viii. Find the first n_{min} node with the minimum $|\Delta v_h|$: $|\Delta \boldsymbol{v_h}|_{\boldsymbol{min}}$
ix. Find the vertical difference, $|\Delta \boldsymbol{V_v}|$
x. Add dummy row vector to $|\Delta \boldsymbol{V_v}|$
xi. Reshape $|\Delta \boldsymbol{V_v}|$ to vector $|\Delta \boldsymbol{v_v}|$
xii. Find the first n_{min} node with the minimum $|\Delta v_v|$: $|\Delta \boldsymbol{v_v}|_{\boldsymbol{min}}$
xiii. $|\Delta \boldsymbol{v}|_{\boldsymbol{min_1}}$={ $|\Delta \boldsymbol{v_h}|_{\boldsymbol{min}} \cap |\Delta \boldsymbol{v_v}|_{\boldsymbol{min}}$}
xiv. Set node 1 and node $mn - n + 1$ as the sinks
xv. Set node n and node mn as the sources
xvi. Repeat step (iii) to (xiii) to obtain $|\Delta \boldsymbol{v}|_{\boldsymbol{min_2}}$
xvii. $\boldsymbol{sn} = \{|\Delta \boldsymbol{v}|_{\boldsymbol{min_1}} \cap |\Delta \boldsymbol{v}|_{\boldsymbol{min_2}}\}$

The algorithm first uses two different sets of sink and source (step (i)-(iii) and step (xiv)-(xvi)) to generate two continuous partition vectors, $\boldsymbol{v}$. The vectors are arranged in matrix form, $\boldsymbol{V}$ to reflects the nodes' positions in the image (step (iv)). Then, I compute the horizontal difference, $|\Delta \boldsymbol{V_h}|$ and vertical difference, $|\Delta \boldsymbol{V_v}|$ between the neighbouring nodes (step (v) and (ix)). In step (vi) and (x), dummy row and column vectors are added to the vertical and horizontal difference matrix. With this step, the difference matrices have the matrix size that reflects the size and dimension of the image, and can be reshaped into a vector according to the defined node numbering later. The nodes with the minimum horizontal and vertical difference (step (xiii)) for

both sets of sink and source (step (xvii)) will be the source candidates that fall in the object. These nodes are the nodes that correspond to the 'flat' region shown in Figure 5.6.

The two different sets of sink and source are shown in Column 1of Figure 5.6. I choose these two sets to avoid the sink and source to fall inside the objects. If any of the sink and sources falls inside the objects, the flat region of the partition vector plot will appears not only in the objects, but also in the background (See the vector plots in Figure 5.3). Hence we want to avoid this situation by using the two sets of sink and sources at the corners. However, the method may fail if the objects are located at the corner of the image. Fortunately, most of the images have the objects situated near the center region of the images.

The reason to use two different sets of sink and sources is to avoid the generation of source candidates in the background. With only one set of sink and source, it is possible that some random background nodes have the minimum horizontal and vertical difference. To reduce this possibility, we use two different sets of sink and source. However, the problem with this scheme is that we may not be able to generate any source candidates because both sets of sink and sources generates different minimum difference nodes that do not intercept each other. This situation can be overcome by changing the parameters in graph construction (r, σ_I, σ_D) or increasing n_{min}.

In *Algorithm 5.1*, the parameter n_{min} is used in step (viii). It is the upper bound to the number of source candidates, sn. Setting a high number of n_{min} generates too many source candidates that may include background nodes. Furthermore, too many candidates may cause longer computation time. Hence, the n_{min} is set to a lower value. However, for k-way image segmentationg, in which we want to segment more than one objects, it is desirable to have a larger n_{min}. Generally, the n_{min} value is set in the range from 0.1 to 0.3% of the total number of nodes.

After obtaining a set of source candidates, depending on the number of objects to be segmented, we have different strategies in selecting the final sources to be used in the 0-1 method. In segmenting a single object from the image, we use the centroid of the candidate nodes as the source. For k-way image segmentation, we use k-means method to find k centroids and use the centroids as the sources. After determining the sources, we can solve the corresponding linear system to obtain the continuous partition vector. Then we perform the discretization to obtain the discrete partition vector, which gives us the segmentation.

5.3 *k*-way Image Segmentation

To illustrate the k-way image segmentation, I created a synthetic image (10 x 10), which contains two objects (Figure 5.7).

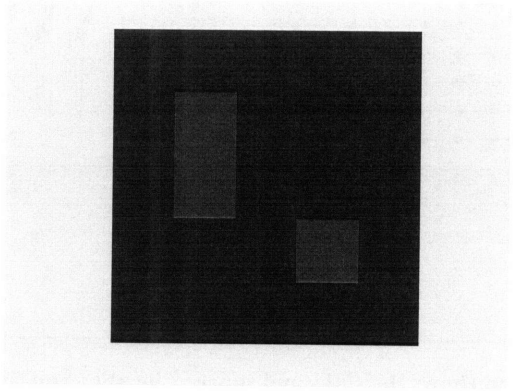


Figure 5.7 **A 10x10 synthetic image which contains two objects: a rectangle and a square.**

Again, if we apply the 0-1 method with the two different sets of sink and sources shown in Column 1 of Figure 5.8, we will again obtain a few flat regions on the partition vector plot (Column 2 of Figure 5.8). The 'flat' regions correspond to the nodes in the rectangle and also the square. Hence, with a suitable n_{min}, *Algorithm 5.1* can give us two groups of source candidates that falls inside the rectangle and the square, respectively.

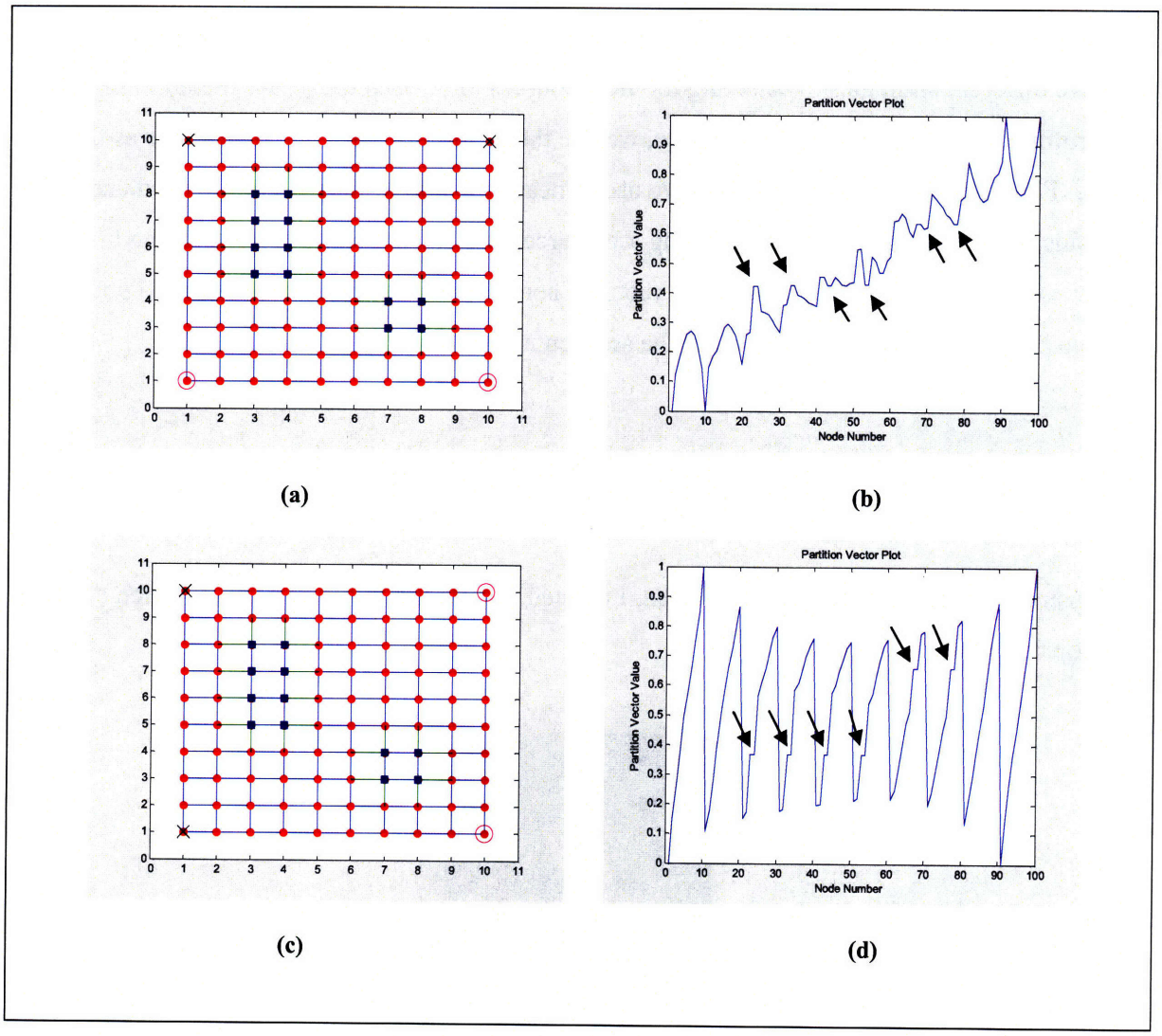


Figure 5.8 The first column shows the sinks and sources' locations on the graph constructed from the image in Figure 5.7. The black 'X's represent the sinks while the red circles represent the sources. The green edges are the weak links. The second column shows the continuous partition vector plots. Notice the flat portion of the plots (pointed by arrows). They correspond to the objects: the rectangle and square.

Extending the example above to k objects, we will obtain k groups of source candidates that fall inside k objects, respectively. Figure 5.9 shows a sample image of size 550 x 384 with two objects: a bowling ball and a shoe. *Algorithm 5.1* has successfully generated two groups of source candidates that fall inside the shoe and the bowling ball, respectively, as pointed by the red arrows in Figure 5.9.

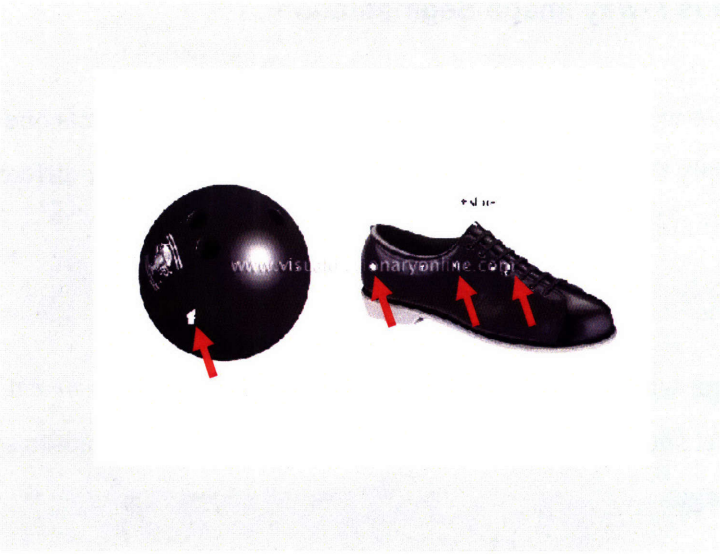


Figure 5.9 Two groups of source candidates with minimum vertical and horizontal difference (pointed by red arrows). One group is all located inside the bowling ball and another group is inside the shoe.

As discussed before, the ability of *Algorithm 5.1* to generate source candidates that cover all the objects in an image depends on the parameter n_{min}. The parameter n_{min} depends on the number of objects and the relative size of the objects in an image. To segment more objects, the parameter n_{min} should be set to a larger number (for example, 0.3). Apart from this, if we have few objects to be segmented, but the size of the objects (relative to the image size) is large, we should consider to use a larger value of n_{min} too. With sufficient number of source candidates generated (by using suitable n_{min} in *Algorithm 5.1*), all the k objects should contain one or a few source candidates. Hence, we can use these source candidates as the sources to segment the objects.

The 0-1 method allows us to segment the k objects out by using the recursive 2-way image segmentation or simultaneous k-way image segmentation.

5.3.1 Simultaneous *k*-way Image Segmentation

In the simultaneous *k*-way image segmentation, we segment the *k* objects one by one from the background. We apply the basic 0-1 method *k* times on the image for *k* different sources. Below is the detailed 0-1 Simultaneous *k*-way Image Segmentation algorithm.

Algorithm 5.2

Given a graph with its Laplacian matrix, $\boldsymbol{L}$ constructed from an $m \times n$ image and we have obtained a set source candidates, $\boldsymbol{sn}$ from *Algorithm 5.1*, we want to segment *k* objects from the image:

i. Find *k* centroids, $c_{i\,,i=1,\ldots,k}$ from $\boldsymbol{sn}$

ii. Set node 1, n, $mn - n + 1$ and mn as the sinks

iii. For $i = 1, \ldots, k+1$

  a. Set node c_i as the only source

  b. Run the Basic 0-1 algorithm to give 0-1 vector, $\boldsymbol{v}$

  c. Apply the Minimum *Ncut* Discretization algorithm to $\boldsymbol{v}$ to obtain the partition $\boldsymbol{u_i}$ (source) and $\overline{\boldsymbol{u}}_{\boldsymbol{i}}$ (sink)

  End For

iv. Calculate *Ncut* value for $\boldsymbol{u_i}$ for $i = 1, \ldots, k$

v. Reorder c_i and $\boldsymbol{u_i}$ in ascending *Ncut* order

vi. Set $\boldsymbol{p_1} = \boldsymbol{u_1}$

vii. Set $j = 2$

viii. For $i = 2, \ldots, k+1$

  If $c_i \notin \boldsymbol{p_{i-1}}$

    1. $\boldsymbol{p_j} = \uplus\, \boldsymbol{p_{k,\ k=1,\ldots,j-1}}$
    2. $\boldsymbol{p_j} = \overline{\boldsymbol{p}}_{\boldsymbol{j}} \cap \boldsymbol{u_i}$
    3. $j = j + 1$

  End If

  End For

In step (i), we find k centroids, $c_{i,i=1,\ldots,k}$ from the source candidates $\boldsymbol{sn}$. If the sources can be grouped into k groups, with each group resides in only one object, then step (i) gives us k centroids that reside in each object, respectively. For every centroid, we apply the 0-1 method with the centroid as the source and the four corners of the image as the sinks to obtain the 0-1 vectors, $\boldsymbol{v}$ (step (iii), a, b). Then we use the Minimum *Ncut* Discretization algorithm to obtain the partition vector $\boldsymbol{u_i}$ containing the source (step (iii), c). Each partition vector gives us a segmented object from an image.

The above description works well if we know how many objects contained in the image beforehand. In reality, we may not know the number of objects contained in an unknown image. There are two scenarios we should consider:

(1) $k >$ number of objects in the image
(2) $k <$ number of objects in the image

In case (1), we have more centroids than the number of objects. In other words, we may encounter the situation that an object contains more than one centroid. Those centroids contained in the same object give similar partitions, which may be redundant. From these similar partitions, we choose the partition that gives a smaller *Ncut* value. We can achieve this by reordering all the partitions according to the *Ncut* value (step (iv) to (v)), and subsequently checking the partition vector in the order for redundant partition vectors. A partition is redundant if its source is contained in other partitions too. Step (viii) checks if the current centroid is contained in the previous partitions. If the current centroid is contained in the previous partitions, the current partition is redundant and will be ignored. If redundant partitions exist, the final number of segmented objects will be less than k.

In case (2), the algorithm may fail. This is because the centroids may fall outside of the objects. For illustration, we consider the example shown in Figure 5.9 (page 135). There are two groups of source candidates that reside in the two objects, respectively. If we set $k = 2$, two centroids will be generated inside the two objects, respectively. However, if we set $k = 1$, the single

centroid will fall in the area between the two objects (outside of the objects). Since the 0-1 method can only segment the objects with the sources in the objects, the method fails. To avoid this situation, we should always set k to a larger value if we do not know the exact number of objects.

It is common that objects in an image are in contact with each other. Hence, it is possible that the partitions generated form step (i) to (vi), $\boldsymbol{u}_{i,\ i=1,\ldots,k}$ intersects each other. To ensure that all the partitions are mutually exclusive, we check the partitions for intersections in step (viii, 1). If intersection exists between the current partition and previous partitions, the intersected part will be removed from the current partition (step (viii, 2)).

5.3.2 Recursive 2-way Image Segmentation

In the recursive *2*-way image segmentation, we first separate all the k objects from the background. Then we recursively segment the segmented out image into k objects. Below is the detailed 0-1 Recursive *2*-way Image Segmentation algorithm.

Algorithm 5.3

Given a graph with its Laplacian matrix, $\boldsymbol{L}$ constructed from an $m\ x\ n$ image and we have obtained a set source candidates, $\boldsymbol{sn}$ from *Algorithm 5.1*, we want to segment k objects from the image:

i. Find k centroids, $c_{i\ ,i=1,\ldots,k}$ from $\boldsymbol{sn}$

ii. Set node 1, n, $mn - n + 1$ and mn as the sinks

iii. Set all the centroids , $c_{i\ ,i=1,\ldots,k}$ as the sources

iv. Run the Basic 0-1 algorithm on $\boldsymbol{L}$ to give 0-1 vector, $\boldsymbol{v}$

v. Apply the Minimum *Ncut* Discretization algorithm to $\boldsymbol{v}$ to obtain the partition $\boldsymbol{u}$ (source) and $\bar{\boldsymbol{u}}$ (sink)

vi. $\boldsymbol{p_{k+1}} = \bar{\boldsymbol{u}}$

vii. $\boldsymbol{c} = c_{i\,,i=1,\ldots,k} \cap \boldsymbol{u}$

viii. $k = n(c)$

ix. Set $j = 1$

x. For $i = 1$

a. $\boldsymbol{L_i} = \boldsymbol{L}(\boldsymbol{u})$

b. Set node c_i as the only source

c. Set node $c_{p,\ p=1,\ldots,k,\ p\neq i}$ as the sinks

d. Run the Basic 0-1 algorithm on $\boldsymbol{L_i}$ to give 0-1 vector, $\boldsymbol{v}$

e. Apply the Minimum *Ncut* Discretization algorithm to $\boldsymbol{v}$ to obtain the partition $\overline{\boldsymbol{u}}$ (source) and $\boldsymbol{u}$ (sink)

f. $\boldsymbol{p_i} = \overline{\boldsymbol{u}}$

End For

xi. $\boldsymbol{p_k} = \boldsymbol{u}$

xii. For $i = 1, \ldots, k$

If $n(\boldsymbol{p_i}) == 1$

1. $\partial p_i = index(\boldsymbol{p_i} > 0)$
2. For $j = 1, \ldots, k$

If $\boldsymbol{p}_{j,\ \partial \boldsymbol{p_i}-1} == 1$

$\boldsymbol{p}_{j,\ \partial \boldsymbol{p_i}} = 1$

End If

End For

End If

End For

xiii. Set $j = 1$

xiv. For $i = 1, \ldots, k$

If $n(\boldsymbol{p_i}) \neq 1$

1. $\boldsymbol{p'}_j = \boldsymbol{p_i}$
2. Set $j = j + 1$

End If

End For

Similar to *Algorithm 5.2*, *Algorithm 5.3* first find the centroids, $c_{i\ ,i=1,\ldots,k}$ from the source candidates, $\boldsymbol{sn}$. However, unlike *Algorithm 5.2*, they are used not only as the sources but also as the sinks. In step (ii), we set again the four corners as the sinks. Different from the simultaneous method, we use all the centroids as the sources in step (iii). By doing this, in one application of the 0-1 algorithm (step (iv)) and discretization algorithm (step (v)), we segment out together all the objects $\boldsymbol{u}$ from the background $\bar{\boldsymbol{u}}$. The segmented objects will be further partitioned, whereas the background will be left untouched as the $(k+1)^{th}$ final partition, $\boldsymbol{p_{k+1}}$ (step (vi)).

To further partitioned the segmented objects $\boldsymbol{u}$, we build the new reduced Laplacian matrix $\boldsymbol{L_i}$ from the segmented objects $\boldsymbol{u}$ (step (x, a)). Then we set the first centroid as the only source (step (x, b)) and the other centroids as the sinks (step (x, c)). With these sinks and sorces, we apply the 0-1 algorithm on the new reduced Laplacian system (step (x, d)) and apply the discretization algorithm (step (x, e)) to obtain the new partitions: $\bar{\boldsymbol{u}}$ and $\boldsymbol{u}$. The partition with the current source $\bar{\boldsymbol{u}}$ will be left untouched as one of the final partitions $\boldsymbol{p_i}$ (step (x, f)) while the other partition will be further partitioned again. The process goes on until k^{th} partition is obtained (step (xi)).

To avoid the segmentation of a single isolated node explained later in Section 5.3.3, step (xii) checks for partitions with single node. If a single node exist in partition $\boldsymbol{p_i}$ with node index ∂p_i (step (xii, 1)), the algorithm find another partition $\boldsymbol{p_j}$ that contains the neighbor node $\partial p_i - 1$ and group the single node to this partition (step (xx, 2)). Step (xiv) eliminates the partitions with a single node.

5.3.3 *k*-means

In the first step of both algorithms, *k* centroids are generated using *k-means* functions provided in MATLAB. The *k-means* function is an iterative heuristic algorithm, which can converge to a local minimum [10]. In this case, the situation of having more than one centroid in an object may

occur. Apart from the issue of the local minimum, the existence of more source candidates in one object than the other objects also causes the multiple sources in one object. This may lead to the failure of *Algorithm 5.3*, in which a single isolated node is segmented out as one of the final partitions. The last few steps *Algorithm 5.3* are used to eleiminate any single node partition. This situation does not appear in *Algorithm 5.2* because this problem is similar to the scenario (2) mentioned in Section 5.3.1.

5.4 Resized Image Segmentation Scheme

As discussed in Section 3.1.1 of Chapter 3, the quality of the image segmentation depends on the graph construction scheme as well. For more complicated image, a larger r value (r-radially connected graph construction scheme) may be needed. However, a graph with a larger r needs longer construction time because of its increased number of edges. Apart from the graph construction, the Minimum Cut algorithm will need to consider more graph edges in finding the minimum cut. Since the resulting Laplacian matrix will be less sparse due to the increased number of edges, the Isoperimetric Partitioning and 0-1 method will need more operations to solve the Laplacian system. The Normalized Cut method too will need even longer time in determining the eigenvectors of the denser Laplacian matrix. In short, all the graph partitioning solvers will require a longer time to partition the graph with a larger r value. Consequently, this further increases the total image segmentation time.

Apart from the consideration of run time, a graph with more links requires more memory space. In some case, the memory space required is too much that the computer runs out of memory. To reduce the total image segmentation time for a complicated image and also the memory space, I propose a scheme called Resized Image Segmentation Scheme as shown in Figure 5.10.

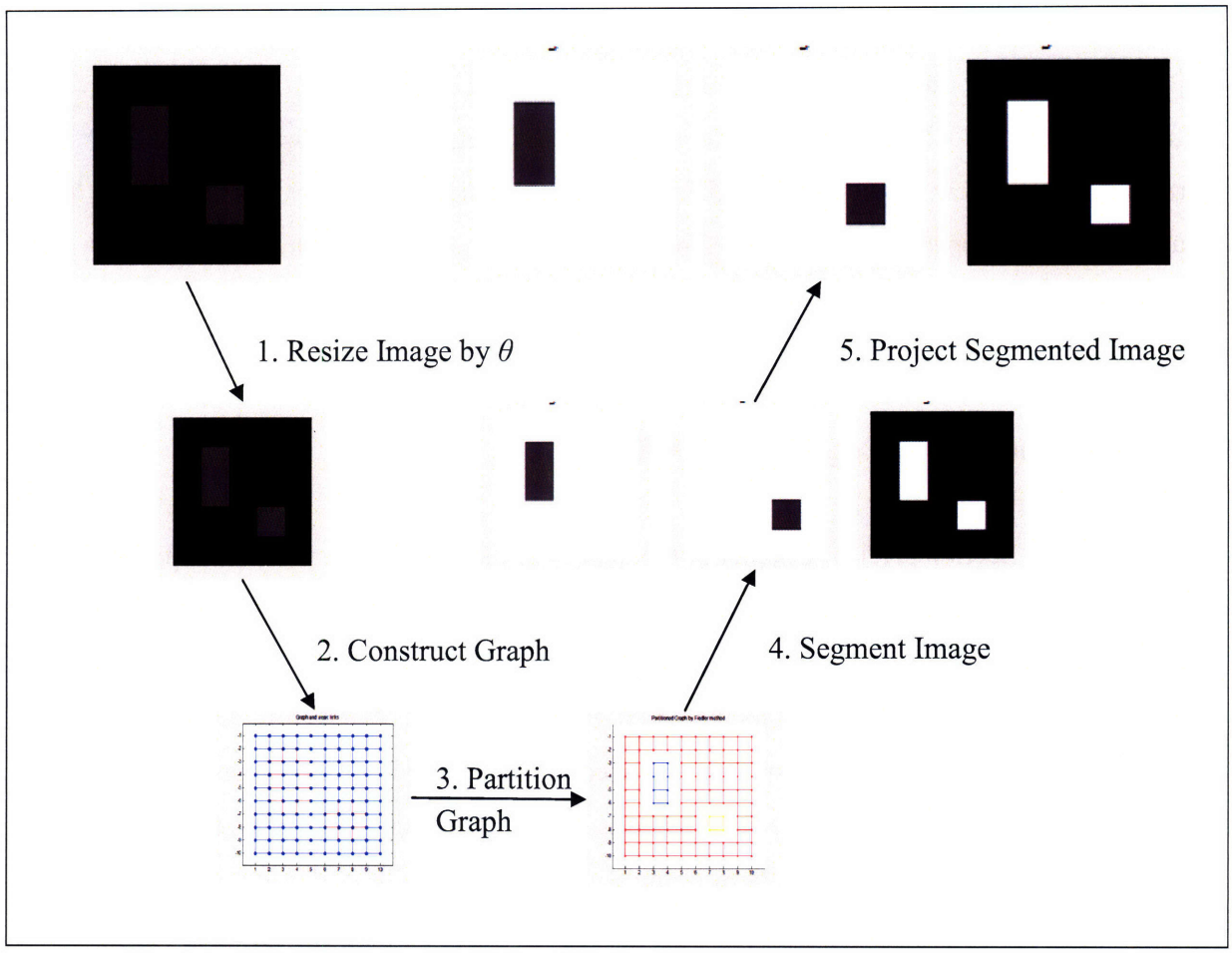


Figure 5.10 Resized Image Segmentation Scheme. The scheme starts with resizing the original large image into a smaller image by a shrinking factor θ (1). Then a graph is constructed from the smaller image (2). The graph can be partitioned by any graph partitioning method (3). The partitioned graph gives the segmented image (4). The scheme ends with projecting the reduced-size segmented image into the original size segmented image (5).

5.4.1 Resize Image

The process of resizing an image can be accomplished by the function *'imresize'* provided by the Image Processing Toolbox in MATLAB. In favor of speed, we want to shrink the image as much as possible. However, shrinking an image too much risks the quality of the image segmentation. This is because by shrinking an image, we lose the fine details of the original image. In image segmentation, this may cause small objects to be ignored by the image segmentation methods. Apart from this, since the image segmentation is done on a smaller image, the projected partitions may leave out some pixels that should be a part of the partitions (or include some

pixels that should not be a part of the partitions) along the boundaries. With increased shrinking factor, the number of left out or included pixels will increase as well. Hence, we should set the shrinking factor θ to an optimum value that balances the speed and the quality.

5.4.2 Project Segmented Image

In the process of projecting the reduced-size segmented image to its original size, I use again the function *'imresize'*. Instead of shrinking, I expand the binary partition vectors. This can be accomplished by reordering the partition vector into the form of an image matrix and applying the function with the inverse of the shrinking factor.

5.4.3 Application in 0-1 Image Segmentation

For large images, if the objects are large as well, using only a source in every object may not be sufficient. Explained in terms of an electrical circuit, a large image corresponds to a large electrical circuit network. With a large circuit network, the accumulated resistance between the source and the nodes further away from the source will be great and thus, the nodes' voltages become low. Even though these nodes are a part of the object, the low voltages cause them to be excluded. As a result, the object is segmented fractionally. To overcome this problem, we can place more sources in the object. However, this is not a good solution because we need to determine another source's location. Another solution to this problem is to use a larger r value in the graph construction. The graph construction scheme allows the source to connect nodes that are further away from it. This is equivalent to adding parallel resistor between the sources and the nodes. The parallel resistors reduce the total accumulated resistance. However, a larger r also increases the computation time. In order to avoid fractional image segmentation and long computation time, we can reduce the size of the image. The resized image gives a smaller graph. Hence the fractional image segmentation can be avoided.

5.4 Refinement

Since all the image segmentation methods, except the Minimum Cut method, are just approximation to the discrete image segmentation problem, the partitions given by the methods does not necessarily give the minimum *Ncut* value. Furthermore, the Resized Image Segmentation scheme causes wrongly left out or included pixels at the boundaries. To improve the segmentation, we implement an iterative refinement process along the boundaries. The Thorough Refinement Algorithm can be summarized as follow:

Algorithm 5.4

Given k partitions $\boldsymbol{p}_{i,\ i=1,\ldots,k}$; the degree matrix, $\boldsymbol{D}$; the adjacency matrix $\boldsymbol{W}$; and the parameter $iter$:

i. For $b = 1, \ldots, iter$
   a. $Ncut = Ncut(\boldsymbol{D}, \boldsymbol{W}, \boldsymbol{p}_{i,\ i=1,\ldots,k})$
   b. For $i = 1, \ldots, k-1$
      i. For $j = 1, \ldots, k$
         1. $\boldsymbol{P}_i = diag(\boldsymbol{p}_i)$
         2. $\boldsymbol{P}_j = diag(\boldsymbol{p}_j)$
         3. $\boldsymbol{B} = \boldsymbol{P}_i * \boldsymbol{W} * \boldsymbol{P}_j$
         4. $\boldsymbol{\partial p}_i = row(\boldsymbol{B} > 0)$
         5. $\boldsymbol{\partial p} = \boldsymbol{\partial p}_i$
         6. For $a = 1, \ldots, n(\boldsymbol{\partial p})$
            a. $\boldsymbol{p}_i(\partial p_a) = \overline{\boldsymbol{p}_i(\partial p_a)}$
            b. $\boldsymbol{p}_j(\partial p_a) = \overline{\boldsymbol{p}_j(\partial p_a)}$
            c. $\Delta Ncut = Ncut(\boldsymbol{D}, \boldsymbol{W}, \boldsymbol{p}) - Ncut$
            d. If $\Delta Ncut > 0$
               i. $\boldsymbol{p}_i(\partial p_a) = \overline{\boldsymbol{p}_i(\partial p_a)}$
               ii. $\boldsymbol{p}_j(\partial p_a) = \overline{\boldsymbol{p}_j(\partial p_a)}$

               End If

         End For a

7. $\boldsymbol{P_i}$=diag($\boldsymbol{p_i}$)
8. $\boldsymbol{P_j}$=diag($\boldsymbol{p_j}$)
9. $\boldsymbol{B = P_i * W * P_j}$
10. $\boldsymbol{\partial p_j} = col(\boldsymbol{B} > 0)$
11. $\boldsymbol{\partial p} = \boldsymbol{\partial p_j}$
12. Repeat step (6) to (11)

End For *j*

End For *i*

End For *b*

In step (3) and (9), the boundary between two adjacent partitions ($\boldsymbol{p_i}$ *and* $\boldsymbol{p_j}$) are obtained by the following equation:

$$\boldsymbol{B = P_i * W * P_j} \ , \tag{5.1}$$

where $\boldsymbol{W}$ is the adjacency matrix; $\boldsymbol{P_i}$ and $\boldsymbol{P_j}$ are the diagonal matrix with the partition vectors $\boldsymbol{p_i}$ and $\boldsymbol{p_j}$ as the diagonals (step (1) – (2), step (7) – (8)), respectively; and $\boldsymbol{B}$ is the binary adjacency matrix for the boundary nodes. The rows of the nonzero entries of $\boldsymbol{B}$ (step (4)) give node numbers of the boundary nodes $\boldsymbol{\partial p_i}$ in partition $\boldsymbol{p_i}$. The columns of the nonzero entries of $\boldsymbol{B}$ (step (10)) give node numbers of the counterpart boundary nodes $\boldsymbol{\partial p_j}$ in partition $\boldsymbol{p_j}$.

We refine the boundaries of the partitions by moving the boundary nodes $\boldsymbol{\partial p_i}$ in partition $\boldsymbol{p_i}$ one by one to the counterpart partition $\boldsymbol{p_j}$ (step (6a, b)) and checking the change in the *Ncut* value, $\Delta Ncut$ (step (6d)). The move can be done by negating the $\boldsymbol{\partial p_i}$ entries of the partitions $\boldsymbol{p_i}$ and $\boldsymbol{p_j}$. If the boundary node's move decreases the *Ncut* value, the boundary node is moved to the counterpart partition $\boldsymbol{p_j}$. We repeat the whole process by testing the boundary nodes $\boldsymbol{\partial p_j}$ of the counterpart partitions $\boldsymbol{p_j}$ (step (12)). The refinement is performed on every partition. For better result, the refinement is performed for $iter$ iterations. The refinement process is costly as we

need to check every boundary nodes. Hence, the number of iterations should be set to a lower value.

In order to speed up the refinement process, we should be selective in checking the boundary nodes. We should not test all the boundary nodes. Instead, we should only test the boundary nodes that have a larger chance in decreasing the *Ncut* value. The higher chance nodes are the boundary nodes with pixel intensity larger or smaller than the average value of their partitions, depending on the average pixel intensity of the counterpart partitions. In this way, we test fewer boundary nodes and reduce the total refinement time. With this new selection criterion, we have the following Fast Refinement algorithm:

Algorithm 5.5

Given k partitions, $\boldsymbol{p}_{i,\ i=1,\dots,k}$; the adjacency matrix $\boldsymbol{W}$; the pixel intensity matrix, $\boldsymbol{im}$ (image); and the parameter $iter$:

i. For $b = 1, \dots, iter$

   a. $Ncut = Ncut(\boldsymbol{D}, \boldsymbol{W}, \boldsymbol{p}_{i,\ i=1,\dots,k})$

   b. For $i = 1, \dots, k-1$

      i. For $j = 1, \dots, k$

         1. $\boldsymbol{P}_i$=$diag(\boldsymbol{p}_i)$
         2. $\boldsymbol{P}_j$=$diag(\boldsymbol{p}_j)$
         3. $\boldsymbol{B} = \boldsymbol{P}_i * \boldsymbol{W} * \boldsymbol{P}_j$
         4. $\boldsymbol{\partial p}_i = row(\boldsymbol{B} > 0)$
         5. $\boldsymbol{\partial p} = \boldsymbol{\partial p}_i$
         6. $im^i_{avg} = avg(\boldsymbol{im}(\boldsymbol{p}_i))$
         7. $im^j_{avg} = avg(\boldsymbol{im}(\boldsymbol{p}_j))$
         8. If $im^i_{avg} < im^j_{avg}$

$$\boldsymbol{\partial p}_i = \boldsymbol{\partial p}_i(im > im^i_{avg})$$

Else

$$\partial \boldsymbol{p}_i = \partial \boldsymbol{p}_i(im < im^i_{avg})$$

End If

9. For $a = 1, \ldots, n(\partial \boldsymbol{p})$
   a. $\boldsymbol{p}_i(\partial p_a) = \overline{\boldsymbol{p}_i(\partial p_a)}$
   b. $\boldsymbol{p}_j(\partial p_a) = \overline{\boldsymbol{p}_j(\partial p_a)}$
   c. $\Delta Ncut = Ncut(\boldsymbol{D}, \boldsymbol{W}, \boldsymbol{p}) - Ncut$
   d. If $\Delta Ncut > 0$
      i. $\boldsymbol{p}_i(\partial p_a) = \overline{\boldsymbol{p}_i(\partial p_a)}$
      ii. $\boldsymbol{p}_j(\partial p_a) = \overline{\boldsymbol{p}_j(\partial p_a)}$

      End If

   End For a
10. $\boldsymbol{P}_i$=*diag*($\boldsymbol{p}_i$)
11. $\boldsymbol{P}_j$=*diag*($\boldsymbol{p}_j$)
12. $\boldsymbol{B} = \boldsymbol{P}_i * \boldsymbol{W} * \boldsymbol{P}_j$
13. $\partial \boldsymbol{p}_j = col(\boldsymbol{B} > 0)$
14. $\partial \boldsymbol{p} = \partial \boldsymbol{p}_j$
15. Repeat step (6) to (11)

End For j

End For i

End For b

In this algorithm, we selectively test the boundary nodes based on the boundary nodes' pixel intensity. For illustration, we consider two partitions: $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$. If $\boldsymbol{p}_i$ has smaller average pixel intensity than $\boldsymbol{p}_j$, any boundary node in $\partial \boldsymbol{p}_i$ with pixel intensity larger than average im^i_{avg} will likely be the wrongly included node (step (8)). On the other hand, if $\boldsymbol{p}_i$ has larger average pixel intensity than $\boldsymbol{p}_j$, any boundary node in $\partial \boldsymbol{p}_i$ with pixel intensity lower than average im^i_{avg} will likely be the wrongly included node (step (8)). The reason is because pixels in the same partitions usually have similar pixel intensity. By testing only the boundary nodes with unusual pixel intensity, we avoid checking every single boundary node.

Algorithm 5.4 can be applied to general graphs while *Algorithm 5.5* is only limited to image segmentation due to the involvement of pixel intensity in the algorithm. Nevertheless, both refinement algorithms are guaranteed to decrease the *Ncut* value. Since we assumed that the *Ncut* value measures the partition quality, the refinement can improve the image segmentation quality.

5.6 Experiments and Results

Using the algorithms decribed in the previous sections, I have produced a few results of the 0-1 image segmentation.

5.6.1 2-way Image Segmentation

This section shows the application of the 0-1 method in extracting the single object from images. Figure 5.11 shows the image segmentation of a 55x55 image of a star. The pixel intensity of the star increases from the bottom to the top. The pixel intensities of the top three vertices are close to the white background. As a result, the tips are excluded.

Figure 5.12 shows a 50 x 50 image of a tiger (a) and the segmented images (b, c). Though the pixel intensity within the tiger's head varies (due to its stripes), the white background gives a good contrast. Hence, the tiger is segmented easily. The unsegmented parts are at the bottom left and right corners. The reason is because they are very close to the locations of the sinks at the corners. The proximity to the sinks causes their 'voltage' to be low. Subsequently, they are excluded.

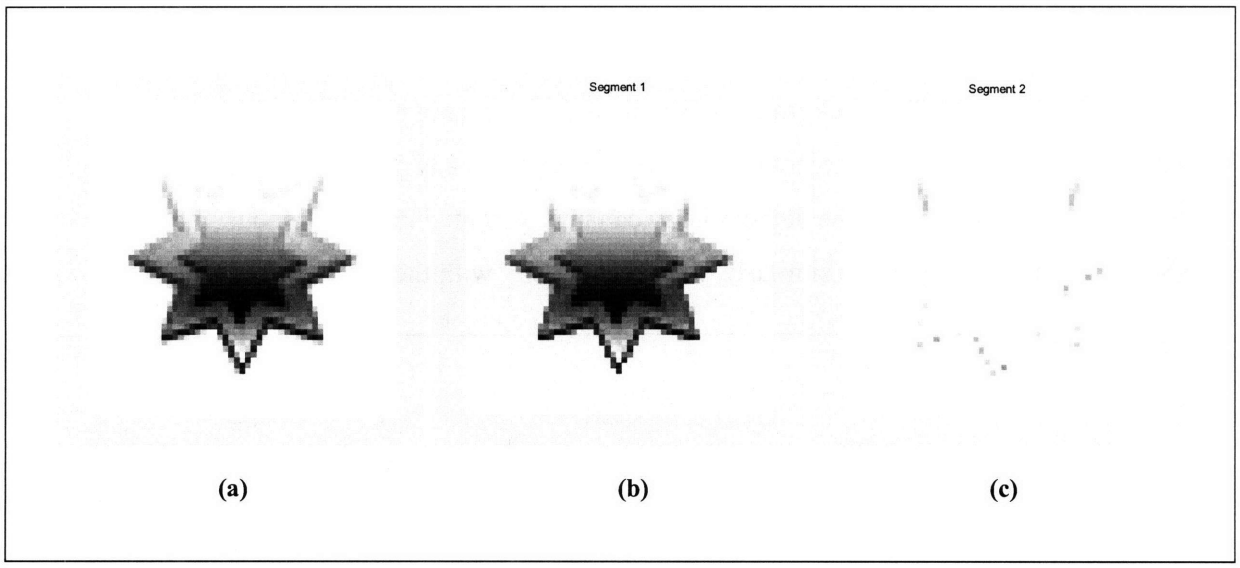


Figure 5.11 Image segmentation of a 55x55 image of a star. Figure (a) shows the original image and figure (b) and (c) show the segmented images. The background is homogeneous and white (a). However, the object is not homogeneous (a). The intensity of the star is fading upwards (a). Using the 0-1 algorithm, the star is segmented (b), without the top three vertices. The reason for this is because of their fading intensity, which becomes similar to the background intensity. The parameters used are: $r = 0$, $\sigma_I = 0.08$ and $n_{min} = 0.1$.

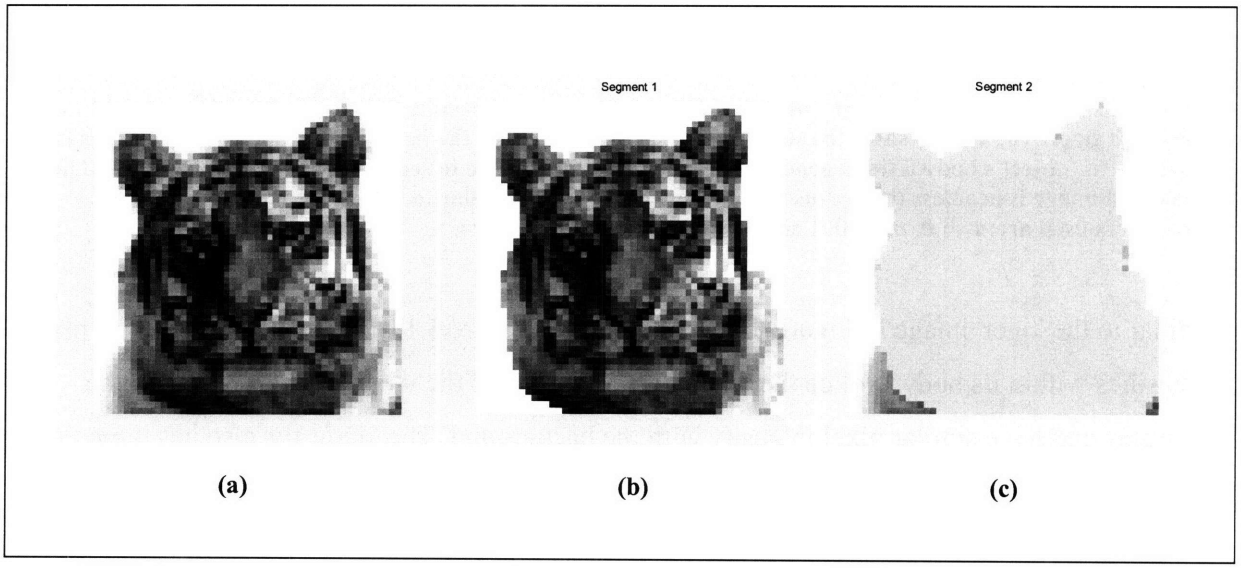


Figure 5.12 Image segmentation of a 50x50 natural image of a tiger. Figure (a) shows the original image and figure (b) and (c) show the segmented images. The background is homogeneous and white (a). However, the object is not homogeneous. The algorithm is able to segment out the people's body. Almost the whole tiger's head is segmented (b). The unsegmented parts are at the bottom left and right corners. The reason for this is because their locations are close to the locations of the sinks. The parameters used are: $r = 0$, $\sigma_I = 0.1$ and $n_{min} = 0.1$.

In the previous two images, the background is white and homogeneous. This makes the segmentation of the objects easier. To further test the ability of the method, the image shown in Figure 5.13 (a) is used. The background of the image has varying pixel intensity (non-homogeneous). However, the object is homogeneous and has a high contrast in pixel intensity with the background. Hence, the object's body can be segmented (b). The head of the people cannot be segmented due to its similarity in pixel intensity with the background.

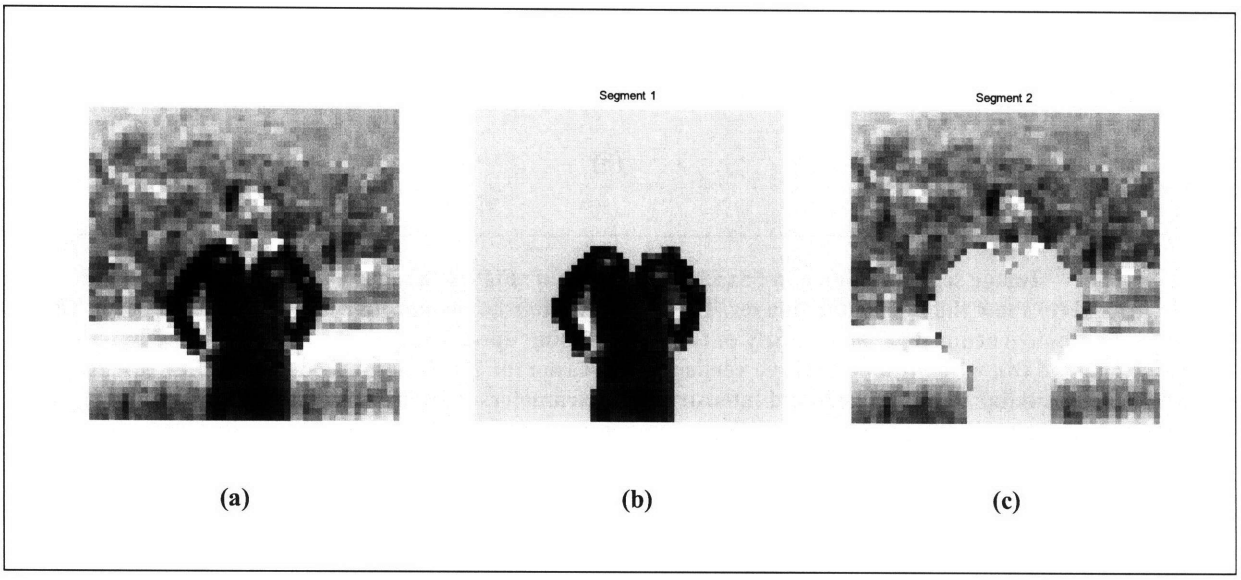


Figure 5.13 Image segmentation of a 48x48 natural image of a people. Figure (a) shows the original image and figure (b) and (c) show the segmented images. Notice that the background is not homogeneous (a). However, the object's body is homogeneous (a). The algorithm is able to segment out the people's body. The segmented image is headless (b) because the objects head is very similar to the background (a, b). The parameters used are: $r = 0$, $\sigma_I = 0.1$ and $n_{min} = 0.1$.

Similar to the 'tiger' image in Figure 5.12, the butterfly (Figure 5.14) has non-homogeneous pixel intensities within its body. Notice the white dots at the top of the wing. They are close to the boundary and have similar pixel intensity with the background. The 'butterfly' also has three tiny legs. The segmentation of the tiny legs requires the use of a larger r value in the graph construction. However, the use of the high r value may cause the white dots to be excluded from the object due to their connectivity with the background. To achieve a balance in between these two considerations, I used $r = 1$. The segmentation result is shown in (b) and (c). The white dots are segmented together with the body of the 'butterfly' and only one of the three legs is segmented.

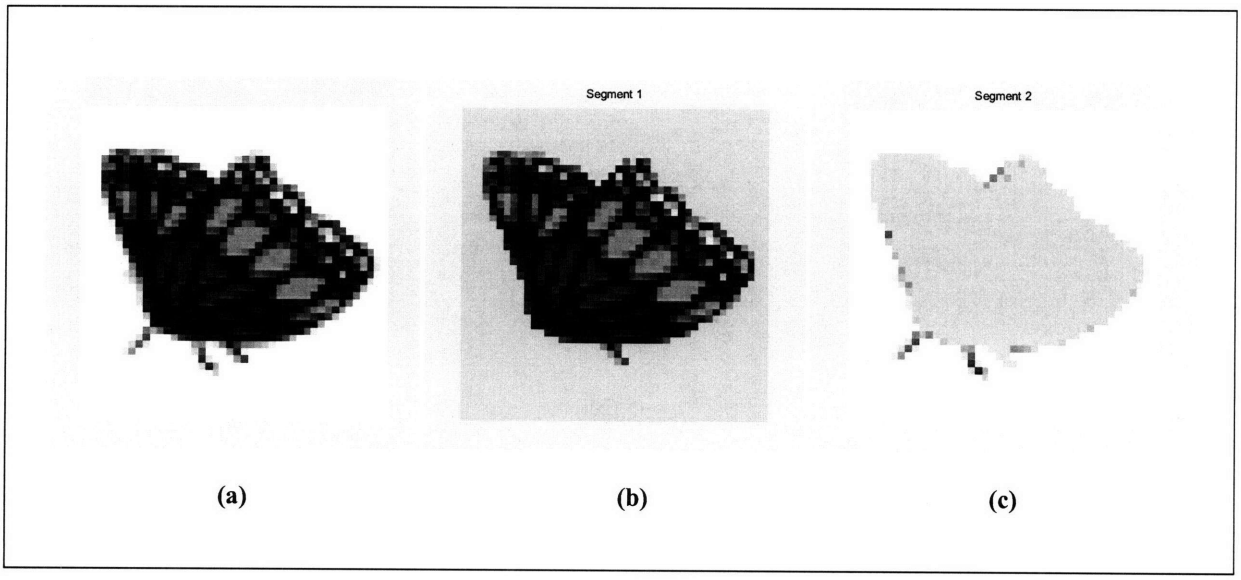


Figure 5.14 Image segmentation of a 44x44 image of a butterfly. Figure (a) shows the original image and figure (b) and (c) show the segmented images. The background is homogeneous and white (a). However, the object is not homogeneous (a). Notice the white dots at the top of the wing. Another feature is its tiny leg. It has three legs. The algorithm is able to segment out the butterfly except its two front legs (a, b). The parameters used are: $r = 1$, $\sigma_I = 0.1$ and $n_{min} = 0.1$.

Figure 5.15 shows the image segmentation of a 50x50 image of an air plane. The challenge of the image is its background and the biplane wings. The background has two homogeneous regions (top and bottom) with different pixel intensities. Most of the time, other image segmentation methods only separate the image into two according to the background. The plane also has a gap between its biplane wings. The 0-1 image segmentation method is able to segment the plane and also its shadow. Notice that the gap between the wings is not segmented out with the plane. This is due to the use of $r = 4$. Though the gap looks isolated in the image, it is actually connected to the background through the r radially connected edges. The r value allows a pixel to connect itself to another pixel that is a few pixels away.

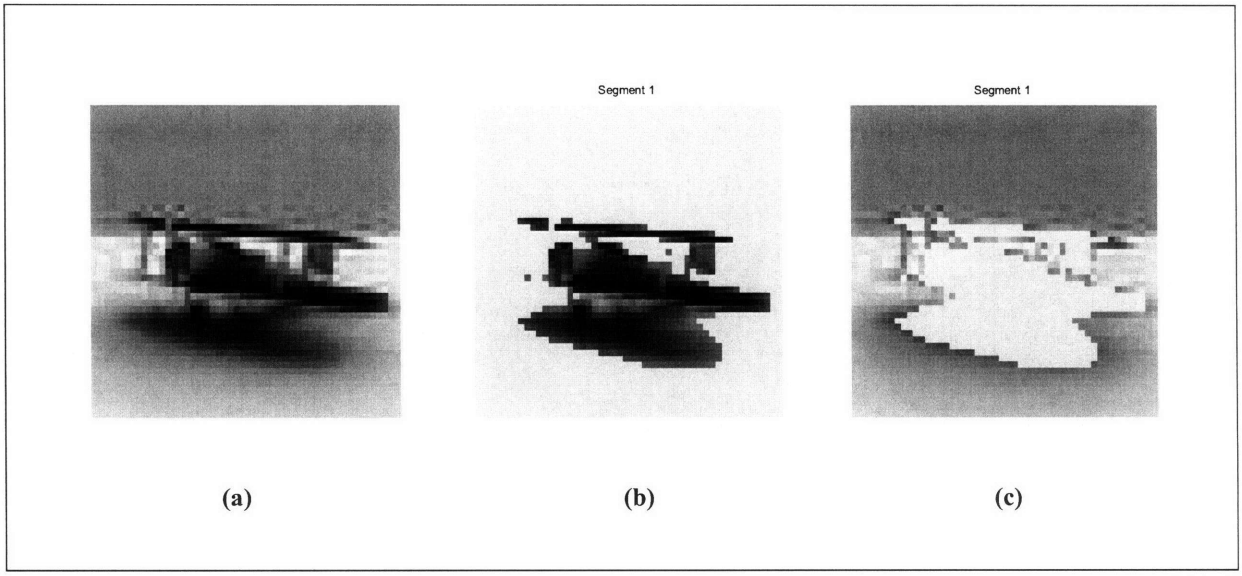


Figure 5.15 Image segmentation of a 50x50 image of an air plane. Figure (a) shows the original image and figure (b) and (c) show the segmented images. The background is homogeneous (a). However, the object's body is not homogeneous (a). The algorithm is able to segment out the plane's fuselage and its double wing; and also its shadow (a, b). Notice that the gap between the wings is not segmented out with the plane. This is due to the use of large r value. Though the gap looks isolated in the image, it is actually connected to the background through the r radially connected edges. The r value allows a pixel to connect itself to another pixel that is a few pixels away. The parameters used are: $r = 4$, $\sigma_I = 0.06$, $\sigma_D = 1$ and $n_{min} = 0.1$.

Table 5.1Table 5. shows the image size and the r value used in the graph construction. It also shows the quantitative measure of the image segmentation: the *Ncut* value and the run time. All the images have similar size. All the images, except 'butterfly.jpg' and 'plane.jpg', use the 4-connected graph construction scheme ($r = 0$). The use of the larger r value in the last two images has caused longer graph construction and image sgmentation times.

Table 5.1 The image sizes, r value, *Ncut* value and run time of the image segmentation of the images in Figure 5.11to Figure 5.15

Image	Size	r	*Ncut*	Graph Construction Time (s)	Image Segmentation Time (s)
star.jpg	55 x 55	0	0.0120	0.152336	0.094588
tiger.jpg	50 x 50	0	0.0055	0.135586	0.080269
people.jpg	48 x 48	0	7.7248e-004	0.121735	0.076531
butterfly.jpg	44 x 44	1	1.3766e-004	5.647105	0.103505
plane.jpg	50 x 50	4	0.0340	14.686836	0.920025

5.6.2 *k*-way Image Segmentation

This section shows the results of the 0-1 *k*-way image segmentation. In Figure 5.16, I performed both the simultaneous and recursive algorithm on the synthetic image created in Section 5.3. Both algorithms are able to segment out the two objects (a rectangle and square) from its background.

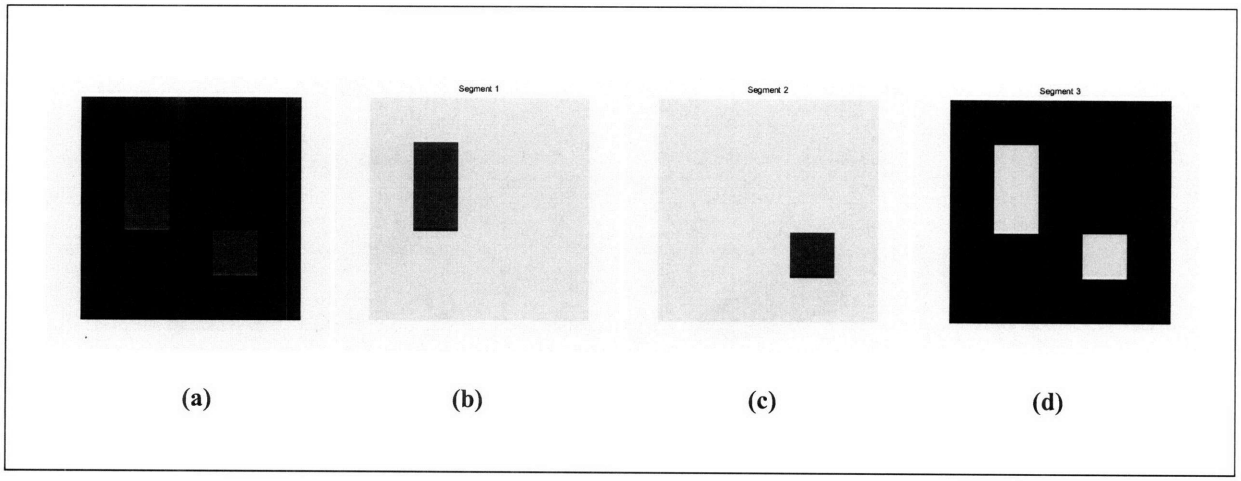


Figure 5.16 Image segmentation of a 10x10 synthetic image. Figure (a) shows the original image and figure (b) shows the segmented images. Both simultaneous and recursive algorithms are able to segment out the two objects (a rectangle and square) from its background. The parameters used are: $r = 0$, $\sigma_I = 0.1$, $n_{min} = 0.1$ and $k = 2$.

Figure 5.17 shows the image segmentation of a 384 x 550 image containing a bowling ball and a shoe. Judging from the size of the image and also the objects, this image is more difficult than the previous synthetic image. Both simultaneous and recursive algorithms are able to segment out the two objects from its background. The difference in segmentation between the methods is unnoticeable. The segmentation is not complete. The edges of the ball and the shoe are left with the background (Segment 3). This is because their pixel intensities are in between the background intensity and the objects' intensity (transition from the object to the background). Hence the pixel intensity difference between the edge nodes and the background is not as great as those between the objects and the background (larger edge weights).

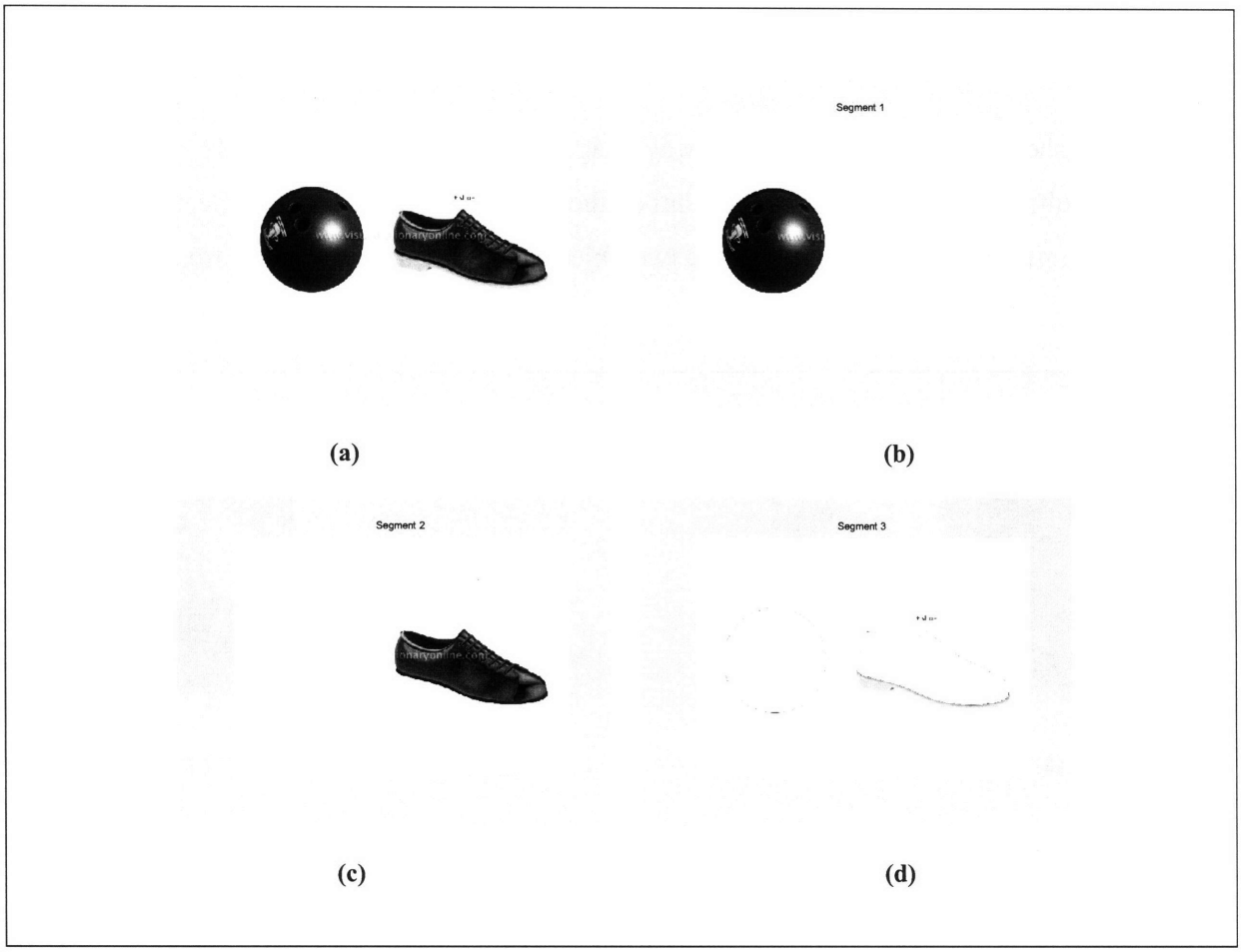


Figure 5.17 Image segmentation of a 384x550 image. Figure (a) shows the original image and figures (b, c, d) show the segmented images. The image contains two objects: a shoe and a bowling ball. Both simultaneous and recursive algorithms are able to segment out the two objects from its background. However the segmentation is not complete. The edges of the ball and the shoe are left with the background (Segment 3). This is because their pixel intensities are in between the background intensity and the objects' intensity. The parameters used are: $r = 0$, $\sigma_I = 0.1$, $n_{min} = 0.1$ and $k = 2$.

In the previous two examples, the objects are well separated. To increase the difficulty, I used the image shown in Figure 5.18. The four objects (basketball, tennis ball, football and base ball) are in contact with each other. Different from the previous examples, the simultaneous and recursive methods give different segmentation (Figure 5.18 and Figure 5.19).

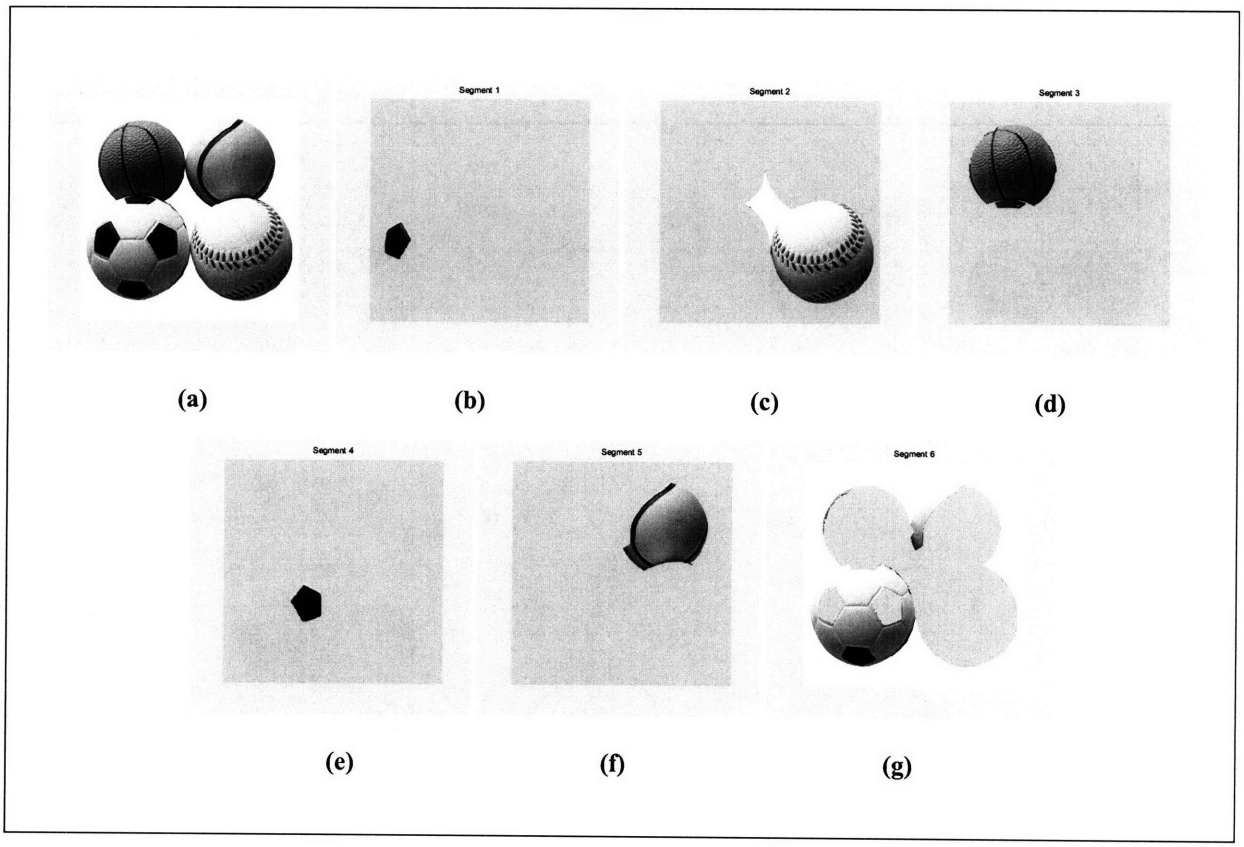


Figure 5.18 Image segmentation of a 360x360 image containing four balls by the simultaneous 0-1 method. Figure (a) shows the original image and figures (b) – (g) show the segmented images. The simultaneous algorithm segments out the two black pentagons of the football (b, e); the baseball with the center white patch (c); the basketball connected with a black pentagon of the football (d); and the tennis ball (f). The football and a part of the tennis ball are left unsegmented from the background (g). The parameters used are: $r = 0$, $\sigma_I = 0.08$, $n_{min} = 0.3$ and $k = 10$. Larger value of n_{min} is used because more objects are to be segmented. Redundancy in partition occurs in this case because the total number of objects segmented is 5, which is less than k (10).

The simultaneous algorithm segments out the two black pentagons of the football (b, e); the baseball with the center white patch (c); the basketball connected with a black pentagon of the football (d); and the tennis ball (f). The football and a part ofthe tennis ball are left unsegmented (g). The center white patch is grouped with baseball because of their similarity in pixel intensity. The remaining parts of the football is not segmented out. The top pentagon of the football is grouped together with the basketball (d) beacuase they are in contact and have very similar pixel intensity. The football is not totally segmented out because of the high contrast between the pentagons and their surrounding. Moreover, the fooball (except the balck pentagons) has similar

pixel intensity with the background. The tennis ball has the same situation. The black stripe separates the tennis ball.

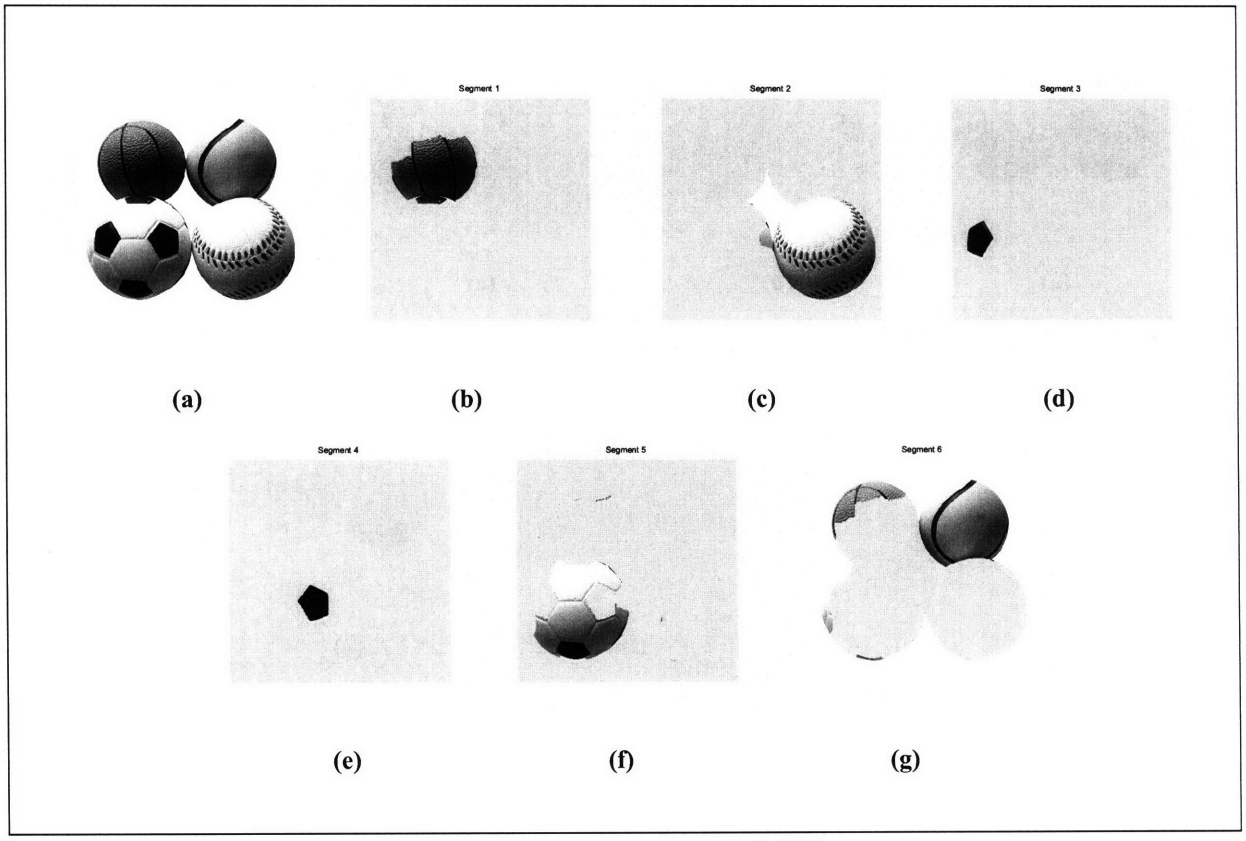


Figure 5.19 Image segmentation of a 360x360 image containing four balls by the recursive 0-1 method. Figure (a) shows the original image and figure (b) shows the segmented images. The recursive algorithm segments out the two black pentagons of the football (d, e); the baseball with the center white patch and a part of the football (c); part the basketball connected with a black pentagon of the football (d); and the partial foot ball (f). A part of the football and basketball and the whole tennis ball are left unsegmented (g).The parameters used are: $r = 0, \sigma_I = 0.08, n_{min} = 0.3$ and $k = 7$. Larger value of n_{min} is used because more objects are to be segmented. Redundancy in partition occurs in this case because the total number of segmented objects is only 5, which is less than k (7).

Similar to the simultaneous 0-1 method, the segmented images include the two black pentagons of the football (b, e); the baseball with the center white patch (c); and the basketball connected with a black pentagon of the football (d). However, the segmentation of the basketball is not complete. The top a part ofthe basketball is not segmented out (b). For the segmented baseball, a small a part ofthe football (in contact with the baseball) is segmented together with the baseball as well (c). Apart from this, the tennis ball is left unsegmented (g). Instead of the tennis ball, the

remaining part ofthe football is segmented out (f). The segmentation is not complte. The recursive method performs poorly compared to the simultaneous method.

In both cases, larger value of $\boldsymbol{n_{min}}$ is used because we want to segment out more objects. Redundancy in partition occurs in both cases because the total number of segmented objects is less than $\boldsymbol{k}$. Both methods give same number of segmented objects (5), but the simultaneous method used $\boldsymbol{k = 10}$ while the recursive method used $\boldsymbol{k = 7}$.

Different from the previous examples, I applied the simultaneous and recursive 0-1 image segmentation methods to an image containing only an object. The image is a of size 50 x 50 and contains a gun. Figure 5.20 and Figure 5.21 show the images and the segmentation results by simultaneous and recursive method, respectively.

For the simultaneous 0-1 image segmentation methods, the results are good (Figure 5.20). The method segments out the distinc feature of the gun: the gun mouth (b); the handle (c); the barrel and trigger; the white gap near the trigger (e); and the background (f).

The segmentation results of the image given by the recursive method (Figure 5.21) are not as good as those given by the simultaneous method. Though it segments out the gun mouth (b) and the white gap near the trigger (e), the barrel is segmented into two parts (d, e). The trigger is divided into two parts too (d, e). The partial barrel in (d) is grouped with a part of the trigger. The handle is grouped together with a part of the trigger and barrel (e).

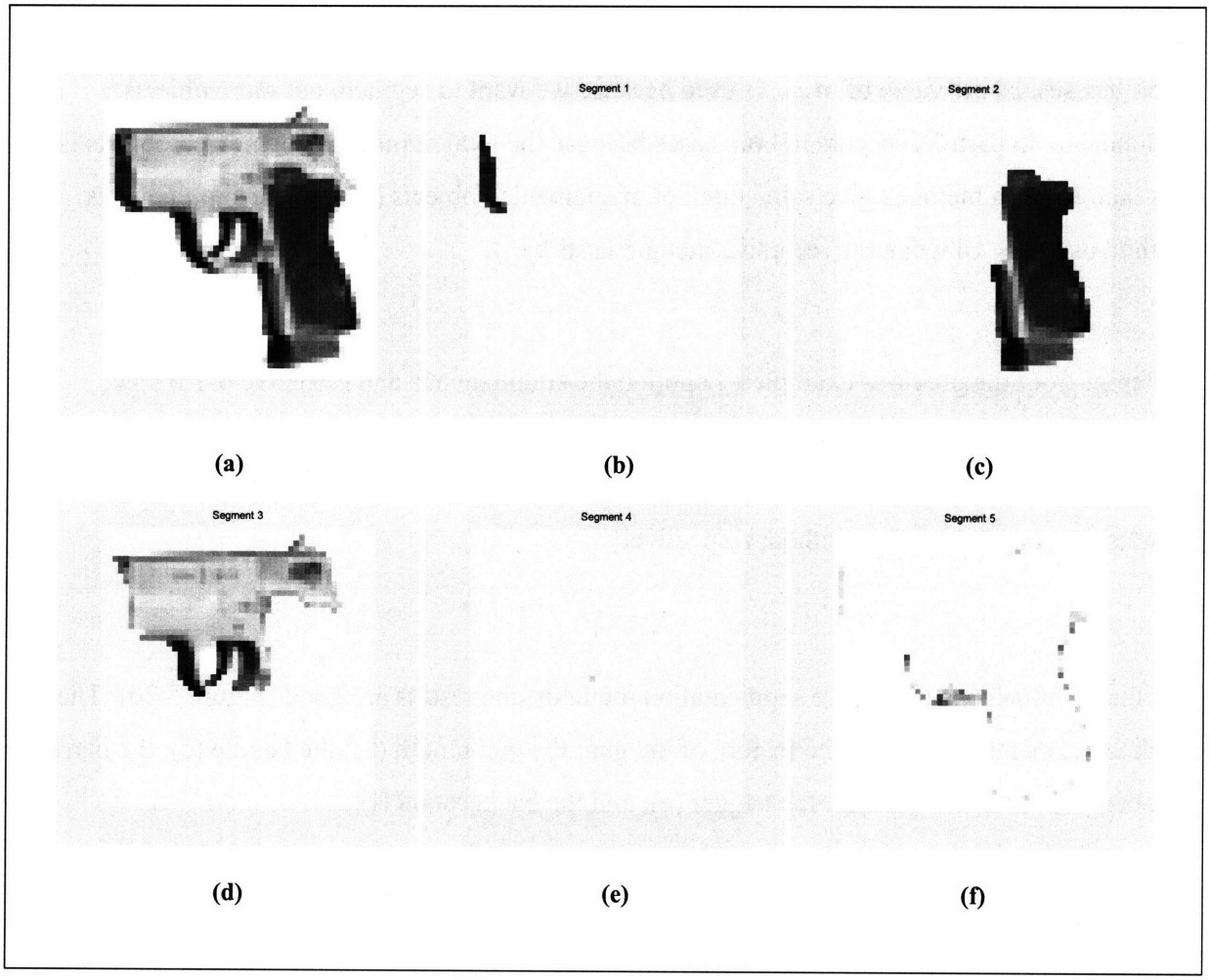


Figure 5.20 **Image segmentation of a 50x50 image of a gun by the simultaneous 0-1 method. Figure (a) shows the original image and figures (b) – (f) shows the segmented images. Notice that the gun has small intensity (dark) at the gun mouth, trigger and handle. These are the distinct features of the gun. The algorithm segmented the distinct features of the gun: gun mouth (b); the handle (c); the barrel and trigger; the white gap near the trigger (e); and the background (f). The parameters used are: $r = 0, \sigma_I = 0.1, n_{min} = 0.2$ and $k = 4$.**

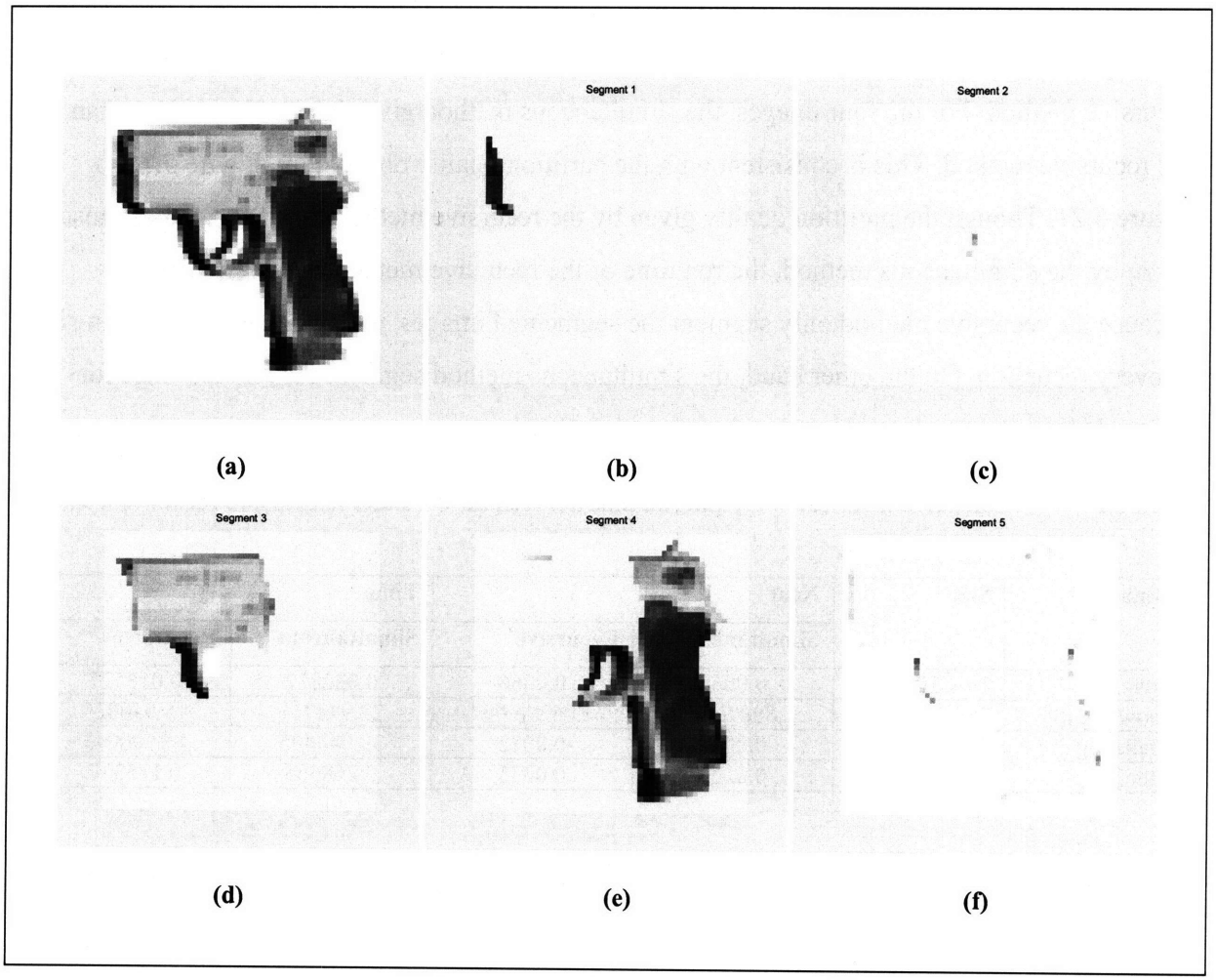


Figure 5.21 **Image segmentation of a 50x50 image of a gun by the recursive 0-1 method. Figure (a) shows the original image and figures (b) – (f) shows the segmented images. Notice that the gun has small intensity (dark) at the gun mouth, trigger and handle. These are the distinct features of the gun. The algorithm segmented the image into the gun mouth (b); the handle (c); the barrel and trigger; and the white gap near the trigger. The disntict features of the gun are not well seperated. The parameters used are: $r = 0$, $\sigma_I = 0.1$, $n_{min} = 0.1$ and $k = 4$.**

From Figure 5.16 to Figure 5.21, we see that both simultanneous and recursive method are able to segment out multiple objects from an image. However, the partition quality given by the simultaneous method is better. The recursive method does not give 'clean' segmentation. The segmentation is often partial. The partition quality can also be reflected in the *Ncut* value. Table 5.2 gives the comparison of the *Ncut* value and the run time between the simultaneous and recursive method. For the four images, the simultaneous method gives lower *Ncut* values than the recursive method. This is consistent with the partition quality observed in Figure 5.16 to Figure 5.21. Though the partition quality given by the recursive method is not as good as those given by the simultaneous method, the run time of the recursive method is shorter. This is because the recursive method only segment the segmented images, which become smaller in size in every recursion. On the other hand, the simultaneous method segment the whole image for every object.

Table 5.2 Comparison of *Ncut* value and run time between the 0-1 simultaneous and recursive image segmentation.

Image	Size	Ncut		Time	
		Simultaneous	Recursive	Simultaneous	Recursive
synthetic	10 x 10	0.0066	0.0066	0.022812	0.015527
bowling.jpg	384 x 550	1.8009e-004	2.0629e-004	22.944711	15.940025
balls.jpg	360 x 360	0.0026	0.0072	39.045495	14.500043
gun.jpg	50 x 50	0.0265	0.0335	0.268565	0.175337

5.6.3 Resized Image Segmentation & Refinement

In this section, I tried to segment natural images of larger size and with more complicated features and background. The images used are obtained from Jiabo Shi's website[4]. The segmentation of these images require the use of larger r value in the graph construction, the resized image segmentation scheme and the refinement process for better segmentation.

Figure 5.22 shows the image segmentation of a 132x130 image of a baby. Due to the large size of the objects and the use of simple 4-connecetd graph construction scheme ($r = 0$), the situation decribed in Section 5.4.3, where the image is segmented fractionally, occured.

[4] http://www.cis.upenn.edu/%7Ejshi/software/

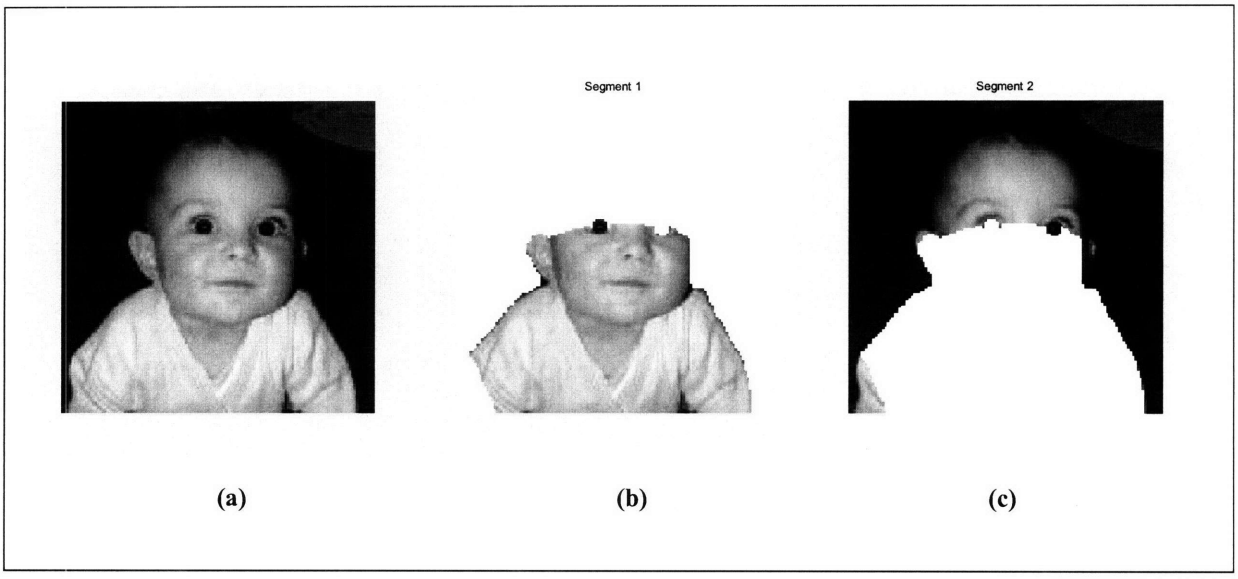


Figure 5.22 Image segmentation of a 132x130 image of a baby using the simple 4-connected graph construction scheme ($r = 0$) . Figure (a) shows the original image and figures (b) and (c) show the segmented images. The segmented baby is fractional. Only half of the face of the baby is segmented out. The parameters used are: $r = 0$, $\sigma_I = 0.1$, $n_{min} = 0.3$ and $k = 1$.

To overcome the fractional segmentation, I use a larger r in the graph construction (Figure 5.23). Alternatively, the Resized Image Segmentation Scheme solves this problem too (Figure 5.24).

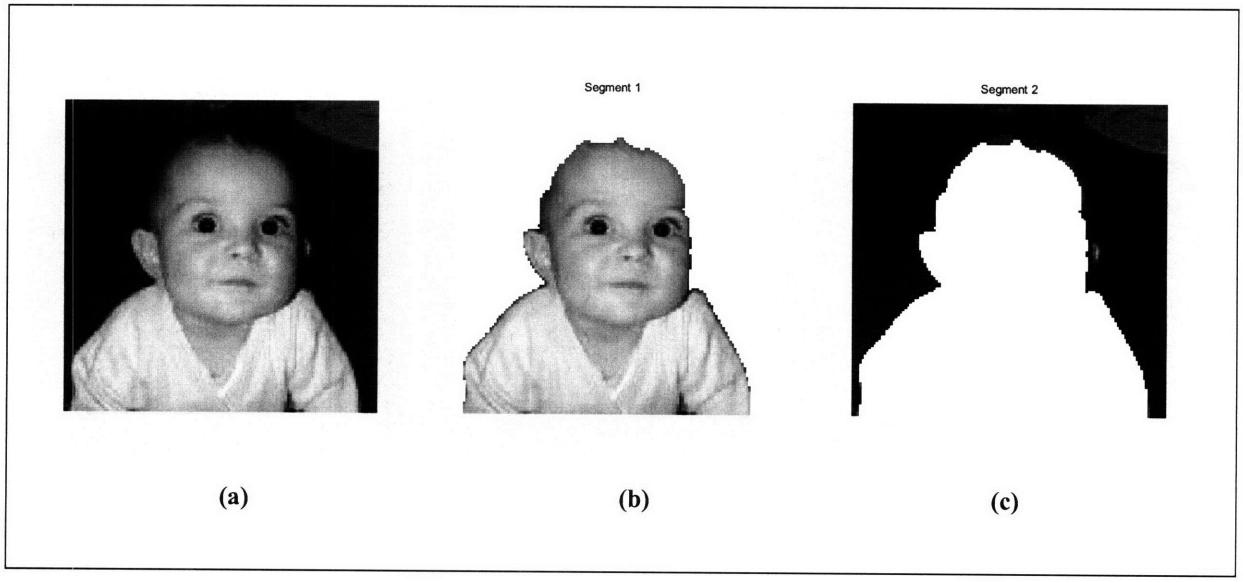


Figure 5.23 Image segmentation of a 132x130 image of a baby using r-radially connected graph construction scheme with $r = 3$. Figure (a) shows the original image and figures (b) and (c) show the segmented images. The segmented baby is no longer fractional. The parameters used are: $r = 3$, $\sigma_I = 0.1$, $\sigma_D = 1$, $n_{min} = 0.3$ and $k = 1$.

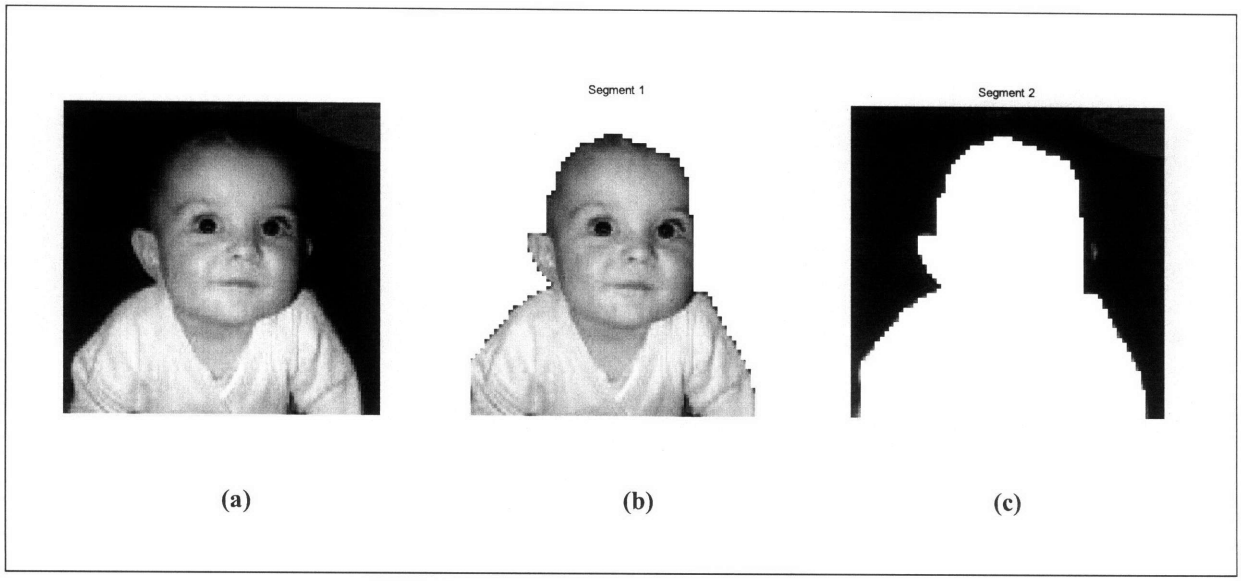


Figure 5.24 Image segmentation of a 132x130 image of a baby using the Resized Image Segmentation Scheme. Figure (a) shows the original image and figures (b) and (c) show the segmented images. The segmented baby is not fractional. However, the boundary of the objects (baby) is not smooth (saw-tooth). The parameters used are: $\theta = 0.5$, $r = 0$, $\sigma_I = 0.1$, $n_{min} = 0.1$ and $k = 1$.

The boundary of the segmented 'baby' by the Resized Image Segmentation Scheme in Figure 5.24 is not smooth (saw-tooth shape) and can be smoothed by the Refinement Scheme (Figure 5.25).

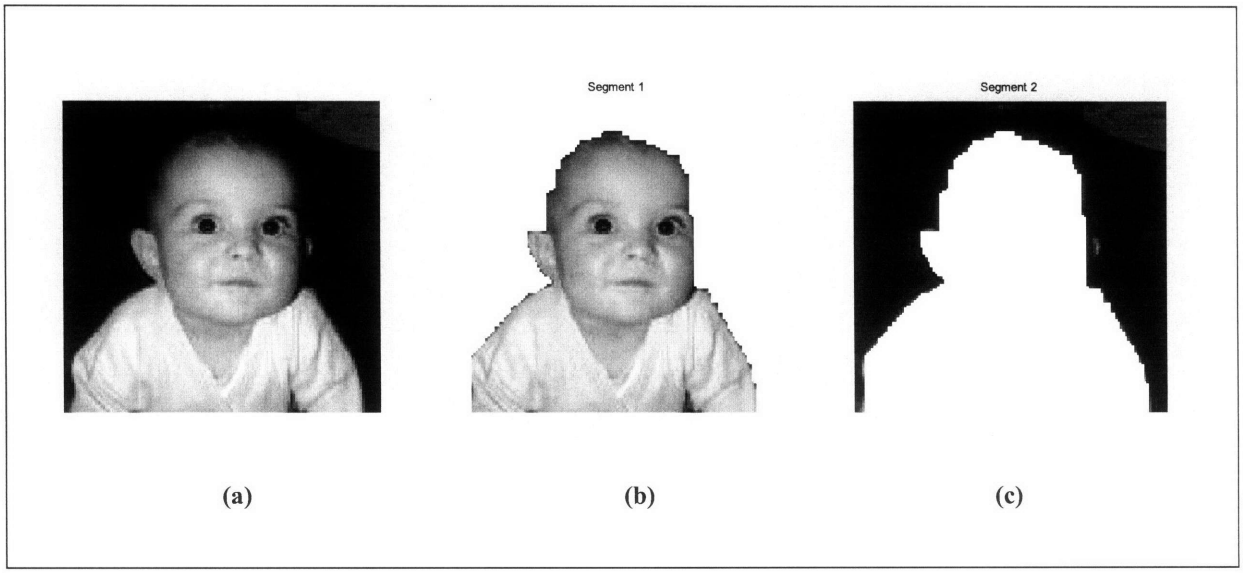


Figure 5.25 Image segmentation of a 132x130 image of a baby using the Resized Image Segmentation Scheme and Refinement Algorithm . Figure (a) shows the original image and figures (b) and (c) show the segmented images. The segmented baby is not fractional. The boundary of the objects (baby) is smoother (fewer saw-tooth edges). The parameters used are: $\theta = 0.5$, $r = 0$, $\sigma_I = 0.1$, $n_{min} = 0.1$ and $k = 1$.

Table 5.3 shows the Comparison of the *Ncut* values and run times for the image segmentation of the baby image using different schemes. The run times include the graph construction time, the image segmentation time, the refinement time and the total time. The image segmentation time measures the time for the graph partitioning methods to partition a graph and give the discrete partitions. The time for resizing and projecting the image is not included because they are negligible compared to the other run times.

Table 5.3 ***Ncut*** **value and run times for image segmentation of the baby image using different schemes.**

No	Scheme	*r*	*Ncut*	Graph Construction Time (s)	Image Segmentation Time (s)	Refinment Time (s)	Total time (s)
1	Original	0	0.0060	1.175968	0.692821	-	1.868789
2	Original	3	0.0071	546.490723	5.718346	-	552.209069
3	Resized (0.5)	0	0.0070	0.234286	0.138188	-	0.372474
4	Resized (0.5) & Refined (Thorough)	0	0.0061	0.232790	0.138427	1.991172	2.362389
5	Resized (0.5) & Refined (Fast)	0	0.0061	0.232790	0.138427	1.844268	2.215485

We see that in Figure 5.22, the segmented baby is fractional. We overcome this problem by using a larger *r* graph construction scheme. However, from Table 5.3, we see that the computation times, especially the graph construction time, increase by a large amount. The graph construction time increases from 1.175968 to 546.490723 seconds. The image segmentation time has increased from 0.692821 to 5.718346. Due to the long run time, I used the Resized Image Segmentation Scheme. The results is non-fractional segmentation (Figure 5.24) and decreased the graph construction time from 546.490723 to 0.234286 seconds. The image segmentation time decreases from 5.718346 to 0.138188 seconds. The resized scheme without refinement has the shortest total time. However, resizing the image causes the saw-tooth edges (Figure 5.24). Hence, I use the Refinement Scheme to smoothe the edges. The refinement process is expensive compared to the other process. The thorough refinement time is 1.991172 seconds while the graph construction time is only 0.232790 seconds and the image segmentation time is 0.138427 seconds. I used the fast refinement scheme to improve the refinement time to 1.844268 seconds.

Though the refinement process is expensive, it is not as expensive as the graph construction time of the second scheme in Table 5.3, and yet the partition quality is comparable. The *Ncut* value after refinement has improved from 0.0070 to 0.0061. Both thorough and fast refinement scheme give same Ncut improvement, but the latter is faster.

Figure 5.26 to Figure 5.28 show three more large natural images segmented by the 0-1 method with the aid of the Resized Image Segmentation Scheme and Fast Refine Scheme. In Figure 5.26, the segmented panther is fractional (without the tail part and a part of the front leg). The segmentation around the right ear includes pixels that do not belong to the panther. This is because of the similarity in intensity between certain parts of the panther and the background. The background of the image is complicated and has varying pixel intensity. The high r value is used ($r = 4$) to segment out the panther form the background. Apart from giving better segmentation, the high r value also has another effect as shown by the two dark patches separated from the panther. The first dark patch at the bottom is the paw of the panther. Another dark patch at the right side of the image is not a part of the panther, but only a stone in the background. These two dark patches are segmented out together with the panther due to the high r value. The last feature to notice is the mouth of the panther. It is not segmented out with the panther due to its bright intensity and the high r value.

Figure 5.27 shows another example of segmenting an object from the complicated background. The background of the image is complicated with white and dark patches. The bear is segmented together with its shadow and some other dark background patches around the bear.

Figure 5.28 shows the image segmentation of a 384x512 image of Boston city using the Resized Image Segmentation Scheme and Fast Refinement Scheme. The image is segmented into two parts: the sky and the bright side of the buildings; the city and the dark side of the buildings. The bright side of the buildings has similar intensity with the sky and hence they are segmented together with the shy. The two patches at the top corners of the image is due to the sinks at the two corners. This is one of the weaknesses of the 0-1 method.

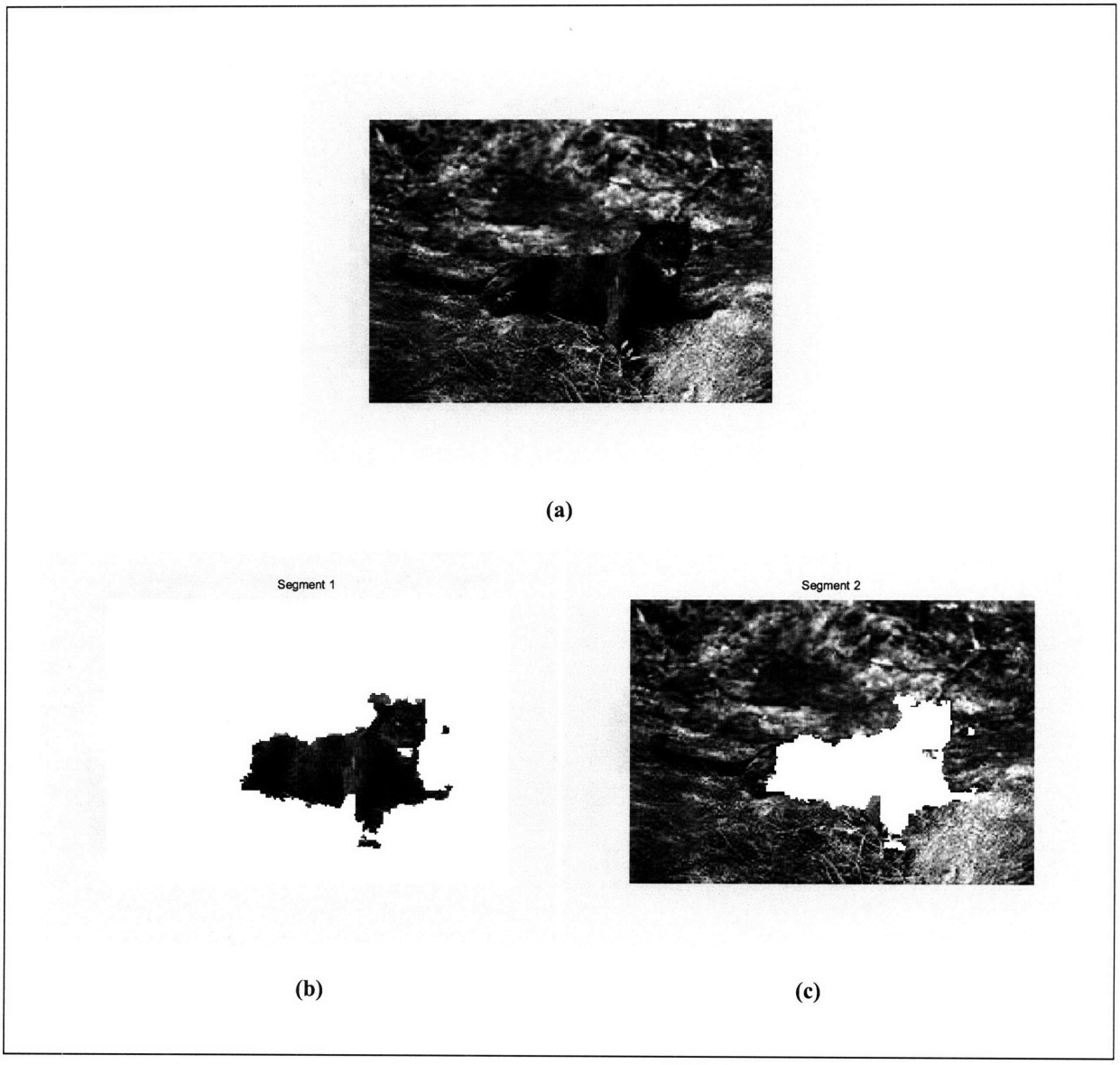
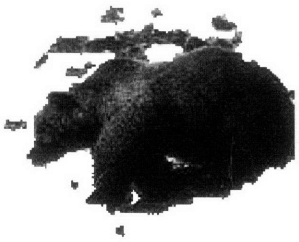


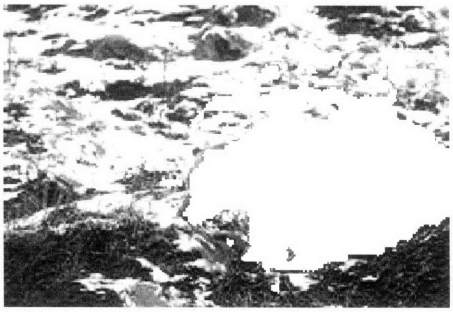
Figure 5.26 Image segmentation of a 232x160 image of a panther using the Resized Image Segmentation Scheme and Fast Refinement Algorithm. Figure (a) shows the original image and figures (b) and (c) show the segmented images. The background of the image is complicated and has varying pixel intensity. Though the panther is relatively darker, some parts of the body are similar in intensity with the background. Consequently, the tail part and a part of the front leg are excluded from the segmented panther in (a). The segmented panther is fractional. Notice the dark patches at the bottom and the right side of the image. The former correspond to the paw of the panther whereas the latter is merely a stone in the background. The last feature to notice is the mouth of the panther. It is not segmented out with the panther due to its bright intensity and also the high r value. The parameters used are: $\theta = 0.25$, $r = 4$, $\sigma_I = 0.06$, $\sigma_D = 1$, $n_{min} = 0.1$ and $k = 1$.

(a)

Segment 1

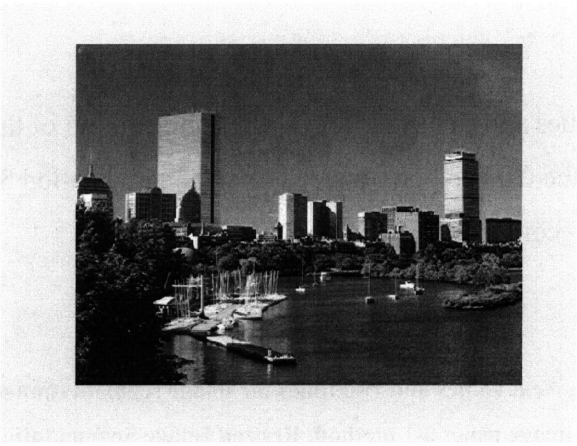


Segment 2

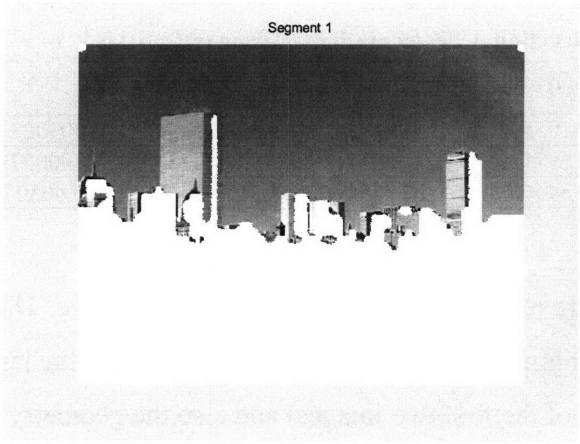


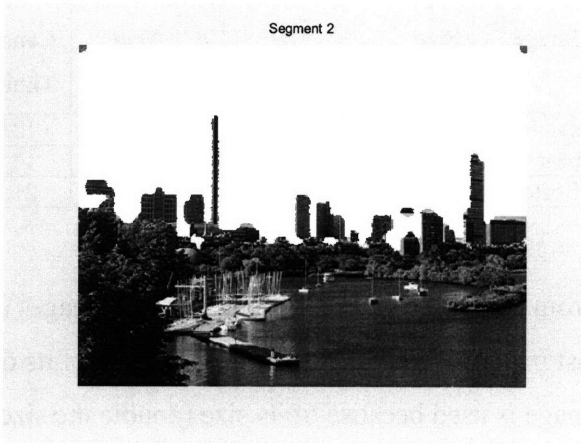
(b) (c)

Figure 5.27 Image segmentation of a 240x160 image of a bear using the Resized Image Segmentation Scheme and Fast Refinement Scheme. Figure (a) shows the original image and figures (b) and (c) show the segmented images. The background of the image is complicated with white and dark patches. The bear is segmented together with its shadow and some other background patches, which have similar intensity. The parameters used are: $\theta = 0.25$, $r = 4$, $\sigma_I = 0.06$, $\sigma_D = 1$, $n_{min} = 0.1$ and $k = 1$.



(a)





(b) (c)

Figure 5.28 Image segmentation of a 384x512 image of Boston city using the Resized Image Segmentation Scheme and Fast Refinement Scheme. Figure (a) shows the original image and figures (b) and (c) show the segmented images. The image is segmented into two parts: the sky and the bright side of the buildings; the city and the dark side of the buildings. Notice that the bright side of the buildings has similar intensity with the sky. The two patches at the top corners of the image is due to the sinks at the two corners. The parameters used are: $\theta = 0.125$, $r = 4$, $\sigma_I = 0.03$, $\sigma_D = 1$, $n_{min} = 0.2$ and $k = 1$.

From the first two examples in Figure 5.26 and Figure 5.27, we see the advantage and disadvantage of the high r value. It helps to segment parts of the object that are not connected to the main object, but at the same time also pick up unwanted objects.

The parameters, *Ncut* values and run times for image segmentation of the panther, bear and Boston city image using the 0-1 method, Resized Image Segmentation Scheme and Fast Refinement Scheme are recorded in Table 5.4.

Table 5.4 **Parameters, *Ncut* values and run times for image segmentation of the panther, bear and Boston city image using 0-1 method, Resized Image Segmentation Scheme and Fast Refinement Scheme.**

Image	Size	θ	r	*Ncut*	Graph Construction Time (s)	Image Segmentation Time (s)	Refinment Time (s)	Total time (s)
panther	232x160	0.25	4	0.0133	12.610554	0.851824	5.040625	18.503003
bear	240x160	0.25	4	0.0082	14.369713	0.926331	8.804013	24.100057
Boston	348x512	0.125	4	0.0043	21.534568	1.174547	141.791389	164.500504

From the table, we see that the first two images are resized to a quarter of their original size. The last image is shrunk further to one eighth of its original size. The low shrinking factor for the last image is used because of its size (double the size of the first two images) and also the geometry of the objects in the image. Since most of the objects are buildings, which have simple boundary (straingt lines), the resizing does not affect the segmentation result much. The resizing allows a high r value to be used in the graph construction for better results. It also reduces the graph construction time. The refinement and the graph construction process are the two most costly processes. The image segmentation time is negligible compared to them. In the first two images, the graph construction time is longer than the refinement time. In contrast, for the last images, the refinement time is enormously larger than the graph construction time. This is because of the long boundary along the skyscrapers.

5.7 Advantages and Disadvantages

Since the 0-1 method is developed from the Isoperimetric Partitioning, it share a few advantages with the Isoperimetric Partitioning. Both methods solve a linear system to obtain the segmentation, which is much faster than solving an eigensystem (Normalized Cut method). The two methods require a low r value in graph construction. In all the examples shown in this chapter, the highest r value used is four. In contrast, Normalized Cut method needs larger r (typically $r > 5$). The low r reduces the graph construction time and the image segmentation time. The methods are also robust to noise and are able to segment large images. The image segmentation of a large image using the Normalized Cut method can be prohibitive.

The advantage not shared by the Isoperimetric Partitioning is its ability to locate the objects in images using multiple sink and sources, as described in Section 5.2.1. Using this concept, we can perform not only the recursive k-way image segmentation but also the simultaneous k-way image segmentation.

Another advantage is that the method focuses on the objects of the image. This means that the method only needs *2*-way image segmentation to segment out an object from an image. Other methods are often not able to segment out an object using only *2*-way. They may need to segment more than one partition (k-way image segmentation) to segment out the object. For illustration, we compare the segmentation given by the 0-1 method in Figure 5.29 and the Normalized Cut method in Figure 5.30.

In Figure 5.30, the Normalized Cut method needs 3-way to totally segment out the bird. On the other hand, the 0-1 method only needs 2-way image segmentation (Figure 5.29). This is because the Normalized Cut method finds the global minimum *Ncut*, which may not be the exact criterion for good image segmentation; while the 0-1 method find the minimum *Ncut* between the sinks

and sources. In this case, the source is inside the bird and the sinks are at the corners. Hence, we obtain the cut around the object, which is between the sinks and source and minimizes the Ncut value.

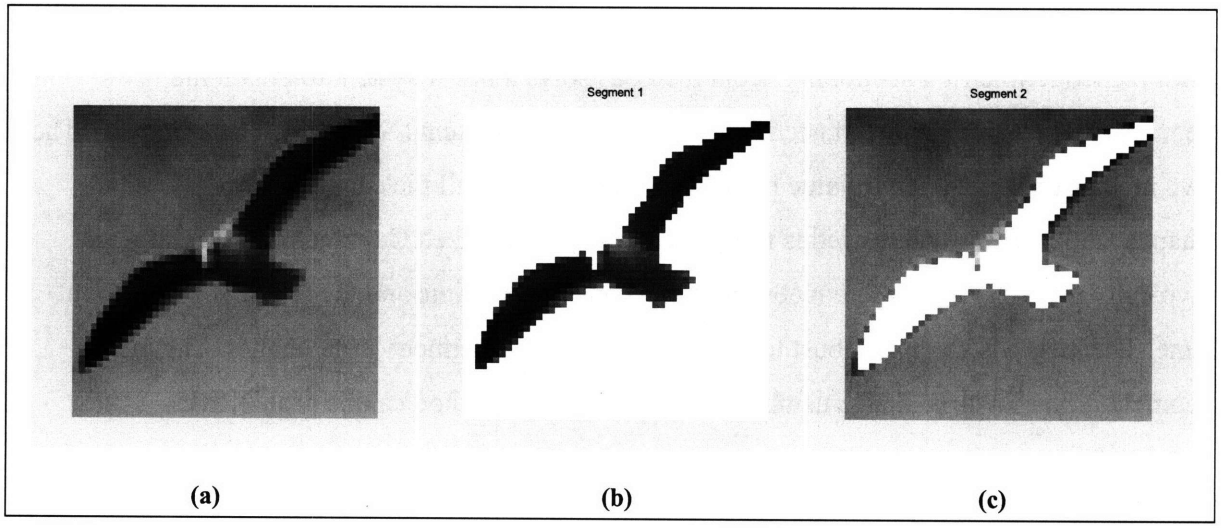


Figure 5.29 Image Segmentation of a 50x50 image of a bird using the 0-1 method. Figure (a) shows the original image and (b) and (c) show the segmented images using 2-way image segmentation. Notice that the bird (object) is segmented out in the first bi-partition (2-way image segmentation).

The disadvantage of the method is its reliability on the sinks and sources. The wrong placement of the sources gives bad segmentation. Apart from the placement of the sources, the location of the sinks at the four corners of the image also limits the image segmentation capability. The method may fail to segment out the objects located near the corners.

Another disadvantage of the method is its parameters. Apart from the parameters used in Normalized Cut method (r, σ_I, σ_D), a few extra parameters are involved in the image segmentation process. They are n_{min} and k. Inputting a k value lower than the actual number of objects may cause failure to the method.

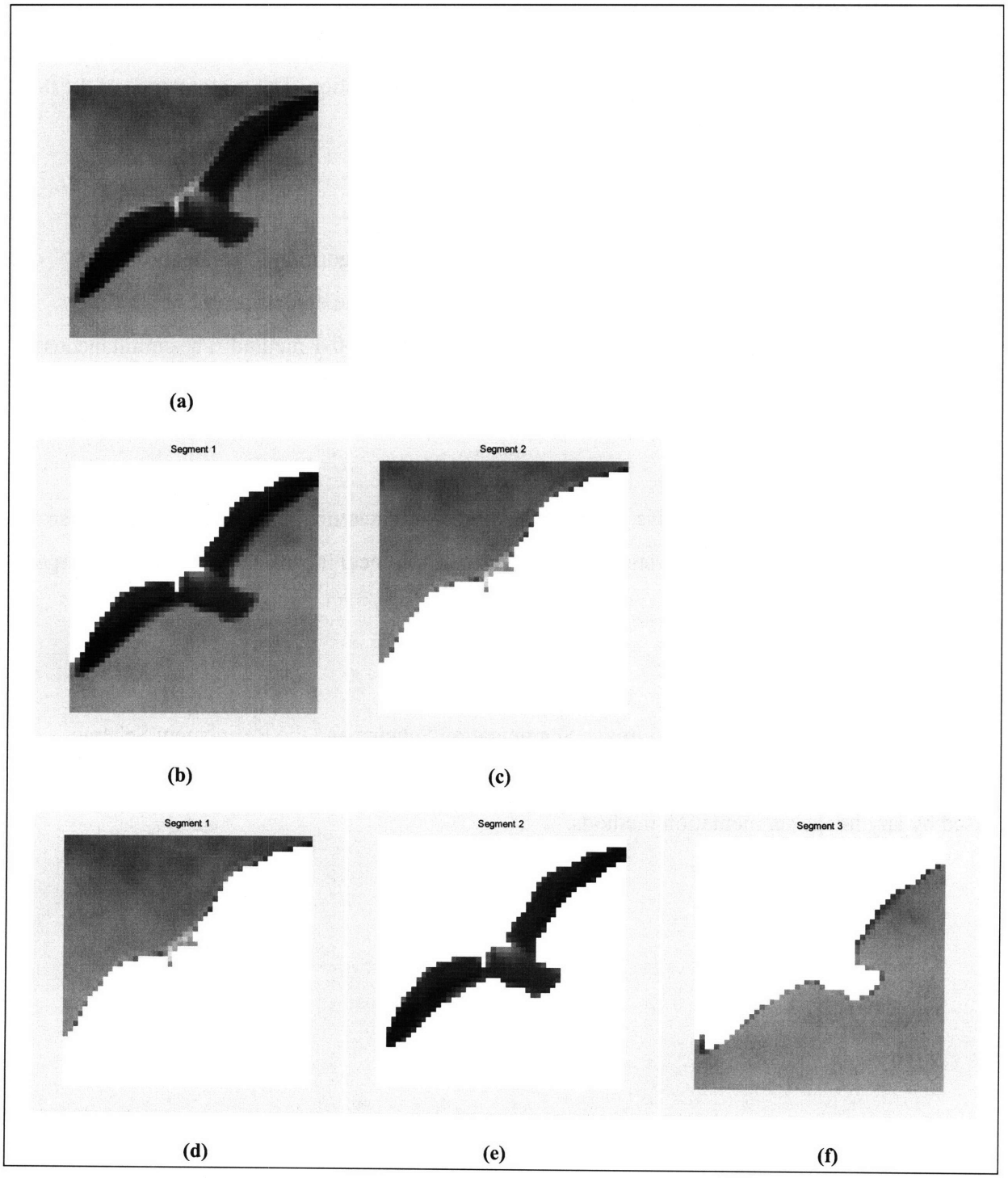


Figure 5.30 Image Segmentation of a 50x50 image of a bird using the Normalized Cut method. The first row shows the original image and the second and third row show the segmented images using 2-way and 3-way image segmentation, respectively. Notice that the bird (object) is only segmented out (e) by the 3-way image segmentation.

5.8 Summary

In this chapter, I have applied the 0-1 method in image segmentation. The performance of the 0-1 method is as good as the Isoperimetric Partitioning.

Based on the concept of the 0-1 method, I have developed the algorithm to locate the objects' location to place the source. The ability to place the sources in the objects enables both the simultaneous and recursive k-way image segmentation using the 0-1 method. The simultaneous method performs better in terms of segmentation quality while the recursive method is faster.

The advantage of the 0-1 method is its speed and object-oriented image segmentation. It is also robust to noise and able to detect small pixel intensity difference. It can segment relatively large image with quality and speed.

I have also developed the Resized Image Segmentation Scheme and the Refinement Scheme (Fast and Thorough), which can speed up the image segmentation process. The schemes can be used by any image segmentation methods.

Chapter 6 Conclusions and Future Works

6.1 Conclusions

Graph partitioning is important because of its application in the important fields such as data clustering, image segmentation and parallel computing. Graph cut techniques like the Minimum Cut method, Normalized Cut method and Isoperimetric Partitioning are used to partition a graph. The Minimum Cut method gives imbalanced partitions. To overcome the imbalanced partitioning, the Normalized Cut method is used. However, it is computationally expensive and hence the application of the method, especially in image segmentation, is limited. The Isoperimetric Partitioning is faster and more stable, but the partitions' quality is compromised due to the limitation of grounding. Hence, there is a need to find a graph partitioning algorithm that is fast and gives good partitions. To achieve this, the current methods need to be studied to know their strengths and weaknesses.

Before coming out with a new graph partitioning algorithm, I have first reviewed the current graph partitioning methods: the Minimum Cut, Normalized Cut and Isoperimetric Partitioning methods. For unweighted graphs, the graph partitioning methods try to give partitions that cut through the least number of edges. The connectivity of graphs is important in the unweighted graph partitioning. Nodes with similar inter-connectivity will be grouped together. If all the nodes have the same connectivity with each others, Isoperimetric Partitioning gives partitions that vary with the sink (ground) location while the Normalized Cut become unstable and give different partitions from time to time. For weighted graph partitioning, the graph partitioning methods try to partition graphs along the links with lower weights (weak links).

To achieve k-way graph partitioning, the Minimum Cut method and Isoperimetric Partitioning needs to be applied recursively on the partitioned graph. On the other hand, the Normalized Cut method can achieve this by either the recursive or the simultaneous way. However, the k partitions given by the two ways may differ.

To measure the partition quality, Isoperimetric Partitioning uses the Isoperimetric constant while the Normalized Cut method uses the normalized cut (*Ncut*) value. Unlike the *Ncut* value, the Isoperimetric constant lacks the ability to measure the *k*-way partitions. However, I also discovered that the *Ncut* is not the universal and precise measurement for good partitions

Though image segmentation is an application of weighted graph partitioning, the success of image segmentation also depends on the construction of graphs from images. Generally, there are three construction schemes: the 4-connected, 8-connected and *r*-radially connected edge construction schemes. The advantage of 4-connected and 8-connected schemes is their simplicity. They only use the usual pixel intensity weighting functions. The *r*-radially connected scheme is advantageous in segmenting images with complicated objects. It allows the use of distance weighting functions besides the intensity weighting functions. Among the four intensity weighting functions (Equation 3.1 – 3.4), the third function is the best as it has faster decay rate, which favors the creation of weak links that helps the graph partitioning.

In Chapter 3, I have compared the image segmentation performance of the three methods and the Spectral Rounding method (a variant of the Normalized Cut method) in terms of their sensitivity to the change in pixel intensity difference between objects and background, their ability to segment large images, their robustness towards noise and their computation speed. The Minimum Cut method performs well in all the criteria except that it is vulnerable to noisy image. The Normalized Cut method performs poorly in all the criteria. The Spectral Rounding method is good at segmenting large images but requires long computation time and is sensitive to the pixel intensity difference between objects and background. The overall performance of the Isoperimetric Partitioning is good. It is sensitive enough to detect small pixel intensity difference, and is good at segmenting large and noisy images. Furthermore, it is fast.

In order to meet the requirement of fast and good partitions, I have developed a new graph partitioning method – 0-1 method by combining the electrical circuit concept of Isoperimetric

Partitioning and the minimum normalized cut discretization of the Normalized Cut method. It requires the placement of sinks and sources. It tries to find a cut in between the sink and sources that minimizes the *Ncut* value.

For unweighted graphs, the criteria for the locations of sinks and sources are:

i. The sink and sources are located as far as possible from each other (for graph with coordinates: mesh).
ii. No link or short path is desirable between the sink and sources.
iii. The sink and sources must be located at the correct segmented parts separately

To fulfill these criteria, I have created the Auto 0-1 algorithm, which finds good sinks and sources based on the assumption that the nodes are numbered in the order of distance. Using this method, I have partitioned ten different unweighted graphs (meshes) and have made comparison with the Normalized Cut method. The method is able to give bi-partitions of comparable quality to that given by the Normalized Cut method. Though the partition quality is not guaranteed to be better than the Normalized Cut method, the 0-1 method is quicker than the Normalized Cut method. In the case of recursive k-way partitioning, the Auto 0-1 algorithm recursively bi-partition graphs to obtain k partitions. The method performs badly. This is because the assumption that the usual node numbering convention is applied in the graph is no longer true in the partitioned graph.

With good sinks and sources, the 0-1 method can produce a 0-1 vector that is close to the Fiedler vector. Based on this fact, I have developed the Fiedler Quick Start algorithm. It applies inverse power method to converge the 0-1 vector to the Fiedler vector. From the experiment results, the Fiedler Quick Start has shown its potential to compute the Fiedler vector faster than solving the generalized eigensystem.

In Chapter 5, I have applied the 0-1 method in image segmentation. From the performance tests, we know that the 0-1 method is equally good in terms of the sensitivity to the change in pixel

intensity difference between objects and background, the ability to segment large images, their robustness towards noise and the computation speed compared to the Isoperimetric Partitioning. Based on the concept of the 0-1 method, I have developed the Source Candidates algorithm to generate source candidates within the objects in images. By using the average of the source candidates as the source, I have applied the 0-1 algorithm in 2-way image segmentation. The method has successfully single out objects from the test images. The ability to place the sources in the objects also enables both the simultaneous and recursive *k*-way image segmentation using the 0-1 method. Both methods are able to segment out objects from the test images. The simultaneous method performs better in terms of segmentation quality while the recursive method is faster.

Apart from the 0-1 image segmentation, I have also developed the Resized Image Segmentation Scheme and the Refinement Scheme (Fast and Thorough), which can speed up the image segmentation process. The Resized Image Segmentation Scheme shrinks an image so that the image segmentation methods can be applied on a smaller image and thus, faster computation is achieved. The Refinement scheme is based on the assumption that lowering the *Ncut* value improves the segmentation. The Thorough Refinement Scheme checks all the boundary nodes for improvement in *Ncut* value; while the Fast Refinement Scheme selectively checks the boundary nodes based on the partitions' average pixel intensity. Both schemes can be used by any graph based image segmentation methods.

The advantage of the 0-1 method is its speed in partitioning a graph. In image segmentation, it allows the use of low *r* graph construction scheme. It is robust to noise and is able to segment large images. It has the ability to locate the objects in images and enable *k*-way image segmentation in both recursive and simultaneous ways. Unlike other image segmentation methods, it focuses on the objects of the image and it is able to segment out an object in the first bi-partition. The disadvantage of the method is its reliability on the sinks and sources.

6.2 Future Works

In Chapter 3, the performance comparison was done using a synthetic image. The comparison result can be more conclusive if natural images are used. In the future, I shall extend the performance tests to include more images of different types.

For unweighted graph partitioning, though the Auto 0-1 algorithm gives fast partitions, it does not guarantee the improvement of the partition quality due to the limitation of sinks and sources. This also caused the *k*-way graph partitioning using the 0-1 method to fail. However, in image segmentation, this problem is solved by the Source Candidates algorithm. Hence, we can work on the extension of the Source Candidates algorithm to general graph partitioning.

Though he Fiedler Quick Start algorithm has the potential to compute the Fiedler vector faster than solving the generalized eigensystem, the method is still not robust enough as a few parameters are involved. The future works to improve the algorithm includes the optimization of the parameters and better sinks and sources selection.

One of the problems faced by the 0-1 *k*-way image segmentation (both recursive and simultaneous) is the redundant partitions. The reason is partly due to *k-means* function used in the algorithms. The *k-means* function is an iterative heuristic algorithm, which can converge to a local minimum [10]. We can improve the algorithms by improving *k-means* function. The improvement includes using a better starting point for the *k-means* iterations [10] or using the average result of a few runs of *k-means* [9].

Though the Refinement scheme can improve the image segmentation, the process is costly, especially when the partitions have long boundaries. Currently, the Fast Refinement scheme is based on the average pixel intensity of the partitions. A faster refinement scheme is possible by

using only the average pixel intensity of the nodes at the boundary region, instead of the whole partition.

REFERENCES

[1] D. Bertsimas, J. N. Tsitsiklis, *Introduction to Linear Optimization,* Athena Scientific, 1997.

[2] Y. Boykov and V. Kolmogorov, "An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision," *Trans. PAMI*, 2004.

[3] Gary L. Miller and David Tolliver, "Graph Partitioning by Spectral Rounding: Applications in Image Segmentation and Clustering," *Technical Report CMU-CS-07*, CMU, 2007.

[4] J. R. Gilbert, G. L. Miller, and S.-H. Teng, "Geometric Mesh Partitioning: Implementation and Experiments," *SIAM Journal on Scientific Computing*, 19 (1998), pp. 2091-2110.

[5] L. Grady, "Space-Variant Computer Vision: A Graph-Theoretic Approach," *PhD Dissertation*, Boston University, 2004.

[6] L. Grady and Marie-Pierre Jolly, "Weights and Topology: A Study of the Effects of Graph Construction on 3D Image Segmentation", Accepted to MICCAI 2008.

[7] L. Grady and E. L. Schwartz, "The isoperimetric algorithm for graph partitioning," *SIAM Journal on Scientific Computing*, 2005.

[8] D. J. Higham, G. Kalna, and M. Kibble, "Spectral clustering and its use in bioinformatics," *J. Computational and Appl. Mathematics*, 204:25:37, 2007.

[9] K-means algorithm, Wikipedia, Retrieved August 10, 2008 from http://en.wikipedia.org/wiki/K-means_algorithm.

[10] *kmeans* :: Functions (Statistic Toolbox), Retrieved August 10, 2008 from MATLAB help file.

[11] A. K. Jain and R. C. Dubes, *Algorithm for Clustering Data*. Prentice Hall, 1998.

[12] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Trans. PAMI*, 22:888-905, 2000.

[13] G. Strang, *Computational Science and Engineering*, Wellesley-Cambridge, 2008.

[14] Z. Wu and R. Leahy, "An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation," *IEEE Trans. PAMI,* 15:1,101-1,113, 1993.