Diploma Thesis

# A Comparative Study of Image Segmentation by Means of Normalized Graph Cut Methods

Christian Bähnisch
[2baehnis@informatik.uni-hamburg.de]
September 10, 2008

Supervisors
Prof. Dr. rer. nat. Leonie Dreschler-Fischer
Dr. rer. nat. Ullrich Köthe

Cognitive Systems Group
Department of Informatics
Faculty of Mathematics, Informatics and Natural Sciences
University of Hamburg
Vogt-Kölln-Strasse 30, 22527 Hamburg, Germany

# Acknowledgments

First of all I would like to thank Ullrich Köthe for providing me with such an interesting and challenging topic and for the useful suggestions and discussions which helped me in actually elaborating it. Then I would like to thank Leonie Dreschler-Fischer who also always helped me with suggestions and discussions valuable in the course of working out my diploma thesis.

Very special thanks go to Hans Meine for the GeoMap, reading parts of my thesis, useful suggestions, and tons of tips on programming and LaTeX related topics which greatly increased the quality of my work (Hans, I hope I was not getting on your nerves too much).

I would also like to thank Rainer for proofreading my thesis and Leonid for encouraging me and giving me the freedom to work on my thesis.

At last but not least, I thank Doreen Jirak who always had the time for reading my work but especially for encouraging me, covering my back and bearing my bad moods.

# Contents

# 1 Introduction

Image segmentation is the task of dividing an image into meaningful parts corresponding to objects or part of objects. It is generally seen as an important step towards the high level interpretation of visual content. Over the last years so called energy function based approaches became very popular as they lead to segmentations fulfilling a global optimality criteria. Such methods are often formulated within a graph based framework. That means, a graph structure is directly created from the pixel elements of an image with edge weights encoding the pairwise similarity of neighboring pixels. Graph cuts can then be used to optimize certain classes of binary energy functions. Especially, normalized versions of the minimum cut problem – which we call *normalized cuts* – have been proposed. The usefulness of them in context of the segmentation problem has been reported in the literature many times.

However, using graphs being derived from image pixels has two significant drawbacks:

1. The number of vertices in the graph equals the number of pixels and is therefore often too high for a fast solution.

2. the pairwise similarities encoded in the edge weights can only be very simple as only the color information at two points in the image is available.

The basic idea of this diploma thesis is therefore to use an initial segmentation step which provides a first aggregation of the image elements into many small sized regions from which the graph is then build. This preprocessing step significantly reduces the number of element to process and makes the incorporation of more sophisticated similarity measures possible. Normalized cuts can then be used to form the final segmentation.

We use two established segmentation methods for the initial segmentation step which are the exact watershed transform developed by Meine and Koethe [2005] and the mean shift method introduced to the field by Comaniciu and Meer [2002]. For the similarity measure we use three different versions; two measures which can be directly derived from the regions of the initial segmentation and one complex measure which is based on the earth mover distance.

The usage of normalized cuts in the described setting is the main interest of this work. For this, we have identified five different types from the literature. Especially, we want to investigate:

- How does the usage of normalized cuts improves the quality of the initial segmentation?
- What are the inherent differences of the final segmentations if different cuts are used?
- Is it possible to identify a normalized cut type which is superior to the other ones?

In order to investigate how the selection of initial segmentation method, similarity measure, and normalized cut impacts on the final segmentation quality we perform an exhaustive evaluation of all possible combinations.

The rest of this thesis is organized as follows:

- **Chapter 2** starts with the definition of the image function which is used here as the mathematical model for representing image data. A more elaborate discussion of the segmentation problem and the graph based approach using graph cuts follows. Moreover, we describe the exact watershed transform and the mean shift method for the initial segmentation step.
- **Chapter 3** first introduces the mathematical concepts needed for graph cuts. We give a more general overview of graph cuts related approaches in computer vision and discuss how they can be used for image segmentation. The different kinds of edge weights used for the graph structure are introduced as well. After this, we focus on the class of normalized cuts giving an overview of recent work and a detailed discussion of five different cut types.
- The evaluation scheme is presented in **chapter 4**. We start with a brief introduction and continue with a detailed discussion of the Normalized Probabilistic *Rand* Index which is used here for assessing the segmentation quality. The chapter ends with a presentation of our evaluation results.
- In **chapter 5** we give a summary of the thesis and our results along with some ideas which could be the object of future work.

## 1.1 Mathematical Notation

We would like to briefly address the mathematical notation used throughout this thesis. For matrices and vectors we always use a letter in bold face and for the components the same letter with non bold face and the corresponding index, e.g. $\boldsymbol{v} = (v_1, \dots, v_n)^{\mathsf{T}}$. We use the symbol "$\triangleq$" in order to state that equality holds per definition. The following table gives an overview of notations which either occur frequently during our work or are not further specified somewhere else as the reader is expected to be familiar with them.

**Numbers and Sets**

| | |
|---|---|
| $\mathbb{R}$ | set of real numbers |
| $\mathbb{R}^+$ | set of positive real numbers including 0 |
| $\mathbb{R}^n$ | space of $n$-dimensional real vectors |
| $\underline{n}$ | the set $\{1, \dots, n\}$ of natural numbers |
| $A^{\mathsf{c}}$ | complement of the subset $A$, $A^{\mathsf{c}} \triangleq A \setminus V$ with $A \subseteq V$ |
| $\|A\|$ | number of elements of $A$ |
| $\|a\|$ | absolute value of $a \in \mathbb{R}$ |

| | |
|---|---|
| $\binom{n}{k}$ | binomial coefficient, $\binom{n}{k} \triangleq \frac{n!}{(n-k)!k!}$ |

**Vectors and Matrices**

| | |
|---|---|
| $\boldsymbol{v}^\mathsf{T}$ | transpose of $\boldsymbol{x}$ |
| $\Vert\boldsymbol{v}\Vert$ | EUCLIDEAN norm of the vector $\boldsymbol{v}$ |
| $\mathbf{1}$ | vector of all ones, $\mathbf{1} \triangleq (1, \ldots, 1)^\mathsf{T}$ |
| $f \star g$ | convolution of two real valued functions $f$ and $g$ |
| $\nabla f(\boldsymbol{x})$ | gradient of $f : \mathbb{R}^n \to \mathbb{R}^n$ at $\boldsymbol{x}$, i.e. $\nabla f(\boldsymbol{x}) = (\frac{\partial f}{x_1}, \ldots, \frac{\partial f}{x_n})^\mathsf{T}$ |
| $\mathsf{diag}(\boldsymbol{v})$ | diagonal matrix with vector $\boldsymbol{v}$ on its main diagonal |

**Graphs**

| | |
|---|---|
| $G$ | always a simple graph, see definition 5 on page 11 |
| $V(G)$ | the vertex set of the graph $G$, see definition 5 on page 11 |
| $E(G)$ | the edge set of the graph $G$, see definition 5 on page 11 |
| $E(A, B)$ | set of edges which have one vertex in $A$ and one vertex in $B$, where $A$ and $B$ are vertex sets of the same graph, see definition 12 on page 29 |
| $\gamma(v)$ | the value of a vertex weight function at the vertex $v$, see definition 10 on page 11 |
| $\gamma(A)$ | vertex weight function on a set of vertices, see page 30 |
| $\omega(e)$ | the value of a edge weight function at the edge $e$, see definition 10 on page 11 |
| $\omega(A, B)$ | edge weight function on a set of edges, see page 30 |

**Miscellaneous**

| | |
|---|---|
| $\boldsymbol{i}$ | image function, see definition 1 on page 5 |
| $\mathcal{R}$ | maps a vertex (edges) of a region adjacency graph to its vertices (union of edges) of the corresponding partition of the plane, see definition 11 on page 13 |

# 2 Image Segmentation

In this chapter we would like to describe the image segmentation problem in more detail and how it is attacked in this thesis.

We start with the definition of the image function along with some related concepts. In the following section we actually discuss the image segmentation problem and after this we come to the graph based segmentation scheme. Finally, we discuss two state of the art segmentation methods which are used here for the initial segmentation step: the exact watershed transform and the mean shift algorithm.

## 2.1 The Image Function

The *image function* is a simple mathematical model for a digital image gained through some kind of image acquisition process:

**Definition 1** (IMAGE FUNCTION)**.** *An* image (function) *is a function*

$$f : \underline{w} \times \underline{h} \to D \quad \textit{with} \quad (x, y) \mapsto f[x, y] \in D$$

*where $w \in \mathbb{N}$ is the* width *and $h \in \mathbb{N}$ is* height *of $f$. In order to indicate the affiliation of the co-domain $D$ to $f$ we also write $D(f)$. An element of $D$ is also called* (image) value *and we use both notations $f[x, y]$ and $f[\boldsymbol{p}]$ to denote the image value at a certain point.*

For a co-domain $D = \underline{256}^3$ the symbol $\boldsymbol{i}$ is used for the image function and all values $\boldsymbol{x} \in D$ are meant to be in the RGB color space. The symbol $\boldsymbol{i}_{\mathsf{Luv}}$ represents an image function with function values converted from $\boldsymbol{i}$ into the CIE LUV color space. It is especially designed to be perceptually uniform with respect of color differences, that is, for two image values $\boldsymbol{x}, \boldsymbol{y}$ the EUCLIDEAN distance $\|\boldsymbol{x} - \boldsymbol{y}\|$ corresponds to perceived difference of a human observer. The conversion can be done in two steps, first transforming an image value $\boldsymbol{x} \in D(\boldsymbol{i})$ into the CIE XYZ space by

$$(X, Y, Z)^{\mathsf{T}} = \boldsymbol{M}\boldsymbol{x}$$

with

$$\boldsymbol{M} \triangleq \begin{bmatrix} 0.41 & 0.36 & 0.18 \\ 0.21 & 0.72 & 0.07 \\ 0.02 & 0.12 & 0.95 \end{bmatrix}$$

and afterwards computing the corresponding image value $(L^*, u^*, v^*)^\mathsf{T}$ in CIE LUV color space by

$$\begin{aligned} L^* &\triangleq \begin{cases} 116\sqrt[3]{Y} - 16 & \text{if} \quad 0.008856 < Y \\ 903.3 \cdot Y & \text{else} \end{cases} \\ u^* &\triangleq 13L^*(u' - u'_n) \\ v^* &\triangleq 13L^*(v' - v'_n) \end{aligned}$$

with

$$\begin{aligned} u' &\triangleq \frac{4X}{X + 15Y + 3Z}, & v' &\triangleq \frac{9Y}{X + 15Y + 3Z} \\ u'_n &\triangleq \frac{4X_n}{X_n + 15Y_n + 3Z_n}, & v'_n &\triangleq \frac{9Y_n}{X_n + 15Y_n + 3Z_n} \end{aligned}$$

[Ford and Roberts 1998].

An image function can be *extended* such that it has image values for the plane $\mathbb{R}^2$ by using B-spline interpolation. More precisely, for an arbitrary position $(x, y) \in \mathbb{R}^2$ the corresponding image value is defined through:

$$\boldsymbol{i}_n(x, y) \triangleq \sum_{\boldsymbol{p} \in \underline{w} \times \underline{h}} c(\boldsymbol{p}) \beta_n(i - x) \beta_n(j - y)$$

using the recursively defined B-spline basis function of *order* $n$:

$$\beta_n \triangleq \frac{1}{n} \left[ \beta_{n-1}\left(\frac{n+1}{2} + x\right)\left(x + \frac{1}{2}\right) + \beta_{n-1}\left(\frac{n+1}{2} - x\right)\left(x - \frac{1}{2}\right) \right] \tag{2.1}$$

$$\beta_0 \triangleq \begin{cases} 0 & \text{if} \quad x < 0 \\ 1 & \text{else} \end{cases} \tag{2.2}$$

The coefficients $c(\boldsymbol{p})$ can be effectively computed by recursive linear filters (more details can be found in [Unser et al. 1993*a*,*b*]). In figure 2.1 examples for spline interpolation with different orders can be seen. For $n = 0$ we also drop the index $n$. Note that we write $\boldsymbol{i}[\cdot]$ for the regular image function and $\boldsymbol{i}(\cdot)$ for the extended version.

**Implementation Notes** The Vigra image library by Köthe [2000] contains functions for several color conversions and data types which implement B-spline interpolation for images as well.

## 2.2 The Segmentation Problem

Image segmentation is generally seen as an important part of image analysis and computer vision. Every standard computer vision textbook contains at least on chapter about image segmentation,

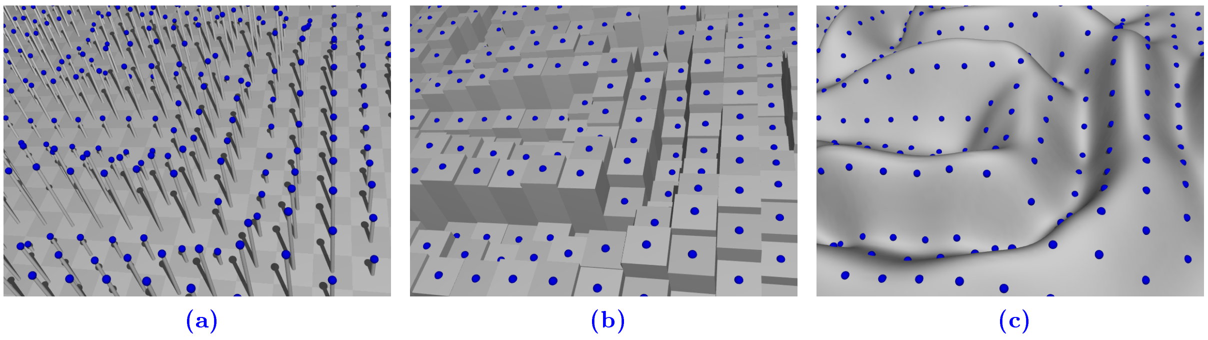


**Figure 2.1:** *Examples for spline interpolation of a scalar valued image function, all images are taken from [*MEINE *2008];(a): height of a ball indicates value of the image function at that position;(b):* $n = 0$ *yields nearest common neighbor interpolation;(c):*$n = 5$ *yields a very smooth interpolation*

e.g. see [BALLARD and BROWN 1982; FORSYTH and PONCE 2002; GONZALEZ and WOODS 2006; SHAPIRO and STOCKMAN 2001; SONKA et al. 2007] for some references. It has been examined and studied by many researchers for almost the last four decades. Following ZHANG [2006] there were almost one thousand publications on different segmentation algorithms up to 1994 with constantly high number of publications over the last decade and with even increasing number of publications over the last few years.

Although there has been such a lot of research on this, it remains unsolved for the general case. The reason for this is that image segmentation is indeed no well-posed problem but rather an intuitive and somewhat vague idea of a certain step in the process of image analysis. This situation is closely related to the lack of a general methodology for extracting semantic information from digital images.

The term "image segmentation" is somewhat ambiguous as it actually has two common meanings, the first is referred here as *low-level segmentation* and the second as *high-level segmentation*. In the following we discuss what is understood under this terms and how they relate to each other before our approach to the image segmentation problem is presented.

We use a small working example for a more intuitive and descriptive discussion. On figure 2.2a an image from the BERKELEY Image Database (see section 4.1) is given. Considering the picture, it is easy for a human observer to tell that the prominent objects shown are two flowers. Effortlessly, he could actually assign each region in an image to the object it belongs to. A possible outcome of such a manual assignment is shown in figure 2.2b. The automation of this task of "distinguishing meaning full parts of an image" is the concern of image segmentation:

**Definition 2** (HIGH-LEVEL IMAGE SEGMENTATION)**.** Image Segmentation *is the division of an image into objects and parts thereof.*

It should be said that this definition does not lead to a well posed problem as it is not always clear what actually constitutes an object or its parts. For our example, in figure 2.2c another manual segmentation is shown which also identifies leafs of the background but does not contain

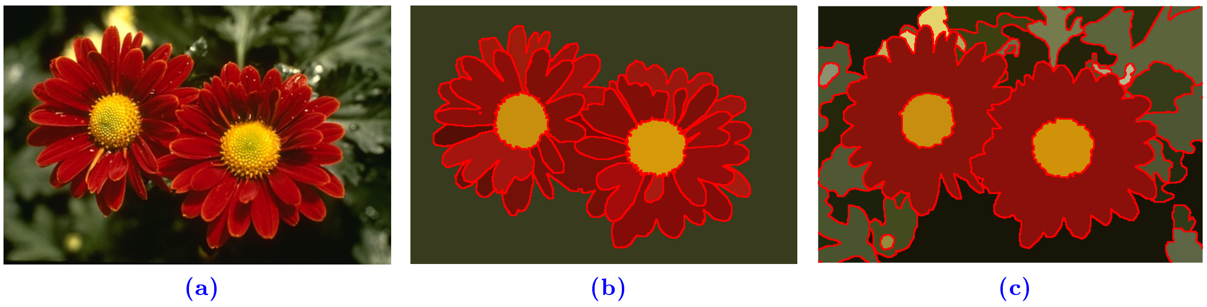


**Figure 2.2:** *(a): an image from the* BERKELEY *Image Database;(b): a manual segmentation; (c): another manual segmentation*

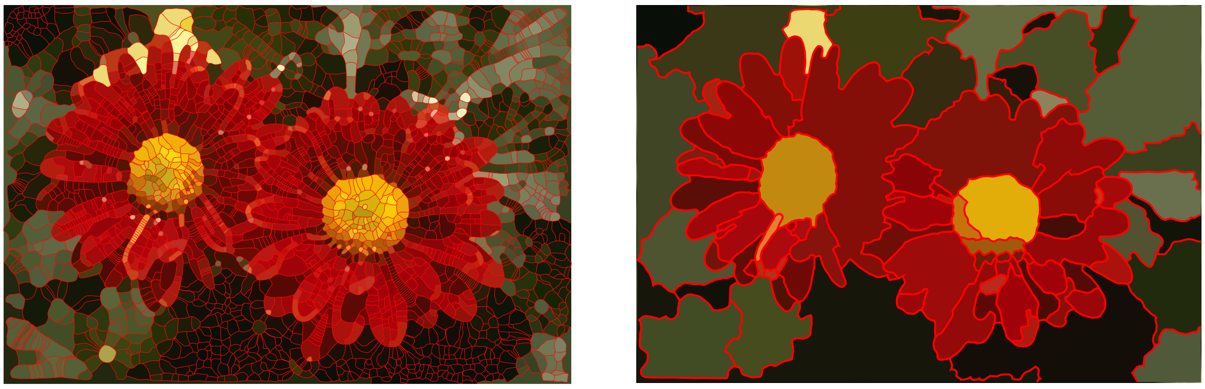
segments for the leafs of the flowers as in figure 2.2b.

For a vision system capable of "detecting flowers" it would be required to actually have knowledge about them, e.g. the parts they are composed of, shape and texture of those parts, the environment they occur and so on. Such high-level knowledge is essential for the solution of complex vision problems and, at the time being, must be appropriately modeled an incorporated for each specific segmentation task.

Nevertheless, it is still reasonable to put effort into the development of general purpose image segmentation methods which do not utilize any domain specific high-level knowledge. So called low-level algorithms only exploit statistical regularities in rather simple features derived from the image data. Common examples for those features are brightness, color, and texture. As our world is visually not chaotic, it is reasonable to assume that regions being uniform with respect to simple image features have a common cause, e.g. result from the same object or the same part of an object. So, it is commonly believed in the computer vision community that image analysis tasks can benefit in general from using such principles of organization irrespective of a concrete problem domain. The following definition of low-level segmentation is adapted from BALLARD and BROWN [1982] from which we think that it consolidates the relevant aspects of what is generally understood under this term:

**Definition 3** (LOW-LEVEL IMAGE SEGMENTATION). *(*Image*)* Segmentation *is the grouping of image elements of a certain image into regions (segments) which are homogeneous with respect to one or more characteristics (or features) which can be assigned to each region.*

Going back to our flower example, figure 2.3 shows two results from low-level segmentation algorithms. Figure 2.3a shows a segmentation performed by a recent algorithm from MEINE and KOETHE [2005] (more details in section 2.4). As one can see, the image is divided into many small segments enclosing areas which are very homogeneous with respect to color. On figure 2.3b this *over-segmentation* has been further improved using another low-level segmentation algorithm (more details follow in section 2.3). As one can see, they perform already quite good together in the sense of giving a first guess how a reasonable segmentation of the image could look like. In summary, we would like to emphasize two aspects how low-level segmentation algorithms can



(a) (b)

**Figure 2.3:** *Two results of low-level image segmentation algorithms (a): oversegmentation with exact watershed transform (b): the foreground cut of section 3.6 further improves this segmentation.*

be useful for solving the high-level segmentation problem:

- A low-level segmentation algorithm can provide a first change of representation, that is, changing from image pixels as atomic image tokens to more complex ones. This way, following process steps can benefit from a first aggregation of information and the reduced number of elements to process.
- It is possible to get already quite close to the desired segmentation result even if no high-level knowledge is used.

In this work we put our attention on the last point, that is, we would like to investigate how good a certain class of low-level segmentation algorithms performs on the high-level segmentation problem; more details are presented in section 2.3. However, it should be clear that due to the ill-posed nature of the high-level segmentation problem statements such as “algorithm A is better then algorithm B” are not easy to make. We ignore this problem for the moment and come back to it in chapter 4 where the evaluation framework used is discussed.

### 2.2.1 Partition of the Plane

In the last section we use phrases like “division of the image” or “grouping of image elements” for describing the segmentation task. We would like to be more specific what the exact meaning of such formulations is actually supposed to be within this thesis.

A classical model for representing segmentation results is the *label image* which is an image function mapping every pixel position to a label indicating the segment the pixel belongs to. For the representation of segmentations with sub-pixel accurate segment boundaries a more sophisticated model is needed. A *partition of the plane* $\mathbb{R}^2$ for the extended image function is suitable for this. The following definition is taken from [Meine 2008, chapter 3, p. 29][1]:

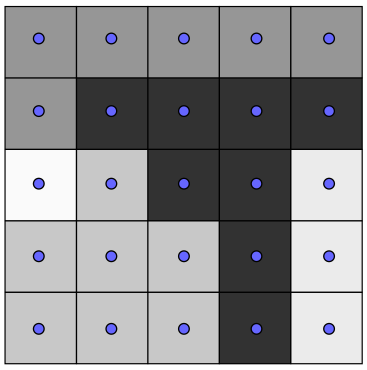
[1]Actually, an equivalent definition was already given in Köthe [2002] and proposed as a model for image

**Definition 4** (Partition of the plane). *A* partition of the plane *is a tuple* $P \triangleq (V, E, F)$ *where*

- $V \subset \mathbb{R}^2$ *is a finite set of non-neighbored points called* vertices,
- $E \subset 2^{\mathbb{R}^2}$ *is a finite set of disconnected, pairwise disjoint, non-self-intersecting, open curves called* edges *such that each* $e \in E$ *is homeomorphic to the open interval* $(0, 1)$ *and has its endpoints (limits) in* $V$*, and*
- $F \subset 2^{\mathbb{R}^2}$ *is the set of connected components of* $\mathbb{R}^2 \setminus (V \cup \bigcup_{e \in E} e)$ *called* faces.

The edges are here always polygonal arcs. A partition of the plane therefore divides $\mathbb{R}^2$ into several regions called faces. Note that there has to be at least one infinite face in each such partition. In the following, each partition of the plane has exactly one such face "enclosing" the image, cf. figure 2.4b.

Further note that we can always derive a partition of the plane from an label-image if we insert polygonal arcs into the plane where pixels having different labels in the nearest neighbor interpolation of the label image abut. An example for this can be seen in figure 2.4. The faces of the partition of the plane then correspond to the 4-connected regions of the label image.


(a)

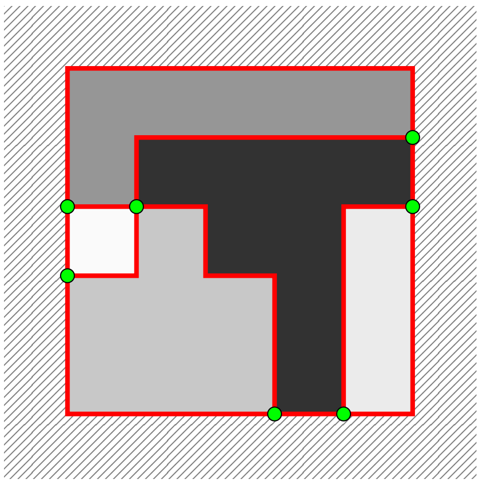

(b)

**Figure 2.4:** *(a): nearest neighbor interpolation of a label image; blue points indicate pixel positions of the original label-image (b): partition of the plane P for the label image; red lines are edges of P; green points indicated vertices of P; the hatched pattern indicates the infinite face*

## 2.3 Graph Based Segmentation

In this section we actually describe the segmentation approach used within this work. For this, some basic concepts from graph theory are needed. Therefore, we start with a series of definitions from this field.

segmentation

### 2.3.1 Basic Terminology

**Definition 5** (SIMPLE GRAPH). *A* simple graph *is a pair of sets written $G := (V(G), E(G))$ where $V(G)$ is a finite set of* vertices, *also called* nodes, *and $E(G)$ is a set of* edges *of the form $\{u, v\}$ with $u, v \in V(G)$ and $u \neq v$. For an edge $e = \{u, v\} \in E(G)$ the vertex $u$ is said to be* incident *to $v$ as well as $e$ is said to be* incident *to $u$. The sets $V(G)$ and $E(G)$ are also written $V$ and $E$ if the graph they belong to is clear from the context. If the set of vertices is given in the form $V(G) = \{v_1, \ldots, v_n\}$ an edge $\{v_i, v_j\}$ of $G$ is also written $e_{ij}$ or $e_{ji}$.*

**Definition 6** (COMPLETE GRAPH). *A graph $G = (V, E)$ is called* complete *iff $E = V \times V$ holds.*

**Definition 7** (SUBGRAPH). *For a graph $G$ a graph $T$ is a* subgraph *of $G$ iff $V(T) \subseteq V(G)$ and therefore $E(T) \subseteq E(G)$. The subgraph $T$ is also said to be* induced *by the set $V(G)$.*

**Definition 8** (CONNECTIVITY OF A GRAPH). *A graph $G$ is* connected *iff for every pair of vertices $u, u' \in V(G)$ there is a sequence $v_1, \ldots, v_n$ of vertices of $G$ such that $u = v_1$, $u = v_n$, and $\{v_i, v_{i+1}\} \in E(G)$ for all $0 \leq i \leq n-1$.*

**Definition 9** (CONNECTED COMPONENT). *A subgraph $T$ of a simple graph $G$ is a connected component of $G$ iff $T$ is connected and adding an arbitrary vertex $v \in V(G)$ with $v \notin V(T)$ destroys the connectivity of $T$.*

**Definition 10** (WEIGHTED GRAPH). *A graph $G$ is a* weighted graph *with* edge weight function *$\omega$ if $\omega$ is a function with*

$$\omega : E(G) \to \mathbb{R}^+.$$

*Similarly, $G$ is a* weighted graph *with* vertex weight function *$\gamma$ if $\gamma$ is a function with*

$$\gamma : V(G) \to \mathbb{R}^+.$$

*All subgraphs of $G$ are then meant to be weighted graphs in the same sense.*

Note that both the edge and vertex weights are always positive real numbers.

### 2.3.2 Energy Functions and Graph Cuts

Over the last few years so called *energy minimization* approaches to the segmentation problem became popular. The idea is to define a function assigning an *energy value* to every possible segmentation of an image. More precisely, if the segmentation problem is defined as assigning a label $\ell \in \underline{k}$ to each image element of a given image with size $w \times h$, we can represent each possible segmentation as a vector $\boldsymbol{x}^{w \cdot h}$ with $x_i \in \underline{k}$. The quality of a segmentation $\boldsymbol{x}$ is then assessed through an energy function

$$E : \underline{k}^{w \cdot h} \to \mathbb{R}.^2$$

A good segmentation then is mapped to a small energy value and the process of segmentation is modeled as a search for an optimizing argument for the energy function $E$. The advantage of this method is that the resulting segmentation is optimal in a *global* sense. So, it is possible to form segment boundaries where local techniques normally would fail.

For the success of such an approach the energy function must be defined properly, i.e. small energies correspond to good segmentations, and there has to be an effective algorithm for optimizing it. Unfortunately, these two aspects are often competing such that a sophisticated designed function often ends in a computationally intractable optimization problem. Therefore, many energy minimization techniques used in vision are formulated within a graph based framework where the image elements are represented through vertices with weighted edges connecting neighboring elements. The advantage of these methods is that for weighted graphs certain binary optimization problems can be effectively solved with “graph cuts”. The energy value then corresponds to the similarity of two resulting components of the bi-partition and is expressed as function of the weights of the edges connecting them. As image segmentation is a non binary problem, if more than two segments are desired, the graph cut methods have to be extended somehow. It has been proposed to use graph cuts as hierarchical clustering tool by recursively splitting a weighted graph and the resulting subgraphs until a stopping criterion is met (more details follow in section 2.3.4). The usefulness of this *recursive bi-partitioning* has been reported many times in the literature (see section 3.2.1.1 for references).

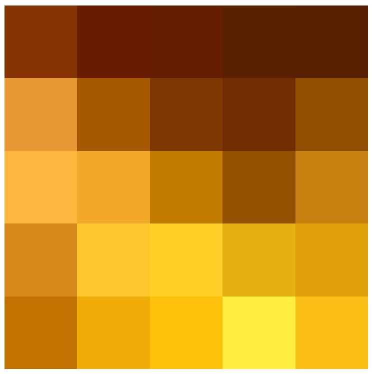
In this work we use graph cuts together with recursive bi-partitioning as the key component for our segmentation scheme. However, the selection of a specific graph cut type is difficult as they have been many suggestions in the literature. Graph cuts especially useful for image segmentation are those which can be seen as normalized versions of the *minimum cut*. The main ambition of this thesis is to investigate how the various exponents of these *normalized cuts* perform in our segmentation scheme. A detailed description of the various graph cuts used follows in chapter 3.

### 2.3.3 The Super-Pixel Approach

As we interested in using graph cuts for image segmentation the image data has to be converted somehow into a weighted graph. Many graph based methods in vision use a graph being directly derived from the pixel grid, see figure 2.5. We call graphs of this kind *pixel grid graphs*; they encode the common neighborhood relations of the image elements and edge weights often represent some kind of affinity between them. The problem of using a pixel grid is twofolds:

1 The number of vertices in such pixel graphs is very high as it equals the number of pixels in the image. For all images used here this number is greater than 120.000. While we above claimed that graph cuts can be effectively computed, the computation time for problem instances of that size is still too high for many applications.

[2]Note that only pixel accurate segmentations can be represented with this method if the components of $\boldsymbol{x}$ are associated with pixel positions.


(a)

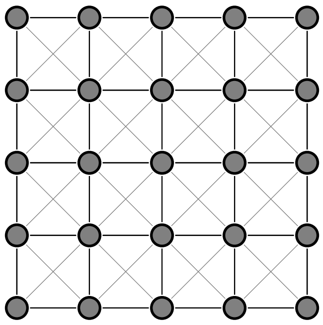

(b)

**Figure 2.5:** *A* $5 \times 5$ *image (a) and the corresponding pixel grid graph (b) representing the 4- or 8-neighborhood.*

[2] The edge weights in this graphs can only be very simple functions of the pixel intensities or colors. This makes the usage of advanced similarity measures derived from color histograms, texture, geometry, and such impossible.

The main idea of this work is therefore to employ a preceding segmentation step and then to further improve this initial segmentation with aid of graph cuts. Hence, the graph used is not derived from the pixels of an image and their topology but from the segments of an inital segmentation step. This results in a *region adjacency graph* which can be derived from the partition of the plane (see section 2.2.1):

**Definition 11** (REGION ADJACENCY GRAPH)**.** *A* region adjacency graph (RAG) *of a partition of the plane* $P \triangleq (V_P, E_P, F_P)$ *is a graph* $G \triangleq (V, E)$ *with:*

- *The set of vertices* $V$ *has exactly one vertex for every face in* $F_P$ *except for the infinite face.*
- *The set of edges* $E$ *has exactly one edge for every pair of vertices of* $G$ *whose faces share at least one edge in* $P$.

*The function*

$$\mathcal{R} : V \cup E \to \mathbb{R} \cup 2^{\mathbb{R}^2}$$

*maps every vertex of* $G$ *to its face in* $P$ *and every edge* $\{u, v\}$ *of* $G$ *to* $\bigcup_{e \in N(u,v)} e$ *with* $N(u, v) \triangleq \{e \in E_P \,|\, e \text{ is a common edge of } \mathcal{R}(u) \text{ and } \mathcal{R}(v)\}$.

This is in the spirit of the *super-pixel* approach of REN and MALIK [2003]. They propose to use an oversegmentation in order to get a first change of representation from pixels as the basic image elements to super-pixels. The use of super-pixels greatly reduces the number of elements to process but maintains most of structure in the image data and thus appropriately addresses drawback [1] of pixel grid graphs. Addressing drawback [2], as super-pixels are segments we can derive more expressive edge weights exploiting the statistics of the enclosed pixels. As an additional advantage, the final segmentation can benefit from special characteristics of the initial segmentation step. For example, the exact watershed transform produces super-pixels with sub-pixel accurate boundaries.

**Implementation Notes** Recently, Meine and Köthe [2006] introduced the GeoMap Framework which is an universal approach for representing segmentation results. Here, we use a partition of the plane together with a region adjacency graph in order to model the geometry and topology of them. The GeoMap is far more expressive then this: For example, it can also appropriately handle non closed segment boundaries. We have decided not to use the GeoMap in order to keep the mathematical model as simple as possible. However, we do use the GeoMap datatype for the implementation of most of the segmentation related algorithms. It makes both the computation of faces and their boundary statistics very easy. Actually, we also use the GeoMap datatype for deriving region adjacency graphs. A detailed introduction to the GeoMap framework can also be found in [Meine 2008].

### 2.3.4 Basic Segmentation Scheme

In the last sections we introduced the basic ideas for the segmentation scheme used in this work. In summary, we perform an initial segmentation of the image which contains as much potential segment boundaries as possible. This results in a partition of the plane from which a region adjacency graph can be derived. For the region adjacency graph, edge weights are computed encoding the pairwise similarity of neighboring faces. The RAG is recursively split using normalized cuts until a stopping criterion is met. In more algorithmic form, the basic segmentation scheme used throughout in this work reads:

**Algorithm 2.3.1:** basic segmentation scheme

```
1 create an initial segmentation represented by the partition of the plane $P$
2 create a region adjacency graph $G$ from $P$
3 compute the edge weights of $G$ by inspection of the underlying image data of faces of $P$
4 by minimization of an energy function create a bi-partition of the vertex set of $G$ yielding
  two subgraphs $T_1$ and $T_2$ and an energy value $\lambda$
5 recursively repeat 4 on $T_1$ and $T_2$ until $\lambda \geq$ stop where stop is a beforehand selected stop
  parameter
6 derive a new partition of the plane from the subgraphs which do not meet the stop criterion
```

We give some final notes on these steps which also give an overview of how the rest of this thesis is organized:

1 We use the exact watershed transform of Meine and Koethe [2005] for oversegmentation

(see section 2.4). It is especially suited for this as it maintains much of the potential segment boundaries and has the additional advantage of producing sub-pixel accurate segment boundaries. In order to get an idea of how exchanging this step can influence the segmentation results we also use the mean shift method of Comaniciu and Meer [2002]. It is only pixel accurate and produces very inhomogeneous segments with respect to size and shape, see section 2.5.

2 A nice property of the region adjacency graphs encountered here is that they are *sparse*, i.e. the number of edges in such a graph is much smaller than the number of possible edges which is $n^2$ with $n$ being the number of vertexes. For example, the region adjacency graphs derived from the watershed-transform of the Berkeley Image Database have only $\approx 0.0025\%$ of the possible edges. The sparsity can then be exploited for algorithms presented in chapter 3.

3 For the edge weights we investigate three different similarity measures which are introduced in section 3.3.

4 We investigate five different graph cut types which are discussed in the next chapter.

5 The parameter `stop` has to be selected beforehand. As the energy value $\lambda$ is interpreted as the quality of the partition of $G$ into $T_1$ and $T_2$, higher values for `stop` permit lower quality partitions. This recursive bi-partitioning together with a stop parameter has been used for several normalized cuts, see section 3.2.1.1.

## 2.4 Watershed Transformation

The watershed transform has its origins in the field of mathematical morphology and was first proposed by Beucher and Lantuejoul [1979]. It is a common tool for region based image segmentation with an intuitive and vivid motivation.

The basic idea of the watershed transform is to use a topographic interpretation for a scalar valued function $f$. Within this interpretation $f$ maps certain positions in a terrain to its level of evaluation. It is therefore common to use geographic terms for a detailed discussion of the watershed transform's concepts.

Imagine rain flooding the landscape described by $f$. Rain drops falling onto the terrain follow a certain path along the surface of the terrain under the influence of gravity. This flowline starts where the corresponding water drop hits the landscape and ends in the lowest point of a nearby valley. Under this assumption, each valley can be seen as a catchment basin corresponding to a local minimum and a surrounding influence zone for which every flowline ends in this minimum. The division lines between catchment basins are called *watersheds* and form a tessellation of the terrain. For image segmentation, we can use a *boundary indicator (function)* that expresses strong evidence for a boundary at a certain image position in terms of a high scalar value.

In an alternative view, flooding is done by piercing holes into the surfaces at local minima. Immersion of the landscape into a lake would cause the catchment basins to gradually fill up. *Watersheds* now correspond to dams build during the flooding process to prevent water of neighboring basins to converge.

In order to be usable for image segmentation, an efficient algorithm for detection of watersheds or catchment basins is needed. Depending on a certain algorithm and its underlying mathematical model of watersheds, various segmentation results are possible. Indeed, there are many algorithms available, see e.g. [Roerdink and Meijster 2000] for an overview. Here, we only use a recent algorithm developed by Meine and Koethe [2005].

### 2.4.1 The Exact Watershed Transformation

The exact watershed transformation is capable of identifying watersheds with sub-pixel accuracy. This is possible if one switches from the discrete domain, i.e. from the pixel grid, to the *continuous* domain by using a spline interpolated version of the boundary indicator. This approach has several advantages [Meine 2008, Chapter 4, p. 88]:

- The geometry of the watersheds is much better compared to their discrete counterparts.
- Discrete Watersheds often produce wrong or miss boundaries as the minimum and watershed detection is limited by the precision of the pixel grid.

Steger [1999] was the first one who used the idea of exact watersheds in connection with Maxwell's definition of watersheds (see below) to develop a computational tractable algorithm for digital terrain models. Meine and Koethe [2005] further improved the method of Steger and made it applicable for image segmentation. A more exhaustive discussion of the exact watersheds method can be found in [Meine 2008, Chapter 4, p. 88 et seqq] on which the following text is mainly based.

If some restrictions on $f$ are imposed, watersheds can be build from critical points of $f$ using Maxwell's definition which says that watersheds are flowlines connecting saddles with maxima. This definition does not exactly match the informal description of watersheds given above, but can be used as a starting point.

Let $f$ be an at least twice differential real valued function with all critical points isolated, i.e. $f$ is a *Morse function*. This restrictions ensure that all critical points are non-degenerate, i.e. there are no plateaus and monkey saddles. Moreover, for every non critical point $\boldsymbol{p} \in D(f)$ there is a unique *flowline*

$$\boldsymbol{x} : [0, \infty) \to D(f)$$

which obeys the first order differential equation

$$\frac{\partial \boldsymbol{x}}{\partial t} = -\nabla f(\boldsymbol{x}(t)) \tag{2.3}$$

with inital condition $\boldsymbol{x}(t) = \boldsymbol{p}$. Each flowline converges to one critical point $\boldsymbol{c}$, i.e.

$$\lim_{t\to\infty} \boldsymbol{x}(t) = \boldsymbol{c} \text{ and } \nabla f(\boldsymbol{c}) = 0,$$

which is either a minimum or a saddle of $f$. Since every catchment basin consists of a minimum $\boldsymbol{m}$ and all points whose flowline ends in $\boldsymbol{m}$, a watershed must consist of flowlines starting near a maximum and ending in a saddle. For each saddle there are exactly two of those flowlines which "enter" it in two opposite directions. So, we determine all interesting flowlines by taking an infinitesimal small step in one of this directions and follow equation (2.3) upwards until a maximum is reached, see figure 2.6.

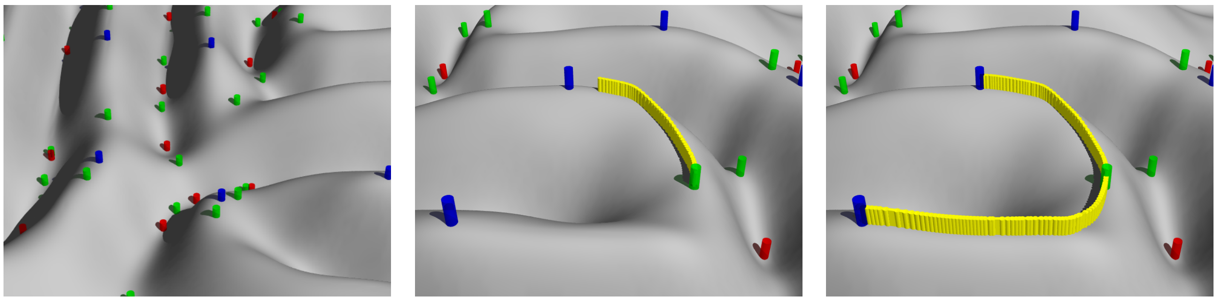
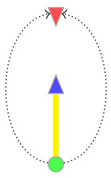


**Figure 2.6:** *A terrain with critical points (maxima (blue), minima(red), and saddles(green)) and flowline tracing (taken from [*MEINE *2008])*



However, some of these flowlines may not correspond to watersheds as depicted in the figure on the right: The configuration of critical points shown can be interpreted as an "isolated mountain". The flowline shown connects a maximum (triangle pointing upward) with a saddle (circle), but drops falling on opposite sites of this flowline would flow to the same minimum (triangle pointing downward) and thus the flowline cannot be a watershed and must be eliminated somehow afterwards (this example is taken from [STEGER 1999]).

Finally, all valid flowlines, and maxima can be connected into a graph in which each pair of flowlines originating in the same saddle corresponds to an edge and the associated maxima are the vertices. The catchment basins enclosed by the flowlines then form a division of the image plane with sub-pixel accurate boundaries.

Putting all together, there are three steps to perform for segmentation:

- Find all critical points of $f$,
- for each saddle point compute both flowlines that end in a maximum, and
- determine all regions enclosed by flowlines and the associated boundaries.

The well known method by NEWTON and RAPHSON is suitable for both effectively and precisely detecting critical points if applied to each pixel with possibly several starting points (see [MEINE 2008, Chapter 4, p. 90 et seqq] for more details).

For the computation of flowlines all critical points must be classified to be either a minimum, a maximum, or a saddle. Since $f$ is a MORSE function, this can be effectively done by examining the eigenvalues $e_1, e_2$ of the HESSIAN $\boldsymbol{H}$ of the boundary indicator $f$ at a critical point $\boldsymbol{p}$: For both $e_1, e_2$ being positive $\boldsymbol{p}$ is maximum, for both $e_1, e_2$ being negative $\boldsymbol{p}$ is a minimum, and otherwise $\boldsymbol{p}$ is a saddle. Then, for the integration of equation (2.3) a Runga-Kutta method with adaptive step size can be used. This yields a polygonal arc $(\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n)$ where

- $\boldsymbol{p}_1$ is a saddle,
- $\boldsymbol{p}_2 = \boldsymbol{p}_1 \pm \tau \boldsymbol{e}_1$ is the starting point for the Runga-Kutta method with $\boldsymbol{e}_1$ being the unit length eigenvector of the positive eigenvalue of $\boldsymbol{H}$ at $\boldsymbol{p}_1$ and $\tau$ being the inital step size (here we use 0.1),
- $\boldsymbol{p}_3, \ldots, \boldsymbol{p}_{n-1}$ are points detected by the Runga-Kutta procedure, and
- $\boldsymbol{p}_n$ is the closest maximum with $|\boldsymbol{p}_{n-1} - \boldsymbol{p}_n| < \tau$.

For the final result “artificial” vertices and flowlines are added: For a flowline which runs near the boundary of the image (measured with some kind of threshold) a new vertex is added on the boundary and the flowline is connected to it. Moreover, all boundary vertices in turn are connected with artificial flowlines which run on the boundary. This results in a partition of the plane in the sense of definition 4.

#### 2.4.1.1 Boundary Indicator

For the application of the watershed algorithm to images a suitable boundary indicator function has to be chosen. Here, we use the magnitude of the common Gaussian *gradient* extended for an image function $\boldsymbol{i}$ with $n$ color channels:

$$\mathsf{ggm}_{\boldsymbol{i},\sigma}(\boldsymbol{p}) \triangleq \sqrt{\|(\nabla_x G_\sigma \star \boldsymbol{i})(\boldsymbol{p})\|^2 + \|(\nabla_y G_\sigma \star \boldsymbol{i})(\boldsymbol{p})\|^2}$$

with

$$G_\sigma(\boldsymbol{p}) \triangleq \frac{1}{2\sigma^2} \exp\left(\frac{\|\boldsymbol{p}\|^2}{2\sigma^2}\right)$$

and

$$(\nabla G_\sigma \star \boldsymbol{i})(\boldsymbol{p}) \triangleq ((\nabla G_\sigma \star i_1)(\boldsymbol{p}), \ldots, (\nabla G_\sigma \star i_n)(\boldsymbol{p}))^\mathsf{T}$$

An example of the Gaussian gradient magnitude for two images of the Berkeley Image Database can be seen in figure 2.7.

#### 2.4.1.2 Discussion

An example of a watershed segmentation described so far is depicted in figure 2.8. Note that the watershed transform produces sub-pixel accurate boundaries and possesses strong over-

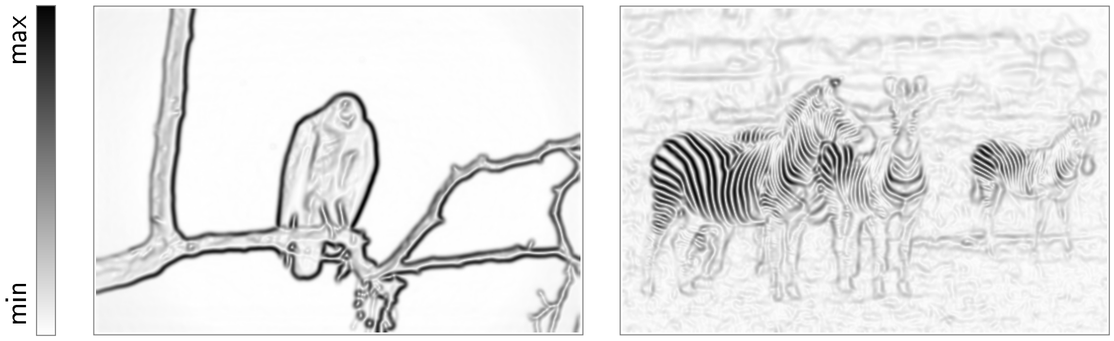


**Figure 2.7:** *The values of* $\mathsf{ggm}_{i,\sigma}$ *for two images of the* Berkeley *Image Database with* $\sigma = 2$

segmentation.

Another feature of watersheds can be seen in figure 2.8e: All flowlines shown converge tangentially to the same maximum (yellow). While this is a typical property of watersheds and not problematic in theory, there are some serious implications: In practice, flowlines can run in parallel into a maximum because of precision issues of the numerical methods used. Recall, we are interested in the regions enclosed by the flowlines, their neighbourhood relations, and similarity measures which can be derived from the enclosed regions and their boundaries. The tangential convergence property of flowlines can impact on these quite considerably:

1 In order to actually derive the regions enclosed by the flowlines and in order to derive the adjacency relations of these regions it is necessary to sort all flowlines ending in a common point with increasing arc length (with an arbitrary starting edge). As it can be seen in figure 2.8e this is a nontrivial task .

2 While the adjacency structure of the faces is consistent, it can have undesired properties as it can be seen by visual inspection of figure 2.8d: Region $r_1$ is both adjacent to $r_2$ and $r_3$, but $r_2$ is not adjacent to $r_3$ due to the parallel part of two flowlines $f_1$ and $f_2$.

3 Most of flowline $f_1$ in figure 2.8d does not belong to the common boundary of region $r_1$ and $r_2$ from a geometrical point of view. As a consequence, similarity measures based on the common boundary of two regions can become completely unreliable.

In order to overcome 1 Meine proposed a sophisticated method for tracing a bundle of parallel running flowlines until they diverge from each other and successively determining the order in which they enter a maximum (see [Meine 2008, p. 94 et seqq] for details). For 2 and 3 one can merge parts of flowlines running in parallel and inserted new vertices at the points where they run apart (again, see [Meine 2008] for details).

Another problematic feature of the watershed segmentation is the occurrence of so called *streamers* between “real” edges running in parallel: They consist of two flowlines each starting from a saddle in the valley between two edges, running uphill on the ridge of the surrounding

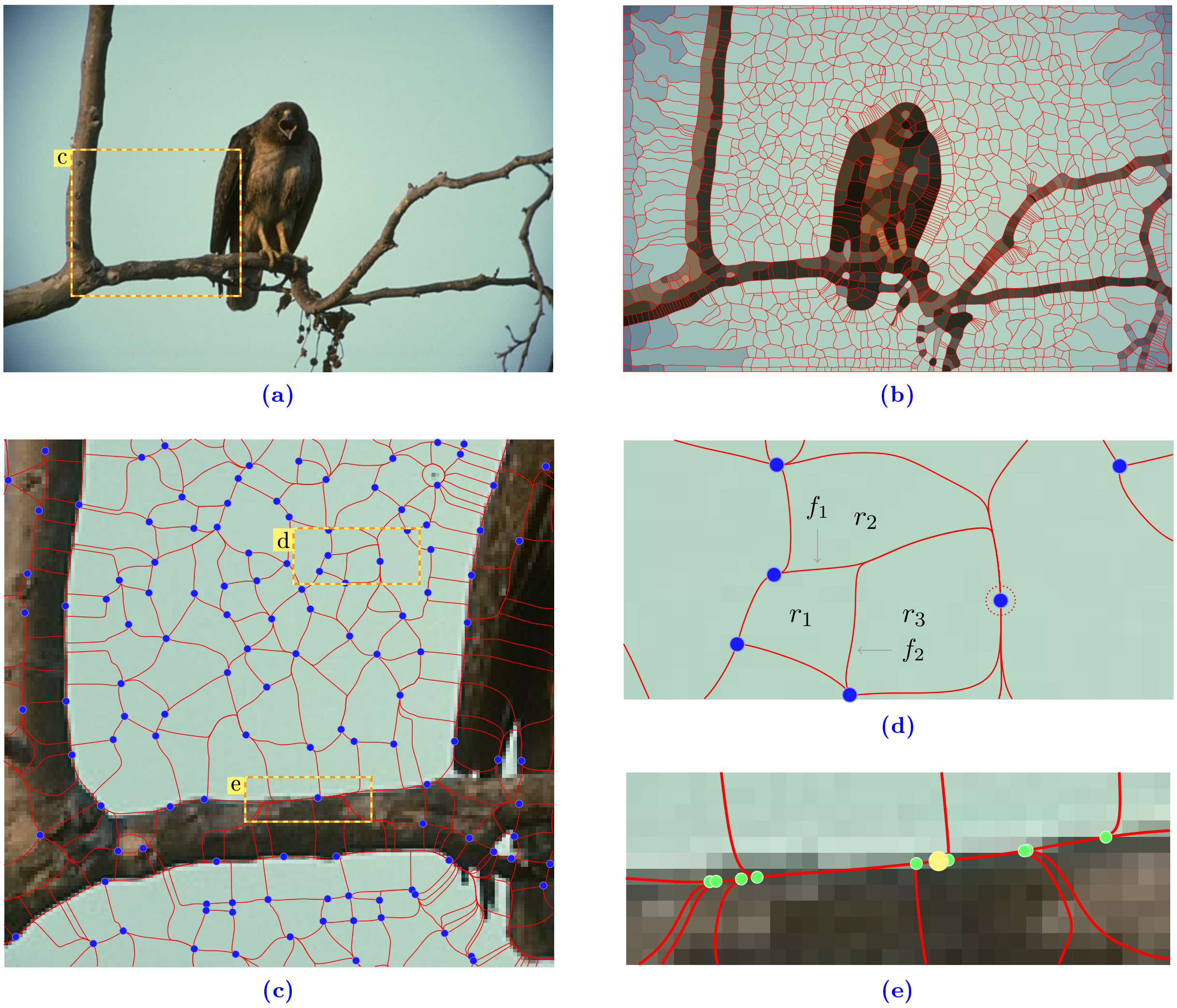


**Figure 2.8:** *(a): an image from the* Berkeley *Image Database; (b): watershed segmentation with gaussian gradient magnitude as boundary indicator ($\sigma = 3.0$); (c): enlarged detail of (a) with maxima as graph vertices (d): enlarged detail of (c) which shows a problematic flowline configurations ($f_1$ and $f_2$ which end in the marked maximum) (e): enlarged detail of (c) which shows a maximum (yellow) with flowlines converging tangentially to it; green points mark positions where the flowlines are break up and new artificial maxima are inserted*

edge, and finally converging to a maximum. Examples of those streamers can be seen in figure 2.9b. Note that they indeed do not correspond to low contrast edges and thus unnecessarily oversegment the image.

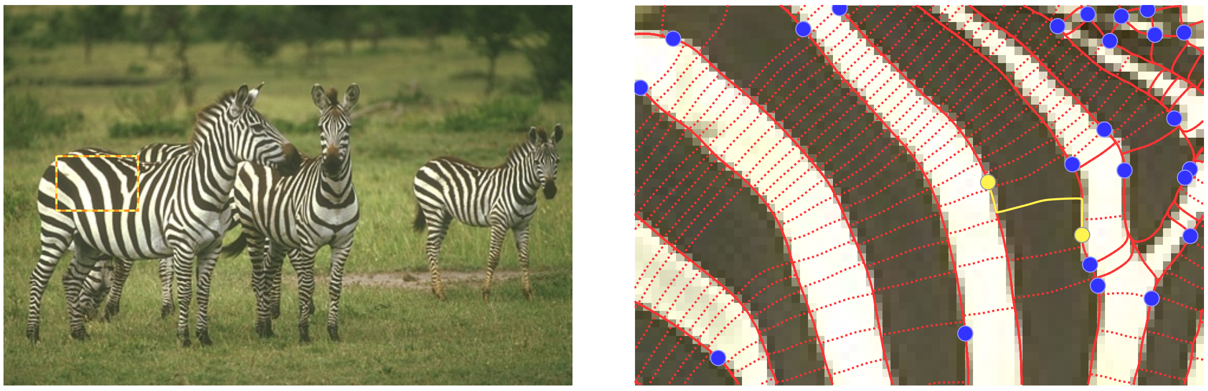

(a) (b)

**Figure 2.9:** *(a): an image from the* BERKELEY *Image Database;*
*(b): part of the corresponding watershed segmentation with gaussian gradient magnitude as boundary indicator ($\sigma = 2.0$). Red line indicate flowlines, dotted edges are streamers which can successfully be removed using the structure tensor ($\sigma' = 1.5\sigma$)*

MEINE [2008] discusses several possibilities to overcome these artifacts. Among the most effective methods is the usage of the *structure tensor* which allows filtering of responsible saddle points even before the flowlines are traced [3]. This method exploits the fact that the streamer's flowlines first run orthogonally towards the nearby edge and that the gradient around the corresponding saddle is thus dominated by the real edges. More formally, let $\boldsymbol{p}$ be the location of a saddle in the boundary indicator $f$ and $\boldsymbol{g} \triangleq \nabla f$ be the gradient at that position. Then the gradient orthogonal to the direction of the flowlines can be computed with

$$\delta_{\mathsf{sog}} \triangleq \sqrt{\boldsymbol{n}^{\mathsf{T}} \cdot \boldsymbol{S}_{\sigma'} \cdot \boldsymbol{n}} \tag{2.4}$$

where $\boldsymbol{S}_{\sigma'} \triangleq (\boldsymbol{g}\boldsymbol{g}^{\mathsf{T}}) \star G_{\sigma'}$ is the *structure tensor* at $\boldsymbol{p}$ and $\boldsymbol{n}$ is an unit length vector perpendicular to the vector pointing in the direction of the flowlines which can be determined from the HESSIAN at point $\boldsymbol{p}$. The parameter $\sigma'$ scales the GAUSSIAN in the structure tensor and thus determines the "influence zone" of it. As the boundary indicator $f$ is computed from the original image by convolution with an GAUSSIAN kernel with scale $\sigma$ (see section 2.4.1.1) $\sigma'$ should be slightly larger than $\sigma$ [MEINE 2008, p. 105 et seqq]. It is even better to discard the strength of the gradient in that direction and solely stick to the directional information instead. This can be done by normalizing $\delta_{\mathsf{sog}}$ by the first eigenvalue of $\boldsymbol{S}_{\sigma'}$ (the index $\sigma'$ is dropped in the following

[3] Actually, using the structure tensor for this was originally suggested by ULLRICH KÖTHE. We cite the text of MEINE [2008] in this context here as for the time of writing only this text was available to the author. Additionally, the further discussion is also based on MEINE [2008] which contains an elaboration of this idea along with comparisons to other filtering methods.

for better reading):

$$\delta_{\mathsf{sdm}} \triangleq \frac{\delta^2_{\mathsf{sog}}}{\lambda_{1,\boldsymbol{S}}}$$

where $\lambda_{1,\boldsymbol{S}}$ can in this case be easily computed [MEINE 2008, p. 94 et seqq]:

$$\lambda_{1,\boldsymbol{S}} = S_{11} + S_{22} + \tfrac{1}{2}\sqrt{(S_{11} - S_{22})^2 + 4 \cdot S_{12}^2}$$

As one can see from figure 2.9b this criterion is suitable for suppressing the occurrence of the streamers even before the actual flowlines are traced which means both an increase in performance and in segmentation quality. We therefore apply this to every watershed segmentation (with $\sigma' = 1.5\sigma$ for equation (2.4)).

Finally, we give two examples for watershed segmentations in figure 2.10.

## 2.5 Mean Shift

The mean shift procedure is a non parametric clustering method for feature spaces with an EUCLIDEAN metric. Originally introduced by FUKUNAGA and HOSTETLER [1975], it was long forgotten until rediscovered and brought back into the field by CHENG [1995] and applied to the problem of image segmentation by COMANICIU and MEER [2002, and references therein]. Unlike many classical clustering techniques, mean shift clustering can be applied to arbitrary structured feature spaces without stating any assumptions about the type of the underlying distribution or the number of clusters present.

One can think of the vectors of a feature space as samples of a probability density function (PDF). Then, dense regions correspond to maxima of the PDF, the *modes*, which are among the points where the gradient of the PDF vanishes. The mean shift procedure employs a kernel to find all modes and their basins of attraction, that is, it maps each feature vector to a mode. This mapping represents the final clustering outcome of the procedure and is effectively done in the manner of a deepest ascent method with adaptive step size. Hence, its parameters only determine the type and size of the kernel.

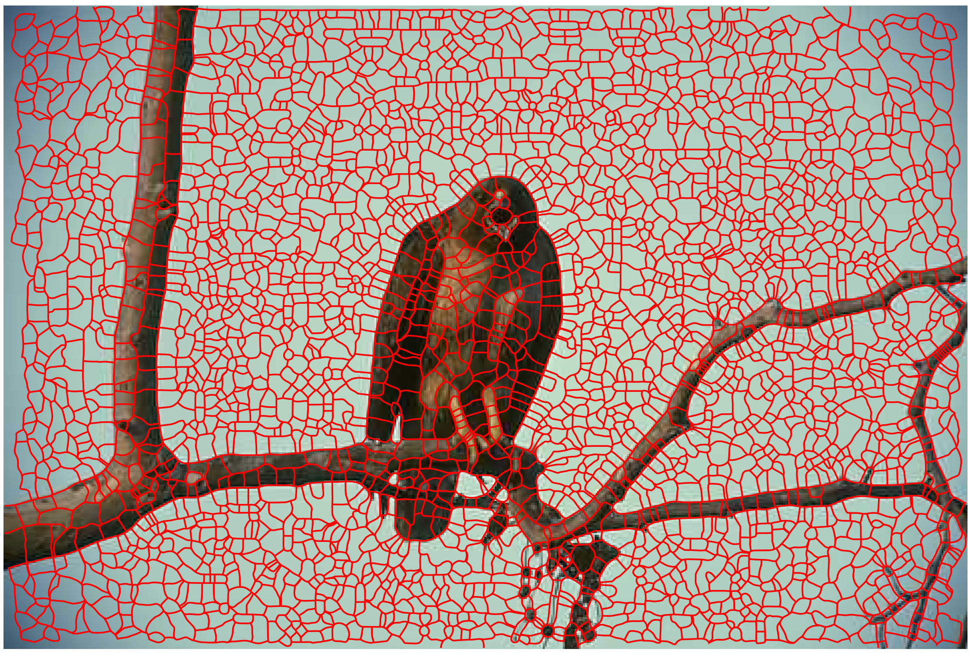
More formally, for a set $S \triangleq \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ of *samples* with $\boldsymbol{x}_i \in \mathbb{R}^d$ (with EUCLIDEAN metric) the well-known *multivariate kernel density estimator* with *bandwidth* $\sigma$ for a PDF $f$ is

$$\tilde{f}_K(\boldsymbol{x}) \triangleq \frac{1}{n\sigma^d} \sum_{i=1}^{n} K\left(\frac{\boldsymbol{x} - \boldsymbol{x}_i}{\sigma}\right).$$

For the mean shift procedure we are only interested in radially symmetric kernels of the form

$$K(\boldsymbol{x}) \triangleq c \cdot k(\|\boldsymbol{x}\|^2)$$

where $k$ is the kernels *profile* and $c$ is in the following always a constant such that the expression



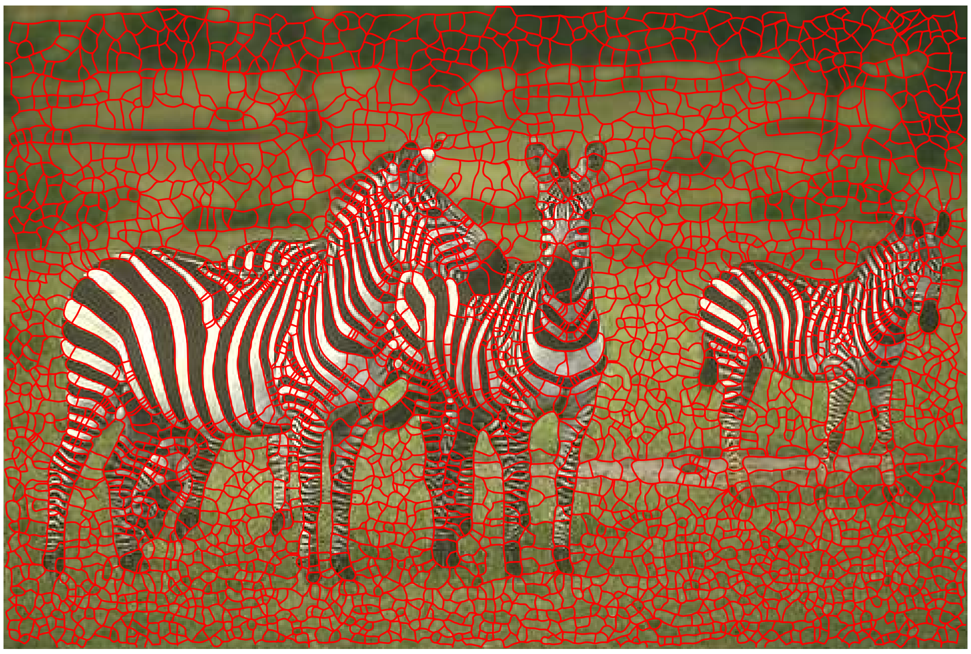


**Figure 2.10:** *Two examples for the exact watershed transform with* $\sigma = 2.0$.

at hand integrates to one. Now, we would like to reveal the structure of $f$ by an estimation of its gradient

$$\begin{aligned}\nabla\tilde{f}_K(\boldsymbol{x}) &= \frac{c}{n\sigma^d}\sum_{i=1}^{n}\nabla k_{\sigma,i}(\boldsymbol{x}) &(2.5)\\ &= \frac{2c}{n\sigma^{d+2}}\sum_{i=1}^{n}(\boldsymbol{x}-\boldsymbol{x}_i)k'_{\sigma,i}(\boldsymbol{x}) &(2.6)\end{aligned}$$

where $k_{\sigma,i}(\boldsymbol{x}) \triangleq k\left(\left\|\frac{\boldsymbol{x}-\boldsymbol{x}_i}{\sigma}\right\|^2\right)$. With $g \triangleq -k'$ and using some simple arithmetic, one gets the following equations

$$\begin{aligned}\nabla\tilde{f}_K(\boldsymbol{x}) &= \frac{2c}{n\sigma^{d+2}}\sum_{i=1}^{n}(\boldsymbol{x}_i-\boldsymbol{x})g_{\sigma,i}(\boldsymbol{x}) &(2.7)\\ &= \frac{2c}{n\sigma^{d+2}}\left[\sum_{i=1}^{n} g_{\sigma,i}(\boldsymbol{x})\right]\underbrace{\left[\frac{\sum_{i=1}^{n}\boldsymbol{x}_i g_{\sigma,i}(\boldsymbol{x})}{\sum_{i=1}^{n} g_{\sigma,i}(\boldsymbol{x})}-\boldsymbol{x}\right]}_{\boldsymbol{m}(\boldsymbol{x})}. &(2.8)\end{aligned}$$

The term $\boldsymbol{m}(\boldsymbol{x})$ denotes the *mean shift* away from $\boldsymbol{x}$ (under the kernel $G$). Note that the first term in equation (2.8) is proportional to $\tilde{f}_G(\boldsymbol{x})$, so we finally arrive at

$$\boldsymbol{m}(\boldsymbol{x}) = \tfrac{1}{2}h^2\sigma\frac{\nabla\tilde{f}_K(\boldsymbol{x})}{\tilde{f}_G(\boldsymbol{x})}. \tag{2.9}$$

Because of equation (2.9), the mean shift at a certain position points in the direction of the estimated gradient and is normalized by an estimation of $f$ at that position. The normalization ensures a large magnitude of the mean shift in sparse regions and a small magnitude in dense regions, i.e. near modes of the density function. Hence, in order to find the mode near a given sample $\boldsymbol{x}$ one can use the simple iterative scheme

$$\boldsymbol{y}^{(i+1)} = \boldsymbol{m}(\boldsymbol{y}^{(i)}) \tag{2.10}$$

with $\boldsymbol{y}^{(0)} = \boldsymbol{x}$ which corresponds to a deepest ascent technique with adaptive step size.

### 2.5.1 Mean Shift and Image Segmentation

For color image segmentation the mean shift method can be applied to the *joint* feature space derived from an image function, i.e. each vector has the form $\boldsymbol{x} = (\boldsymbol{x}_s, \boldsymbol{x}_f)$ with $\boldsymbol{i}_{\mathsf{Luv}}(\boldsymbol{x}_s) = \boldsymbol{x}_f$ and $\boldsymbol{x_s} \in \underline{w} \times \underline{h}$ with $w$ being the width and $h$ being the height of the image function. We also write $\boldsymbol{x}_i$ in order to denote the corresponding vector of the $i$-th pixel given an arbitrary but fixed enumeration of the pixels. The CIE LUV color space (see section 2.1) is used instead of the RGB color space to take account for that the EUCLIDEAN distance is used for the kernels.

Both the EPANECHNIKOV and normal kernel was shown to be suitable for image segmentation

[Comaniciu and Meer 2002]. Here, we use the normal kernel

$$K_N(\boldsymbol{x}) \triangleq \frac{c}{\sigma_s^2 \sigma_f^3}\, k_N\left(\left\|\frac{\boldsymbol{x}_s}{\sigma_s}\right\|^2\right) k_N\left(\left\|\frac{\boldsymbol{x}_f}{\sigma_f}\right\|^2\right)$$

with profile

$$k_N(x) \triangleq \exp\left(-\tfrac{1}{2}x\right).$$

Note that the normal kernel is decomposed in a spatial and in a feature part in order to account for proper normalization. Moreover, for a fast implementation the kernel should have finite support, so the spatial part is truncated for $\|\boldsymbol{x}_s\| > 3\sigma_s$.

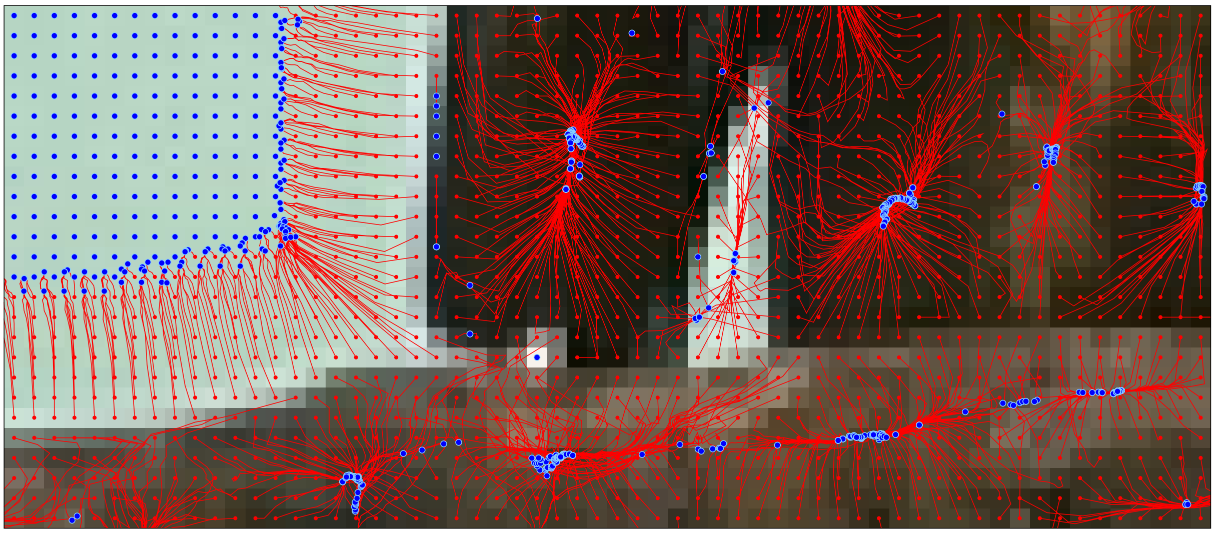


**Figure 2.11:** *Part of the bird image; red points indicate starting points of the mean shift filtering, red polygon lines describe mean shift steps, and blue points are convergence points*

We run the iteration scheme represented by equation (2.10) for each pixel as a starting point. Convergence is then assured for the normal kernel within infinite number of steps (see [Comaniciu and Meer 2002] for more details); for a finite number of steps an upper bound on the magnitude of the mean shift is sufficient or equivalently by setting an upper bound on the difference of consecutive iteration steps, i.e. $\left\|\boldsymbol{y}^{(i)} - \boldsymbol{y}^{(i+1)}\right\| < \tau$ for some small $\tau$, e.g. $10^{-5}$. The resulting mode of the procedure for each pixel is written $\boldsymbol{z}_i$. This is also called *discontinuity preserving filtering*. An illustration of the iteration steps taken for some pixels can be seen in figure 2.11; the filtering of a whole image is depicted in figure 2.12.

For the final segmentation outcome, two pixels are assigned to the same segment if they are in the same equivalence class of the relation formed by the transitive closure of

$$\boldsymbol{x}_i \sim \boldsymbol{x}_j \quad \text{iff} \quad \|\boldsymbol{z}_{s,j} - \boldsymbol{z}_{s,i}\| < \sigma_s \wedge \|\boldsymbol{z}_{f,j} - \boldsymbol{z}_{f,i}\| < \sigma_f .$$

For an effective implementation of this fast access to the neighboring modes of a given pixel must be possible. This can be achieved with a *kd-tree* data structure [Bentley 1975]. The final outcome of the technique is a partition of the plane with pixel accurate segment boundaries, see figure 2.13 for examples.

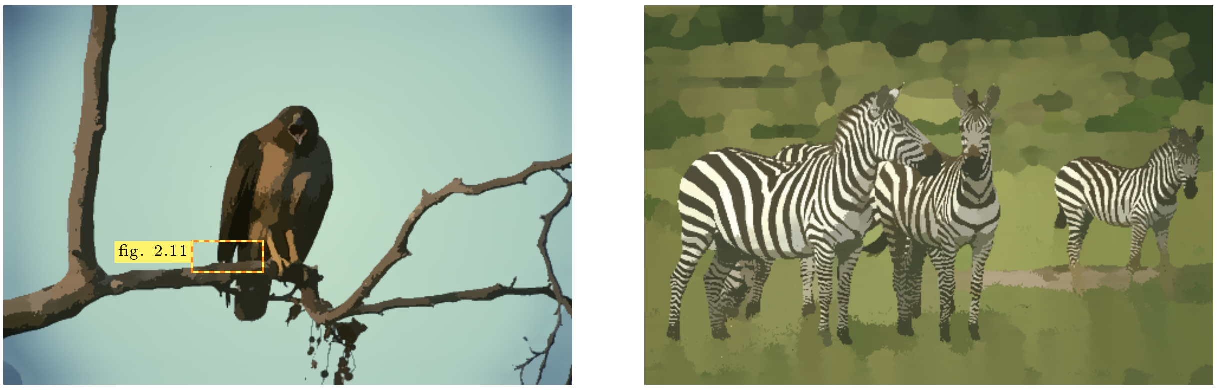
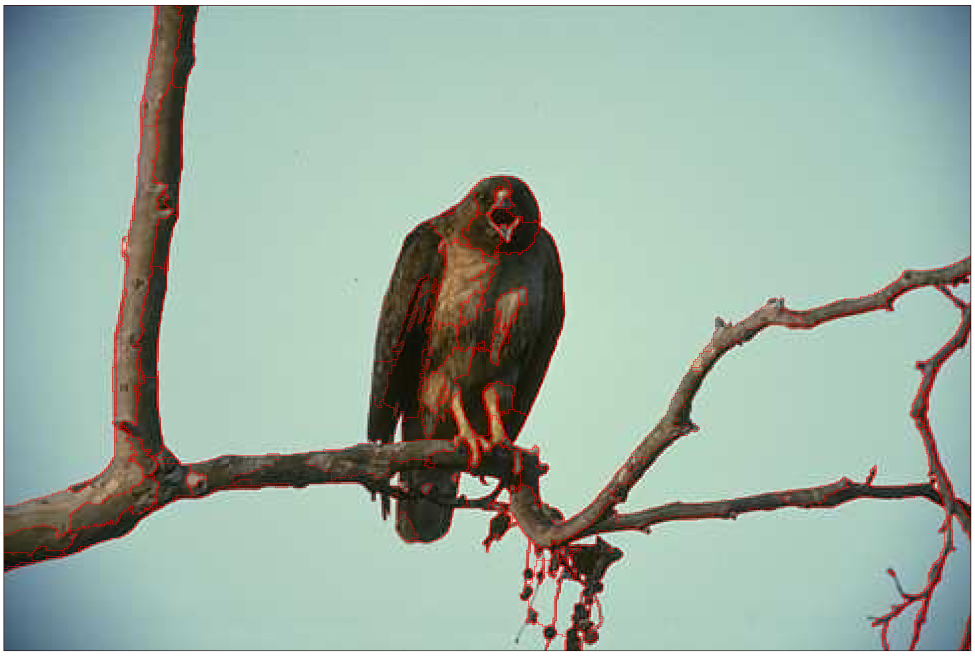


**Figure 2.12:** *Two images after discontinuity preserving filtering with $\sigma_s = 8$ and $\sigma_f = 10$*



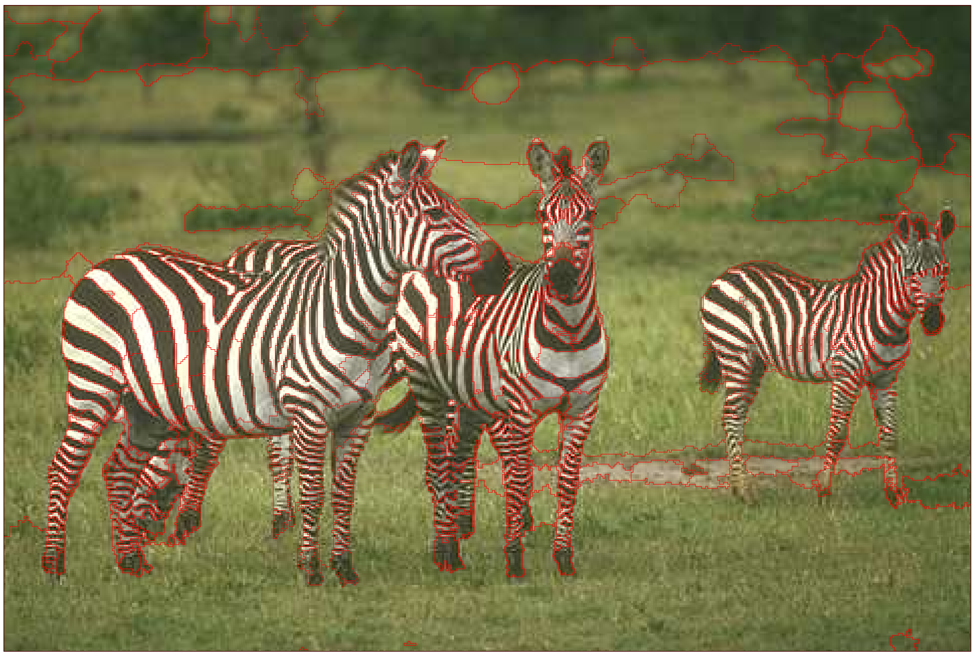


**Figure 2.13:** *Two mean shift segmentations with* $\sigma_s = 8$ *and* $\sigma_f = 10$

# 3 Graph Cuts

The last chapter introduced our image segmentation framework which in its heart relies on minimization of an energy function assessing the segmentation quality. In this chapter we want to shed some light on the nature of that functions. To be specific, all energy functions in the focus of this thesis are tightly connected with the term "graph cuts" and we now turn to a more exhaustive discussion of this concept.

The remainder of this chapter is organized as follows: A formal definition of graph cuts is given first along with some useful terminology. An overview of different graph cut based applications in the field of computer vision follows. Then, we present some similarity measures used here for the weighted graphs. As already noted in the last chapter, only the class of so called *normalized cuts* is in the focus of this work. A detailed discussion of each normalized cut used here and an algorithm to compute it is presented at the end of this chapter.

## 3.1 Basic Terminology

We start with a series of definitions which are used thoroughly for the rest of this chapter. Central to these definitions is the concept of graph cuts:

**Definition 12** (CUT). *Let $G$ be a connected graph. For two sets $A, B \subseteq E(G)$ the term $E(A,B)$ denotes the set of all edges of $G$ which have one vertex in $A$ and the other vertex in $B$, i.e.*

$$E(A,B) \triangleq \{\{u,v\} \in E(G) \,|\, u \in A \wedge v \in B\}\,.$$

*A set of edges $C \subset E(G)$ is a* (graph) cut *iff*

$$\exists A \subseteq V(G) : E(A, A^{\mathsf{c}}) = C.$$

*We emphasize this connection by the formulation $A$* induces a cut on $G$. *A cut is* nontrivial *iff both $A$ and $A^{\mathsf{c}}$ are nonempty and* connected *iff both the subgraphs induced by $A$ and $A^{\mathsf{c}}$ are connected. The sets $A$ and $A^{\mathsf{c}}$ are also called the* components *of the cut.*

So removing all edges in a cut from a graph $G$ "cuts" $G$ into two disconnected parts and therefore imposes a bi-partition on the graph, i.e. a partition of the vertices of $G$ into the sets $A$ and $A^{\mathsf{c}}$.

**Definition 13** (TERMINAL CUT). *A cut $E(A, A^{\mathsf{c}})$ is a $(s,t)$*-cut *iff for two selected* terminals *$s, t \in V(G)$ – also called* source *and* sink *– $s \in A$ and $t \in A^{\mathsf{c}}$ holds. A $(s,t)$-cut is also called* *(*two*)* terminal cut*. The set $A \setminus \{s\}$ is the* source set *and all edges incident to the source are called* source edges*, the terms* sink set *and* sink edges *are defined analogously. A terminal cut is* trivial *iff the source set only contains the source or if the sink set only contains the sink.*

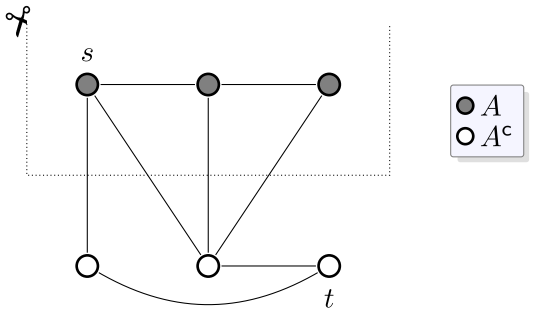


**Figure 3.1:** *A small example for graph cuts: The cut shown (dotted line) cuts several edges of the graph yielding a partition into the sets $A$ and $A^\mathsf{c}$. This is also a valid terminal cut if the terminals are $s$ and $t$.*

**Definition 14** (CROSSING CUTS)**.** *Two cuts $E(A_1, A_1^\mathsf{c})$ and $E(A_2, A_2^\mathsf{c})$* cross *iff all of the following sets*

$$A_1 \cap A_2, A_1 \cap A_2^\mathsf{c}, A_2 \cap A_1^\mathsf{c} \textit{ and } A_1^\mathsf{c} \cap A_2^\mathsf{c}$$

*are nonempty.*

Figuratively speaking, two cuts cross if two lines running through all edges of one cut intersect or formulated the other way round, two cuts do *not* cross if one component of one cut is completely contained in one component of the other cut.

**Definition 15** (CUT VALUE)**.** *If $G$ is a weighted graph with edge weight function $\omega$, the term*

$$\omega(A, A^\mathsf{c}) \triangleq \sum_{e \in E(A, A^\mathsf{c})} \omega(e)$$

*denotes the* costs *or* cut value *of the cut induced by $A$.*

Note, that we use the symbol "$\omega$" both for a function defined on edges and as a function defined on a set of edges and that we also write $\omega(A, A^\mathsf{c})$ instead of $\omega(E(A, A^\mathsf{c}))$. The same applies for "$\gamma$" which we use for a function on vertices and as a function on a set of vertices:

$$\gamma(A) \triangleq \sum_{v \in A} \gamma(v).$$

This is used very often in the remainder of this chapter but should be unproblematic since the meaning is always clear from the context.

## 3.2 Graph Cuts in Computer Vision

A classical problem of extremal graph theory is to find a cut with minimal costs:

**Definition 16** (MINIMUM CUT). *For a graph $G$ a cut $C = (S, S^c)$ is a* minimum cut *if its costs are minimal, i.e.*

$$\omega(S, S^c) = \min_{A \subset V(G)} \omega(A, A^c).$$

*The problem of determining a minimum cut for a given graph $G$ is then* the minimum cut problem *and* the minimum terminal cut problem *is the problem of determining a minimum cut which is also a terminal cut for two beforehand selected terminals.*

Note that for a given graph there can be several cuts which fulfill this property. We occasionally use formulations of the kind "the bi-partition gained from application of a cut" by which we refer to the two components solving the minimum cut problem (or a normalized cut problem) for a specific graph.

For the field of computer vision both optimization problems, the minimum cut problem and the minimum terminal cut problem, have become quite popular for a wide range of applications. The minimum terminal cut problem is tightly connected to a certain class of energy functions which are very useful for many computer vision applications. It is dual to the well-known maximum flow problem [FORD and FULKERSON 1962]. This connection enables us to use one of several algorithms for the maximum flow problem which are of polynomial runtime with small constant factors. An overview of many minimum terminal cut algorithms along with a performance evaluation with focus on computer vision can be found in [BOYKOV and KOLMOGOROV 2004]. The more general minimum cut problem, i.e. without terminals, can be used as a starting point for several graph based hierarchical clustering tools which are here subsumed under the term *normalized cuts*[1].

In the following, we give an overview of how the minimum (terminal) cut problem is related to applications in the field of computer vision. This investigation is headed against the question how these approaches can be used for image segmentation which is the main interest of this work.

### 3.2.1 Recent Work

The first application of minimum terminal cuts as combinatorial optimization tool in vision was presented in [GREIG et al. 1989] for binary image restoration. Being unnoticed by the community by a long time a similar approach based on this work was reported in [BOYKOV et al. 1997] and used for the stereo correspondence problem. In the following years the usefulness of this approach was realized. This led to the development of several similar techniques which can be applied to other fields of vision such as motion, image synthesis, multi-camera scene reconstruction and semi automatic/interactive image segmentation (for an overview with references see e.g. [BOYKOV 2005; BOYKOV and FUNKA-LEA 2006; KOLMOGOROV and ZABIN 2004]). As far as computer vision is concerned, the term "graph cut" is therefore often associated to this kind of applications.

[1]the term "normalized cut" is often associated with the well known work of SHI and MALIK [2000]; we use this term also in a more general meaning for all cuts which can be seen as normalized versions of the minimum cut.

The essence of these methods lies in the minimization of an energy function of the form

$$E(\boldsymbol{x}) = \sum_i E^i(x_i) + \sum_{i<j} E^{i,j}(x_i, x_j) \tag{3.1}$$

with $\boldsymbol{x} = (x_1, \dots, x_n)$ and $x_i \in \underline{k}$. While many authors used their own graph constructions along with some theory to show the correctness of their results, a general framework is presented in [Kolmogorov and Zabin 2004]. It shows under which conditions minimizing arguments for energy functions of the form (3.1) can be found with minimum terminal cuts in appropriate constructed graphs. A general construction scheme for these graphs is given as well. Note, that energy functions of the form (3.1) correspond to optimization problems with possibly $k$ different assignments to the components of the input variable $\boldsymbol{x}$. A main contribution of the work of Kolmogorov and Zabin is how these more general problems can be attacked with the aid of minimum terminal cuts. However, solutions to these problems can not guaranteed to be optimal except for the binary case.

As image segmentation is of main interest here, we briefly want to investigate how these energy functions can be used for it. Most of the work published within this context relates to interactive image segmentation where image data is divided into "back-" and "foreground" items based on inter-pixel affinities and some kind of user input. Examples of those segmentation techniques are presented in [Boykov and Jolly 2001; Boykov and Kolmogorov 2003; Boykov and Funka-Lea 2006; Li et al. 2004]. All of them use the following energy function

$$E(\boldsymbol{x}) = \sum_i \underbrace{[(1 - x_i)p_F(i) + x_i \cdot p_B(i)]}_{E^i(x_i)} + \sum_{i<j} \underbrace{\omega(i,j)(x_i - x_j)^2}_{E^{i,j}(x_i,x_j)} \tag{3.2}$$

with $x_i \in \{0, 1\}$. The function $\omega$ encodes the affinity between nodes as a function of the corresponding pixel intensities and distances (or as a function of features of super-pixels ([Li et al. 2004]), voxels [Boykov and Funka-Lea 2006] or other entities). The functions $p_B$ and $p_F$ represent the possibility that a pixel belongs to the background or foreground. These possibilities can be estimated if samples of both distributions are known which in turn requires some complex preprocessing or manual input. Functions of this form can be minimized via minimum terminal cuts in graph constructions depicted in figure 3.2 (for $n = 3$). This approach is especially qualified for semi-automatic image segmentation since some image elements can be manually specified to be in the fore- or background. These pixels are often called *seeds* as the functions $p_B$ and $p_F$ can then be computed from them. A minimum terminal cut in the associated graphs then corresponds to an argument for (3.2) which minimizes the affinities between pixels on the boundary of the fore- and background and maximizes the probabilities that the pixels within the two segments were classified correctly. However, this approach is not investigated further here as for non interactive segmentation the seeds have to be computed automatically. Nevertheless, we can use a similar approach for the foreground cut which only requires the selection of one seed for the background (see section 3.6 for more details).

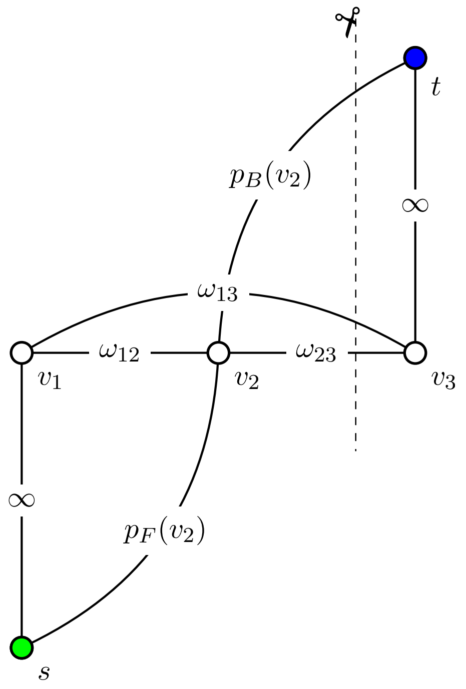


**Figure 3.2:** *An example of the graph construction for three pixels for a binary segmentation problem. The cut yields the bipartition* $\{\{v_1, v_2\}, \{v_3\}\}$ *and has cut value* $p_B(1) + w_{13} + w_{23}$*. Note that the seeds are* $\{v_1\}$ *for the foreground and* $\{v_3\}$ *for the background and therefore have edge weights of infinity in the graph.*

An unsupervised segmentation procedure based on energy functions similar to (3.1) was published in [Ishikawa and Geiger 1998]. Their work is an example of how minimum terminal cuts can be used in graphs not being a "simple" extension of a pixel grid graph such as in figure 3.2. Instead, they use a fairly complex graph construction in order to solve a non binary segmentation problem directly with minimum terminal cuts. The selection of seeds is not done manually but is based on junction detectors which identify points in the resulting segmentation.

A rather distinct approach was demonstrated in [Veksler 2000]. He uses a simple extension of a pixel grid graph's dual with edge weight function $\omega$ to find the smallest $\lambda \in [\mathsf{min}_e\omega(e), \mathsf{max}_e\omega(e)]$ and a series $(S_1, S_1^\mathsf{c}), \dots, (S_m, S_m^\mathsf{c})$ of noncrossing cuts with

$$\omega(S_i, S_i^\mathsf{c}) \leq \ell \cdot \lambda \text{ and } k \leq |S_i| \leq |V| - k \quad \forall i \in \underline{m} \tag{3.3}$$

where $\ell$ is the number of pixels on the border of an image and $k$ a free parameter avoiding trivial solutions of the optimization problem and small segments. The desired cuts can be found with binary search on $\lambda$ and $\mathcal{O}(|V|)$ invocations to a minimum terminal cut algorithm for each different $\lambda$. In fact, Veksler problem formulation can be seen as instance of the *parametric* terminal cut problem which we use in section 3.6 for the foreground cut problem and therefore could be solved more effectively without binary search.

#### 3.2.1.1 Normalized Cuts for Image Segmentation

We now turn to the above-mentioned normalized cuts which can be seen as the second main class of graph cut based methods in computer vision. They are especially suited for unsupervised image segmentation since no sophisticated preprocessing for seeds selection is needed. All cuts presented in the rest of this section are investigated in detail within this work such that the following text is meant to be only an introducing overview and motivation.

The work of Wu and Leahy [1990, 1993] can be seen as the starting point for this branch of segmentation methods. Instead of using minimum terminal cuts for extended pixel grid graphs, the idea of Wu and Leahy was to directly search for minimum terminal cuts in the "pure" pixel grid graph. If the edge weights in such graphs correspond to pairwise similarities of pixels, a cut with minimal costs splits the graph, and thus the image, into two parts divided by a common high contrast boundary. Wu and Leahy presented an effective algorithm capable of effectively determining all possible minimum terminal cuts of a graph which can be used to create a globally optimal segmentation. They successfully used this segmentation method in the field of tissue classification in MR images and for segmenting aerial images.

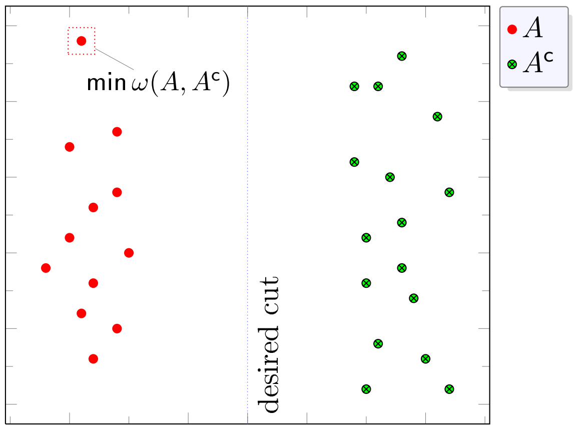


**Figure 3.3:** *Bias of the minimum cut to small regions; all points in the figure represent nodes of a complete graph (every node is connected to all other nodes) and the position of each point indicates the edge weights of the graph (nodes close to each other are very similar). A desired cut in such a graph would separate group $A$ from group $A^{c}$.*

However, as already noted by the authors, the resulting segmentations suffer from many small regions. This bias to small regions is due to the minimum cut on which their optimality criterion is based. The reason for this can be seen from figure 3.3: Assume a node for each point in the figure connected to all other nodes with edge weight inversely proportional to their Euclidean distances. The arrangement of the points suggests a separation into a left and a right half but a minimum cut separates one node from the others. The minimum cut has this tendency to separate small sets of isolated nodes as the costs of the cut also increase with number of edges in it.

This situation initiated many suggestions in the literature on how to appropriately extend the minimum cut problem. The basic idea of all of them is to scale the costs of the edges in the cut such that more balanced partitions are favored:

$$\omega(A, A^{\mathsf{c}}) \longrightarrow \frac{\omega(A, A^{\mathsf{c}})}{f(A, A^{\mathsf{c}})} \tag{3.4}$$

where $f$ is usually a function defined on the edges in the cut or a function defined on the vertices in $A$ and $A^{\mathsf{c}}$. Due to the form of the resulting energy functions, they are subsumed here under the term *normalized cuts*.

The exact nature of the energy function used strongly depends on the existence of an effective algorithm for determining an optimizing cut. While the classical minimum cut and the foreground cut problem is solvable in polynomial time all other cut problems presented below are NP-hard for general graphs. The reason why they can be used for image segmentation nevertheless is twofold. On the one hand some cost measures become computational tractable if they are used only for a more restricted class of graphs such as the class of planar graphs. On the other hand it is sometimes possible to use some kind of relaxation of the original problem which can be effectively computed and used as an approximating solution. An overview of the graph cuts which are in the scope of this work is depicted in table 3.1.

| **name** | **formula** | **complexity** | **section** |
|---|---|---|---|
| minimum cut | $\mathsf{cut}(A, A^{\mathsf{c}}) = \omega(A, A^{\mathsf{c}})$ | P | 3.4 |
| mean cut | $\mathsf{mcut}(A, A^{\mathsf{c}}) \triangleq \dfrac{\omega_1(A, A^{\mathsf{c}})}{\omega_2(A, A^{\mathsf{c}})}$ | NP | 3.5 |
| foreground cut | $\mathsf{fcut}(A, A^{\mathsf{c}}) \triangleq \dfrac{\omega(A, A^{\mathsf{c}})}{\gamma(A)}$ | P | 3.6 |
| isoperimetric cut | $\mathsf{icut}(A, A^{\mathsf{c}}) \triangleq \dfrac{\omega(A, A^{\mathsf{c}})}{\mathsf{min}\,\{\gamma(A), \gamma(A^{\mathsf{c}})\}}$ | NP | 3.7 |
| normalized cut | $\mathsf{ncut}(A, A^{\mathsf{c}}) \triangleq \dfrac{\omega(A, A^{\mathsf{c}})}{\gamma(A)} + \dfrac{\omega(A, A^{\mathsf{c}})}{\gamma(A^{\mathsf{c}})}$ | NP | 3.8 |

**Table 3.1:** *Overview of the different normalized cuts used within this work*

In order to overcome the problems of the minimum cut Wang and Siskind [2003] presented a normalized version they refer to as the "ratio cut". Their cut measure yields a perceptual appealing image segmentation if recursive bi-partitioning is applied starting from a pixel grid graph. The ratio cut is a generalization of the "mean cut" which was introduced in a previous paper [Wang and Siskind 2001] and has better runtime complexity. Since only the generalized version is investigated here, we use the term "mean cut" instead of "ratio cut" nonetheless. A minimum mean cut can be effectively computed if the encountered graph is restricted to be planar and only integer edge weights are used. The corresponding algorithm is presented in detail in section 3.5.

A popular approximation scheme for cut measures is *spectral relaxation* where an eigenvalue decomposition of a graph's LAPLACIAN is used in connection with thresholding techniques. An example of such a relaxation scheme was presented by SHI and MALIK [1997, 2000] who were also the first using a normalized version of the minimum cut for image segmentation. SHI and MALIK also used their normalized cut for recursively splitting a pixel grid graph. It is probably the most prominent example of using graph cuts in this field. Another example of using spectral methods is the "average cut" of SARKAR and SOUNDARARAJAN [2000] which can be seen as a more simpler variant of the normalized cut of SHI and MALIK. Both cut measures are discussed in more detail in section 3.8.

Spectral methods can also be used for the "foreground cut" [PERONA and FREEMAN 1998]. PERONA and FREEMAN did not explicitly tested their method for image segmentation but for line grouping and as clustering tool in general. For the foreground cut we do not use spectral relaxation but instead present a polynomial algorithm in section 3.6.

GRADY and SCHWARTZ [2006*a*,*b*] proposed the "isoperimetric cut" for image segmentation. They also used recursive bi-partitioning of a pixel grid graph. Although their are also spectral methods for this cut (see [GRADY and SCHWARTZ 2006*a*] and references therein) a new simpler linear approximation technique was presented, see section 3.7 for more details.

## 3.3 Similarity Measures

In this section the different kinds of edge weights measuring the similarity of neighboring faces in a region adjacency graph of the segmentation framework described in section 2.3.4 are discussed. Given a partition of the plane $P$ and the corresponding region adjacency graph $G$, we define three edge weight functions which measure the dissimilarity of two neighboring faces. They can be transformed into a similarity measure with an additional transformation which we describe below. These three dissimilarity measures are:

1 The difference of the mean color of two segments is defined as:

$$\mathsf{colorMean}(v_1, v_2) \triangleq \|\mathsf{cm}(v_1) - \mathsf{cm}(v_2)\|$$

with

$$\mathsf{cm}(v) \triangleq \frac{\int_{\mathcal{R}(v)} \boldsymbol{i}(\boldsymbol{p})\,\mathrm{d}\boldsymbol{p}}{\mathsf{area}(v)}$$

being the mean color of the face associated with $v$. Recall that $\boldsymbol{i}$ is an image function (see section 2.1) and that $\mathcal{R}$ maps every vertex of $G$ to its face in $P$. The term $\mathsf{area}(v)$ is the area of the face $\mathcal{R}(v)$ and can be simply computed if all edges of $P$ are polygonal arcs.

2 The average gradient magnitude along the common boundary of two segments is defined as:

$$\mathsf{averGrad}(v_1, v_2) \triangleq \frac{\int_{\mathcal{R}(\{v_1,v_2\})} \|\nabla \boldsymbol{i}(\boldsymbol{p})\| \, \mathrm{d}\boldsymbol{p}}{\mathsf{length}(\{v_1, v_2\})}$$

where $\mathsf{length}(\{v_1, v_2\})$ denotes the length of the common edges $\mathcal{R}(\{v_1, v_2\})$ of the corresponding faces. Again, it can be simply computed if the edges of $P$ are polygonal arcs.

3 While the former measure are quite simple, we also would like to use a more complex one which analyzes the distribution of the pixels in each face. We use the *earth mover distance* for this:

$$\mathsf{EMD}(v_1, v_2) \triangleq \mathsf{emd}(\mathsf{sig}(v_1), \mathsf{sig}(v_2))$$

where the function $\mathsf{sig}$ maps every vertex of the graph to the *signature* of the corresponding face and $\mathsf{emd}$ is the earth mover distance between them. More details on this follow in the next section.

In order to get an similarity measure we apply a nonlinear transformation to each edge weight function: For a given edge weight function $s$ the final edge weight for an edge $e$ of the graph $G$ is

$$\mathsf{length}(e) \cdot \exp\left(-\beta \cdot s(e)^2\right) \tag{3.5}$$

where $\beta$ is the *scaling parameter* of $s$. The incorporation of the common boundary length reflects the geometric properties of the regions in the underlying segmentation. This way the cut value of a cut now increases with the total length of the common boundary of its two components and not with the number of edges in the cut as it is the case if we do not use the scaling factor $\mathsf{length}(e)$.

The nonlinear transformation allows one to adjust how sensitive a cut is with respect to increasing edge weights. Especially, for the normalized cuts the dominance of the normalization factor can be adjusted this way. Similar transformations are used in the literature, e.g. in [Cox et al. 1996; Grady and Schwartz 2006*b*; Shi and Malik 2000; Wu and Leahy 1993]. Here we only add the incorporation of the common boundary length for reasons already given above.

## 3.3.1 The Earth Mover Distance

The earth mover distance (EMD) is a distance measure for signatures [Rubner et al. 1998, 2000]. A *signature* $s \triangleq \{(\boldsymbol{x}_1, w_1), \dots, (\boldsymbol{x}_m, w_m)\}$ is a set of ordered pairs with $\boldsymbol{x}_i \in \mathbb{R}^n$ being a *feature vector* and $w_i \in \mathbb{R}$ being its *weight*. Similar to a histogram, a signature represents an estimate of an unknown probability density function based on a set of samples. In fact every histogram can be represented by a signature: For a histogram we select the feature vectors such that for a given $i \in \underline{n}$ the $i$-th components of the feature vectors form a sequence $x_{1_i} \leq \cdots \leq x_{m_i}$ of equidistant monotone increasing numbers. The feature vectors then represent the midpoints of the bins and the weights $w_i$ equal the number of samples which lie in the bin associated with the $i$-th feature vector.

However, a drawback of histograms is that they are static structures, i.e. the quantization of the feature space is everywhere the same but often the samples are located in isolated regions of it. As a result, only a small fraction of the bins actually contains significant information. So, one has to choose between a high resolution of the data and a manageable number of bins. Signatures do not have this drawback as the feature vectors can be arbitrarily distributed in the feature space. As an example, the feature vectors can represent clusters of the samples and its weights the number of samples falling into the corresponding cluster.

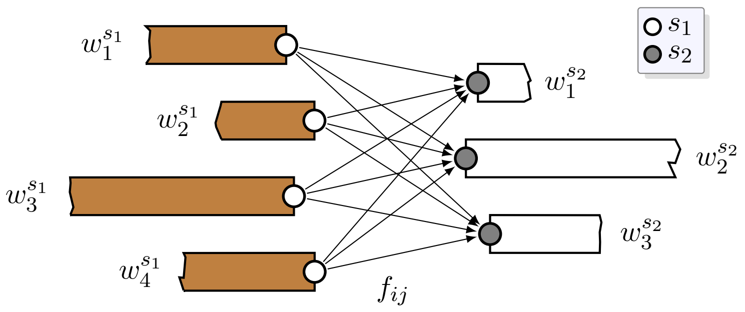


**Figure 3.4:** *Computing the earth mover distance corresponds to solving a transport problem in a bipartite graph. Each signature is represented by several vertices (one for each feature/weight pair). The signature with the greater total amount of weights is the associated with the piles of earth ($s_1$) and the other signature is associated with the holes ($s_2$). The height of the earth piles (holes) corresponds to the weight of the features vectors.*

The idea of the EMD is to imagine one signatures as piles of earth located in the feature space at points given by the feature vectors with height equal to their weights and to imagine the other signature as holes defined analogously. Then, the EMD measures the amount of work to fill the holes with earth where the work is measured in units of earth times units of distance the earth is moved. In order to assure that all holes can be filled the signature with smaller total weight is associated with them. More precisely, for two signatures $s_1 \triangleq \left\{(\boldsymbol{x}_1, w_1^{s_1}), \ldots, (\boldsymbol{x}_l, w_l^{s_1})\right\}$ and $s_2 \triangleq \{(\boldsymbol{y}_1, w_1^{s_2}), \ldots, (\boldsymbol{y}_m, w_m^{s_2})\}$ the earth mover distance is defined as [Rubner et al. 2000]:

$$\mathsf{emd}(s_1, s_2) \triangleq \frac{\sum_{i=1}^{l}\sum_{j=1}^{m} c_{ij} f_{ij}}{\sum_{i=1}^{l}\sum_{j=1}^{m} f_{ij}} \tag{3.6}$$

with $c_{ij} \triangleq \|\boldsymbol{x}_i - \boldsymbol{y}_j\|$.[2] The terms $f_{ij}$ present the earth transported from pile $i$ of one signature to hole $j$ of the other signature. They are selected such that the expression

$$\sum_{i=1}^{l}\sum_{j=1}^{m} c_{ij} f_{ij}$$

[2]Actually, $c_{ij}$ can be any distance defined on feature vectors, we use this straightforward selection.

is minimal subject to the conditions:

$$f_{ij} > 0, \quad \forall i \in \underline{l}\, \forall i \in \underline{m} \tag{3.7}$$

$$\sum_i f_{ij} = w_j^{s_2}, \quad \forall j \in \underline{m} \tag{3.8}$$

$$\sum_j f_{ij} \leq w_i^{s_1}, \quad \forall i \in \underline{l} \tag{3.9}$$

$$\sum_{i=1}^{l} \sum_{j=1}^{m} f_{ij} = \mathsf{min}\Big\{\sum_{i=1}^{l} w^{s_1}, \sum_{j=1}^{m} w_j^{s_2}\Big\} \tag{3.10}$$

where condition (3.7) ensures that earth is only moved in one direction, i.e. from piles to holes, condition (3.8) ensures that all holes are filled, condition (3.9) ensures that from each pile only as much earth is transported than available, and condition (3.9) ensures that the maximum of earth is transported which means that the signature with smaller total weight is the signature associated with the holes. The normalization factor of equation (3.6) then assures that signatures with smaller total weight are not favored.

The $f_{ij}$'s can be computed by solving the *transport problem* (see [Rubner et al. 2000] and references therein) which is defined for directed bipartite weighted graphs, see figure 3.4. [3]

#### 3.3.1.1 Signature Computation

Given a partition of the plane $P$ and the corresponding RAG $G$ the signatures of the faces are computed as follows: For every vertex $v \in V(G)$ we determine the set of pixels which are in the face associated with $v$:

$$\{\, \boldsymbol{i}_{\mathsf{Luv}}(\boldsymbol{p}) \,|\, \boldsymbol{p} \in \underline{w} \times \underline{h} \wedge \boldsymbol{p} \in \mathcal{R}(v)\} \tag{3.11}$$

where $\boldsymbol{i}_{\mathsf{Luv}}$ is the image function with width $w$ and height $h$ from which $P$ has been created with color values converted to the CIE LUV color space (cf. section 2.1). From the set (3.11) $n$ clusters with centroids $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n$ with $\boldsymbol{c}_i \in \mathbb{R}^3$ are computed with the well known K-means clustering algorithm. For the vertex $v$ the term $\mathsf{sig}(v) \triangleq \{(\boldsymbol{c}_1, w_1), \ldots, (\boldsymbol{c}_n, w_n)\}$ then denotes the signature of $v$ where $w_i$ is the number of points which are in cluster $i$ normalized by the total number of points in (3.11). In order to get a good representation for the color values in each face, we have decided to choose $n = 15$ which is a fairly high number of clusters for each face in $P$: In [Rubner et al. 2000] $n = 8.8$ on average is used for a whole image with size comparable to size of the images of the Berkeley Image Database.

[3] At the time of writing an implementation in C of the EMD was available at `http://www.cs.duke.edu/~tomasi/software/emd.htm`

## 3.4 Minimum Cut

The minimum cut problem concerns the simplest cost measure considered here. Recall its definition of section 3.2: For a weighted graph $G$ with edge weight function $\omega$ we seek for a nontrivial cut $E(S, S^c)$ whose cut value is minimal, i.e.

$$\omega(S, S^c) = \min_{A \subset V(G)} \omega(A, A^c)$$

Note that every minimum cut is also a minimum terminal cut for every pair of terminals $(s, t)$ with $s \in A$ and $t \in A^c$ and as a first characterization we can show that every minimum terminal cut must be connected:

**Theorem 1.** *Let $C = E(A, A^c)$ be a minmum terminal cut in a connected simple graph $G$. Then both $A$ and $A^c$ must be connected.*

*Proof.* Without loss of generality assume that $A$ is not connected and let $A_1, \ldots, A_n$ be its connected components. One of these components must contain one of the terminals, say $A_i$. This also means that $E(A_i, A_i^c)$ is a minimum terminal cut with strictly smaller cut value than $C$ which means that $C$ cannot be a minimum terminal cut. □

There are several algorithms for solving the minimum cut problem. As there are many effective algorithms for finding a minimum terminal cut and there are $n(n-1)/2$ different terminal pairs in a graph, with $n$ being the number of vertices, one could in a naive approach simply search for the smallest terminal cut among all terminals. However, a both very simple and fast algorithm was developed by Stoer and Wagner [1997] with runtime complexity $\mathcal{O}(|V| \cdot |E| + |V|^2 \log |V|)$ for an undirected simple graph $G = (V, E)$. It is based on *vertex condensation* and uses a scheme very similar to the one of an algorithm by Gomory and Hu [1961] (see below).

However, for the minimum cut we do not use the recursive segmentation scheme of section 2.3.4 but instead use the clustering scheme of Wu and Leahy [1993] which uses several minimum terminal cuts of a graph. The reason for this is twofold: Firstly, we already discussed the bias of the minimum cut to produce very small segments in section 3.2.1.1. Indeed, we have experienced this behavior, if the minimum cut is recursively applied to region adjacency graphs from an initial segmentation. The result is bad performance of the overall procedure as the sizes of the encountered subgraphs decay very slow in practice. The clustering scheme of Wu and Leahy also has this bias but is far more effective to compute.

Secondly, it is possible to effectively compute all minimum terminal cuts of a simple graph with an algorithm by Gomory and Hu [1961]. They can be encoded in a tree structure, which we call here Gomory-Hu tree, and used to form a partition of the vertex set of the graph into several components which fulfill a global optimality criterion. Note, that for the recursive scheme only binary optimization problems are solved for the single encountered subgraphs. We discuss this aspect in more detail in section 3.4.2.

The idea of using an Gomory-Hu tree for the segmentation problem was already presented by Wu and Leahy [1993] for pixel grid graphs. They developed an modified version of the algorithm of Gomory and Hu which has better runtime performance in practice but is harder to implement. Since the problem size of the region adjacency graphs computed here are much smaller than the pixel grid graphs of the associated image, the original algorithm will also do.

Finally, we would like to note that we have found several times in the literature that the work of Wu and Leahy has been cited as an example of recursively searching for minimum cuts, e.g. [Shi and Malik 2000; Soundararajan and Sarkar 2003; Veksler 2000]. In fact that is not true and it can be shown that the method of Wu and Leahy, that we are about to describe in the following, actually does not give the same results. We give a simple counter example in section 3.4.3.

### 3.4.1 Gomory-Hu Trees

The algorithm of Gomory and Hu offers an effective way for revealing the complete minimum terminal cut structure of a graph. More precisely, it constructs a tree which can be used to compute a minimum terminal cut for every possible pair of nodes in the given graph. For the construction of the tree itself only $|V(G)| - 1$ calls to an arbitrary minimum terminal cut algorithm are needed. Even better, due to *vertex condensation* most of these calls encounter much smaller problem instances (see below). Using the tree an arbitrary minimum terminal cut can then be computed with linear runtime complexity.

In order to clarify how this tree can be useful for solving the $k$-partition problem described above, it is necessary to get a deeper understanding of the nature of this tree structure. For this, consider a connected weighted simple graph $G$ and the complete graph $G^*$ which has the same vertex set as $G$ and an edge weight function $\omega^*$ which assigns to every edge $\{u, v\}$ of $G^*$ the cut value of the minimum $(u, v)$-cut in $G$. A classical result of the work of Gomory and Hu [1961] is that there is always a minimum spanning tree $T$ of $G^*$ with the following properties: If for a given pair $(s, t)$ of terminals the edge $\{s, t\}$ is not in $T$ the cut value of the corresponding minimum terminal cut is equal to the smallest edge weight of the edges on the unique path leading from $s$ to $t$. Moreover, if this edge is removed from $T$ the tree falls into two connected components which are also the components of the minimum $(s, t)$-cut. So, this means that we can effectively compute every desired minimum terminal cut of $G$ if we have the spanning tree $T$. For a given pair of terminals, one only has to determine the smallest edge on the path from the terminal to the other and compute the connected components of the tree resulting from the removal of this edge. This tree is called *Gomory-Hu tree* here and its mathematical characterization is captured in the following definition:

**Definition 17** (Gomory-Hu tree)**.** *A graph $T_G$ with edge weight function $\omega_T$ is a* Gomory-Hu tree *of a connected simple graph $G$ with edge weight function $\omega$ iff it has the following properties:*

- *$T_G$ is a spanning tree of the complete graph with the vertex set $V(G)$.*

- *For every pair of terminals $s, t \in V(G)$ the corresponding minimum terminal cut in $G$ can be computed from $T_G$ as follows: Let $e$ be the smallest edge with respect of $\omega_T$ on the unique path connecting $s$ with $t$ in $T_G$. The removal of $e$ disconnects $T_G$ into two parts $S, S^c$ such that $C = E(S, S^c)$ is a minimum $(s,t)$-cut with $\omega(C) = \omega_T(e)$.*

The following propostion is given without proof as it requieres alot of theory not introduced here. A stringent proof can be found in [Gomory and Hu 1961; Hu et al. 1982] instead; a less formal argumentation can be found in [Christofides 1975].

**Theorem 2.** *For every connected simple graph there is a Gomory-Hu tree.*

The existence of a Gomory-Hu tree for each simple graph has some interesting implications. At first note, since each Gomory-Hu tree is also a minimum spanning tree, it has $|V| - 1$ edges. This also means that only $|V| - 1$ different minimum terminal cuts are needed to have a minimum terminal cut for every possible pair of terminals. There can be still $\binom{|V|}{2}$ different minimum terminal cuts in a graph but they can only take on $|V| - 1$ different cut values. The existence of a Gomory-Hu tree for every connected simple graph makes also the following property of minimum terminals cuts easy to derive:

**Lemma 1.** *For every two pairs of terminals there are two corresponding minimum terminal cuts in a connected simple graph which do not cross.*

*Proof.* This follows from the fact that for every pair of terminals there is a minimum terminal cut represented by an edge in a Gomory-Hu tree of the corresponding graph. So, for two pairs of terminals this minimum cuts are either identical, as they are represented by the same edge; otherwise they are represented by two different edges, say $e_1$ and $e_2$. If one these edges, e.g. $e_1$, is removed form the tree, $e_2$ must be in one of the resulting components. Therefore, removing $e_2$ from the tree yields two components with one being entirely contained in the components from the first removal which proves the statement. □

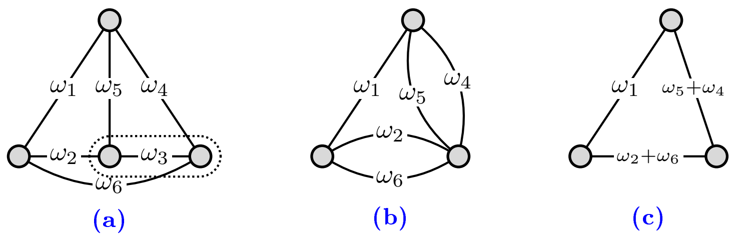


**Figure 3.5:** *Example of vertex condesation; the two nodes enclosed by the dotted line (in (a)) are to be condensed; the condensed graph is in shown (c)*

We now give an algorithm for actually computing a Gomory-Hu tree. For this we need the concept of *vertex condensation*:

**Definition 18** (VERTEX CONDENSATION)**.** *Let $G$ be a simple graph with edge weight function $\omega$ and $S \subseteq G(V)$ a set of vertices. For another graph $G'$ with edge weight function $\omega'$ we say $G'$* can be created from $G$ by vertex condensation of $S$ (into $s$) *iff the following holds:*

- $V(G') = V(G) \setminus S \cup \{s\}$ *with* $s \notin V(G)$
- $E(G') = \{\{u, v\} \in E(G) \mid u, v \notin S\} \cup \{\{u, s\} \mid \exists v : \{u, v\} \in E(S, S^c)\}$
- $\omega'(u, v) = \begin{cases} \omega(\{u\}, S) & \textit{if} \quad v = s \\ \omega(u, v) & \textit{else} \end{cases}$

Stated less formal, vertex condensation of a set $S$ of vertices in a weighted graph means to replace this set with a new node $s$ with all edges having a common starting point $u \in S^c$ replaced by a new edge $\{u, s\}$ with edge weight equal to the sum of the weights of these edges. A small example for this operation can be seen in figure 3.5. The following theorem shows the usefulness of vertex condensation:

**Theorem 3.** *Let $E(A, A^c)$ be a minimum terminal cut in a connected simple graph $G$. For two terminals $s, t \in A$, a minimum $(s, t)$-cut in the graph created by vertex condensation of $A^c$ is also a minimum $(s, t)$-cut in $G$. The same applies for $s, t \in A^c$ with vertex condensation of $A$.*

Again, we would like to refer to [GOMORY and HU 1961] for a proof. Note that theorem 3 also means that for a pair of those minimum terminal cuts holds that they do not cross.

The computation of a GOMORY-HU tree now can be achieved by a series of minimum terminal cuts in condensed graphs, see algorithm 3.4.1. Additionally, we would like to give some comments on certain steps of the algorithm:

**1** The tree is initialized with a single vertex which represents the whole vertex set of $G$. During the algorithm, this set is step by step split into smaller sets. So, the vertices of the tree are actually subsets of vertices of $G$. The algorithms is finished if all sets of the tree build only contain one vertex of $G$.

**2** + **3** Strictly speaking, the step 2 subsumes several vertex condensations in the sense of definition 18. The minimum terminal cuts are thus not computed in the original graph but in a condensed version (step 3). Note that due to theorem 3 a minimum terminal cut in the condensed graph $G'$ is also minimum terminal cut in the original graph $G$ as the to be condensed vertex sets have been computed by minimal terminal cuts in the preceding steps of the algorithm. The condensation also ensures that all minimum terminal cuts computed do not cross each other. As a sideffect, the runtime of whole algorithm benefits from this as the condensed graphs have less vertices as the original graph.

**4** The tree is modified in each iteration based on the minimum terminal cut of step 3: The vertex set $N$ is split into two sets $v_1$ and $v_2$ which become new vertices of the tree with an

**Algorithm 3.4.1:** gomoryHuTree

```
Input: a connected and undirected graph $G = (V, E)$ with edge weight function $\omega$
Output: a Gomory-Hu tree of $G$
begin
1   $V_T \leftarrow \{V\}$, $E_T \leftarrow \emptyset$
    while $|V_T| \neq |V|$ do
        select $N \in V_T$ with $|N| > 1$ arbitrarily;
2       $G' \leftarrow$ the graph which can be created from $G$ by vertex condensations of the
            vertices in each $M \in V(T)$ with $M \neq N$;
        select $n_1, n_2 \in N$ arbitrarily;
3       determine a minimum $(n_1, n_2)$-cut $C$ in $G'$ and $A \subset V(G)$ such that $C = E(A, A^{\mathsf{c}})$;
4       $v_1 \leftarrow A \cap N$; $v_2 \leftarrow A^{\mathsf{c}} \cap N$;
        $V_T \leftarrow V_T \setminus \{N\} \cup \{v_1, v_2\}$;
        $E_T \leftarrow \{\{u, v\} \in E_T \mid u \neq N \wedge v \neq N\} \cup \{\{v_1, v_2\}\} \cup$
            $\{\{u, v_1\} \mid \{u, N\} \in E_T \wedge u \subset A\} \cup$
            $\{\{u, v_2\} \mid \{u, N\} \in E_T \wedge u \subset A^{\mathsf{c}}\}$;
        $\omega_T(e) \leftarrow \begin{cases} \omega(v_1, v_2) & \text{if } e = \{v_1, v_2\} \\ \omega_T(\{u, N\}) & \text{if } e = \{v_1, u\} \vee e = \{v_2, u\} \\ \omega_T(\{u, v\}) & \text{else} \end{cases}$
    return $(V_T, E_T)$ with $\omega_T$;
end
```

connecting edge with weight equal to the cut value of the minimum terminal cut. All edges $\{u, N\}$ in the tree are replaced by new edges incident to $\{u, v_1\}$ or $\{u, v_2\}$ with the old edge weight depending on whether $u \subset A$ or $u \subset A^{\mathsf{c}}$ holds. Note that due to the vertex condensations of step 2 only these two cases can occur. These operations ensure that in each iteration for every edge in the tree holds that it represents a minimum terminal cut with components composed of the vertices of $G$ which are on either sides of this edge with cut value equal to its weight. The algorithm is then guaranteed to terminate in $|V| - 1$ steps.

### 3.4.2 Gomory-Hu Tree Based Clustering

This section discusses how a Gomory-Hu tree can be used to divide a simple graph into several connected components and how the quality of the resulting partition can be characterized.

We know from section 3.4.1 that given a graph $G$ with $n$ vertices there are $n - 1$ different minimum terminal cuts such that for every possible pair $(s, t)$ of terminals there is a minimum $(s, t)$-cut among them. Additionally, these cuts have the property that they do not cross each other and can be represented by a Gomory-Hu tree $T_G$.

We can use these facts for clustering in the following way: The smallest of these minimum terminal cuts is also a minimum cut of $G$. If the edges of this cut are removed from $G$ the graph falls into two connected components. If we further remove the edges of the second smallest minimum terminal cut, the graph falls into exactly three connected components as all pairs of minimum terminal cuts are non-crossing. This idea can be further generalized to form a $k$-partition of the vertex set by removing the edges of $k-1$ minimum terminal cuts. If these cuts are also the $k-1$ smallest ones, the resulting partition is the optimal partition among all possible $k$-partitions in the sense that the largest *inter-component cut value* is minimal. The inter-component cut value is one way to characterize the pairwise similarity of the components:

**Definition 19** (COMPONENT CUT VALUE)**.** *Let $G$ be a connected simple graph with edge weight function $\omega$. For two connected components $S_1, S_2 \subset V(G)$ the* inter-component cut value between $S_1$ and $S_2$ *written $\lambda(S_1, S_2)$ is the cut value $\omega(C)$ of the cut $C = E(A, A^c)$ for $A \subset V(G)$ such that*

1 *$C$ separates $S_1$ from $S_2$, i.e. $S_1 \subseteq A$ and $S_2 \subseteq A^c$ and*

2 *the cut value of $C$ is minimal, i.e. for all other cuts $C'$ having property* 1 *holds*

$$\omega(C') \geq \omega(C).$$

Note that the inter-component cut value is equal to the inter-subgraph maximum flow in [WU and LEAHY 1993]. The later is not used here as it would require the definition of the maximum flow problem which is dual to the minimum terminal cut problem.

Now we have the following theorem which can be proofed with aid of a GOMORY-HU tree of the corresponding graph:

**Theorem 4.** *Let $G$ be a connected simple graph with edge weight function $\omega$ and $T_G$ its GOMORY-HU tree. The connected components $C_1, \ldots, C_k$ of $T_G$ after removing the $k-1$ smallest edges from $T_G$ form a $k$-partition on $G$ such that*

1 *each of the $k$ components is connected in $G$ and*

2 *the largest inter-component cut value, i.e.*

$$\max_{i<j} \{\lambda(C_i, C_j)\}$$

*is minimal among all possible $k$-partitions of $G$ and equal to the largest cut value of the $k-1$ cuts.*

*Proof (based on [WU and LEAHY 1993]).* First note that since $T_G$ is spanning tree of $G$ removing $k-1$ edges from the tree imposes a $k$-partition on the vertex set of $G$. Further note that for two of these components their inter-component cut value is equal to the minimum of the edge weights on the unique path in $T_G$ which connects the components. This follows from the properties of a GOMORY-HU tree given in definition 17. Hence the inter-component cut value between any pair of components cannot exceed the largest weight of the removed edges. Additionally, for those components being linked only by a single edge in $T_G$, the inter-component cut value must be equal to the weight of that edge. Hence, the largest inter-component cut value is equal to the largest weight of the $k-1$ removed edges, which is minimized when the $k-1$ edges with the smallest weight are removed. □

So theorem 4 suggest to use a GOMORY-HU tree for getting a partition into a desired number of components. However, instead of removing a fixed number of edges, we can also use a threshold which serves as a upper bound for all edges to remove. We know form theorem 4 that the threshold is then also an upper bound for the inter-component cut value. WU and LEAHY [1993] first suggested to use a GOMORY-HU tree in this way and we do as well. Finally, algorithm 3.4.2 describes the resulting clustering scheme in pseudo-code.

**Algorithm 3.4.2:** gomoryHuTreeClustering

```
Input: a connected simple graph $G$ with edge weight function $\omega$ and stop $\in \mathbb{R}$
Output: a map $c : V(G) \to \underline{k}$ for a $k \in \mathbb{N}$
begin
    compute the GOMORY-HU tree $T_G$ and its edge weight function $\omega_T$ using the
    gomoryHuTree algorithm;
    sort all edges of $T_G$ in increasing order;
    remove all edges from $T_G$ with $\omega_T(e) <$ stop;
    compute the connected components of $T_G$ and enumerate them arbitrary;
    $c \leftarrow$ a function mapping each vertex to the component it belongs to;
    return $c$;
end
```

**Time Complexity** The dominating operations of the clustering algorithm are the computation of the GOMORY-HU tree and the sorting step. The computation of a GOMORY-HU tree can be done with $\mathcal{O}(n \cdot N(n,m))$ operations where $N(n,m)$ is number of steps needed for computing a minimum terminal cut in a graph with $n$ vertices and $m$ edges. GOLDBERG's *preflow* algorithm needs $\mathcal{O}(n^2\sqrt{m})$ operations for this, see [GOLDBERG and TSIOUTSIOULIKLIS 2001] and references therein. The complexity of the sorting step is $\mathcal{O}(n \log n)$. As the number of edges is proportional to the number of vertices with a small factor in a sparse graph (cf. 2 in section 2.3.4) the total time complexity is $\mathcal{O}(n^{\frac{5}{2}} + n \log n)$.

**Implementation Notes** For the computation of the GOMORY-HU tree algorithm we used the graph library LEMON [JÜTTNER 2008] which contains an implementation of this algorithm. However, this implementation does not follow the algorithm of GOMORY and HU but instead uses the approach of GUSFIELD which has the same time complexity. GUSFIELD's algorithm does not use vertex condensation and is therefor much easier to implement. An description of GUSFIELD's algorithm along with an experimental study of the computational performance of both algorithms can be found in [GOLDBERG and TSIOUTSIOULIKLIS 2001].

### 3.4.3 Discussion

We already noted that some authors refereed to the work of Wu and Leahy [1993] as an example of recursively searching for minimum cuts in the sense of the segmentation scheme given in section 2.3.4. We give here a simple counter example to proof that this is actually not true. In figure 3.6 a graph $G$ with four vertices is shown; the numbers in the figure are the edge weights. The smallest and the second smallest minimum terminal cut are indicated with dotted lines. Removing the edges of these two cuts would result in the partition $\{\{v_1\}, \{v_2\}, \{v_3, v_4\}\}$ while the recursive scheme would result in $\{\{v_1\}, \{v_2, v_3\}, \{v_4\}\}$.

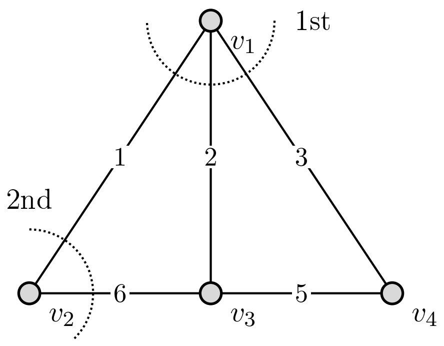


**Figure 3.6:** *A counter example which shows that the Gomory-Hu tree based clustering scheme is not equivalent to the recursive application of the minimum cut; see text for more details.*

## 3.5 Minimum Mean Cut

The last section introduced the minimum cut as a clustering tool for image segmentation. While the successive application of the minimum terminal cuts is effective computational by using a GOMORY-HU tree its bias to small clusters is problematic. A more sophisticated cost measure, that gives small costs to salient boundaries irrespective of their length would be desirable.

To overcome this situation, WANG and SISKIND [2003] proposed to normalize a cut via a second edge weight function:

**Definition 20** (MINIMUM MEAN CUT). *Let $G$ be a simple graph with two edge weight functions $\omega_1$ and $\omega_2$. All edge weights $\omega_1(e)$ are called* first edge weights *and all weights $\omega_2(e)$ are called* second edge weights *respectively. Then the* mean cut value *of a cut $E(A, A^{\mathsf{c}})$ in $G$ is defined as*

$$\mathsf{mcut}(A, A^{\mathsf{c}}) \triangleq \frac{\omega_1(A, A^{\mathsf{c}})}{\omega_2(A, A^{\mathsf{c}})} \tag{3.12}$$

*and each cut minimizing* (3.12) *is a* (minimum) mean cut *of $G$.*

The first edge weights encode some kind of inter-vertex affinity as it is the case with the minimum cut. For the choice of the second edge weights there are at least two possibilities: At first, one straightforward choice would be $\omega_2(e) = 1$ for all edges of $G$ in order to normalize the cut value by the number of edges in the cut, i.e. $|E(A, A^{\mathsf{c}})|$. Then, $\mathsf{mcut}(A, A^{\mathsf{c}})$ represents the average affinity of the common boundary of $A$ and $A^{\mathsf{c}}$. This should overcome the problem inherent to the minimum cut to penalize long boundaries. However, selecting the second edge weights this way ignores the geometric properties associated with each edge. Since each edge in the region adjacency graph derived from an initial segmentation also represents the common boundary of the associated regions, the corresponding boundary length should also be taken into account. Otherwise, boundaries with many short segments (see figure 3.7 for example) would be preferred instead of boundaries with long segments. So, in order to take into account the geometric interpretation of the graph structures, we use a function of the common boundary length for two neighboring regions in a partition of the plane as the second edge weight function, i.e. $\omega_2(e) = \mathsf{length}(e)$ for all edges (see section 3.3). Then $\mathsf{mcut}(A, A^{\mathsf{c}})$ represents the average affinity *per unit length* of the common boundary of $A$ and $A^{\mathsf{c}}$.

While the minimum cut problem is solvable in polynomial time things become more complicated with the minimum mean cut. Unfortunately, for simple graphs the problem is NP-hard in general. A proof can be found in [WANG and SISKIND 2003]. If the problem class is restricted to planar graphs and all cuts must be connected, the minimum mean cut problem can be solved in polynomial time for $\omega_2(e) = \mathsf{const}$ and in *pseudo-polynomial* time for positive second edge weights. Here, the term "pseudo-polynomial" means that the number of processing steps depends on the precision of the edge weights and is only bounded by a polynomial function if the edge weights are constrained to be integers.

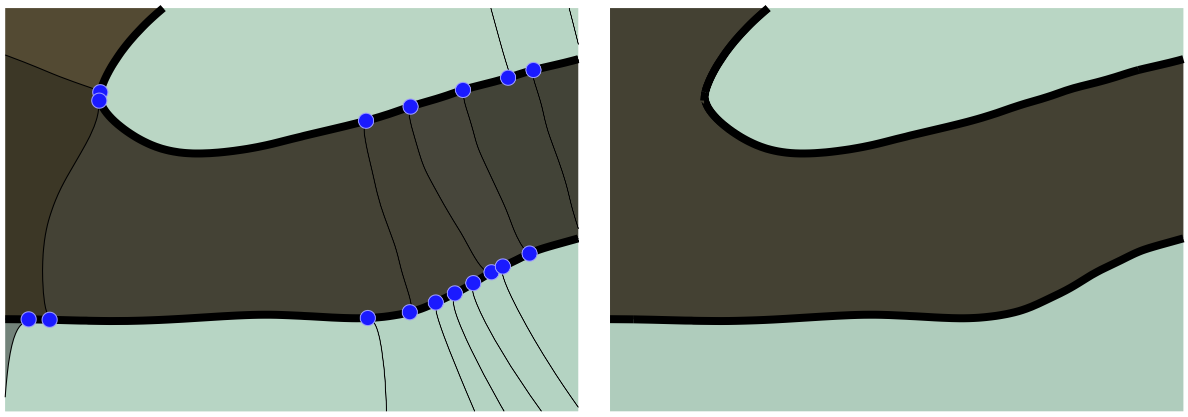


**Figure 3.7:** *The image shows the partition of the plane from a watershed segmentation, the faces (colored with mean of the underlying pixels) are enclosed by lines (edges of the partition of the plane). The boundary marked with thick lines consists out of several both long and very short segments. Each boundary segment should contribute to the cost of the whole boundary depending on its length.*

Before we give an algorithm for solving the minimum mean cut problem, we briefly want to discuss the constraints just mentioned. As far as the planarity constraint is concerned: Since all graphs encountered here are region adjacency graphs and thus planar, this restriction is not problematic. A very important question is if the minimum mean cut problem is still well defined with the connectedness constraint imposed. For this, we have to proof that there is always at least one connected cut among all minimum mean cuts of a graph:

**Lemma 2.** *For a connected graph $G$ there is always a minimum mean cut $E(A, A^{\mathsf{c}})$ such that both $A$ and $A^{\mathsf{c}}$ are connected components of the corresponding graph $G$.*

*Proof.* Assume there is a cut $E(A, A^{\mathsf{c}})$ where $A$ is not connected. We can always take the connected component of $A$ with minimum mean cut value among all other components to get a cut which is as least as costly as $A$. More precisely, let $A_1, \dots, A_m$ be the connected components of $A$. Then, we have

$$\mathsf{mcut}(A, A^{\mathsf{c}}) = \frac{\sum_{i=1}^{m} \omega_1(A_i, A_i^{\mathsf{c}})}{\sum_{i=1}^{m} \omega_2(A_i, A_i^{\mathsf{c}})} \tag{3.13}$$

and thus

$$\mathsf{mcut}(A, A^{\mathsf{c}}) \geq \min_{i \in \underline{m}} \frac{\omega_1(A_i, A_i^{\mathsf{c}})}{\omega_2(A_i, A_i^{\mathsf{c}})}. \tag{3.14}$$

Since $E(A, A^{\mathsf{c}})$ is a minimum mean cut, $E(A_k, A_k^{\mathsf{c}})$ is also a minimum mean cut if $k$ is the index which minimizes (3.14). The same arguments apply if $A^{\mathsf{c}}$ is not connected. If both $A$ and $A^{\mathsf{c}}$ are not connected one can use this method this first for $A$ and then for $A^{\mathsf{c}}$ in order to get a minimum mean cut. □

Additionally, the author thinks that connectedness requirement is not problematic if the minimum mean cut is used for clustering but rather a desirable feature, since both $A$ and $A^c$ then correspond to a spatially connected region which is a perceptually appealing bi-partition. One may argue that there might be situations where this constraint forbids desirable segmentations such as in figure 3.8: The region patches forming the connected component $A$ represent a salient segment. However, the cut $E(A, A_1 \cup A_2)$ is not connected. This is no problem since one can first find the cut $E(A, A_1)$ and afterward the cut $E(A, A_2)$.

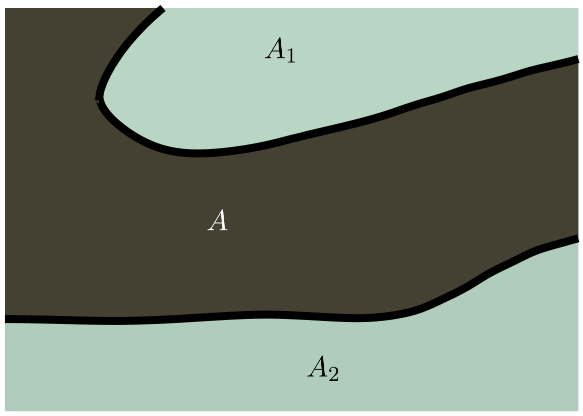


**Figure 3.8:** *The example of figure 3.7 with only the boundary between the components $A$, $A_1$, and $A_2$ drawn. The cut $E(A, A_1 \cup A_2)$ contains both boundary segments marked and is not connected.*

#### 3.5.0.1 Planar Graphs and Duals

We already mentioned planar graphs to be essential for the solution of the mean cut problem. Before we proceed with the discussion we would like to shed some light of the meaning of this concept. A planar graph is a graph which can be drawn such that its edges only intersect at their endpoints. The classical concept for formalizing this idea is the *embedding* of a graph. We already introduced the partition of the plane (definition 4 in section 2.2.1); we can use this for the definition of planar graphs:

**Definition 21** (PLANAR GRAPH)**.** *A simple graph $G = (V, E)$ is* planar *if and only if it can be* embedded *into the plane $\mathbb{R}^2$ which means that there is a partition of the plane $P = (V_P, E_P, F_P)$ such that:*

- *There is a bijection $f : V \rightarrow V_P$ with*
- *$\{u, v\} \in E$ if and only if there is an edge in $V_E$ which has $f(u)$ and $f(v)$ as its endpoints.*

*The partition of the plane $P$ is then also called the* embedding *of $G$.*

From the embedding of a planar graph $G$ one can derive its *dual graph* $G^*$. The construction is similar to the region adjacency graph (definition 11): For every face in the embedding we have exactly one vertex in $G^*$ and for every pair of faces sharing one edge in the embedding we have exactly one *dual* edge in $G^*$. This also means that $G^*$ cannot guaranteed to be simple if two faces share several edges. In this case $G^*$ has *multi-edges* which cannot be modeled with simple graphs. There can also be *loops* if one edge has the same face on both sides. For this we need the concept of *multi-graphs* which is not further specified here. However, this informal introduction to dual graphs should be sufficient for our needs. The graph $G$ is also called the *primal of* $G^*$ in this context. More details can also be found in [Even 1979]. An example for a dual graph shows figure 3.9. As a final note, as there can be several embeddings of a graph there can be also several duals; but this is no problem, as any dual will fulfill our needs.

## 3.5.1 A Pseudo-Polynomial Algorithm

We only present the more general pseudo-polynomial algorithm. The basic ideas and line of argumentation is based on [Wang and Siskind 2003] but with differences in use of terminology and style of presentation. The algorithm for constant second edge weights is very similar and can be found in [Wang and Siskind 2001].

The algorithm of Wang and Sisking consists of a series of reduction steps. The constraints given above allow a reformulation of the original problem such that it becomes solvable with standard techniques of graph theory.

Recall that the first constraint only admits planar graphs and the second requires every cut to be connected. These constraints allow us to establish a connection of connected cuts in a graph and *simple cycles* in its dual:

**Definition 22** (simple cycle). *For a graph $G$ a* simple cycle *is a set $C \subseteq E(G)$ of edges such that the vertices incident to the edges in $C$ are pairwise distinct and can be arranged as a series $(v_1, v_2, \dots, v_n)$ with $\{v_i, v_{i+1}\} \in C$ for $i \in \underline{n-1}$ and $\{v_n, v_1\} \in C$.*

The first key observation for approaching the minimum mean cut problem is then given with the following theorem:

**Theorem 5.** *For every simple planar graph there is an one to one correspondence between connected cuts and simple cycles in its duals.*

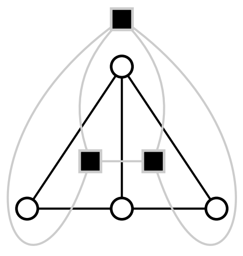
A proof of this theorem can be found in [Even 1979]. Instead we give a small example: Figure 3.9a shows an embedding of a simple graph together with the corresponding dual. Recall, the dual contains a dual edge for every edge in the original graph and the dual can have self loops and multi-edges. Figure 3.9b shows a connected cut and the corresponding simple cycle whereas figure 3.9c shows a none-connected cut and the corresponding none-simple cycle.

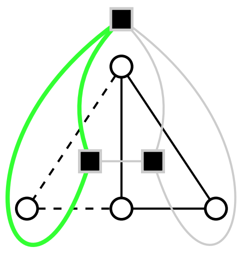
If $G^*$ is a dual graph of $G$ and $f$ is a function which takes every edge of $G$ to its corresponding dual edge, the *costs* of a simple cycle $C$ are defined as

$$\mathsf{mcyc}(C) \triangleq \mathsf{mcut}(f^{-1}(C)). \tag{3.15}$$

A cycle which minimizes equation (3.15) is then equivalent to a minimum mean cut in $G$. Therefore, instead of searching for a minimum mean cut in a simple graph we can turn to the problem of finding a *minimum mean cycle* in its dual:

**Lemma 3.** *For a simple planar graph $G$ every connected minimum mean cut has a corresponding simple minimum mean cycle in a dual of $G$ provided that the edges of $G$ have the same edge weights as their duals.*


(a)


(b)

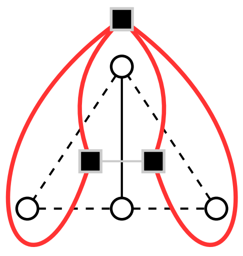

(c)

**Figure 3.9:** *(a): an embedding of a simple graph (round vertices) with a dual (rectangular vertices), note that the dual is not a simple graph (b): same construction with a* connected *cut in the primal graph (dashed edges) and the corresponding* simple *cycle (green bold edges) in the dual (c): the same construction with a* none-connected *cut in the primal graph (dashed edges) and the corresponding* none-simple *cycle (red bold edges) in the dual*

Now the question is how to find a minimum mean cycle in the dual of a graph. In general, a quite popular approach for solving an optimization problem of a fractional energy function such as (3.12), or equivalently (3.15), is to use a linearized versions of it and then to perform a search for an optimizing argument. More formally, we recast the problem into the question

$$\text{“}\exists C : \omega_1(C) - b \cdot \omega_2(C) < 0 \text{ ?''} \tag{3.16}$$

and do a search on the parameter $b$. If the answer to question (3.16) is “yes” we know that there is a cycle $C$ such that $\mathsf{mcyc}(C) < b$ and thus that the costs of the minimum mean cycle must be smaller than $b$. Likewise, if the answer is “no” we know that the costs of the minimum mean cycle must be greater than or equal to $b$. Now, one can gradually approach to the optimal value of $b$ by using a binary search technique. Of course, this method only works well if there is an effective algorithm which answers question (3.16) and if it is also possible to extract the corresponding cycle effectively. That is why we switch from the minimum mean cut problem to the minimum mean cycle problem as we can give a fast algorithm which solves the parameter question for cycles instead of cuts and also gives such a cycle if the answer is positive. Sometimes it is also possible to use the solution of one parameter to speed up the computation of a subsequent call to the algorithm for another parameter; some problems are even more tractable since they allow a linear instead of a binary search on the parameter. Both applies to the algorithm described in the following. At first we fix the idea of using a linear transformation with the next definition which together with lemma 4 provides the decisive part for using this method.

**Definition 23** (LINEAR TRANSFORMED WEIGHTED GRAPH)**.** *For a graph $G$ with first edge weight function $\omega_1$ and second edge weight function $\omega_2$ the term $G_b$ with* parameter $b \in \mathbb{R}$ *denotes the same graph but with the first edge weights replaced with*

$$\omega_b(e) = \omega_1(e) - b \cdot \omega_2(e), \forall e \in E(G).$$

Note that the ordering of simple cycles with respect to their mean cycle costs is invariant under this linear transformation, i.e. for two cycles $C_1, C_2 \subset E(G^*)$ with $\mathsf{mcyc}(C_1) \leq \mathsf{mcyc}(C_2)$ also holds in $G_b^*$

$$\mathsf{mcyc}(C_1) = \frac{\omega_b(C_1)}{\omega_2(C_1)} = \frac{\omega_1(C_1)}{\omega_2(C_1)} - b \quad \leq \quad \frac{\omega_1(C_2)}{\omega_2(C_2)} - b = \frac{\omega_b(C_2)}{\omega_2(C_2)} = \mathsf{mcyc}(C_2).$$

Thus every minimum mean cycle in $G_b^*$ is also a minimum mean cycle in $G^*$ and vice versa. This implies that if $b$ is set to the cost of a minimum mean cycle of $G^*$, say $C'$, all minimum mean cycles in $G_b^*$ have zero costs, i.e.

$$\mathsf{mcyc}(C') = \frac{\omega_b(C')}{\omega_2(C')} = \frac{\omega_1(C')}{\omega_2(C')} - b \overset{!}{=} 0 \qquad \text{for } b = \min_{C} \mathsf{mcyc}(C)$$

The other way round, if a minimum mean cycle in $G_b^*$ has zero costs then $b$ is equal to the cost of a minimum mean cycle in $G^*$.

**Lemma 4.** *A graph $G$ has a minimum mean cycle $C$ with cycle ratio $b$ if and only if $C$ is a minimum mean cycle of $G_b$ with zero cycle costs.*

So, due to lemma 4 instead of searching for a minimum mean cycle in $G^*$ it is sufficient to find a cycle with zero costs in $G_b^*$ if $b$ is selected properly.

It is both possible to test whether a graph $G_b^*$ has a negative cost cycle and to compute such a cycle in one go using an auxiliary graph derived from $G_b^*$ which is described in more detail in the next section. With such an algorithm together with lemma 4 it is possible to employ the search technique described above: Let $o$ be the optimal parameter – the costs of a minimum mean cycle of $G^*$ – if for any other parameter $b$ the graph $G_b^*$ has a cycle with negative costs we know $b > o$. Otherwise, if there is no such cycle we know $b \leq o$. If we select an initial guess on $b$ which is guaranteed to be greater than $o$, the algorithm for determining a negative cost cycle outputs such a cycle $C$. If $b$ is then set to $\mathsf{mcyc}(C)$ the cycle $C$ has zero costs in $G_b^*$ and a negative cost cycle $C'$ in $G_b^*$ has lower mean cost, i.e. $\mathsf{mcyc}(C') < \mathsf{mcyc}(C)$ . This way one gets a series of cycles with monotone decreasing mean costs until no further negative cost cycle can be detected. The last cycle computed this way is the wanted minimum mean cycle.

Since there is only a finite number of cycles in a finite graph this procedure must end. Unfortunately this method has exponential runtime for the number of cycles grows exponentially with the size of the graph. However, if all edge weights are constrained to take only on integer values, the number of iterations is bounded on the number of different possible parameters which is $\omega_1(E(G)) \cdot \omega_2(E(G))$. For the selection on the initial guess $b = \frac{\max_e \omega_1(e)}{\min_e \omega_2(e)}$ is sufficient. Putting all together, we finally arrive at algorithm 3.5.1.

**Algorithm 3.5.1:** minimumMeanCut

```
Input: a connected and planar graph $G$ and two functions $\omega_1, \omega_2 : E(G) \to \underline{n}$
Output: a minimum mean cut $C \subset E(G)$
begin
    compute a dual $G^*$ of $G$ and $f : E(G^*) \to E(G)$ with $e \mapsto f(e) = \mathsf{dual\ of}\ e$;
    $a \leftarrow \min \omega_2(e)$;
    $b \leftarrow \max \omega_1(e)$;
    $\omega'(e) \leftarrow a \cdot \omega_1(f(e)) - b \cdot \omega_2(f(e)), \forall e \in E(G^*)$;
    $C \leftarrow$ negativCostCycle($G^*$,$\omega'$);
    while $C \neq \emptyset$ do
        $C' \leftarrow C$;
        $a \leftarrow \omega_2(C)$;
        $b \leftarrow \omega_1(C)$;
        $\omega'(e) \leftarrow a \cdot \omega_1(f(e)) - b \cdot \omega_2(f(e)), \forall e \in E(G^*)$;
        $C \leftarrow$ negativCostCycle($G$,$\omega'$);
    return $f(C')$
end
```

### 3.5.2 Determining a Negative Cost Cycle

Now we only have to find an effective method for determining a negative cost cycle. It is possible to attack this problem by an additional reduction step. We construct a new graph from the dual of the initial graph and then use minimum cost perfect matchings in order to test whether there is a cycle with negative costs. If there is such a cycle the perfect matching can also be used for reconstructing it.

**Definition 24** (AUXILIARY GRAPH)**.** *For a simple graph $G$ with edge weight function $\omega$ the graph $\widehat{G}$ with edge weight function $\widehat{\omega}$ denotes its* auxiliary *graph with the following properties:*

- *For every $v \in V(G)$ there are two* real *vertices $v_1, v_2$ and one* virtual *edge $\{v_1, v_2\}$ in $\widehat{G}$ with $\widehat{w}(v_1, v_2) = 0$.*
- *For every edge $\{u, v\} \in E(G)$ there are two* virtual *vertices $u_v, v_u$ in $\widehat{G}$, one* real *edge $\{u_v, v_u\} \in E(\widehat{G})$ with $\widehat{\omega}(u_v, v_u) = 0$ and four edges $\{u_1, u_v\}, \{u_2, u_v\}, \{v_1, v_u\}, \{v_2, v_u\}$ all being mapped by $\widehat{\omega}$ to ${}^{\omega(u,v)}/_2$ and with $v_1, v_2$ being the real vertices of $v$ and $u_1, u_2$ being the real vertices of $u$.*

Figure 3.10 shows an example of such a construction. In the following the one to one correspondence between edges of a simple graph and real edges of its auxiliary graph is represented by the

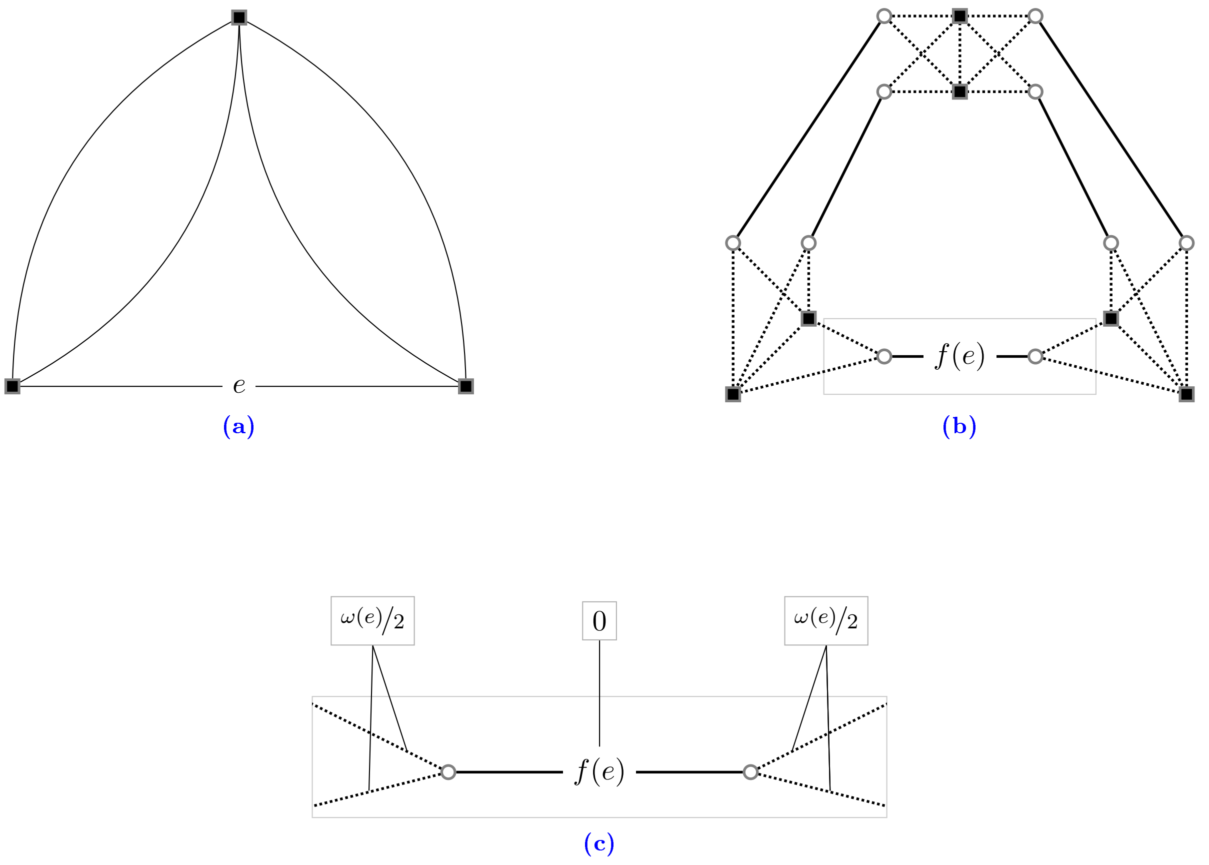


**Figure 3.10:** *(a): The dual graph of figure 3.9 slightly differently drawn (b): the corresponding auxiliary graph with real edges (solid), virtual edges (dotted), real vertices (black,rectangular) and virtual vertices (white,circular) (c): the division of the edge weights (for edge $f(e)$).*

function $f$. In order to proceed the concept of "perfect matchings" is needed:

**Definition 25** (PERFECT MATCHING). *A* perfect matching *of a graph $G$ with edge weight function $\omega$ is a set $M \subseteq E(G)$ of edges for which every vertex in $G$ is incident to exactly one edge in $M$. A perfect matching is a* minimum perfect matching *if it has minimal edge costs, i.e. for every other perfect matching $M'$ it is $\omega(M) \leq \omega(M')$.*

For a graph $G$ its auxiliary graph is especially designed such that a perfect matching $M$ of it can be used to derive a set of cycles in $G$ which has the same total edge costs than the costs of the edges in $M$. One can form such a set by taking each edge of $G$ whose associated real edge in $\widehat{G}$ is **not** contained in the perfect matching $M$, i.e.

$$\mathcal{C} \triangleq \{e \in E(G) \,|\, f(e) \notin M\}.$$

Now, the subgraph of $G$ induced by the set $\mathcal{C}$ only contains two-degree nodes. Instead of a rigid proof of this proposition, we only describe the underlying idea of the auxiliary graph construction using a small working example. From this a proof can then easily be derived.

Consider the small graph in figure 3.11. Following the definition of an auxiliary graph the node $r$ has two corresponding real nodes $r_1$ and $r_2$ in $\widehat{G}$ which are connected through a virtual edge $e$. If this edge is in the perfect matching $M$, as in figure 3.11 b), none of the remaining virtual edges incident to $r_1$ and $r_2$ can be in $M$. So, in order to form a valid perfect matching all real edges in $\widehat{G}$ which are connected to $r_1$ and $r_2$ through a virtual edge also have to be in $M$ ($e_1$ and $e_2$ in figure 3.11 b). The other way round, if the edge $e$ is not in $M$ there must be exactly two virtual edges, one for $r_1$ and one $r_2$, in $M$ ($v_1$ and $v_2$ in figure 3.11 c). These two virtual edges are each incident to a real edge which therefore cannot be in $M$. Putting all together, for every pair of real nodes there are either exactly two of its associated real edges not in $M$ or all.

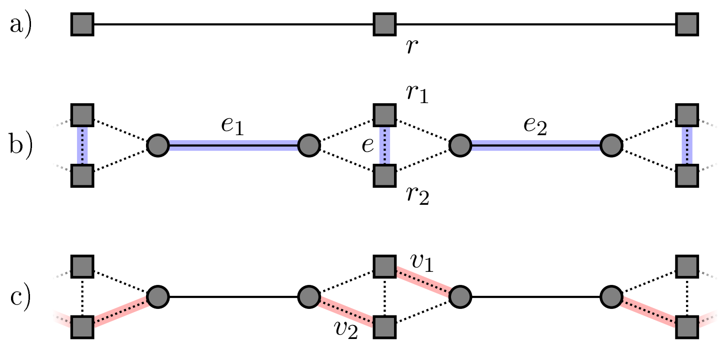


Figure 3.11: *A small graph (first), its auxiliary graph with two different perfect mathings marked*

So, if $\mathcal{C}$ only contains two-degree nodes it must be an union of simple cycles. Moreover, since there are exactly two virtual edges in a perfect matching for each real edge not in $M$ and only those edges have none-zero edge costs in $\widehat{G}$ it follows $\widehat{\omega}(M) = \omega(\mathcal{C})$ which in turn implies that if

$\widehat{G}$ has a perfect matching with negative costs one of the cycles in the corresponding set $\mathcal{C}$ must be a simple cycle with negative costs. A more complex example of a cycle together with its perfect matching is shown in figure 3.12.

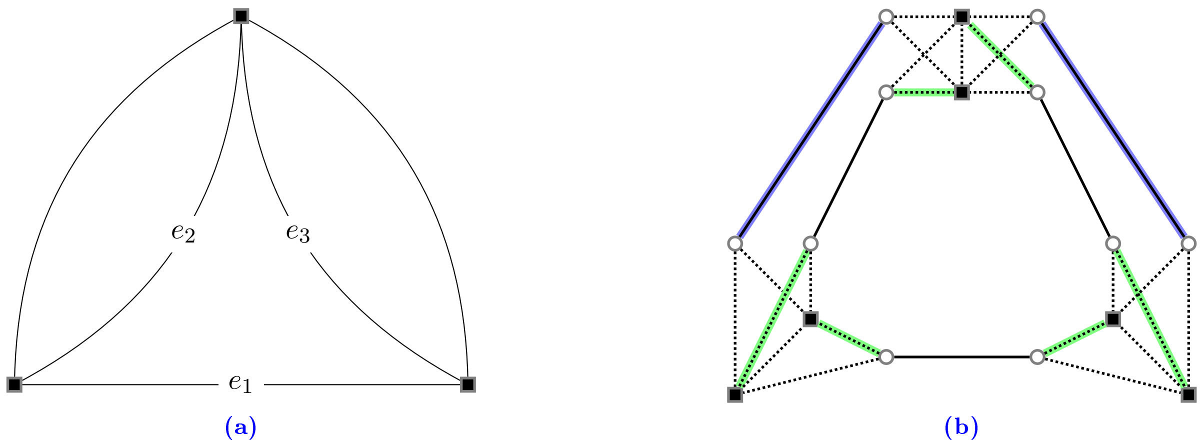


**Figure 3.12:** *(a): the dual graph from figure 3.9 with cycle $C = \{e_1, e_2, e_3\}$ (b): the corresponding auxiliary graph with a perfect matching: green and blue edges are in the matching where green edges (virtual edges) form the cycle $C$ and blue edges (real edges) do not correspond to a cycle.*

Conversely, for a simple cycle in a graph it is always possible to construct a perfect matching in the corresponding auxiliary graph with the same total edge costs. The following lemma subsumes the last observations:

**Lemma 5.** *A simple graph has a negative cost cycle if and only if its auxiliary graph has a perfect matching with negative costs.*

We finally arrive at the question whether an auxiliary graph has a perfect matching with negative costs. This can be easily answered by computing a minimum cost perfect matching which is a standard problem in graph theory and for which effective algorithms exist, see [COOK and ROHE 1999] for an overview. Algorithm 3.5.2 then gives the steps to take for finding a negative cost cycle.

**Time Complexity** The computation of minimum perfect matchings can be done in $\mathcal{O}(|V| \cdot |E| \log |V|)$ [GALIL et al. 1986]. The overall performance of the minimum mean cut algorithm is dominated by these computations. WANG and SISKIND already reported that they experienced only a small number of iterations needed to find a minimum mean cycle. In order to investigate the convergence behavior of this method for our segmentation scheme, we clustered all subgraph sizes encounter during recursive application of the algorithm on one hundred images using the segmentation scheme described in section 2.3.4, with the watershed method for the initial segmentation, the function colorMean (section 3.3) for the edge weights, and the optimal parameters found by parameter search as described in section 4.3.1. The mean number of iterations has been

**Algorithm 3.5.2:** negativeCostCycle

```
Input: a connected and planar graph $G$ and a functions $\omega : E(G) \to \underline{n}$
Output: a simple cycle $C \subset E(G)$ with negative costs, $\emptyset$ if no such cycle exists
begin
    build the auxiliary graph $\widehat{G}$ with edge weights $\widehat{\omega}$ from $G$ and $\omega$;
    compute a minimum cost perfect matching $M$ from $\widehat{G}, \widehat{\omega}$;
    if $\widehat{\omega}(M) = 0$ then
        return $\emptyset$
    $\mathcal{C} \leftarrow \{e \in E(G) \mid f(e) \notin M\}$;
    compute $\mathcal{S}$ the set of connected components of $\mathcal{C}$;
    return $\arg\min_{C \in \mathcal{S}} w(C)$
end
```

assigned to each cluster and the results can be see in figure 3.13. Indeed, as one can see the number of iterations needed is very small and does not grow with the size of the graphs.

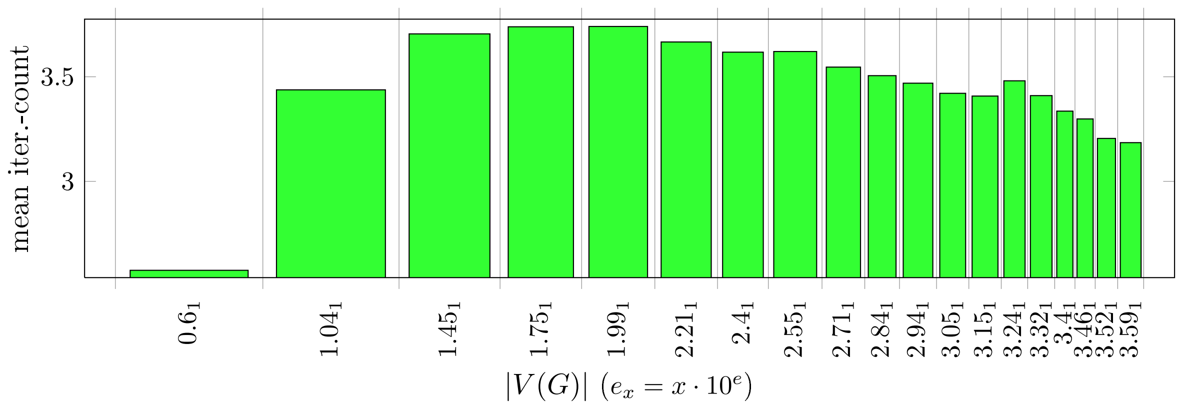


**Figure 3.13:** *This figure shows the dependence between number of nodes in the graph and number of iterations used by the algorithm for the minimum mean cut; the scaling for x-axis is logarithmic and for the numbers we have $e_x = x \cdot 10^e$; see text for the data generation.*

**Implementation Notes** The algorithm was implemented using LEMON a C++ open source graph library [Jüttner 2008]. The crucial routines used from LEMON for this include routines for computing connected components, planar embeddings and maximum perfect matchings. All edge weights are transformed to the range $[1, 1200]$ and rounded to the nearest integers.

## 3.6 Foreground Cut

This section introduces the *foreground cut* for another way of normalizing the minimum cut. Whereas the minimum mean cut uses a second weight function defined on the edges, the foreground cut takes another approach for normalizing the minimum cut; a vertex weight function is used for this purpose:

**Definition 26** (Foreground Cut)**.** *For a graph $G$ with edge weight function $\omega$ and vertex weight function $\gamma$ the* foreground cut value *of a cut $E(A, A^\mathsf{c})$ is defined as*

$$\mathsf{fcut}(A, A^\mathsf{c}) \triangleq \frac{\omega(A, A^\mathsf{c})}{\gamma(A)}. \tag{3.17}$$

*A cut minimizing* (3.17) *with $A^\mathsf{c} \neq \emptyset$ is then a* (minimum) foreground cut*.*

A foreground cut thus takes into account the vertices contained in *one* of the two components and can be considered as an asymmetric version of the isoperimetric cut (section 3.7) or the normalized cut (section 3.8) which use *both* components of the cut. An interesting question is how this asymmetric nature of the foreground cut impacts on the segmentation quality. We expect it to have a bias to large regions such that the denominator in equation (3.17) is maximized. A more detailed discussion on this aspect is given in section 4.4.

The component $A$ which is favored this way is called *foreground* (*component*) and analogously $A^\mathsf{c}$ is the *background* (*component*). For the background we have the following property:

**Lemma 6.** *If $E(A, A^\mathsf{c})$ is a foreground cut in a connected graph, then $A^\mathsf{c}$ is also connected.*

*Proof.* Suppose that for a minimal foreground cut $C = E(A, A^\mathsf{c})$ the component $A^\mathsf{c}$ is not connected and let $A^\mathsf{c}_1, \cdots, A^\mathsf{c}_k$ be its connected components. Since all weights are positive, we have

$$\begin{aligned} \mathsf{fcut}(A, A^\mathsf{c}) &= \frac{\sum_{i=1}^k \omega(A, A^\mathsf{c}_i)}{\gamma(A)} \\ &\geq \frac{\omega(A, A^\mathsf{c}_j)}{\gamma(A) + \sum_{i \neq j} \gamma(A^\mathsf{c}_i)} \end{aligned}$$

for any $j \in \underline{k}$ which means that $C$ cannot be minimal. □

Similarly to the minimum mean cut, there are two straightforward choices for the vertex weight function. At first, one could simply select $\gamma(v) = 1$ for all vertices. Then the weights of the edges in the cut are normalized by the number of vertices in the foreground. However, this selection would not attend to the different features of the underlying image patches. Their geometric properties can be incorporated via the vertex function $\gamma(v) = \mathsf{area}(v)$ such that the total weight of the boundary is normalized by the enclosed area, but it would be more desirable to respect the underlying image statistics of the enclosed area. An attempt for this is to use the *weighted* *degree* which is the total weight of the incident edges, i.e $\gamma(v) \triangleq \omega(\{v\}, V)$. Regions with a high total weighted degree have then a high inter region similarity measured in terms of the edge weights. Choosing the vertex weights this way has already been proposed in [Grady and Schwartz 2006*a*; Shi and Malik 2000] and here we do likewise.

The foreground cut was already introduced in [Perona and Freeman 1998] within the context of spectral clustering. Therein, some insights of the relation between the spectrum of a graph and equation (3.17) are given. The connection of the foreground measure to the spectrum of a graph can be used to derive an approximation technique in the spirit of the methods described in section 3.8. However, the theory around these methods often lacks of a clear statement about upper approximation bounds and such: If no exact solution to a problem is known, and one has to resort to an approximation scheme instead, it would be desirable to have at least evidence about how bad the approximating solution can be in worst case. Fortunately, the minimum foreground problem can be efficiently solved with *parametric network flows.* The simple algorithm presented here was motivated by an email correspondence with Satish Rao [4] who pointed the author to the basic graph construction which includes the derivation of equation (3.21). The detailed elaboration presented in the following and the idea of using parametric network flows is due to the author of this thesis.

### 3.6.1 A Polynomial Algorithm

For the solution of the foreground cut problem an auxiliary graph can be employed which in this case is a rather simple extension of the original graph. For this, we simply take the initial graph of interest and add two different kind of vertices together with some edges: At first we have to determine a set of *seeds* among the set of the original vertices. The exact nature of these seeds is described below. Then we add a new node to the graph which is called the *sink.* Each seed is connected to the sink via a *sink edge* with edge weight of infinity. Moreover, all vertices except the sink are connected to another new vertex – the *source* – via *source edges.* For each vertex $v$, the corresponding newly added source edge has an edge weight equal to $\gamma(v) \cdot \lambda$ where $\lambda$ is the *parameter* of the graph and $\gamma$ is the vertex weight function of the original graph. The parameter $\lambda$ is crucial for the solution of the foreground cut problem. The following definition characterizes such an auxiliary graph in rigorous terms:

**Definition 27** (auxiliary graph)**.** *An* auxiliary graph $G_\lambda$ *with the* parameter $\lambda$*, a set* $T \subset V(G)$ *of* seeds*, and an edge weight function* $\omega_\lambda$ *of a simple graph* $G$ *with edge weight function* $\omega$ *and vertex weight function* $\gamma$ *is a graph with*

- $V(G_\lambda) \triangleq V(G) \cup \{s,t\}$ *with* source *and* sink $s, t \notin V(G)$*,*
- $E(G_\lambda) \triangleq E(G) \cup E_s(G_\lambda) \cup E_t(G_\lambda)$ *with*
  - $E_s(G_\lambda) \triangleq \{\{s,v\} \mid v \in V(G)\}$ *a set of* source edges *and*

[4] Satish Rao is professor at the University of California, Berkeley

- $E_t(G_\lambda) \triangleq \{\{t, v\} \mid v \in T\}$ *a set of* sink edges

- *and*

$$w_\lambda(u,v) \triangleq \begin{cases} \lambda \cdot \gamma(v) & \text{if} \quad \{u,v\} \in E_s \\ \omega(u,v) & \text{if} \quad \{u,v\} \in E(G) \\ \infty & \text{if} \quad \{u,v\} \in E_t \end{cases}$$

An example of such an auxiliary graph can be seen in figure 3.14 for a graph with three vertices. Note that the structure of the auxiliary graph is similar to the graphs used for the minimization

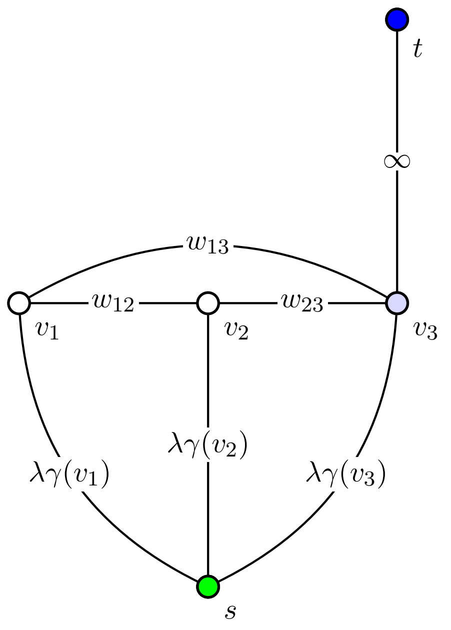


**Figure 3.14:** *The auxiliary graph for the complete graph with vertex set $\{v_1, v_2, v_3\}$. All edges are marked with their edge weight in the auxiliary graph. Edges connected to $s$ are source edges and edges connected to $t$ are sink edges.*

of energy functions used by Komologorov and Boykev (see section 3.2 along with equation (3.1)). In fact, these energy functions are quite similar to the costs of a minimum terminal cut in an auxiliary graph. More precisely, if $E(A, A^c)$ is a terminal cut in an auxiliary graph $G_\lambda$ with terminal $s$ and $t$, for the corresponding cut value holds

$$\omega_\lambda(A, A^c) = \omega_\lambda(\{s\}, S^c) + \omega_\lambda(S, S^c) + \omega_\lambda(\{t\}, S), \tag{3.18}$$

where $S$ is the source set, i.e. $S = A - \{s\}$, and $S^c$ is in the following always the complement of $S$ with respect to $V(G)$, i.e. $S^c = G_\lambda - S - \{t\}$. Recall from the definition of a terminal cut, that it has to divide both terminals from each other. In order to achieve this, it can directly cut some source edges, some edges from the original graph, and directly cut some sink edges which respectively corresponds to the first, second, and third term in the sum of equation (3.18). Since all sink edges have weights of infinity the last term in (3.18) is de facto zero if we are seeking for a terminal cut with minimal cut value. Moreover, the set of possible minimum cuts depends on the parameter $\lambda$. Irrespective of the actual selection of $\lambda$, all minimum $(s,t)$-cuts obey

$$\omega_\lambda(\{s\}, S^{\mathsf{c}}) + \omega_\lambda(S, S^{\mathsf{c}}) \leq \omega_\lambda(\{s\}, G(V)) \tag{3.19}$$

with $S$ being again the source set as above. Due to the special selection of the edge weights of an auxiliary graph the inequality (3.22) can also be written in the form

$$\lambda \cdot \gamma(S^{\mathsf{c}}) + \omega(S, S^{\mathsf{c}}) \leq \lambda \cdot \gamma(V). \tag{3.20}$$

Since $\gamma(V) - \gamma(S^{\mathsf{c}}) = \gamma(S)$ and if $S \neq \emptyset$ we finally get

$$\omega(S, S^{\mathsf{c}}) - \lambda \cdot \gamma(S) \leq 0 \tag{3.21}$$

and equivalently

$$\frac{\omega(S, S^{\mathsf{c}})}{\gamma(S)} \leq \lambda. \tag{3.22}$$

So, if $\lambda = \mathsf{min}_A\, \mathsf{fcut}(A)$ a minimum $(s,t)$-cut in $G_\lambda$ is also a minimum foreground cut in $G$. Further note that (3.21) is the foreground cut version of "the question" of section 3.5 used for an optimal parameter search for the minimum mean cut problem; likewise, we can perform a search on the parameter $\lambda$: If it is selected "too small" a minimal $(s,t)$-cut simply cuts all source edges and thus yields the trivial cut with $S = \emptyset$ and $\omega(S, S^{\mathsf{c}}) = \gamma(S) = 0$ and we know $\lambda < \mathsf{fcut}(G)$. That means, in order to solve the minimum foreground cut problem we must search for the first parameter which gives a nontrivial cut which in turn can be done with binary search and requires the solution of the minimal terminal cut problem for each selection on $\lambda$.

However, similar to the minimum mean cut problem a more effective search technique is possible: one can show that the optimal parameter can be found with at most $|V(G)|$ iterations. Even better, as the problem of finding a minimum $(s,t)$-cut in $G_\lambda$ is an instance of the *parametric maximum flow problem*, the optimal value can be found with an algorithm which has the same time complexity as a single invocation to the minimum terminal cut algorithm if the preflow algorithm of GALLO et al. [1989] is used. How this works is described in the next section.

### 3.6.2 Optimal Parameter Search

The key observation for an effective parameter search technique is that minimum terminal cuts in an auxiliary graph with terminals selected appropriately are *nested* in the following sense:

**Lemma 7.** *Let $G$ be a simple graph and $G_{\lambda_1}$ and $G_{\lambda_2}$ two auxiliary graphs of $G$ with $\lambda_1 \geq \lambda_2$ and with the same seed set. Let further $s, t$ be the source and the sink of the auxiliary graphs*

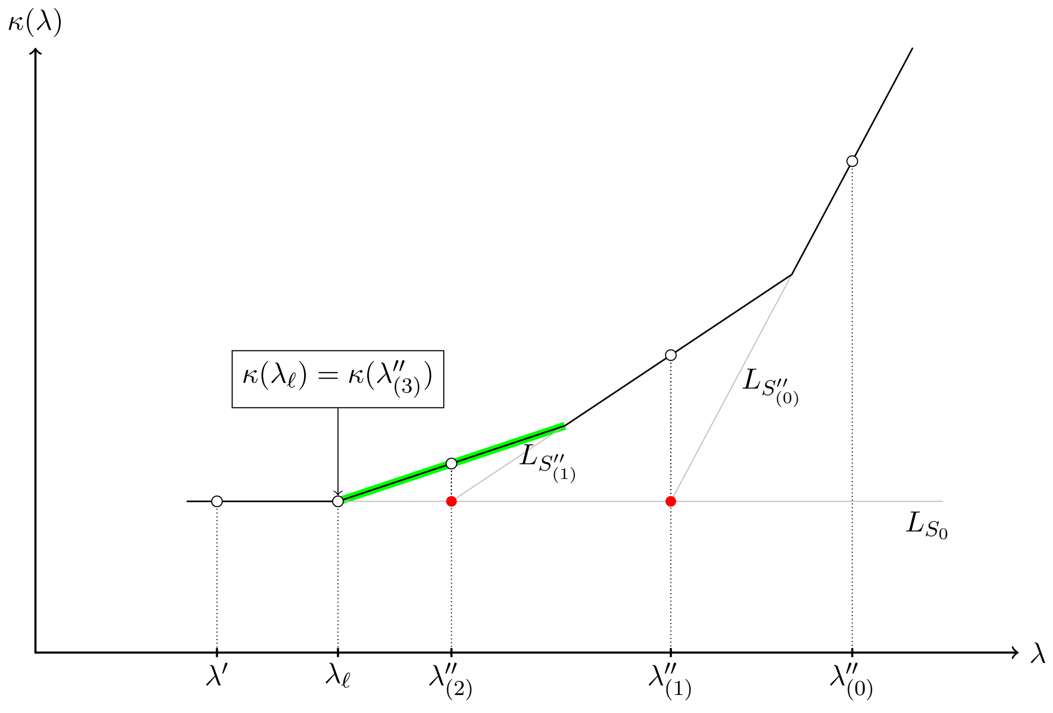


**Figure 3.15:** *The modified newton method for determining the first breakpoint – here $\lambda_\ell$ – of $\kappa$ in the range $[\lambda', \lambda'']$ . The computation starts with $\lambda''_{(0)} = \lambda''$ . The first intersection point of the corresponding line segments $L_{S_0}$ and $L_{S''_{(0)}}$ is $\lambda''_{(1)}$. The first linesegment after $L_{S_0}$, framed green, is the linesegment of the foreground cut.*

*according to definition 27. If $S_1$ is the source set of a minimum $(s,t)$-cut in $G_{\lambda_1}$ and $S_2$ is the source set of a minimum $(s,t)$-cut in $G_{\lambda_2}$ then also holds $S_1 \subseteq S_2$.*

*Proof.* Due to the construction of an auxiliary graph, all source edges in $G_{\lambda_2}$ have edge weights smaller than or equal to all weights of the source edges in $G_{\lambda_1}$. Therefore, a source edge which is in a minimum terminal cut of $G_{\lambda_1}$ must also be in $G_{\lambda_2}$. $\square$

That means that there is a maximal sequence $\lambda_1 > \lambda_2 > \cdots > \lambda_\ell$ of monotone increasing parameters where the source set of the corresponding minimum terminal cuts of $G_{\lambda_i}$ strictly expands, i.e. one gets a sequence $S_1 \subset S_2 \subset \cdots \subset S_\ell$ of strictly nested source sets. If $\kappa(\lambda)$ denotes the cut value of a minimum $(s,t)$-cut in $G_\lambda$ as a function of $\lambda$, it follows that $\kappa$ is a monotone piecewise linear function with at most $\ell \leq |V(G)|$ discontinouties. These discontinouties are also called *breakpoints*.

Now recall that we are interested in the smallest $\lambda$ for which the left side of (3.21) is nonzero. The desired parameter is $\lambda_\ell$, i.e. the smallest breakpoint of $\kappa$. It can be determined with a modified newton method which we adapt from [Gallo et al. 1989]. Since $\kappa$ is a piecewise linear function, it can be represented as composition of line segments, the intersection point of these line segments being the breakpoints of $\kappa$. Each line segment is given through

$$L_{S_i}(\lambda) \triangleq \omega(S_i, S_i^{\mathsf{c}}) + \lambda \cdot \gamma(S_i^{\mathsf{c}}),$$

if $S_i$ is the source set of the minimum $(s,t)$-cut in $G_\lambda$ for $\lambda \in (\lambda_i, \lambda_{i+1}]$. The first breakpoint within a given range $\lambda \in [\lambda', \lambda'']$ can be found by first determining the slope at $\kappa(\lambda')$ and $\kappa(\lambda'')$ which requires the computation of two minimum terminal cuts. Then we replace $\lambda''$ with the point of intersection of the two line segments. If then $\kappa(\lambda'') = L_{S_i}(\lambda'')$ we know that $\lambda''$ must be the first breakpoint after $\lambda'$ otherwise we go on with this procedure until we are done, see figure 3.15 for an example. The computation of the exact value of the breakpoint may be problematic in practice due to precision issues. However, this is not really a problem as we are mainly interested in the cut itself and not in the exact cut value (which can be easily computed if we have the optimal cut anyway). Therefore, it is sufficient to know if we are on the first line segment after the line segment of $\lambda'$ which is the case if the computed cut for the intersection point is the same as the cut for $\lambda'$ or $\lambda''$.

For the final algorithm, we have to select an appropriate interval which is guaranteed to contain the optimal parameter and we have to determine at least one seed vertex. For the interval we select

$$\frac{\mathsf{min}_{v \in V(G)} \{\omega(v)\}}{\gamma(V(G))}$$

as the lower bound and

$$\frac{\omega(V(G))}{\mathsf{min}_{v \in V(G)} \{\gamma(v)\}}$$

for the upper bound. The seed selection is discussed in the next section.

**Algorithm 3.6.1:** minForegroundCut

```
Input: a connected simple graph $G$ with edge weight function $\omega$ and vertex weight
       function $\gamma$
Output: a minimum foreground cut
begin
    select $\lambda_0, \lambda_1$ such that $\lambda_0 \leq \lambda_\ell \leq \lambda_1$;
    $v_s \leftarrow$ vertex with maximal weighted degree, i.e. $\max\{\omega(\{v\}, V(G))\}$;
    compute the auxiliary graph $G_{\lambda_1}$ with $v_s$ as seed;
    $S_1 \leftarrow$ source set of a minimum $(s,t)$-cut of $G_{\lambda_1}$/* $s$ and $t$ denote the source and the
    sink of $G_\lambda$ according to definition 27 */;
    $\alpha \leftarrow \omega(S_1, S_1^c)$;
    $\beta \leftarrow \gamma(S_1)$;
    $\lambda' \leftarrow \lambda_1$;
    repeat
        $\lambda \leftarrow \lambda'$;
        update edge weights of $G_\lambda$;
        $S \leftarrow$ source set of a minimum $(s,t)$-cut of $G_\lambda$;
        $\lambda' \leftarrow (\alpha - \omega(S, S^c)) \cdot (\beta - \gamma(S))^{-1}$    /* intersection point of $L_{S_i}$ and $L_{S_\ell}$ */;
    until $L_{S_0}(\lambda) = L_S(\lambda)$ ;
    return $E(S, S^c)$
end
```

### 3.6.3 Seed Selection

For the solution of the foreground cut problem with the method just described it is necessary to automatically select at least one seed for the auxiliary graph. Alternatively, one could also select seeds at hand which would correspond to an interactive segmentation method but this is not further examined here. Graph cut segmentation methods which require the selection of one seed have already been described in the literature: Cox et al. [1996] solved the foreground cut problem in planar graphs by searching for a closed contour in the dual of a pixel grid graph. Their algorithm requires the selection of an edge which is guaranteed to be in this contour. For this they select the edge with the smallest weight for the seed; the idea is that this edge is likely to be in the cut. Grady and Schwartz [2006$a$] proposed an approximation scheme for the isoperimetric cut which is described in detail in section 3.7. As with the foreground cut method used here their algorithm also requires the selection of a seed which is guaranteed to be in the background. For this they choose the vertex with maximal weighted degree, i.e. $\max\{\omega(\{v\}, V(G))\}$. Their idea is that a seed selected in this was is likely not to be incident to an edge in the cut. A result of both the work of Cox et al. [1996] and Grady and Schwartz [2006$a$] is that selecting the seed based on the local features encoded by the edge weights is a good heuristic for seed selection. So, we also choose the vertex with maximal weighted degree as the seed. The resulting algorithm 3.6.1 then solves the minimum foreground cut problem with the additional constrain that the background of the optimal cut contains the selected seed.

As a final note, it is also possible to use all vertices as seed in turn and select the cut with minimal foreground cut value as the final result. The resulting cut is then the optimal foreground cut. However, this would also greatly increase the computation time and is therefore not used here. Nevertheless, the impact of the seed selection on the final segmentation can be further investigated in this way and could be the object of future work.

**Time Complexity** The most expensive part in computing the foreground cut is the computation of the minimum terminal cuts. The preflow algorithm of Gallo et al. [1989] can be used for this. It runs with time $\mathcal{O}(nm \log n^2/m)$ with $n$ being the number of nodes and $m$ being the number of edges in the graph. As the number of edges is proportional to the number of vertices with a small factor in a sparse graph (cf. 2 in section 2.3.4) the total time complexity is $\mathcal{O}(n^2 \log n)$. Moreover, using reoptimization techniques Gallo et al. showed that all breakpoints can be computed by $\mathcal{O}(1)$ invocations of this algorithm. We have not implemented these optimization techniques due to time constraints. However, we have found that for our experiments the actual number of iterations needed only scales very slightly with the number of nodes in the graph, see figure 3.16 for comparison.

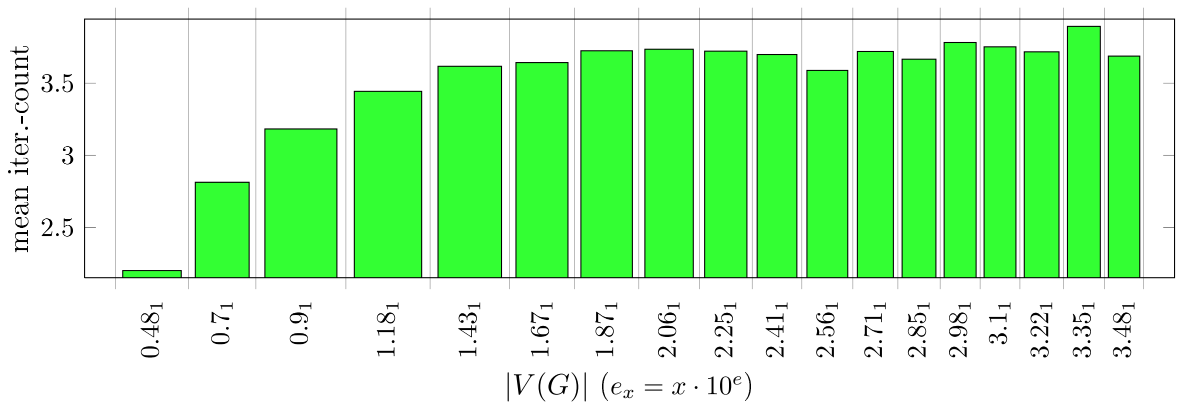


**Figure 3.16:** *This figure shows the dependence between number of nodes in the graph and number of iterations for the foreground cut algorithm method, the scaling for x-axis is logarithmic and for the numbers we have $e_x = e \cdot 10^x$; for data generation same method as described on page 57 has been used.*

**Implementation Notes** For the computation of the minimum terminal cuts we also used the open source graph library LEMON [Jüttner 2008].

## 3.7 Isoperimetric Cut

In this section we want to propose the *isoperimetric cut* of a graph as a candidate for our segmentation tool. Computing it is a classical problem of extremal graph theory [Chung 1997]. Isoperimetric problems, in general, also have a long history in the field of geometry. Originally, it describes the problem of determining a closed curve with fixed length which encloses a maximal area [Chung 1997]. In graph theory, we are faced with the same problem if volume and perimeter are defined properly. In general, all optimization problems concerned with the problem of optimizing some kind of relation between perimeter and volume can be regarded as isoperimetric problems. The foreground cut (section 3.6) and the normalized cut problem (section 3.8), for example, could thus also be put into this category. However, the term often means the problem of determining the *isoperimetric number* of a graph respectively the corresponding *isoperimetric cut*:

**Definition 28** (isoperimetric cut). *For a simple graph $G$ with edge weight function $\omega$ and vertex weight function $\gamma$ a cut $E(A, A^c)$ for which the term*

$$\mathsf{icut}(A, A^c) \triangleq \frac{\omega(A, A^c)}{\min\{\gamma(A), \gamma(A^c)\}}. \tag{3.23}$$

*is minimal is called* (minimum) isoperimetric cut (of $G$) *and the cut value with respect to this function, i.e.* $\mathsf{icut}(A, A^c)$, *is called the* isoperimetric number *of $G$.*

The cut measure icut is thus a balanced version of the foreground cut (section 3.6) which only uses the term $\gamma(A)$ for normalization. For the selection of the vertex weights we have the same options as with the foreground cut. Again, we choose the weighted degree which is also used in [Grady and Schwartz 2006*a*].

The isoperimetric cut possesses an even stronger connectedness property than the foreground cut (for a proof see [Grady and Schwartz 2006*a*] and references therein):

**Theorem 6.** *If $E(A, A^c)$ is an isoperimetric cut of a simple graph then holds that both $A$ and $A^c$ are connected.*

Unfortunately, computing a minimum isoperimetric cut is NP-hard in general (see [Grady and Schwartz 2006*a*] and references therein). The problem becomes more tractable if the class of considered graphs is constrained to the class of planar graphs. The currently best known algorithms for determining the isoperimetric cut within a planar graph is due to Park and Phillips [1993]. They have developed two exact algorithms which run in $\mathcal{O}(n^3W)$ and $\mathcal{O}(n^2W \cdot \log nW)$ time respectively where $W \triangleq \gamma(V(G))$ is the total vertex weight and $n$ is the number of vertices in the graph. Hence, both algorithm have exponential time complexity if the vertex weights are given in binary form and are therefore *pseudo-polynomial.* We think that the run-time performance of these algorithms is too slow for graph structures encountered here. Instead we use an approximation scheme which has recently been published by Grady and SCHWARTZ [2006*a*]. An interesting question for further work would be to investigate how close the method of GRADY and SCHWARTZ is to the optimal solution.

### Linear Approximation

GRADY and SCHWARTZ [2006*a*,*b*] proposed a new, simple approximation technique for finding isoperimetric cuts in simple graphs. It only requires the solution of a system of linear equations with an additional sorting step.

For a simple graph $G$ with vertex set $V \triangleq \{v_1, \dots, v_n\}$, a cut induced by $S \subseteq V$ can be represented with an *indicator vector* $\boldsymbol{x}$ with

$$x_i \triangleq \begin{cases} 1 & \text{if} \quad v_i \in S \\ 0 & \text{else} \end{cases}$$

Then, the total costs of the edges in the cut $E(S, S^{\mathsf{c}})$ can be written

$$\omega(S, S^{\mathsf{c}}) = \boldsymbol{x}^{\mathsf{T}} L \boldsymbol{x}$$

if $\boldsymbol{L}$ is the well-known LAPLACIAN of $G$ which is defined as

$$L_{ij} \triangleq \begin{cases} \omega(\{v_i\}, V) & \text{if} \quad i = j \\ -\omega(v_i, v_j) & \text{if} \quad \{v_i, v_j\} \in E(G) \\ 0 & \text{else} \end{cases}$$

In addition, the total vertex weights of a set $S \subseteq V(G)$ can be written

$$\gamma(S) = \boldsymbol{x}^{\mathsf{T}} \boldsymbol{\gamma}$$

with $\boldsymbol{\gamma} = (\gamma(v_1), \dots, \gamma(v_n))^{\mathsf{T}}$. Putting all together, we can cast equation (3.23) into the form

$$\mathsf{icut}(A, A^{\mathsf{c}}) = \frac{\boldsymbol{x}^{\mathsf{T}} \boldsymbol{L} \boldsymbol{x}}{\mathsf{min}\, \{\boldsymbol{x}^{\mathsf{T}} \boldsymbol{\gamma}, (\boldsymbol{1} - \boldsymbol{x})^{\mathsf{T}} \boldsymbol{\gamma}\}} \tag{3.24}$$

with $\boldsymbol{x}$ being the indicator vector of $A$. The crux of the isoperimetric cut problem is the minimum operator in denominator of equation (3.24). The incorporation of both the fore- and background vertices into the optimization problem makes it very difficult to handle. However, it is possible to overcome this situation if we first remove the minimum operator and add an additional constrain instead. Moreover one can "relax" the problem by letting the variables $x_i$ take on the continuous interval $[0, 1]$ instead of only the two values $\{0, 1\}$. So, if we apply the first modification minimizing (3.24) is equivalent to minimizing the simpler expression

$$\frac{\boldsymbol{x}^{\mathsf{T}} \boldsymbol{L} \boldsymbol{x}}{\boldsymbol{x}^{\mathsf{T}} \boldsymbol{\gamma}} \tag{3.25}$$

subject to the constrain $\boldsymbol{x}^\mathsf{T}\boldsymbol{\gamma} = k \leq {}^1\!/_2\mathbf{1}^\mathsf{T}\boldsymbol{\gamma}$. The second modification, allows one to insert the constrain just added into 3.25 with aid of a LAPLACIAN Multiplier:

$$Q(\boldsymbol{x}) \triangleq \boldsymbol{x}^\mathsf{T}\boldsymbol{L}\boldsymbol{x} - \Lambda(\boldsymbol{x}^\mathsf{T}\boldsymbol{\gamma} - k),$$

such that finding an optimal argument for (3.25) can be done by finding an optimal argument for $Q(\boldsymbol{x})$. Since $\boldsymbol{L}$ is positive semi-definite and $\boldsymbol{x}^\mathsf{T}\boldsymbol{\gamma}$ is non-negative the expression $Q(\boldsymbol{x})$ is minimized at any critical point [GRADY and SCHWARTZ 2006*a*]. Differentiating $Q(\boldsymbol{x})$ with respect to $\boldsymbol{x}$ yields

$$Q'(\boldsymbol{x}) = 2\boldsymbol{L}\boldsymbol{x} - \Lambda\boldsymbol{\gamma}$$

Setting $Q'(\boldsymbol{x}) = 0$ we finally get after some rearranging:

$$2\boldsymbol{L}\boldsymbol{x} = \Lambda\boldsymbol{\gamma}.$$

Since we are only interested in the relative values of $\boldsymbol{x}$ we can safely drop the scalars 2 and $\Lambda$. Unfortunately, $\boldsymbol{L}$ is singular, i.e. all columns and rows sum to zero, which also means that it is not invertible. In order to overcome this situation GRADY and SCHWARTZ propose to assign one beforehand selected *seed vertex* $v_i \in V(G)$ to the "background" $S^\mathsf{c}$ which is equivalent to setting $x_i = 0$. This way, we can remove the $i$-th row and column from $\boldsymbol{L}$ yielding $\boldsymbol{L}_i$ and the $i$-th component from $\boldsymbol{x}$ and $\boldsymbol{\gamma}$ yielding $\boldsymbol{x}_i$ and $\boldsymbol{\gamma}_i$. So we finally arrive at solving the system of linear equations represented by

$$\boldsymbol{L}_i\boldsymbol{x}_i = \boldsymbol{\gamma}_i. \tag{3.26}$$

So solving equation (3.26) results in a vector $\boldsymbol{x} \in [0,1]^{n-1}$. This vector can be converted into a cut by application of a threshold $t$ on each component, i.e. by setting

$$S = \{v_i \in V(G) \,|\, x_i < t\}\,. \tag{3.27}$$

The resulting cut $E(S, S^\mathsf{c})$ is then not guaranteed to be connected. However, GRADY and SCHWARTZ [2006*a*] show that at least the component of the resulting cut containing the seed vertex is always connected. In summary, starting with a region adjacency graph with given edge and vertex weights we have the following steps to take in order to get an approximating solution to the isoperimetric cut problem:

**Algorithm 3.7.1:** isoperimetric cut

```
1 Enumerate all vertices of the graph arbitrarily and compute the entries for $\boldsymbol{L}$ and $\boldsymbol{\gamma}$,
2 determine a vertex $v_i$ which functions as seed vertex,
3 solve the system $\boldsymbol{L}_i\boldsymbol{x}_i = \boldsymbol{\gamma}_i$ of linear equations, and
4 threshold the solution $\boldsymbol{x}_i$ to take only on binary values.
```

We briefly discuss these steps

2 For the selection of the seed vertex Grady and Schwartz [2006*a*] suggest to use the vertex with the maximal weighted degree. The motivation of this is that a vertex with high degree is likely not to have an edge in the optimal cut. They also have found that the selection of the seed vertex is only crucial in pathological cases. Therefore, we adopt this selection method here.

3 Solving the linear system of equations is computationally the most expensive step. Since $\boldsymbol{L}_i$ is symmetric and semi-definite the conjugate gradient method can be used which is both effective and accurate [Press et al. 2002, p. 87 et seqq]. Moreover, since the Laplacians of the region adjacency graphs derived from an initial segmentations are sparse (cf. 2 in section 2.3.4), the matrix-vector multiplications needed for the conjugate gradient can be effectively computed. We used the conjugate gradient method available in the scipy python module for this [5]. Figure 3.17 shows the solution of the linear system of equations for a region adjacency graph derived from the watershed segmentation of an image from the Berkeley Image Database.

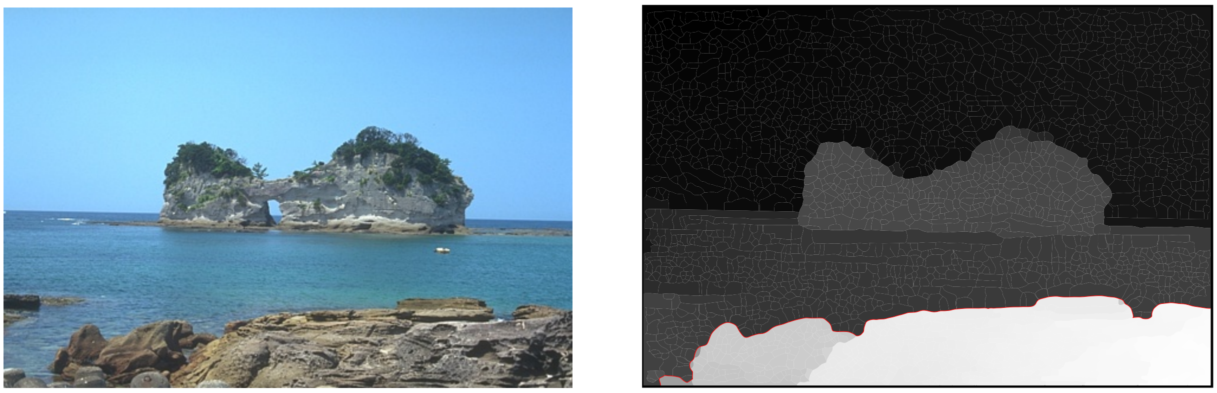
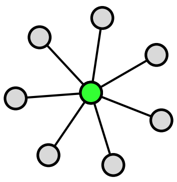


**Figure 3.17:** *An example for the isoperimetric cut approximation; the image on the right shows the entries for the vector $\boldsymbol{x}_i$ after solving the linear system of equations and the resulting cut (red line) after thresholding. The region adjacency graph has been derived form a watershed segmentation and has edge weights from* colorMean.



However, it should be noted that the solution of the linear system of equations can be problematic for small problem instances which correspond to “star” graphs, see the figure on the right. We encounter such graphs during the recursive application of the isoperimetric cut. If the seed vertex happens to be the vertex in the middle the corresponding Laplacian only contains one row which means that we cannot use the approximation scheme. The solution to this problem is very simple: Since the isoperimetric cut has to be connected with both components being nonempty, it can only cut one isolated vertex from the star configuration being unequal to the vertex in the middle. So, the computation of the isoperimetric cut for such configurations is straightforward.

4 Grady and Schwartz investigated several ways to actually determine such a threshold

[5]For the time of writing (September 10, 2008) this software was available to the author at `http://www.scipy.org`.

value. Here we chose the method which also yields the smallest isoperimetric number. In order to achieve this, one could successively select all $x_k$ as a threshold and compute the isoperimetric number of the resulting partition and select the partition with the optimal value. This would require $\mathcal{O}(n^2)$ operations. However, it is possible to determine the optimal threshold in $\mathcal{O}(n \cdot \log n)$ operations requiring an additional sorting step [Grady and Schwartz 2006*a*]: If the components of an indicator vector $\boldsymbol{x}$ are sorted one can simple successively use its components $x_k$ in increasing order as the threshold value. So, if we set $t = x_k$, the cut value of the cut satisfying equation (3.27) can be computed from the Laplacian $\boldsymbol{L}$ and the vertex weight vector $\boldsymbol{\gamma}$ which now have only for the components $L_{ij}$,$\gamma_i$ with $i, j < k$ nonzero entries. This also means that the cut value can be computed with $\mathcal{O}(n)$ operations if the indicator vector is already sorted.

**Time Complexity** The time complexity for the algorithm is composed of the time complexity of solving the linear system of equations and the time complexity of the sorting step which is $\mathcal{O}(n \log n)$. For the linear system of equations the dominant operations is the matrix-vector multiplication for the Laplacian which is for sparse matrices $\mathcal{O}(m)$ if $m$ is the number of nonzero entries. Since $m < c \cdot n$ for $n$ being the number of vertices in the graph and $c$ a constant, the time complexity for this operations is $\mathcal{O}(n)$. This makes a total complexity of $\mathcal{O}(N \cdot n + n \log n)$ where $N$ is the number of iterations needed for the conjugate gradient method. Grady and Schwartz [2006*a*] assumed $N$ to be constant. Figure 3.18 shows the dependence of the number of vertices in the graph and the number of iterations used by the conjugate gradient method for our experiments. As one can see, the number of iterations indeed increases much with increasing graph size.

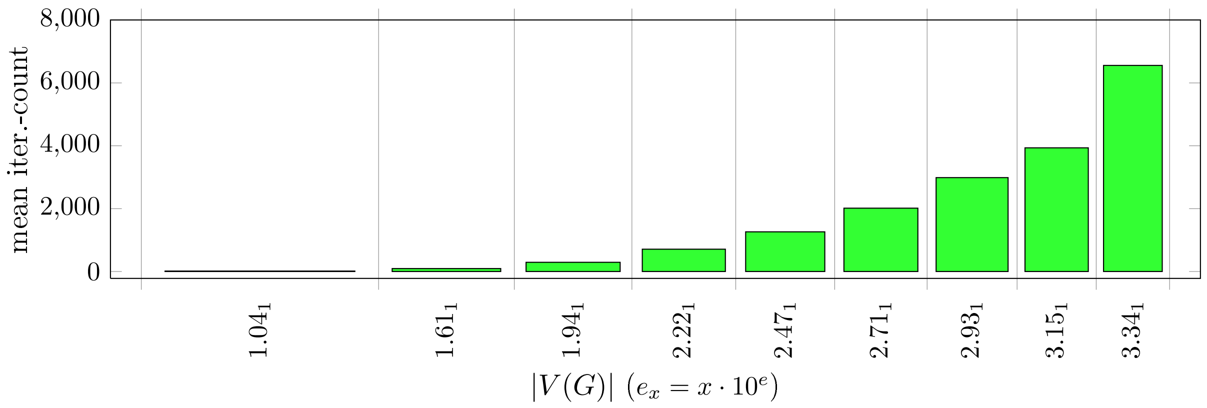


**Figure 3.18:** *This figure shows the dependence between number of vertices in the graph and number of iterations for the conjugate gradient method, the scaling for x-axis is logarithmic and for the numbers we have $e_x = x \cdot 10^e$; for data generation same method as described on page 57 has been used.*

## 3.8 Normalized Cut

This section introduces the cut of Shi and Malik [2000] which they call “the normalized cut”. It is probably the most famous normalized version of the minimum cut and we therefore also adopt this name here despite of the fact that all cut measures used in this work are normalized versions of the minimum cut.

Before we discuss the actual normalized cut of Shi and Malik [2000] we introduce are more generalized version. Similarly to the isoperimetric cut, the normalized cut also uses a weight function defined on the vertices of a graph for proper normalization:

**Definition 29** (normalized cut)**.** *For a graph $G = (V, E)$ with edge weight function $\omega$ and vertex weight function $\gamma$ the* normalized cut value *of a cut $(A, A^{\mathsf{c}})$ with $A \in V$ is defined as*

$$\mathsf{ncut}(A, A^{\mathsf{c}}) \triangleq \frac{\omega(A, A^{\mathsf{c}})}{\gamma(A)} + \frac{\omega(A, A^{\mathsf{c}})}{\gamma(A^{\mathsf{c}})} \tag{3.28}$$

*and a cut with minimal normalized cut costs is refereed to as a* normalized cut.

The usage of a vertex weight function in the normalized cut leads to cuts which are balanced with respect to the *volume* of the resulting subgraphs, which is defined as sum of the vertex weights in the subgraph.

For the actual definition of the volume we have two common candidates for selecting the value of $\gamma(v)$ for $v \in V$: The number of edges incident to $v$, i.e. $\gamma(v) \triangleq |E(\{v\}, V)|$, and the sum of the weights of the incident edges, i.e $\gamma(v) \triangleq \omega(\{v\}, V)$ which is the weighted degree of section 3.6. Using the first suggestion the normalized cut defined by equation (3.28) becomes the so called *average cut* and the second gives the normalized cut as defined by Shi and Malik [2000].

Although the average cut has already been used in the field of computer vision [Sarkar and Soundararajan 2000] we concentrate here on the normalized cut of Shi and Malik for the following reason: The average cut is only balanced with respect to the number of vertices in the components of the cut and therefore does not take into account the relations of the vertices to each other encoded in the edge weights. The normalized cut has this property which can be further investigated if we look at the *normalized association* defined as [Shi and Malik 2000]:

$$\mathsf{nassoc}(A, A^{\mathsf{c}}) \triangleq \frac{\omega(A, A)}{\omega(A, V)} + \frac{\omega(A^{\mathsf{c}}, A^{\mathsf{c}})}{\omega(A^{\mathsf{c}}, V)}. \tag{3.29}$$

It describes how tightly connected the nodes in each component of the cut are. Since $\omega(A, A^{\mathsf{c}}) = \gamma(A) - \omega(A, A)$ and similar for $A^{\mathsf{c}}$ the connection of the normalized association to the normalized cut can be seen from:

$$\begin{aligned}
\mathsf{ncut}(A, A^{\mathsf{c}}) &= \frac{\omega(A, A^{\mathsf{c}})}{\gamma(A)} + \frac{\omega(A, A^{\mathsf{c}})}{\gamma(A^{\mathsf{c}})} \\
&= \frac{\gamma(A) - \omega(A, A)}{\gamma(A)} + \frac{\gamma(A^{\mathsf{c}}) - \omega(A^{\mathsf{c}}, A^{\mathsf{c}})}{\gamma(A^{\mathsf{c}})} \\
&= 2 - \frac{\omega(A, A)}{\gamma(A)} + \frac{\omega(A^{\mathsf{c}}, A^{\mathsf{c}})}{\gamma(A^{\mathsf{c}})} \\
&= 2 - \frac{\omega(A, A)}{\omega(A, V)} + \frac{\omega(A^{\mathsf{c}}, A^{\mathsf{c}})}{\omega(A^{\mathsf{c}}, V)} \\
&= 2 - \mathsf{nassoc}(A, A^{\mathsf{c}})
\end{aligned}$$

Thus minimizing $\mathsf{ncut}(A, A^{\mathsf{c}})$ also maximizes $\mathsf{nassoc}(A, A^{\mathsf{c}})$.

## Spectral Relaxation

Unfortunately, finding a normalized cut is NP-hard in general (see [Shi and Malik 2000] for a proof) and for the time being it is not known if imposing restrictions on the structure of the graph makes it more tractable. However one can find an approximating solution using spectral relaxation which means using the eigenvectors of a scaled version of the Laplacian. Note that we already use the Laplacian in section 3.7 for linear relaxation of the isoperimetric cut problem. The general idea is similar for the normalized cut; at first we proof that we can express the normalized costs of equation (3.28) using the Laplacian of a graph together with some constraints and then get an approximating solution if these constraints are relaxed somehow.

More precisely, we can represent a cut $E(A, A^{\mathsf{c}})$ in a graph $G$ with vertex set $V \triangleq \{v_1, \ldots, v_n\}$ with an *indicator vector* $\boldsymbol{x}$ defined as

$$x_i \triangleq \begin{cases} +1 & \text{if} \quad v_i \in A \\ -1 & \text{else} \end{cases}$$

Using the same definition of section 3.7 for the *Laplacian* of $G$ which is

$$L_{ij} \triangleq \begin{cases} \omega(E(\{v_i\}, V)) & \text{if} \quad i = j \\ -\omega(v_i, v_j) & \text{if} \quad \{v_i, v_j\} \in E(G) \\ 0 & \text{else} \end{cases} ,$$

we can write

$$\omega(A, A^{\mathsf{c}}) = \tfrac{1}{4}\boldsymbol{x}^{\mathsf{T}}\boldsymbol{L}\boldsymbol{x}.$$

For the volume of the hole vertex set $V$ we can write

$$\gamma(V) = \boldsymbol{x}^\mathsf{T}\boldsymbol{\Gamma}\boldsymbol{x}$$

if $\boldsymbol{\Gamma}$ is defined as

$$\Gamma_{ij} \triangleq \begin{cases} \gamma(v_i) & \text{if} \quad i = j \\ 0 & \text{else} \end{cases} .$$

The following theorem establishes the connection of these definitions to the normalized cut problem. This theorem, its proof, and the subsequent line of argumentation is based on [Keuchel 2004, chapter 3, p. 29 et seqq] [6]:

**Theorem 7.** *If $(A, A^\mathsf{c})$ is a solution of the normalized cut problem, then the optimization problem*

$$\min_{\boldsymbol{y}} \frac{\boldsymbol{y}^\mathsf{T}\boldsymbol{L}\boldsymbol{y}}{\boldsymbol{y}^\mathsf{T}\boldsymbol{\Gamma}\boldsymbol{y}} \tag{3.30}$$

*subject to the constraints*

$$\boldsymbol{y} \in \big\{ -\alpha, \tfrac{1}{\alpha^2} \big\}^n \;\; \textit{and} \quad \boldsymbol{y}^\mathsf{T}\boldsymbol{\Gamma}\boldsymbol{1} = 0$$

*with $\alpha \triangleq \sqrt{\frac{\gamma(A)}{\gamma(A^\mathsf{c})}}$ is equivalent to the normalized cut problem.*

*Proof (based on [Keuchel 2004, chapter 3, p. 29 et seqq]).* First note that since $\gamma(V) = \gamma(A^\mathsf{c}) + \gamma(A)$ equation (3.30) can also be written as

$$\mathsf{ncut}(A, A^\mathsf{c}) = \gamma(V)\frac{\omega(A, A^\mathsf{c})}{\gamma(A)\cdot\gamma(A^\mathsf{c})} \tag{3.31}$$

Using the alterantive vector and matrix notation as describe above, we have

$$\begin{aligned} \mathsf{ncut}(A, A^\mathsf{c}) &= \gamma(V)\frac{\frac{1}{4}\boldsymbol{x}^\mathsf{T}\boldsymbol{L}\boldsymbol{x}}{\gamma(A)\cdot\gamma(A^\mathsf{c})} \\ &= \frac{\frac{\gamma(V)^2}{\gamma(A^\mathsf{c})^2}}{\frac{\gamma(V)^2}{\gamma(A^\mathsf{c})^2}} \cdot \frac{\frac{1}{4}\boldsymbol{x}^\mathsf{T}\boldsymbol{L}\boldsymbol{x}}{\gamma(A)\cdot\gamma(A^\mathsf{c})} \qquad \text{since } (ab)\boldsymbol{x}^\mathsf{T}\boldsymbol{L}\boldsymbol{x} = (a\boldsymbol{x})^\mathsf{T}\boldsymbol{L}(b\boldsymbol{x}) \\ &= \frac{(\frac{1}{2}\frac{\gamma(V)}{\gamma(A^\mathsf{c})}\boldsymbol{x})^\mathsf{T}\boldsymbol{L}(\frac{1}{2}\frac{\gamma(V)}{\gamma(A^\mathsf{c})}\boldsymbol{x})}{\frac{\gamma(A)}{\gamma(A^\mathsf{c})}\cdot\boldsymbol{x}^\mathsf{T}\boldsymbol{\Gamma}\boldsymbol{x}} \end{aligned}$$

[6] The results of Keuchel lead us to same algorithm as in [Shi and Malik 2000]. We use Keuchel derivation as we think that it is more intuitiv and easier to follow.

for better reading we further define $\beta_\oplus \triangleq 1+\alpha^2$ and $\beta_\ominus \triangleq 1-\alpha^2$ such that we can proceed with

$$\begin{aligned}
\mathsf{ncut}(A,A^\mathsf{c}) &= \frac{(\frac{1}{2}\beta_\oplus \boldsymbol{x})^\mathsf{T}\boldsymbol{L}(\frac{1}{2}\beta_\oplus \boldsymbol{x})}{\alpha^2\cdot \boldsymbol{x}^\mathsf{T}\boldsymbol{\Gamma}\boldsymbol{x}} \\
&= \frac{(\frac{1}{2}\beta_\oplus \boldsymbol{x})^\mathsf{T}\boldsymbol{L}(\frac{1}{2}\beta_\oplus \boldsymbol{x})}{\alpha^2\cdot \boldsymbol{x}^\mathsf{T}\boldsymbol{\Gamma}\boldsymbol{x}} + \underbrace{\frac{\frac{1}{4}\beta_\ominus^2\mathbf{1}^\mathsf{T}\boldsymbol{L}\mathbf{1}+\frac{1}{2}\beta_\ominus\beta_\oplus\mathbf{1}^\mathsf{T}\boldsymbol{L}\boldsymbol{x}}{\alpha^2\cdot \boldsymbol{x}^\mathsf{T}\boldsymbol{\Gamma}\boldsymbol{x}}}_{=0 \text{ since } \mathbf{1}\boldsymbol{L}\boldsymbol{v}=0\forall \boldsymbol{v}} \\
&= \frac{(\frac{1}{2}\beta_\oplus \boldsymbol{x})^\mathsf{T}\boldsymbol{L}(\frac{1}{2}\beta_\oplus \boldsymbol{x}) + (\frac{1}{2}\beta_\ominus\mathbf{1})^\mathsf{T}\boldsymbol{L}(\frac{1}{2}\beta_\ominus\mathbf{1}) + 2\cdot\left[(\frac{1}{2}\beta_\ominus\mathbf{1})^\mathsf{T}\boldsymbol{L}(\frac{1}{2}\beta_\oplus\boldsymbol{x})\right]}{\alpha^2\cdot \boldsymbol{x}^\mathsf{T}\boldsymbol{\Gamma}\boldsymbol{x}} \\
&= \frac{\left(\frac{1}{2\alpha}\left(\beta_\oplus\boldsymbol{x}+\beta_\ominus\mathbf{1}\right)\right)^\mathsf{T}\boldsymbol{L}\left(\frac{1}{2\alpha}\left(\beta_\oplus\boldsymbol{x}+\beta_\ominus\mathbf{1}\right)\right)}{\boldsymbol{x}^\mathsf{T}\boldsymbol{\Gamma}\boldsymbol{x}}.
\end{aligned}$$

Further we can derive

$$\begin{aligned}
\boldsymbol{x}^\mathsf{T}\boldsymbol{\Gamma}\boldsymbol{x} &= \gamma(A)+\gamma(A^\mathsf{c}) \\
&= \alpha^2\gamma(A^\mathsf{c}) + \tfrac{1}{\alpha^2}\gamma(A) \\
&= \tfrac{1}{2}(\tfrac{1}{\alpha^2}+\alpha^2)(\gamma(A)+\gamma(A^\mathsf{c})) + \tfrac{1}{2}(\tfrac{1}{\alpha^2}-\alpha^2)(\gamma(A)-\gamma(A^\mathsf{c})) \\
&= (\tfrac{1}{4\alpha}\beta_\oplus^2+\tfrac{1}{4\alpha}\beta_\ominus^2)(\gamma(A)+\gamma(A^\mathsf{c})) + (\tfrac{1}{2\alpha}\beta_\ominus\beta_\oplus)(\gamma(A)-\gamma(A^\mathsf{c})) \\
&= (\tfrac{1}{4\alpha}\beta_\oplus^2)\boldsymbol{x}^\mathsf{T}\boldsymbol{\Gamma}\boldsymbol{x} + (\tfrac{1}{4\alpha}\beta_\ominus^2)\mathbf{1}^\mathsf{T}\boldsymbol{\Gamma}\mathbf{1} + (\tfrac{1}{2\alpha}\beta_\ominus\beta_\oplus)\mathbf{1}^\mathsf{T}\boldsymbol{\Gamma}\boldsymbol{x} \\
&= (\tfrac{1}{2\alpha}\beta_\oplus\boldsymbol{x})^\mathsf{T}\boldsymbol{\Gamma}(\tfrac{1}{2\alpha}\beta_\oplus\boldsymbol{x}) + (\tfrac{1}{2\alpha}\beta_\ominus\mathbf{1})^\mathsf{T}\boldsymbol{\Gamma}(\tfrac{1}{2\alpha}\beta_\ominus\mathbf{1}) + 2\cdot\left[(\tfrac{1}{2\alpha}\beta_\ominus\mathbf{1})^\mathsf{T}\boldsymbol{\Gamma}(\tfrac{1}{2\alpha}\beta_\oplus\boldsymbol{x})\right] \\
&= (\tfrac{1}{2\alpha}\left(\beta_\oplus\boldsymbol{x}+\beta_\ominus\mathbf{1}\right))^\mathsf{T}\boldsymbol{\Gamma}(\tfrac{1}{2\alpha}\left(\beta_\oplus\boldsymbol{x}+\beta_\ominus\mathbf{1}\right))
\end{aligned}$$

So, if we set

$$\begin{aligned}
\boldsymbol{y} &\triangleq \tfrac{1}{2\alpha}\left(\beta_\oplus\boldsymbol{x}+\beta_\ominus\mathbf{1}\right) \\
&= \tfrac{1}{2}\big((\tfrac{1}{\alpha}+\alpha^2)\boldsymbol{x}+(\tfrac{1}{\alpha}-\alpha^2)\mathbf{1}\big),
\end{aligned}$$

one can see that actually $\boldsymbol{y}\in\big\{-\alpha,\frac{1}{\alpha^2}\big\}^n$ holds. For the second constraint, observe that

$$\begin{aligned}
\boldsymbol{y}\boldsymbol{\Gamma}\mathbf{1} &= \tfrac{1}{2\alpha}\left(\beta_\oplus\boldsymbol{x}\boldsymbol{\Gamma}\mathbf{1}+\beta_\ominus\mathbf{1}\boldsymbol{\Gamma}\mathbf{1}\right) \\
&= \tfrac{1}{2}\left(2\tfrac{1}{\alpha}\omega(A)-2\alpha\omega(A^\mathsf{c})\right) \\
&= \tfrac{1}{2}\left(\frac{\omega(A^\mathsf{c})}{\omega(A)}\omega(A)-\omega(A^\mathsf{c})\right) \\
&= 0
\end{aligned}$$

finally proves the stated equivalence.

□

While theorem 7 does not alter the intractability of the normalized cut problem – the value $\alpha = \sqrt{\gamma(A)/\gamma(A^c)}$ has to been known in advance – it offers nevertheless a way for an approximation scheme: One could drop the constraint $\boldsymbol{y} \in \{-\alpha, \frac{1}{\alpha^2}\}^n$, that is, *relax* this binary constraint on $\boldsymbol{y}$ and instead minimize equation (3.28) for real valued entries of $\boldsymbol{y}$ by solving the generalized eigenvalue problem

$$\boldsymbol{L}\boldsymbol{y} = \lambda \boldsymbol{\Gamma} \boldsymbol{y} \tag{3.32}$$

Since $\boldsymbol{L}$ is symmetric and positive semi-definite [Shi and Malik 2000] and $\boldsymbol{\Gamma}$ is positive definite (it is a diagonal matrix with all entries being positive) all solutions to equation (3.32) are guaranteed to be positive. Moreover, since $\boldsymbol{1}$ is an eigenvector of the Laplacian with eigenvalue 0 ($\boldsymbol{L}\boldsymbol{1} = 0$) Shi and Malik [2000] suggest to use the second smallest eigenvalue of equation (3.32) for an approximating solution of the normalized cut problem. The problem can be even simplified as equation (3.32) can be transformed into a standard eigenvalue problem [Shi and Malik 2000]: Due to the definition of $\boldsymbol{\Gamma}$ it is symmetric positive definite and we can write $\boldsymbol{\Gamma} = \boldsymbol{F}\boldsymbol{F}^\mathsf{T}$ with $\boldsymbol{F} = \boldsymbol{\Gamma}^{\frac{1}{2}} = \mathsf{diag}(\gamma^{\frac{1}{2}})$. Thus, we can rewrite equation (3.32) into the form

$$\boldsymbol{\Gamma}^{-\frac{1}{2}} \boldsymbol{L} \boldsymbol{\Gamma}^{-\frac{1}{2}} = \lambda \boldsymbol{z} \tag{3.33}$$

with $\boldsymbol{z} \triangleq \boldsymbol{\Gamma}^{\frac{1}{2}} \boldsymbol{y}$. So, solving (3.32) is equivalent to solving (3.33). Since $\boldsymbol{L}$ is positive semi-definite this also holds for $\boldsymbol{\Gamma}^{-\frac{1}{2}} \boldsymbol{L} \boldsymbol{\Gamma}^{-\frac{1}{2}}$ which means that all its eigenvectors are positive. Moreover, as $\boldsymbol{z}_1 \triangleq \boldsymbol{\Gamma}^{\frac{1}{2}} \boldsymbol{1}$ is an eigenvector of (3.33) with eigenvalue 0 which is also perpendicular to all other eigenvectors of (3.33) we have

$$\boldsymbol{z}_1^\mathsf{T} \boldsymbol{z}_2 = \boldsymbol{y} \boldsymbol{\Gamma} \boldsymbol{1} = 0$$

if $\boldsymbol{z}_2$ is the second smallest eigenvector of (3.33). This means that solving either (3.32) or (3.33) also automatically satisfies the second constraint in theorem 7. Note, that if (3.32) is solved the resulting eigenvector $\boldsymbol{z}_2$ has to be transformed by $\boldsymbol{y}_2 = \boldsymbol{\Gamma}^{-\frac{1}{2}} \boldsymbol{z}_2$ in order to get the final result.

As with the isoperimetric cut (section 3.7) this real valued solution can then be transformed into a cut of the corresponding graph by thresholding. So, finally we arrive at the following steps for finding a solution for the normalized cut problem:

**Algorithm 3.8.1:** normalized cut

```
1  Given a graph enumerate all its vertices arbitrarily and compute the entries for $\boldsymbol{L}$ and $\boldsymbol{\Gamma}$,
2  either solve the generalized eigenvalue problem given by equation (3.32) or the standard
   eigenvalue problem given by equation (3.33), and
3  threshold the solution $\boldsymbol{y}$ to take only on binary values.
```

We would like to give some remarks on the last two steps:

2 Solving the generalized eigenvalue problem is computationally a very expensive step. As already noted for the isoperimetric cut, the Laplacian for the graphs used here are sparse.

Therefore one can use optimized eigenvalue solvers. We used the implementation available with the PYSPARSE python package[7] which uses the routines of the ARPACK fortran library. Figure 3.17 shows the solution of the eigenvalue problem for a region adjacency graph derived from the watershed segmentation of an image form the BERKELEY Database.

3 For thresholding we used the same method as described in section 3.7 but appropriately modified for the normalized cut.

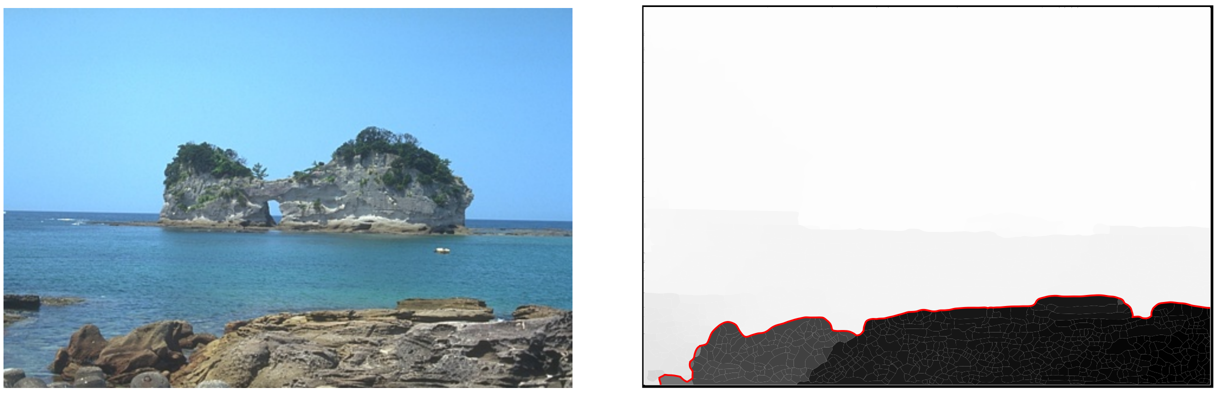


**Figure 3.19:** *An example for the normalized cut approximation; the image on the right shows the entries for the second smallest eigenvector and the resulting cut (red line) after thresholding. The region adjacency graph has been derived form a watershed segmentation and has edge weights from* colorMean.

**Time Complexity** The most expensive part for the computation of the normalized cut approximation is solving the eigenvalue problem. This can by done with the LANZCOS Method which runs in $\mathcal{O}(nm) + \mathcal{O}(nk)$ time with $n$ being the size of the matrix, i.e. the number of nodes in the graph, $m$ being the number of iterations needed for the LANZCOS Method, and $k$ being the costs of a matrix-vector multiplication [SHI and MALIK 2000]. Since $\boldsymbol{L}$ is sparse this can be done using $\mathcal{O}(n)$ operations. The number of steps $m$ can not be further determined but figure 3.20 shows that for our experiments it slowly grows with the number of nodes in the graph.

[7]For the time of writing (September 10, 2008) this software was available to the author at `http://sourceforge.net/projects/pysparse/`.

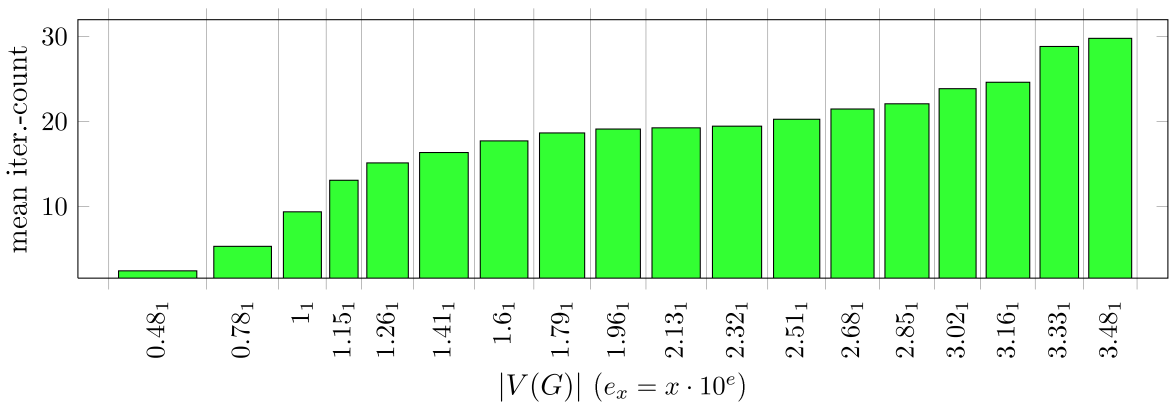


**Figure 3.20:** *This figure shows the dependence between number of nodes in the graph and number of iterations for the normalized cut algorithm method, the scaling for x-axis is logarithmic and for the numbers holds $e_x = x \cdot 10^e$; for data generation the same method as described on page 57 has been used.*

# 4 Evaluation

In this chapter we discuss the evaluation method used in order to assess the segmentation quality of the various instances of the basic segmentation scheme (as described in section 2.3.4). We start with a brief discussion of the evaluation problem for image segmentation algorithms in general. Then we give an overview of the work of Martin et al. in conjunction with the Berkeley Image Database. After this the Normalized Rand Index is discussed which is used here to actually assess the segmentation results. Finally, the evaluation scheme is described and the results are presented and discussed.

## 4.1 Image Segmentation Evaluation

As the main interest of this thesis is to qualify the impact on segmentation results if different types of graph cuts and edge costs are used it is necessary to actually tell what a “good” segmentation is. Moreover, for an exhaustive comparison of different segmentation algorithms on a huge set of images it is necessary to express the segmentation quality in terms of an index being effectively and automatically computational. The ill posed nature of the segmentation problem renders this task very difficult.

There have already been approaches to this problem in the early eighties while the number of publications increased much in recent years. For an overview of progress on this topic Zhang published several surveys [Zhang 1996, 2001, 2006].

Following Zhang, evaluation methods can be divided into *analytical* and *empirical* methods. Analytical methods, qualify segmentation methods from a theoretical standpoint on algorithmic characteristics such as runtime-complexity, stability under variation of intrinsic parameters or input data. Such attributes can certainly only be given in addition to ones which are more concerned with the actual segmentation quality.

Empirical methods can be further divided into *goodness* and *discrepancy* measures. Goodness measures assess segmentation results indirectly by using quality measures motivated by human intuition on properties a good segmentation should have. Examples for goodness measures are region homogeneity or inter-region contrast respectively. Discrepancy measures actually use a reference segmentation, also called *ground truth*, which is seen as being optimal in some sense. A discrepancy measure then measures how close a segmentation is to the ground truth.

Here we use an empirical method, namely, the Normalized Rand Index together with ground truth data from the Berkeley Image Database.

### The BerkeleyImage Database

The Berkeley Database is an image database of natural scenes with several manual performed segmentations for each image. It has been introduced by Martin et al. [2001] and later described in more detail in [Martin 2003].

The main intention of Martin et al. is to provide a solid basis of ground truth data for empirical evaluation of segmentation algorithms. For this Martin et al. have taken a subset of images from the well known Corel Image Database. The selected images show mainly natural scenes that contain at least one discernible object such as animals, human beings, or man-made structures, see figure 4.1 for some examples. This excludes images with difficult photometric phenomena such as prominent shadows, reflections, or translucence. All images have been segmented

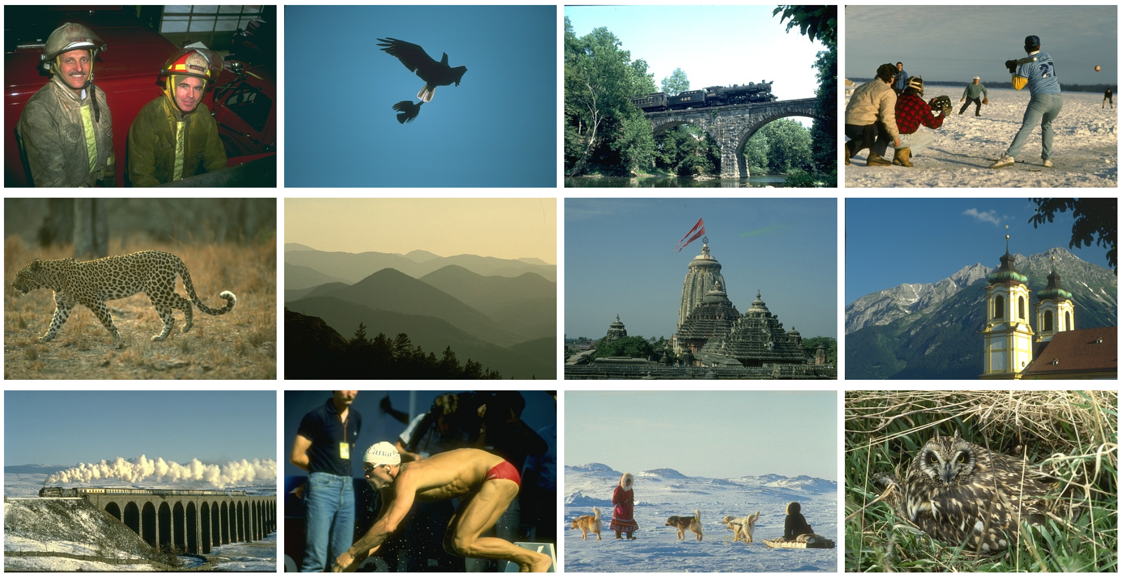


**Figure 4.1:** *some examples of images contained in the Berkeley Image Database*

manually by non computer vision experts briefed with the following instructions [Martin 2003, Chapter 2, p. 11]:

> *You will be presented a photographic image. Divide the image into some number of segments, where the segments represent "things" or "parts of things" in the scene. The number of segments is up to you, as it depends on the image. Something between 2 and 30 is likely to be appropriate. It is important that all of the segments have approximately equal importance.*

These instructions were intentionally formulated in a vague way in order not to bias the experiment and to encourage each subject to segment each image in a natural manner and as a naive observer. The segmentations were performed using a Java tool, and importantly, each image has been segmented several times from *different* subjects. For evaluating segmentation quality we choose a subset of one hundred randomly chosen images from the Berkeley Image

Database with 4 − 9 (5.43 on average) different ground truth segmentations for each image [1]. Some examples of the ground truth data can be seen in figure 4.2

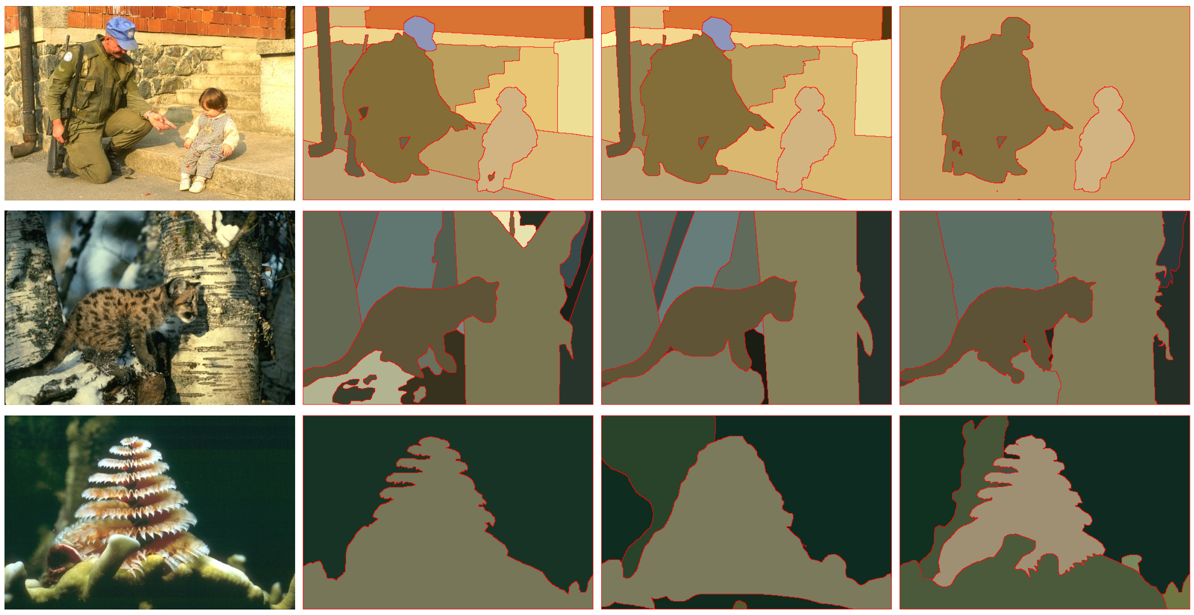


**Figure 4.2:** *some examples of manual segmentations from the* BERKELEY *Image Database. The first column shows the image on which segmentation has been performed. The remaining columns show the manual segmentation where each of them has been performed by a different subject (at least in the same row). These examples also show the occurrence of mutual refinement.*

As one can see from these examples, the various manual segmentations differ in level of granularity on which the division into segments has been performed: For example, in the second row of figure 4.2 one can see that parts of the image have been segmented into several regions by one subject where another subject only segmented the same region into one group and the other way around. This phenomenon is called *mutual refinement*. The important thing to note is that difference in segmentations performed by humans can be mainly reduced to this aspect and are therefore perceptually consistent. Hence, MARTIN et al. [2001] argue that the idea of comparing segmentation results to all perceptually valid interpretations of an image is a reliable basis for an evaluation framework.

## 4.2 Rand Index and Extensions

In this section we discuss the Normalized Probabilistic RAND Index (NPR) introduced to the field of computer vision by UNNIKRISHNAN et al. [2007] for automatically assessing the quality of segmentation results. The intention of UNNIKRISHNAN et al. was to design an empirical evaluation measure which can be used if several ground truth segmentations for an image are available

[1]at the time of writing (September 10, 2008) the image data together with the ground truth data was available in internet at `http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench/`

and obeys certain additional requirements such a measure should fulfill. The requirements are:

1 nondegeneracy

2 no assumption about data generation

3 accommodation to mutual refinement

4 comparable scores

1 Nondegeneracy means that there should be no pathological configurations for which a segmentation gives very high scores although it does not fit well to the ground truth data. This is one main aspect which motivated UNNIKRISHNAN et al. [2007] to introduce the NPR since other measures have problems with degenerated segmentations which for example only contain one big region or many very small sized regions. Therefore those measures required the segmentation to be assessed and the ground truth data to have comparable number of regions. If this is not the case those measures produce misleading high scores. The NPR does not have such requirements.

2 The measure should not make assumptions on the number of regions or their size.

3 The phenomenon of mutual refinement as described above should be handled appropriately. This means that the measure should not punish difference in segmentation granularity being consistent with the ground truth data and do punish difference in granularity not being consistent with the ground truth.

4 To be usable as an evaluation measure the scores of the NPR should allow comparison of segmentations from different images using the same algorithm as well as comparison on one image using different algorithms.

The NPR is based on the Probabilistic RAND Index which in turn is based on the RAND Index which are now presented in detail. For this we assume that segmentation results are represented with label-images, which are image functions $\boldsymbol{\ell} : \underline{w} \times \underline{h} \rightarrow \mathbb{N}$ assigning each pixel of an image to an unique label. The problem of incorporating sub-pixel accurate segment boundaries into an evaluation framework using label images is discussed on page 86.

## Rand Index

Similarity of two different segmentations can be expressed through the pixel pairwise comparison of the associated labels from label images: For two label images $\boldsymbol{\ell}, \boldsymbol{\ell}'$ defined on a set of pixel positions $\{\boldsymbol{p}_1, \dots, \boldsymbol{p}_n\}$ the *RAND Index* is defined as

$$\mathsf{R}(\boldsymbol{\ell}, \boldsymbol{\ell}') \triangleq \binom{n}{2}^{-1} \sum_{i<j} [c_{\boldsymbol{\ell}}(i,j) \cdot c_{\boldsymbol{\ell}'}(i,j) + (1 - c_{\boldsymbol{\ell}}(i,j))(1 - c_{\boldsymbol{\ell}'}(i,j))] \tag{4.1}$$

with

$$c_{\boldsymbol{\ell}}(i,j) \triangleq \begin{cases} 1 & \text{if} \quad \boldsymbol{\ell}[\boldsymbol{p}_i] = \boldsymbol{\ell}[\boldsymbol{p}_j] \\ 0 & \text{else} \end{cases}$$

(named after William Rand, see [Unnikrishnan 2005] and references therein). So, the expression $\mathsf{R}(\boldsymbol{\ell}, \boldsymbol{\ell}')$ simply denotes the number of agreements of the two label images on the question whether two given pixels are in the same segment normalized by the number of possible pixel-pairs. A value of zero indicates maximal dissimilarity and a value of one indicates maximal similarity of two segmentations. Note that the Rand Index does not require two segmentations to have the same number of segments. Moreover, it also does not require two segmentations to agree on a certain label for a segment. That means we can permute all labels of a label image without affecting the value of the Rand Index.

As shown in [Unnikrishnan et al. 2007], the Rand Index can be effectively computed with $\mathcal{O}(n+L)$ operations with $L$ being the number of different labels: Equation (4.1) can be rearranged to the form

$$\mathsf{R}(\boldsymbol{\ell}, \boldsymbol{\ell}') = \binom{n}{2}^{-1} \Bigg[ \underbrace{\Big( \sum_{i<j} c_{\boldsymbol{\ell}}(i,j) \cdot c_{\boldsymbol{\ell}'}(i,j) \Big)}_{C_1} + \underbrace{\Big( \sum_{i<j} (1 - c_{\boldsymbol{\ell}}(i,j))(1 - c_{\boldsymbol{\ell}'}(i,j)) \Big)}_{C_2} \Bigg]$$

Where term $C_1$ denotes the number of pixel pairs which are assigned to the same segment by both $\boldsymbol{\ell}$ and $\boldsymbol{\ell}'$ and, similarly, term $C_2$ denotes the number of pixel pairs which are assigned to different segments by the two label-images.

For the computation of $C_1$, we define the *contingency table (of $\boldsymbol{\ell}$ and $\boldsymbol{\ell}'$)*

$$\mathsf{ct}(\ell_1, \ell_2) \triangleq \big| \{ \boldsymbol{p} \in \underline{w} \times \underline{h} \,|\, \boldsymbol{\ell}[\boldsymbol{p}] = \ell_1 \wedge \boldsymbol{\ell}'[\boldsymbol{p}] = \ell_2 \} \big|$$

which denotes the number of pixels pairs which have label $\ell_1$ in $\boldsymbol{\ell}_1$ and also label $\ell_2$ in $\boldsymbol{\ell}_2$. Hence, this term equals the number of pixels shared by the associated segments in the two segmentations. Now we can write

$$C_1 = \sum_{\ell, \ell'} \binom{\mathsf{ct}(\ell, \ell')}{2} \tag{4.2}$$

which means that $C_1$ can be computed using $\mathcal{O}(n)$ operations.

As the second term represents the number of pixel pairs with different labels in both segmentations, one can simply compute it by subtracting all pairs of pixels which do not belong to that category from the total number of possible pixel pairs. There are four categories and one of these is already represented by equation (4.2). For the two categories left, we further define

$$\mathsf{ct}(\ell, \cdot) \triangleq \sum_{\ell'} \mathsf{ct}(\ell, \ell') \quad \text{and} \quad \mathsf{ct}(\cdot, \ell) \triangleq \sum_{\ell} \mathsf{ct}(\ell, \ell')$$

which are the numbers of pixels with label $\ell$ in $\boldsymbol{\ell}$ and the number of pixels with label $\ell$ in $\boldsymbol{\ell}'$ respectively. Now, the term

$$C_3 \triangleq \sum_{\ell} \binom{\mathsf{ct}(\ell,\cdot)}{2} - C_1$$

denotes the number of pixel pairs which have the same label in $\boldsymbol{\ell}$ but different labels in $\boldsymbol{\ell}'$. Likewise, the term

$$C_4 \triangleq \sum_{\ell'} \binom{\mathsf{ct}(\cdot,\ell)}{2} - C_1$$

denotes the number of pixel pairs which have different labels in $\boldsymbol{\ell}$ but the same label in $\boldsymbol{\ell}'$. Putting all together, we can write the second term in the form

$$\begin{aligned} C_2 &= \binom{n}{2} - C_1 - C_3 - C_4 \\ &= \binom{n}{2} + \sum_{\ell,\ell'} \binom{\mathsf{ct}(\ell,\ell')}{2} - \sum_{\ell} \binom{\mathsf{ct}(\ell,\cdot)}{2} - \sum_{\ell'} \binom{\mathsf{ct}(\cdot,\ell)}{2} \qquad (4.3) \end{aligned}$$

So, $C_2$ is computable in $\mathcal{O}(n+L)$ steps which is also the overall complexity of for the Rand Index.

**Implementation Notes** For the actual computation of $\mathsf{R}$ all values of a contingency table $\mathsf{ct}(\boldsymbol{\ell},\boldsymbol{\ell}')$ are determined and put into an array data structure where the $i$-th row contains all entries of $\mathsf{ct}$ for label $i$ of label image $\boldsymbol{\ell}$. Note that if for two label-images $\boldsymbol{\ell}_1$ and $\boldsymbol{\ell}_2$ holds that $\boldsymbol{\ell}_1$ can be transformed into $\boldsymbol{\ell}_2$ by merging two regions with label $i$ and $j$, the contingency table of $\boldsymbol{\ell}_2$ and $\boldsymbol{\ell}'$ can be computed by merging the $i$-th and $j$-th row in the contingency table of $\boldsymbol{\ell}_1$ and $\boldsymbol{\ell}'$. More precisely, if $\mathsf{ct}_1$ is the contingency table of $\boldsymbol{\ell}_1$ and $\boldsymbol{\ell}'$, and $\mathsf{ct}_2$ is the contingency table of $\boldsymbol{\ell}_2$ and $\boldsymbol{\ell}'$, we have

$$\mathsf{ct}_2(k,\ell) = \mathsf{ct}_1(i,\ell) + \mathsf{ct}_1(j,\ell) \quad \forall \ell \in L$$

if $L$ is the set of different labels in $\boldsymbol{\ell}'$ and $k$ is the label of the region in $\boldsymbol{\ell}_2$ which is composed of the regions with label $i$ and $j$ in $\boldsymbol{\ell}_1$.

Similarly, we can go in the opposite direction by “splitting” the appropriate row in a contingency table if one region is split into two parts. Using these simply operations we experienced a serious performance gain when computing the Rand Index in conjunction with recursive application of graph cuts (see section 4.3).

### Probabilistic Rand Index

Unfortunately, the Rand Index does not fulfill the requirements given above. Especially, it treats each disagreement of two segmentations on label assignment with equal importance. Hence, mutual refinement is not handled appropriately.

Nevertheless, it is possible to extend the Rand Index such that it fulfills our needs. The

Probabilistic RAND Index, first proposed by UNNIKRISHNAN [2005], assumes the existence of a BERNOULLI distribution $p(i,j)$ which models the probability that two pixels $\boldsymbol{p}_i, \boldsymbol{p}_j$ belong to the same segment. Instead of comparing a certain segmentation to one ground truth segmentation, a set $\mathcal{L}$ of different ground truth segmentations is employed. This way, we can turn to a probabilistic interpretation in which each agreement on label assignment of two pixels is weighted by the probability that this is correct. This probability can be estimated on basis of the ground truth data. Such a generalization called the *Probabilistic RAND Index* is defined as follows:

$$\mathsf{PR}(\boldsymbol{\ell} \,|\, \mathcal{L}) \triangleq \binom{n}{2}^{-1} \sum_{i<j} \left[c_{\boldsymbol{\ell}}(i,j) \cdot p_{\mathcal{L}}(i,j) + (1 - c_{\boldsymbol{\ell}}(i,j))(1 - p_{\mathcal{L}}(i,j))\right] \tag{4.4}$$

where $p_{\mathcal{L}}(i,j)$ is the estimated probability that pixel $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ have the same label, i.e. belong to the same segment, based on the set $\mathcal{L}$ of ground truth segmentations.

For the concrete definition of $p_{\mathcal{L}}(i,j)$ UNNIKRISHNAN [2005] proposes to use the sample mean estimator

$$p_{\mathcal{L}}(i,j) \triangleq \frac{1}{|\mathcal{L}|} \sum_{\boldsymbol{\ell} \in \mathcal{L}} c_{\boldsymbol{\ell}}(i,j) \tag{4.5}$$

With this definition the Probabilistic RAND Index can be reduced to the mean of all RAND Indexes between the segmentation to be assessed and all elements of the ground truth data:

$$\mathsf{PR}(\boldsymbol{\ell} \,|\, \mathcal{L}) \triangleq \frac{1}{|\mathcal{L}|} \sum_{\boldsymbol{\ell}' \in \mathcal{L}} \mathsf{R}(\boldsymbol{\ell}, \boldsymbol{\ell}')$$

### Normalized Probabilistic Rand Index

In order to be more comparable on different images a baseline correction is performed. So, we finally arrive at the *Normalized Probabilistic RAND Index* if equation (4.4) is normalized by the expected value of the index in the following way:

$$\mathsf{NPR}(\boldsymbol{\ell} \,|\, \mathcal{L}) \triangleq \frac{\mathsf{PR}(\boldsymbol{\ell} \,|\, \mathcal{L}) - \mathbb{E}[\mathsf{PR}]}{\max\{\mathsf{PR}\} - \mathbb{E}[\mathsf{PR}]}. \tag{4.6}$$

For the computation of $\mathbb{E}[\mathsf{PR}]$ we have:

$$\begin{aligned}
\mathbb{E}[\mathsf{PR}(\boldsymbol{\ell}, \mathcal{L})] &= \mathbb{E}\Big[\binom{n}{2}^{-1} \sum_{i<j} \big[c_{\boldsymbol{\ell}}(i,j) p_{\mathcal{L}}(i,j) + \big(1 - c_{\boldsymbol{\ell}}(i,j)\big)(1 - p_{\mathcal{L}}(i,j))\big]\Big] \\
&= \binom{n}{2}^{-1} \sum_{i<j} \Big[\mathbb{E}[c_{\boldsymbol{\ell}}(i,j)] p_{\mathcal{L}}(i,j) + \mathbb{E}[(1 - c_{\boldsymbol{\ell}}(i,j))](1 - p_{\mathcal{L}}(i,j))\Big] \\
&= \binom{n}{2}^{-1} \sum_{i<j} \left[p'_{ij} p_{\mathcal{L}}(i,j) + (1 - p'_{ij})(1 - p_{\mathcal{L}}(i,j))\right].
\end{aligned}$$

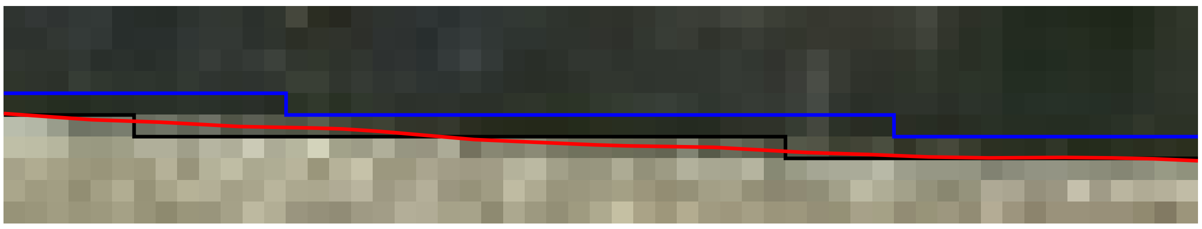
with $p'_{ij} = \mathbb{E}[c_{\boldsymbol{\ell}}(i,j)]$. The term $p'_{ij}$ is estimated from all ground truth segmentations in the database:

$$p'_{ij} \triangleq \frac{1}{M}\sum_{m=1}^{M}\frac{1}{|\mathcal{L}_m|}\sum_{\boldsymbol{\ell}\in\mathcal{L}_m} c_{\boldsymbol{\ell}}(i,j) \tag{4.7}$$

where $\mathcal{L}_m$ is the ground truth data of the $m$-th image in the database. The computation of $p'_{ij}$ is very expensive for a large dataset. Unnikrishnan et al. [2007] used 50,000 randomly chosen instances of $c_{\boldsymbol{\ell}}(i,j)$ instead. They reported negligible change in the results if more samples were used. The range of the indexes is now not longer constrained to the interval $[0,1]$ and can also be negative. For our experiments we encountered values ranging from $-1.24$ up to 0.98.

### Subpixel Boundaries

As the exact watershed transform from section 2.4 produces subpixel accurate segment boundaries it is necessary to convert the partition of the image-plane produced by the watersheds into a label-image. We take a simple solution to this problem: We assign an unique label to each face in the partition of the plane and construct a label-image $\boldsymbol{\ell}$ by assigning to $\boldsymbol{\ell}[\boldsymbol{p}]$ the label of the face that contains the point $\boldsymbol{p}$.


(a) *three boundary segments: two from manual segmentations (ground truth data) from different subjects (blue, and black) and one from exact watersheds after using a graph cut. As one can see the manual boundaries differ in accuracy.*

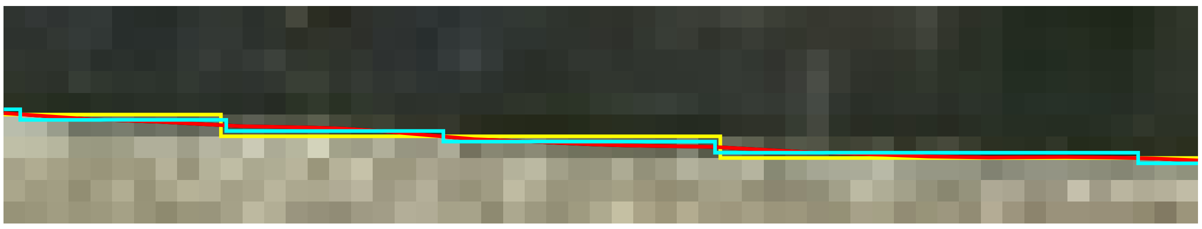

(b) *subpixel boundary (red) and same boundary segment with pixel accuracy (yellow) and doubled pixel accuracy (cyan). The NPR Index is only affected slightly to this loss in accuracy.*

**Figure 4.3:** *Part of an image from the Berkeley Image Database with subpixel segment boundary and manual segment boundaries.*

Certainly, this method means a loss of boundary accuracy which is actually the strength of subpixel accurate segmentation methods such as the exact watershed transform. However, this loss makes no big difference to the NPR-value of the resulting label-image: The boundary accuracy of the ground truth data from the Berkeley Image Database is poor in the sense that manual performed segmentation often differ near segment boundaries, see figure 4.3a. The Probabilistic RAND Index is very robust to this phenomenon [UNNIKRISHNAN et al. 2007] such that discretising subpixel boundaries does not matter as far as the comparison to the ground truth is concerned.

In order to confirm this we doubled the accuracy of the conversion step by applying the above described method not only at pixel positions $\boldsymbol{p}$ but also at subpixel positions $\boldsymbol{p}+(\pm 0.5, \pm 0.5)^\mathsf{T}$, see for example figure 4.3b. In this way we get doubled sized label images and compare them to the ground truth data appropriately scaled. The mean absolute difference of the NPR-values between these both methods on the watershed segmentations on the whole database is $8.02 \cdot 10^{-4}$ with variance $1.07 \cdot 10^{-6}$. For comparison, the mean absolute difference of all pairs of segmentations from the pixel precise method is 0.17 with variance $2.7 \cdot 10^{-2}$.

## 4.3 Evaluation Framework

In this section the evaluation framework for the various segmentation algorithms is described. For this we briefly review the segmentation scheme describe in section 2.3.4: At first we perform an initial segmentation for each image using either the watershed transform (section 2.4) or the mean shift algorithm (section 2.5). From this segmentation a region adjacency graph is derived with edge weights being either the average gradient along the segment boundary (averGrad), the difference of the color means (colorMean), or the earth mover distance (EMD), see section 3.3. These edge weights are non-linearly scaled using an exponential function with a scaling parameter $\beta$. The weighted graph is then recursively split using one of the graph cuts of sections 3.4 to 3.8 until a threshold parameter stop on the cut values is reached. Therefore, we have in summary $2 \cdot 3 \cdot 5 = 30$ different segmentation algorithms each adjustable with parameters.

### 4.3.1 Parameter Selection

Since every segmentation algorithm investigated here depends on several parameters it is necessary to determine the optimal values for them.

In order to make the evaluation manageable we select hand chosen parameters for the initial segmentation step. For the watershed segmentation we have to determine the scaling parameter $\sigma$ for the boundary indicator (see section 2.4). We choose a rather small $\sigma = 2.0$ to ensure that no potential boundaries are missed which results in 3000 segments on average (ranging from 1500 to 6000). For the mean shift we also have to select scaling parameters, one $\sigma_s$ for the spatial part and one $\sigma_f$ for the feature part (see section 2.5). Similarly, we choose $\sigma_s = 10.0$ and $\sigma_f = 5.0$ in order to ensure that the resulting segmentations miss as few as possible potential boundaries. For the mean shift the number of segments among different images is much more scattered ranging from 30 to 3000 segments.

Given an initial segmentation, a graph cut, and an edge weight function the final result only depends on the scaling parameter $\beta$ for the edge weights and the threshold parameter stop for stopping the recursion. For the parameter $\beta$ we chose 30 uniformly distributed values in the range $[5, 300]$ as we experienced values not being in this range to produce degenerate segmentations. For each of these scaling parameters the optimal stop parameter is computed using the method described below. Then we have 30 pairs of parameters for each image from which the pair with the best NPR-value is chosen yielding 100 parameters for the complete image set. For each of these pairs we do a leave-one-out cross-validation, that means each optimal parameter pair for a certain image is used to perform the segmentation on the other images. Finally we select the parameter pair which performs best on average.

### Parameter Search for Graph Cuts

For the step of finding an optimal stop parameter on the recursive segmentation scheme we use the same method for each cut except for the minimum cut which is described in the next section.

For a chosen scaling parameter the corresponding optimal stop parameter can be easily determined. To simplify the ongoing discussion a little bit we define the *cut value of a subgraph* as the cut value of an optimal normalized cut in this subgraph. The basic idea now is: Since the parameter stop is compared to the cut values of each sub graph created in course of the recursion and there can only be a finite number of recursion steps, the optimal parameter must be among the cut values produced during the recursion. We perform therefore a complete recursion for a region adjacency graph, i.e. until the subgraphs only contain one vertex and thus cannot be further split. In order determine which cut value corresponds to the best segmentation we have to determine the NPR indexes for each cut value. For this we can compute the NPR index each time a cut is applied, i.e. the current segmentation is further refined, and associate the cut value of this cut with the NPR index. In this way we get a series $((\lambda_1, s_1), \ldots, (\lambda_n, s_n))$ of cut value/NPR-index pairs from which one can select the pair with maximal NPR-index.

The only difficulty with this approach is that we have to execute the recursion such that each time a NPR-index $s_i$ is computed the corresponding cut value $\lambda_i$ is *valid* in the sense that if we set $\mathsf{stop} = \lambda_i$ we get the same segmentation for which $s_i$ has been computed and this is only the case if $\lambda_j \leq \lambda_i$ for $j < i$ and $\lambda_j > \lambda_i$ for $j > i$ holds.

In order to achieve this we actually perform the recursion in an ordered way. More precisely, starting with the region adjacency graph from an initial segmentation, every computed subgraph is put into a priority queue according to its cut value, i.e., the next item in the queue is always the subgraph with the smallest cut value. The subgraphs in the queue are successively drawn and their connected components, after removing all edges in the optimal cuts for the graphs, are pushed as new items into the queue. The cut value $\lambda_i$ of each drawn subgraph is compared to a variable $\lambda_{\mathsf{max}}$ which holds the maximum cut value of all drawn subgraphs. If $\lambda_i > \lambda_{\mathsf{max}}$ we output $\lambda_{\mathsf{max}}$ as a valid cut value, set $\lambda_{\mathsf{max}} = \lambda_i$ and continue in this manner until all remaining subgraphs cannot be further split. As each draw from the priority queue corresponds to a split of the associated subgraph, the change in the current segmentation can be concerned by updating the current NPR according to a split operation of the contingency table (see implementation notes in section 4.2). The last NPR index before the valid cut value $\lambda_{i_{j+1}}$ is then the correct NPR index of $\lambda_{i_j}$.

### Parameter Search for Minimum Cut

As we are not using the recursive segmentation scheme for the minimum cut but instead employ a Gomory-Hu tree instead for forming the final segmentation we also use a different parameter search for determining the optimal stop parameter. More precisely, after building the Gomory-Hu tree for a certain graph at hand the edges of the tree are sorted with decreasing edge weights. We first remove all edges from the Gomory-Hu tree and then successively add edges in order of decreasing weights. Reinserting an edge into the tree then corresponds to a merge of two regions in the initial segmentation. This way one gets a series of coarser becoming segmentations starting with the initial segmentation and ending in the segmentation with one big region (e.g. the whole image). Note that this exactly corresponds to the reversed clustering scheme as described in section 3.4.

The corresponding NPR-values can be computed using the merge operation on the contingency table (see above). The cut value with maximum NPR-value is then the optimal parameter for that image with respect to the scaling parameter of the edge weights used. The cross validation is then performed in the same manner as described above.

## 4.4 Evaluation Results

We now turn to a detailed discussion of the evaluation results for the various instances of the segmentation scheme. We have two NPR indexes for each image and each instance of the segmentation scheme; one index gained from the optimal parameter-search as described in section 4.3 and one index from the parameters performing best on all images on average after cross-validation. A listing of these parameters from cross-validation can be seen in table A.1. As we have chosen a subset of 100 images from the Berkeley Image Database, this makes $2 \cdot 100$ NPR indexes for each instance of the segmentation scheme. For the rest of this section we concentrate on the 100 indexes found by cross-validation. A NPR index is in the following also called (NPR) *score*.

We present statistics of the scores and examples of the corresponding segmentation results. Whenever we compare two different instances of the segmentation scheme, we not only compare their mean NPR scores but also apply two well-known statistical tests on the both distributions to see if the difference are actually significant: The paired t-test and its nonparametric counterpart, the Wilcoxon signed rank tests (see for example [Meintrup and Schäffle 2005]). For the significant level we use the standard value $\alpha = 0.05$.

In order to make the following discussion more observable we put most of the statistics and segmentation results in the appendix of this thesis. These are:

- Figure A.1 and Figure A.2 contain several plots. Each plot shows the mapping of a certain value for the scaling parameter $\beta$ for the edge weights to the mean NPR index found by parameter search over all images for that $\beta$.
- We give several histograms in Figure A.4 (page 106) and figure A.5 (page 107) in order to give an overview on how the NPR indexes are distributed for each instance of the segmentation scheme. The green histograms in the first rows present the best NPR score on each image found by parameter search. Red histograms present the NPR scores for the parameter which has been found during cross-validation. From this, one can also see how the selection of one parameter for all images impacts on the NPR scores. In figure A.3 the two histograms of the NPR indexes for the initial segmentation steps are given.
- Table A.2 gives a more compact overview through several simple statistics of the NPR scores, i.e. the mean, minimal, and maximal NPR-score for the various segmentation algorithms. The last column in each table gives the fraction of segments of the inital segmentation which are also in the final segmentation; from this one can get an idea on how strong a certain algorithm is biased to over- or under-segmentation.
- Table A.4 gives a ranking of the cuts for each combination of initial segmentation step and edge weight function. The entry in the first row is always the cut which performs best on average. The results of some statistical tests are contained in the remaining columns in order to verify that the ranking is also significant. Especially, the scores of each particular row are compared to the first and the preceeding row. The last two columns compare the NPR scores

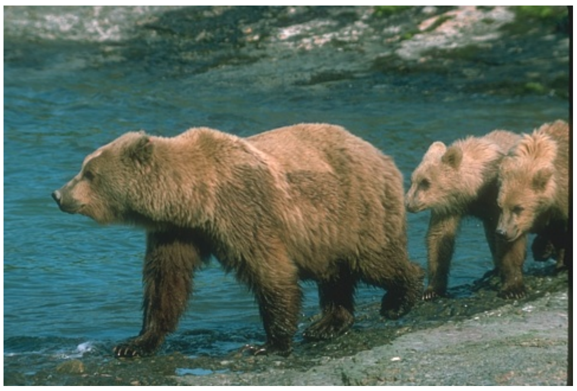
with the scores of the initial segmentation step in order to see if the improvement of the initial segmentation by using a cut is significant.

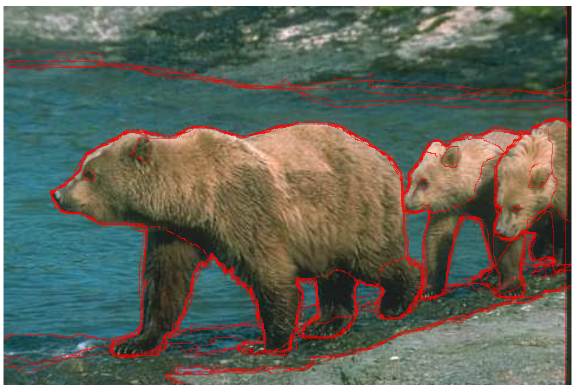
- For comparison on how changing from mean shift to watersheds as initial segmentation step influences the NPR scores see table A.3 on page 109. As before, we also inserted the results from the statistical tests.
- A similar comparison is given for the edge weights in table A.5.
- Finally, in section A.3 (page 113-130) one can see several segmentation results on hand selected images from which we think that they are a representative subset of all segmentation results. We also included the results for one of these images in figure 4.4 for a first glimpse.

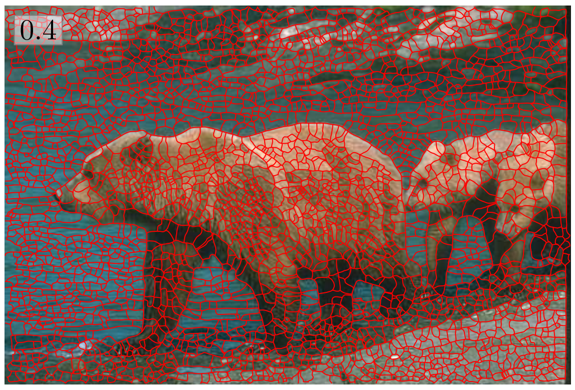
Whenever these content is needed for the ongoing discussion we insert the appropriate reference to it. From these statistics, comparisons, and segmentation examples we draw the following conclusions:

1. The usage of graph cuts always improves the quality of the initial segmentation on average. However, the improvement does not seem to be always significant.
2. It is not possible to clearly identify an instance of the segmentation scheme which is superior to all other ones.
3. The minimum cut is inferior if compared to the other cuts.
4. The cuts being normalized by the weighted degree – the foreground, the isoperimetric, and the normalized cut – produce comparable results with respect to the NPR scores and the number of segments in the final result.
5. For the different selections on the edge weights we found that the average gradient magnitude (averGrad) performs significantly worse if watersheds are used for the initial segmentation step.
6. The usage of the earth mover distance for the edge weights (EMD) does not lead to significantly better results and is comparable to the color mean (colorMean).
7. The selection of the mean shift as initial segmentation can be problematic for the cuts being normalized by the weighted degree. Except for the minimum cut using watersheds leads to better results if the averGrad is not used for the edge weights.

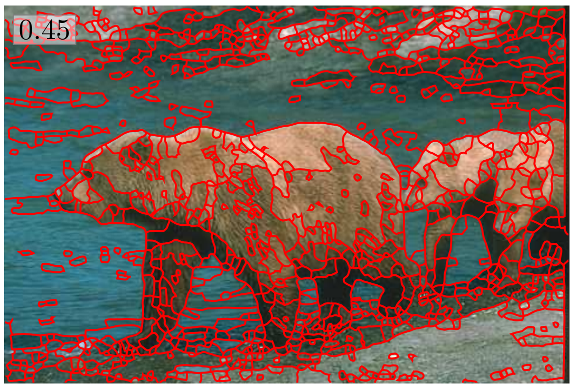
Statement 1 to 3 are the most important ones as they address the main question we formulated at the beginning of this work. While it is not possible to identify a certain instance being superior we can be more specific on the inherent characteristics of the different cuts. So, we give a final discussion on each cut and some remarks on the edge weights below.


(a) *original image*

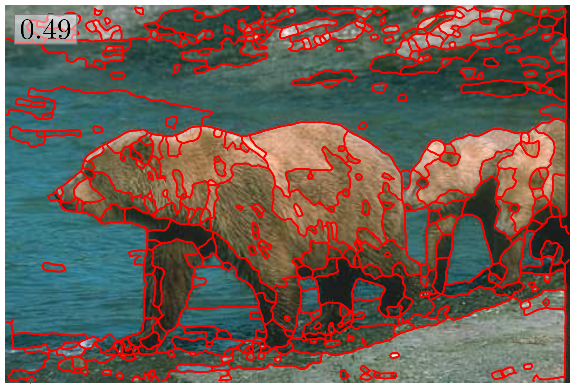

(b) *ground truth*



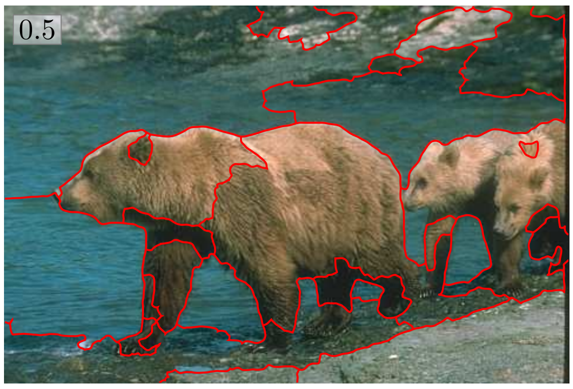
(c) *watershed segmentation*



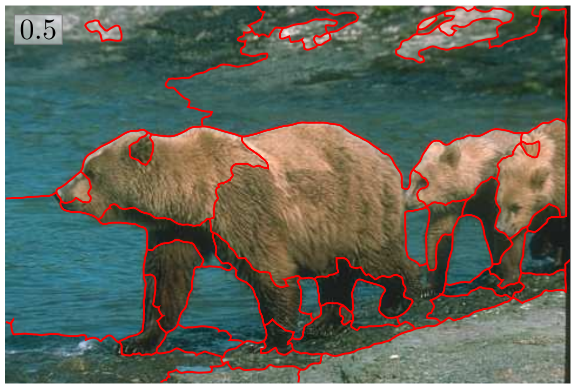
(d) *minimum cut*



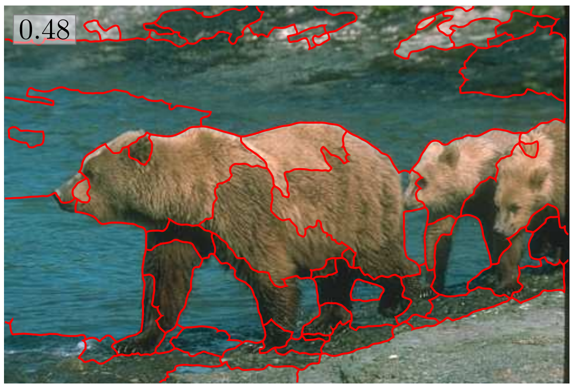
(e) *mean cut*



(f) *foreground cut*



(g) *isoperimetric cut*



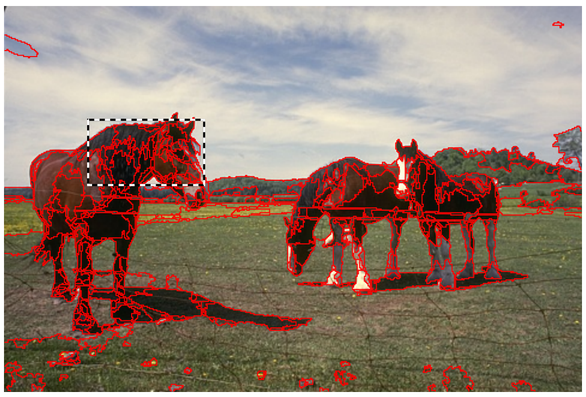
(h) *normalized cut*

**Figure 4.4:** *segmentation results of all cuts on one image for watersheds as initial segmentation and* colorMean *for the edge weights*

### Minimum Cut and Mean Cut

The minimum cut (as described in section 3.4) is the simplest graph cut investigated here. It can be seen as the cut with the worst overall performance. Its NPR scores are often significant smaller than the scores of the other cuts. We already discussed the bias of the minimum cut to produce strong over-segmentation in section 3.2.1.1 as the costs of a cut grow both with increasing edge weights and number of edges in the cut. Nevertheless, it can be still used to remove much of the over-segmentation of the watershed transform in regions being quite homogeneously colored. For example for the image on page 116 the minimum cut produces a very high score. The high ranking of the minimum cut together with mean shift in table A.4 can be explained by the facts that the mean shift already produces good initial segmentations and that the weighted degree normalized cuts perform systematically worse due to the "single vertex" problem (see below). In fact, for the image on page 122 on can see an example where the minimum cut removes almost no edges at all and the NPR score is mainly due to the mean shift.

The mean cut successfully attacks the drawbacks of the minimum cut by normalizing the costs of the cut by the total length of the produced boundary. It is therefore less biased to over-segmentation and can also create segments with long boundaries. Especially, segments having a large perimeter and a small area, which are problematic for all other cuts, can be produced with the mean cut. However, the mean cut also often produces very small isolated segments as it can be seen from the example segmentations. An example on how the mean cut successfully overcomes the problems of the minimum cut can be seen in figure 4.5: The rectangle in figure 4.5a enclosed a region where the minimum cut possess over-segmentation; the arrow in figure 4.5b marks a very long boundary which is not problematic for the mean cut but for the minimum cut. However, figure 4.5 is also an example where the minimum cut has a higher NPR score as the mean cut since the marked segment is not in the ground truth data.


(a) *mean shift with minimum cut and* colorMean

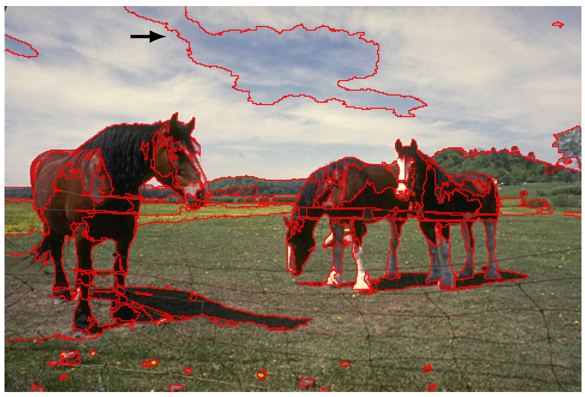

(b) *mean shift with mean cut and* colorMean

**Figure 4.5:** *Final segmentation result from one image for the minimum cut and the mean cut.*

### Foreground, Isoperimetric, and Normalized Cut

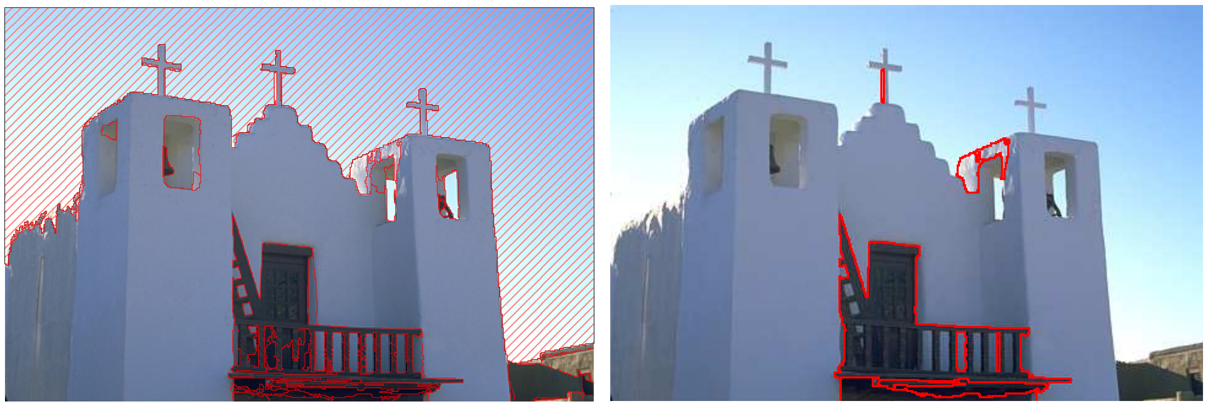
We now turn to a more detailed discussion of the foreground, isoperimetric, and normalized cut. We already noted that the performance of these cuts is very comparable. Especially, the foreground cut is not inferior if compared to the other two balanced cuts. Recall, that the foreground cut is only normalized by the total vertex weight of one component in the cut whereas the isoperimetric and the normalized cut take both components into account. The number of segments in the final segmentations are also very similar especially for the watershed transform as initial segmentation step. We would like to finish our discussion with two aspects which especially attracted our attention: The "single vertex problem" and "shortcuts".

As all of these three cuts use the weighted degree for the vertex weight function all of them share the same problem which especially becomes evident with the mean shift as initial segmentation step: If we look at the definitions of the three cuts with the weighted degree inserted for the vertex weight function, i.e.

$$\begin{aligned}
\mathsf{fcut}(A, A^{\mathsf{c}}) &= \frac{\omega(A, A^{\mathsf{c}})}{\omega(A, V(G))} \\
\mathsf{icut}(A, A^{\mathsf{c}}) &= \frac{\omega(A, A^{\mathsf{c}})}{\mathsf{min}\,\{\omega(A, V(G)), \omega(A^{\mathsf{c}}, V(G))\}} \\
\mathsf{ncut}(A, A^{\mathsf{c}}) &= \frac{\omega(A, A^{\mathsf{c}})}{\omega(A, V(G))} + \frac{\omega(A, A^{\mathsf{c}})}{\omega(A^{\mathsf{c}}, V(G))},
\end{aligned}$$

we can see that cuts having only one vertex in one of their components can be problematic. More precisely, if for the foreground cut the component $A$ only contains one vertex the cut value evaluates to $\mathsf{fcut}(A, A^{\mathsf{c}}) = 1$. Similarly, for the isoperimetric cut the cut value also evaluates to $\mathsf{icut}(A, A^{\mathsf{c}}) = 1$ if the weighted degree for this vertex is smaller than the total weighted degree of the other vertices. For the normalized cut we have $\mathsf{ncut}(A, A^{\mathsf{c}}) > 1$ if $A$ or $A^{\mathsf{c}}$ has only one vertex. For comparison, the $\mathsf{stop}$ parameter found by parameter search is much smaller than 1 for all cuts (cf. table A.1). Figure 4.6 shows an example where this can be problematic: The mean shift method has been used for the initial segmentation and successfully identified large homogeneous regions. For example, a cut which has only the region hatched in figure 4.6a in one component is very costly for the isoperimetric cut such that the final segmentation result (figure 4.6b) is worse than the result for the minimum cut (figure 4.6a).

Another characteristic of these three cuts is the tendency of producing what we call *shortcuts*. The meaning of this can be seen in figure 4.7. The segment boundary marked runs through a region in the image where almost no "real" boundary information is present. The reason for this is that during the recursive scheme subgraphs can occur where the normalizing factor dominates: For example, in figure 4.7a the subgraph corresponding to the sky is created after several applications of the isoperimetric cut; it is then split into the two components indicated with the hatched patterns. This is only possible as the total length of the edges in the cut (marked with an arrow) is very short and the smaller one of the two components has still a high

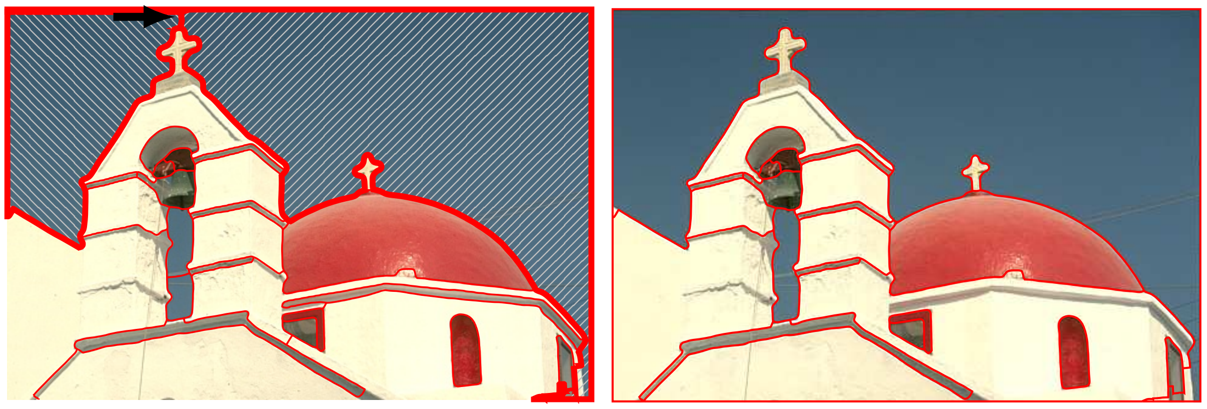


(a) *minimum cut* (b) *isoperimetric cut*

**Figure 4.6:** *Example for isolated vertex problem. Both segmentations have been generated with mean shift as initial segmentation step. The minimum cut successfully segments the big segment (hatched on the left image).*

total vertex weight. If the church spire were closer to the left image border, the cut would have a higher cut value. While for example the normalized cut does not show this behavior for this image (figure 4.7b) similar examples can be found for both the normalized and the foreground cut.

Also note that the segmentations shown in figure 4.7b match completely except for the discussed cut. However, the NPR score for the isoperimetric cut is much smaller than for the normalized cut.



(a) *isoperimetric cut,* NPR = 0.55 (b) *normalized cut,* NPR = 0.74

**Figure 4.7:** *Example for "shortcut" of isoperimetric cut*

## Edge Weights

For the selection of the edge weights the most interesting aspect is whether the usage of sophisticated weights improves the overall performance. In order to investigate this we decided to use the earth mover distance. We already noted above (statement 6) that we have not experienced a significant improved if compared to the rather simple edge weights. Especially, the results are often similar to the results for colorMean. This can also be seen from the various segmentation examples we included in section A.3.

We also already noted that the selection of averGrad can be problematic for the watersheds. This becomes especially apparent if one looks at the segmentation results of the mean cut, see figure 4.8 for an example. As one can see, there are many small satellite segments near high contrast segment boundaries. A possible explanation for this is that the flowlines running orthogonally to such a high contrast boundary still have a certain part on this boundary such that the average gradient magnitude for them becomes high enough.

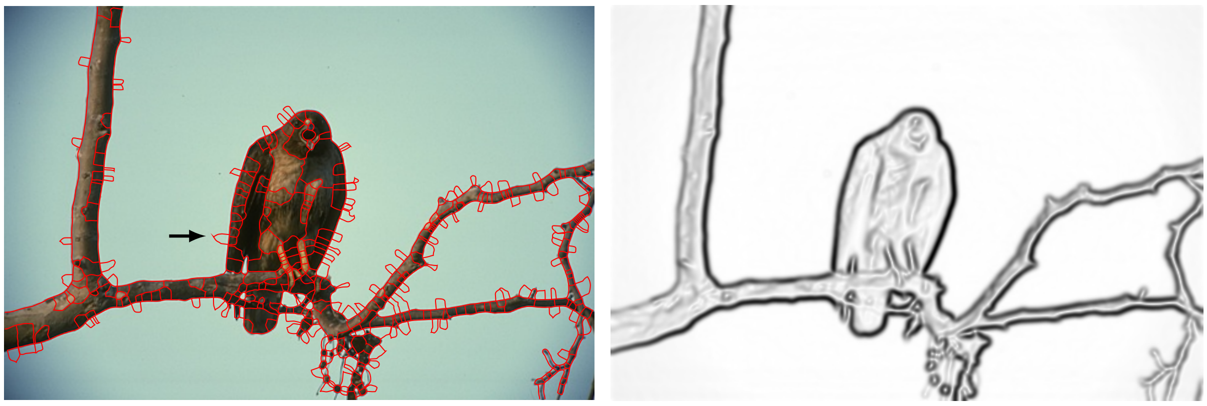


**Figure 4.8:** *Segmentation result for the mean cut with edge weights from* averGrad.

# 5 Summary and Future Work

In this chapter we give a summary of our work and present some possible extension for future work.

## 5.1 Summary

In this thesis we described the image segmentation problem and an energy function based approach to it. Using energy function for segmentation has been proposed in the literature many times. Especially, the graph cuts methods are very popular as they can be used for effectively solving binary optimization problems which can then be extended with a recursive scheme in order to form a segmentation consisting out of more than two segments.

We gave an overview on graph cuts related work in computer vision and identified a certain class of graph cuts which can be seen as normalized versions of the minimum cut. We presented and implemented algorithms for five different normalized cuts. Except for the foreground cut, we did this on the basis of the relevant publications. For the foreground cut we worked out the details of an algorithm due to SADISH RAO which has not been published so far.

In contrast to the classical application of cuts employing a graph structure which is directly derived from the pixels of an image we use an initial segmentation step from which the graph is build. This way the encountered graphs are much smaller and can have more sophisticated edge weights. We used the exact watershed transform of MEINE and KÖTHE [2006] and the mean shift method of COMANICIU and MEER [2002] for the initial segmentation step.

For the edge weights, we used the average gradient magnitude along the common boundary of two neighboring segments of the initial segmentation, the difference of the mean color of those segments, and the earth mover distance. The later can be seen as a rather complex similarity measure if compared to the first two.

The main intention of this thesis was to investigate how the different selections of the initial segmentation step, edge weights, and graph cuts impacts on the segmentation quality. For assessing the segmentation quality we used the BERKELEY Image Database in connection with the Normalized Probabilistic RAND Index of UNNIKRISHNAN et al. [2007]. We performed an exhaustive search for the optimal parameter, i.e. the scaling parameter for the edge weights and the stop parameter for the recursive scheme, by means of a leave-one-out cross-validation.

For the evaluation results we presented and discussed several statistics of NPR scores along with several segmentation examples. From this we found that all cuts are capable of significantly improving the initial segmentation. This means that our basic segmentation scheme indeed is useful for attacking the segmentation problem. We also found that the different cuts produce comparable segmentation results with respect to the NPR scores except for the minimum cut which is inferior as expected. This also means that the foreground cut is not inferior if compared to the isoperimetric and the normalized cut where the later two correspond to NP-hard optimization problems. We further found that the common usage of the weighted degree for the vertex weight function can be problematic especially for initial segmentations being less over-segmented.

For the edge weights we found that the selection has no big influence on the segmentation quality except for average gradient magnitude in connection with the exact watershed transform. Especially, the earth mover distance, being quite expensive with respect to the computation time needed, does not lead to better segmentations within our setting.

## 5.2 Future Work

Primarily, possible extension and modifications for future work can be done for all aspects of the basic segmentation scheme, i.e.

1 for the initial segmentation step,

2 for the edge and vertex weights,

3 for the graph cut used, and

4 for the stopping criterion.

For the rest of this work we would like to be more specific on these aspects.

1 **Initial Segmentations** As the segmentation scheme does not depend on a specific selection for the initial segmentation step all kinds of methods available can be used for it. It is even possible to add a further processing step to the segmentation scheme, e.g. reuse its outcome as the initial segmentation for further improving the segmentation. For example for the watershed segmentation one could try to first remove a certain amount of the over-segmentation in order to get more expressive edge weights from the coarser segmentation.

2 **Edge and Vertex Weights** Unfortunately, we were not able to investigate two kinds of edge weights from which we think that they could improve the quality of the final segmentations: Texture and geometry related edge weights.

The earth mover distance, which we used as the most complex similarity measure, only analyzes the distribution of the colors contained in a region of the initial segmentation. Texture based measures can be used to also incorporate the spatial distribution of the pixel elements.

Geometry based similarity measures are especially interesting in connection with the exact watershed transform. Observing the image in figure 5.1 it is easy for a human observer to identify prominent object boundaries although there is no color information available but only the exact watersheds. Measures rating the geometrical saliency of polygonal arcs have already been discussed in the literature many times; an overview being also in the context of exact watersheds can be found in [Meine 2008, Chapter 5, p. 129 et seqq]. To be usable for the graph cuts discussed here the saliency of each edge of a partition of the plane must be computed in advance and therefore cannot be a complex function of all edges being contained in a cut. However, this would be desirable for geometrical measures, e.g. the curvature of the common boundary. If it is nevertheless possible to define appropriate saliency measures usable for graph cuts would be an interesting object for future work. We also discuss in brevity an extension of a graph cut method below which allows to overcome this limitation.

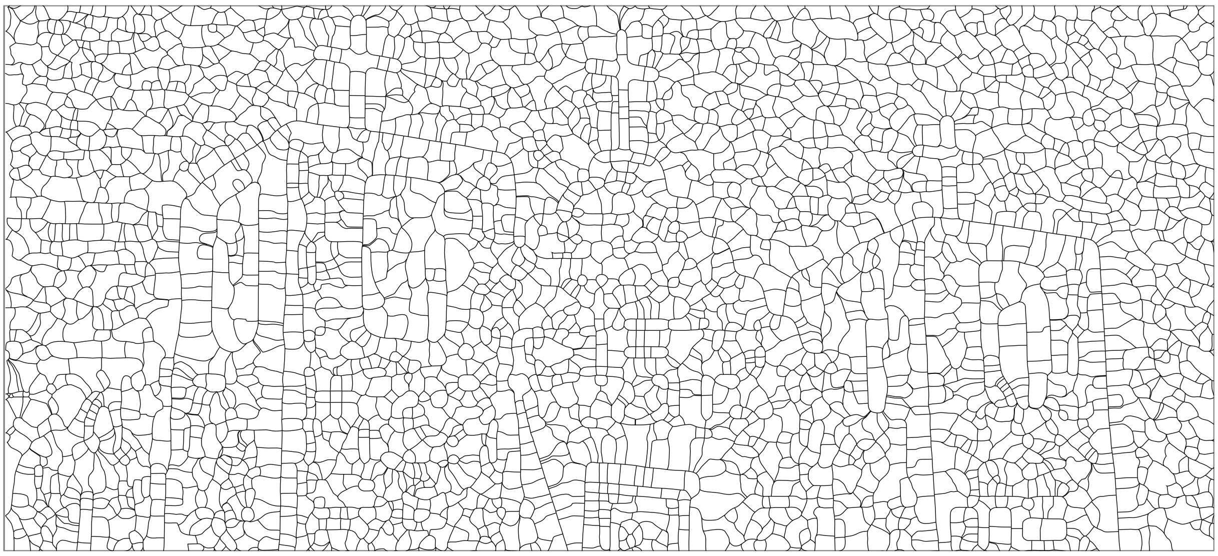


**Figure 5.1:** *Example for geometrical saliency of watershed boundaries; prominent boundaries immediately "pop-out" to a human observer (example is based on [Meine 2008, Chapter 5, p. 129 et seqq])*

As a further extension, one can also combine all kinds of different edges weights into a single edge weight function. Especially, one can combine feature and geometry based weights in order to improve their discrepancy power. The importance of each different edge weight could than be adjusted with appropriate selected scaling factors.

For the graph cuts using a vertex weight function, i.e. the foreground, the isoperimetric, and the normalized cut, we found that using the weighted degree for this can be problematic. In order to overcome this problem further work on choosing other vertex weight functions is needed.

3 **Graph Cuts** Although, an extensive search in the present literature has been our intention there are certainly other publications on normalized graph cuts related clustering methods which we have not found but could also be used in our segmentation scheme. Moreover, we have found two other cuts from which we think that they are worth to be investigated but we were not able to include due to time constraints: The work of Keuchel [2004]; Keuchel et al. [2003], and an extension of the mean cut presented in [Wang et al. 2005].

Keuchel et al. used techniques from semi definite programming to solve the minimum cut problem with additional balancing constraints. The usefulness of this approach has been demonstrated among others within the tasks of binary image restoration, line grouping, and image segmentation. Using balancing constraints on the resulting components, e.g. constraints on the size or total vertex weight, of a bi-partition is an alternative way to overcome the bias of the minimum cut to small regions. If it is possible to increase the quality of our segmentations this way would be an interesting question to be investigated.

Wang et al. [2005] show how the minimum mean cut algorithm (as presented in section 3.5) can be used to connect line segments produced by an edge detector into globally optimal contours. The corresponding energy function has a similar form as with the minimum mean cut:

$$\omega(C) = \frac{\sum_{e\in C} \omega_1(e) + \omega_C(e)}{\sum_{e\in C} \omega_2(e)} \tag{5.1}$$

where $C$ is a set of edges forming a cycle. The edge weight functions $\omega_1$ and $\omega_2$ can only be simple functions mapping each edge to a beforehand selected weight as it is the case with all edge weight functions used in this work. However, $\omega_C$ can be a more complex edge weight function whose function value for an edge $e$ also depends on the two edges preceding and succeeding $e$ in the cycle $C$. More precisely, for $C = (e_1, \dots, e_n)$ it can have the form

$$\omega_C(e_i) = f(e_{i-1 \,\mathsf{mod}\, n}, e_i) + f(e_i, e_{i+1 \,\mathsf{mod}\, n}) \tag{5.2}$$

with $f$ being an arbitrary function defined on pairs of edges. Instead of deriving a region adjacency graph from a partition of the plane $P$ one can construct a *boundary graph* encoding the topology of the vertices and edges in $P$ (that is, $P$ is an embedding of the boundary graph, see definition 21). The idea is that a cycle optimizing equation (5.2) then forms a perceptually appealing bi-partition as with the graph cuts if the edge weights are defined appropriately. The usefulness of this becomes clear if one likes to incorporate geometric saliency into the energy function: The edge weight functions $\omega_1$ and $\omega_2$ can be defined analogously as with the minimum mean cut, i.e. $\omega_1$ can be a function of the pairwise similarity of faces sharing an edge in $P$ and $\omega_2$ can be function of the length of this common edge. For $\omega_C$, $f$ can be a function mapping pairs of edges to a high value if they are good continuation from each other in a geometric sense.

Additionally, we have already mentioned some other possible aspects for further work before: In section 3.6 we used a simple heuristic for the seed selection for the foreground cut problem. The impact on the optimality can be investigated by trying each vertex of the region adjacency graph in turn and by comparing the best bi-partition with the one gained from the heuristic. In section 3.6 we mentioned the optimal algorithm for the isoperimetric cut problem of Park and Phillips [1993] for planar graphs. The algorithm is not used here as we think that it is too laborious to implement and too expensive for the graph sizes encounter here. At least a comparative study on how close the approximation scheme of Grady and Schwartz [2006*a*] is to the optimal solution would be possible with this algorithm nevertheless.

4 **Stopping criteria** Finally we would like to mention possible modification on the stopping criterion for the recursive application of the graph cuts. The quality of the final segmentations also depends on the selection of the stop parameter which is used as an upper bound for the cut values and therefore stops the recursion. Alternatively, one could use the stop parameter on different similarity measures defined on the components of a cut. For this one could use all kinds of measures ranging from measures incorporating all pixels in the both components over measures using histograms or signatures derived from the pixels to statistical tests which are only defined on the mean and variance of the color values. As a possible advantage, one could try to overcome the problem of some cuts to produce shortcuts in this way (cf. section 4.4).

# A Evaluation Results

| | | | $\beta$ | stop |
|---|---|---|---|---|
| cut | mean shift | averGrad | 105 | $4.37 \cdot 10^{-2}$ |
| | | colorMean | 155 | $1.98 \cdot 10^{-2}$ |
| | | EMD | 105 | 0.11 |
| | watersheds | averGrad | 145 | $2.47 \cdot 10^{-2}$ |
| | | colorMean | 135 | $2.59 \cdot 10^{-2}$ |
| | | EMD | 105 | $7.84 \cdot 10^{-2}$ |
| mcut | mean shift | averGrad | 31 | 0.23 |
| | | colorMean | 31 | 0.37 |
| | | EMD | 31 | 0.47 |
| | watersheds | averGrad | 16 | 0.36 |
| | | colorMean | 36 | 0.17 |
| | | EMD | 36 | 0.14 |
| fcut | mean shift | averGrad | 65 | $9.99 \cdot 10^{-2}$ |
| | | colorMean | 65 | $5.36 \cdot 10^{-2}$ |
| | | EMD | 55 | $2.96 \cdot 10^{-2}$ |
| | watersheds | averGrad | 45 | $2.59 \cdot 10^{-3}$ |
| | | colorMean | 95 | $4.81 \cdot 10^{-4}$ |
| | | EMD | 85 | $6.55 \cdot 10^{-4}$ |
| icut | mean shift | averGrad | 35 | 0.3 |
| | | colorMean | 45 | $4.99 \cdot 10^{-2}$ |
| | | EMD | 75 | $4.57 \cdot 10^{-2}$ |
| | watersheds | averGrad | 65 | $6.02 \cdot 10^{-3}$ |
| | | colorMean | 95 | $7.86 \cdot 10^{-4}$ |
| | | EMD | 75 | $2.17 \cdot 10^{-3}$ |
| ncut | mean shift | averGrad | 55 | 0.35 |
| | | colorMean | 65 | 0.14 |
| | | EMD | 65 | $7.26 \cdot 10^{-2}$ |
| | watersheds | averGrad | 25 | $1.34 \cdot 10^{-2}$ |
| | | colorMean | 135 | $5.95 \cdot 10^{-4}$ |
| | | EMD | 115 | $6.26 \cdot 10^{-4}$ |

**Table A.1:** *Parameters found by cross validation for each instance if the basic segmentation scheme. The parameter $\beta$ is the scaling factor for the edge weights and* stop *is stop parameter for the recursion.*

## A.1 Statistics

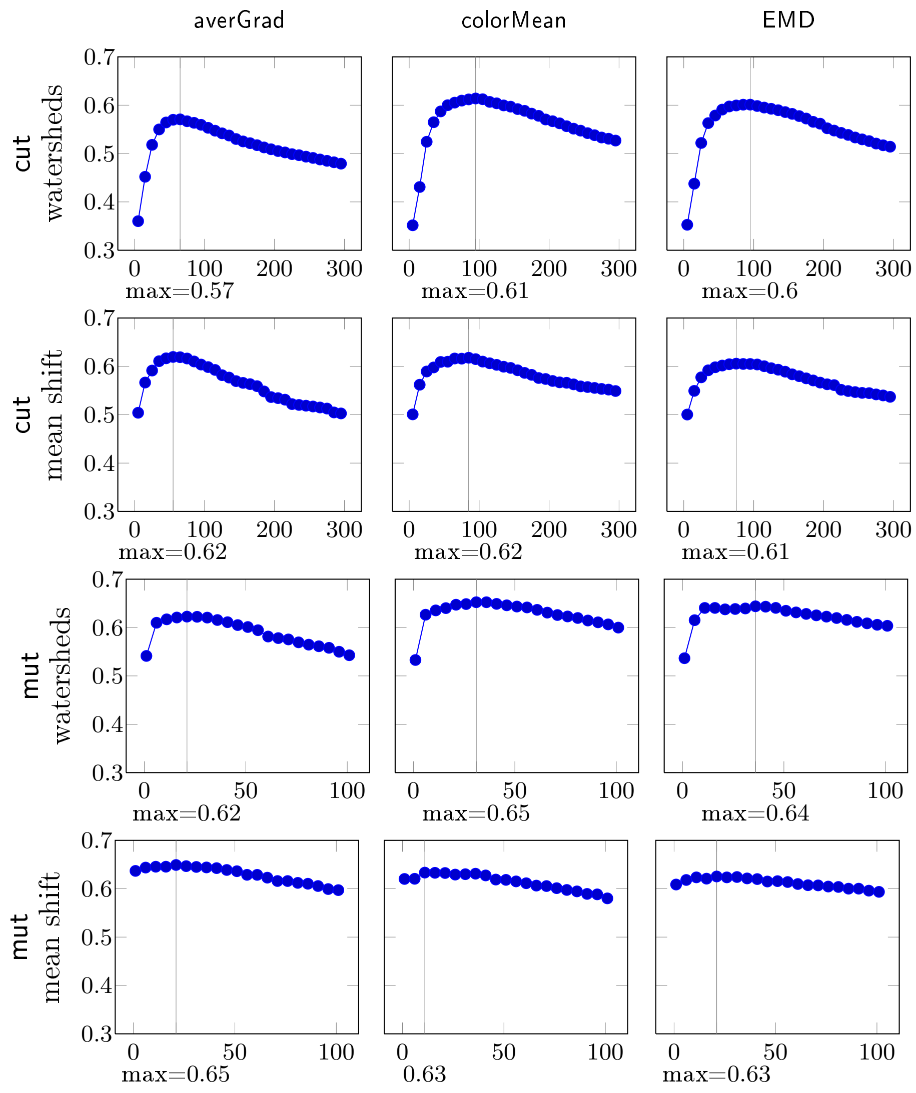


**Figure A.1:** *Each plot shows for a certain instance of the segmentation scheme the mapping of a scaling parameter $\beta$ for the edge weights (x axis) to the mean of the NPR-indexes over all images (y axis) found by optimal parameter search for $\beta$; here for the minimum cut and mean cut.*

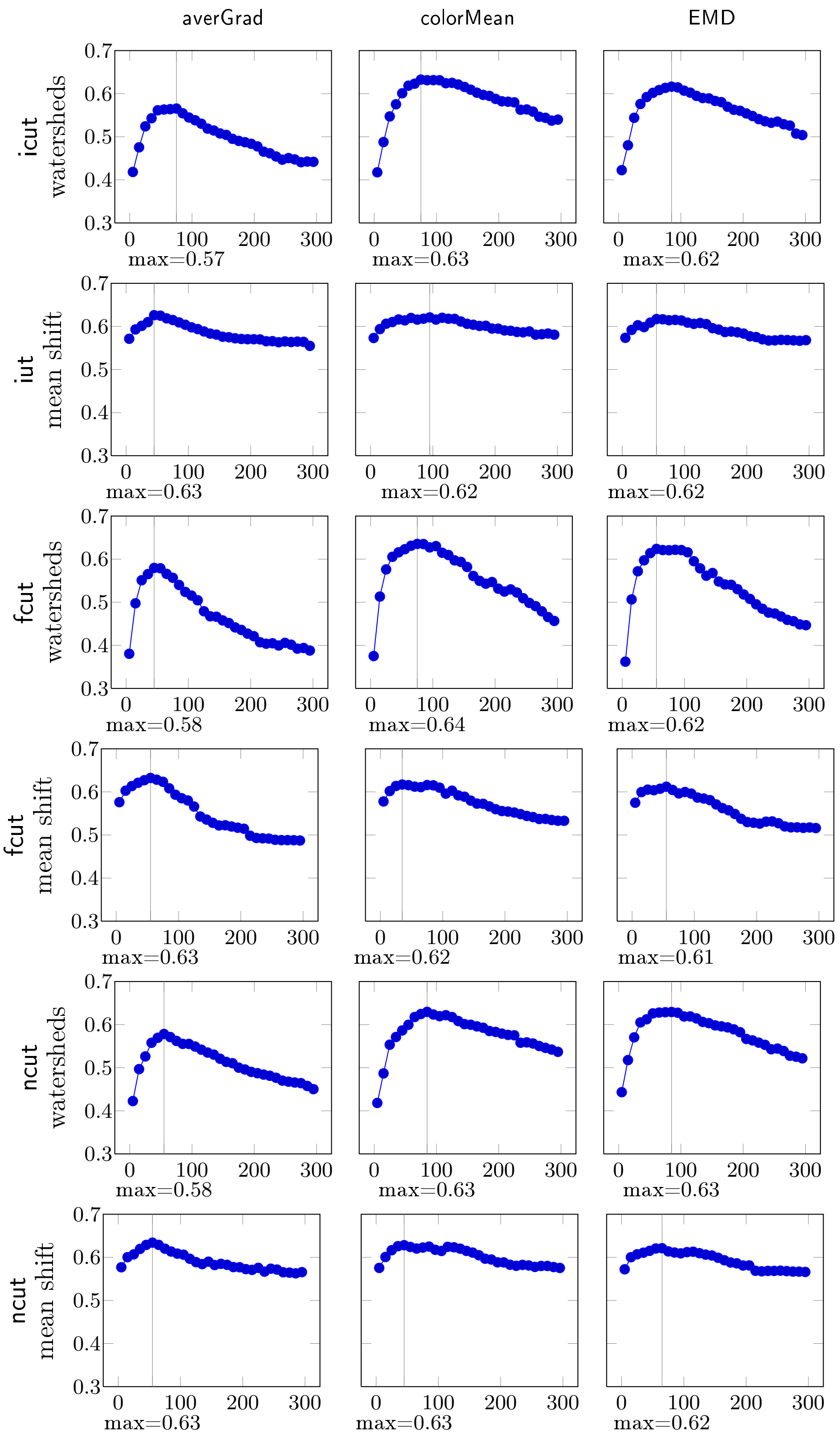
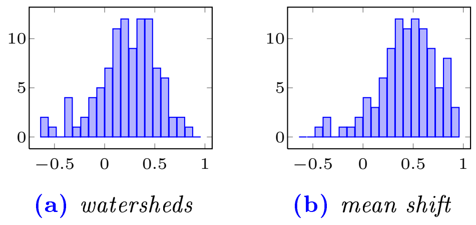


**Figure A.2:** *Each plot shows for a certain instance of the segmentation scheme the mapping of a scaling parameter $\beta$ for the edge weights (x axis) to the mean of the NPR-indexes over all images (y axis) found by optimal parameter search for $\beta$; here for the foreground, isoperimetric, and normalized cut.*



|  | mean | variance | max | min |
|---|---|---|---|---|
| (a) | 0.26 | $8.67 \cdot 10^{-2}$ | 0.86 | −0.58 |
| (b) | 0.48 | $8.12 \cdot 10^{-2}$ | 0.98 | −0.43 |

**Figure A.3:** *Histograms and simple statistics of the NPR indexes for the watershed transform and the mean shift method.*

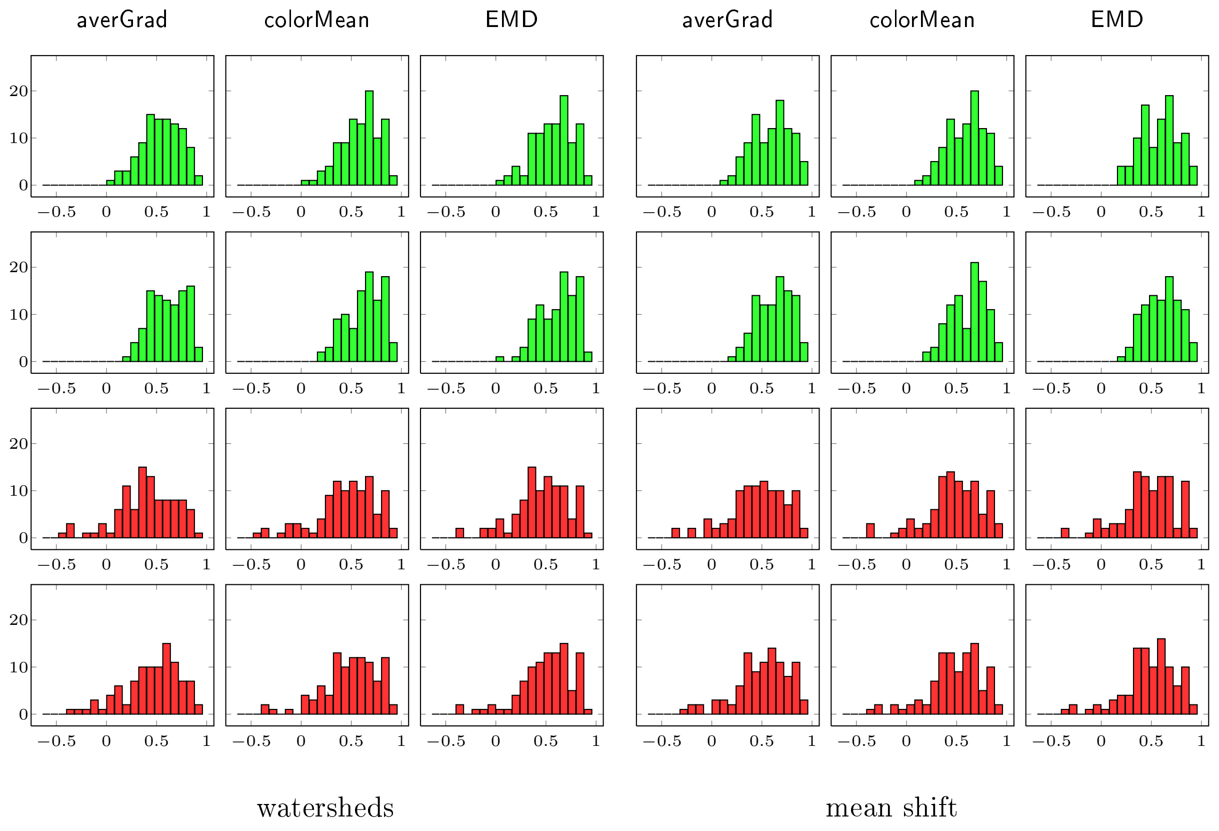


**Figure A.4:** *Histograms for the NPR scores for the minimum cut and the mean cut. The first two rows are the histograms of the best NPR score for each image: minimum cut (**first row**) and mean cut (**second row**). The remaining rows are the histograms for the best parameter from cross-validation: minimum cut (**third row**) and mean cut (**fourth row**).*

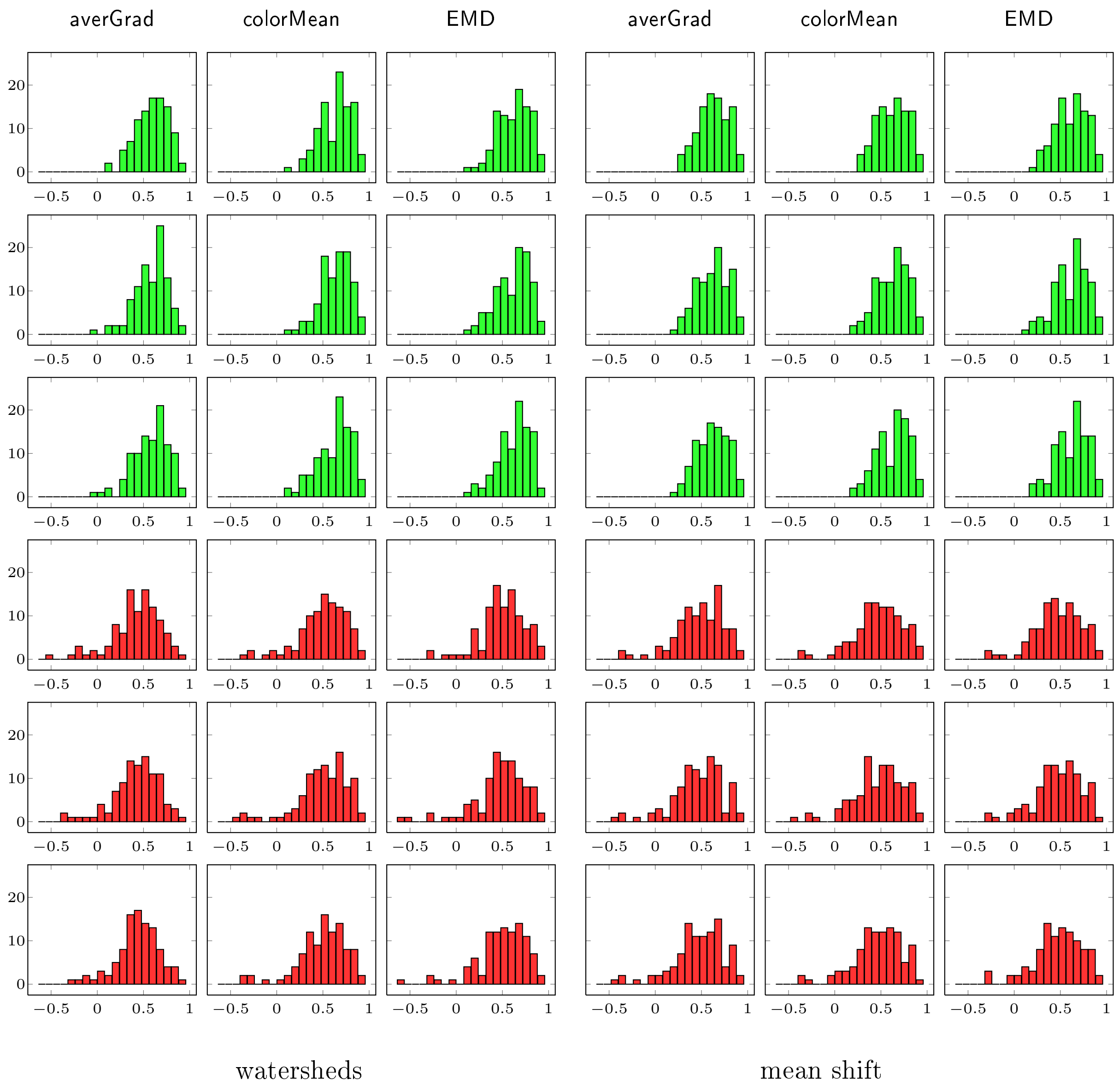


**Figure A.5:** *Histograms for the NPR scores for the foreground cut (**first row**), the isoperimetric cut (**second row**), and the normalized cut (**third row**). The first three rows are the histograms of the best NPR score for each image; the remaining rows are the histograms for the best parameter from cross-validation.*

| | mean | | variance | | maximum | | minimum | | segs |
|---|---|---|---|---|---|---|---|---|---|
| *(watersheds)* | best | cross | best | cross | best | cross | best | cross | % |
| averGrad | 0.58 | 0.44 | $3.89e{-2}$ | $8.34e{-2}$ | 0.96 | 0.95 | 0.11 | −0.43 | 55.99 |
| colorMean | 0.63 | 0.49 | $3.60e{-2}$ | $8.53e{-2}$ | 0.97 | 0.97 | 0.12 | −0.42 | 25.59 |
| EMD | 0.61 | 0.50 | $3.80e{-2}$ | $7.04e{-2}$ | 0.97 | 0.97 | 0.12 | −0.32 | 30.43 |
| *(mean shift)* | | | | | | | | | |
| averGrad | 0.63 | 0.50 | $3.67e{-2}$ | $7.74e{-2}$ | 0.98 | 0.94 | 0.12 | −0.32 | 85.41 |
| colorMean | 0.63 | 0.52 | $3.42e{-2}$ | $7.34e{-2}$ | 0.98 | 0.98 | 0.18 | −0.32 | 76.59 |
| EMD | 0.62 | 0.52 | $3.50e{-2}$ | $7.16e{-2}$ | 0.98 | 0.98 | 0.22 | −0.31 | 81.86 |

| | mean | | variance | | maximum | | minimum | | segs |
|---|---|---|---|---|---|---|---|---|---|
| *(watersheds)* | best | cross | best | cross | best | cross | best | cross | % |
| averGrad | 0.65 | 0.47 | $3.00e{-2}$ | 0.12 | 0.98 | 0.97 | 0.26 | −1.24 | 10.16 |
| colorMean | 0.67 | 0.53 | $3.10e{-2}$ | $7.29e{-2}$ | 0.98 | 0.98 | 0.25 | −0.31 | 13.67 |
| EMD | 0.66 | 0.55 | $3.29e{-2}$ | $6.73e{-2}$ | 0.98 | 0.98 | 0.12 | −0.30 | 11.67 |
| *(mean shift)* | | | | | | | | | |
| averGrad | 0.66 | 0.53 | $2.99e{-2}$ | $8.42e{-2}$ | 0.98 | 0.98 | 0.21 | −0.63 | 37.22 |
| colorMean | 0.66 | 0.53 | $3.08e{-2}$ | $7.03e{-2}$ | 0.98 | 0.98 | 0.26 | −0.30 | 43.17 |
| EMD | 0.65 | 0.52 | $3.00e{-2}$ | $6.78e{-2}$ | 0.98 | 0.98 | 0.22 | −0.30 | 46.47 |

| | mean | | variance | | maximum | | minimum | | segs |
|---|---|---|---|---|---|---|---|---|---|
| *(watersheds)* | best | cross | best | cross | best | cross | best | cross | % |
| averGrad | 0.63 | 0.47 | $2.81e{-2}$ | $8.37e{-2}$ | 0.98 | 0.97 | 0.12 | −0.80 | 0.70 |
| colorMean | 0.68 | 0.52 | $2.62e{-2}$ | $7.02e{-2}$ | 0.98 | 0.98 | 0.20 | −0.58 | 1.70 |
| EMD | 0.66 | 0.52 | $2.98e{-2}$ | $7.37e{-2}$ | 0.98 | 0.98 | 0.20 | −0.58 | 1.32 |
| *(mean shift)* | | | | | | | | | |
| averGrad | 0.67 | 0.50 | $2.55e{-2}$ | $7.48e{-2}$ | 0.98 | 0.98 | 0.29 | −0.35 | 5.91 |
| colorMean | 0.66 | 0.52 | $3.06e{-2}$ | $7.02e{-2}$ | 0.98 | 0.98 | 0.21 | −0.28 | 13.42 |
| EMD | 0.65 | 0.51 | $2.99e{-2}$ | $7.27e{-2}$ | 0.98 | 0.96 | 0.20 | −0.26 | 10.20 |

| | mean | | variance | | maximum | | minimum | | segs |
|---|---|---|---|---|---|---|---|---|---|
| *(watersheds)* | best | cross | best | cross | best | cross | best | cross | % |
| averGrad | 0.62 | 0.46 | $3.22e{-2}$ | $6.37e{-2}$ | 0.98 | 0.96 | 0.01 | −0.32 | 1.09 |
| colorMean | 0.68 | 0.54 | $2.79e{-2}$ | $7.80e{-2}$ | 0.98 | 0.97 | 0.15 | −0.43 | 1.63 |
| EMD | 0.66 | 0.52 | $3.24e{-2}$ | $7.97e{-2}$ | 0.98 | 0.97 | 0.19 | −0.58 | 1.39 |
| *(mean shift)* | | | | | | | | | |
| averGrad | 0.66 | 0.50 | $2.87e{-2}$ | $7.40e{-2}$ | 0.98 | 0.98 | 0.25 | −0.39 | 16.43 |
| colorMean | 0.66 | 0.51 | $3.03e{-2}$ | $7.27e{-2}$ | 0.98 | 0.98 | 0.20 | −0.38 | 8.59 |
| EMD | 0.65 | 0.51 | $3.35e{-2}$ | $6.58e{-2}$ | 0.98 | 0.98 | 0.16 | −0.27 | 10.93 |

| | mean | | variance | | maximum | | minimum | | segs |
|---|---|---|---|---|---|---|---|---|---|
| *(watersheds)* | best | cross | best | cross | best | cross | best | cross | % |
| averGrad | 0.62 | 0.48 | $3.61e{-2}$ | $5.13e{-2}$ | 0.98 | 0.98 | 0.01 | −0.21 | 0.79 |
| colorMean | 0.67 | 0.54 | $3.31e{-2}$ | $6.87e{-2}$ | 0.98 | 0.97 | 0.13 | −0.33 | 1.96 |
| EMD | 0.67 | 0.54 | $3.11e{-2}$ | $7.07e{-2}$ | 0.98 | 0.96 | 0.19 | −0.58 | 1.04 |
| *(mean shift)* | | | | | | | | | |
| averGrad | 0.66 | 0.51 | $2.81e{-2}$ | $7.39e{-2}$ | 0.98 | 0.96 | 0.24 | −0.38 | 4.03 |
| colorMean | 0.67 | 0.51 | $3.04e{-2}$ | $6.71e{-2}$ | 0.98 | 0.94 | 0.20 | −0.30 | 9.27 |
| EMD | 0.66 | 0.51 | $3.10e{-2}$ | $6.50e{-2}$ | 0.98 | 0.98 | 0.22 | −0.27 | 5.89 |

**Table A.2:** *Various basic statistics of the evaluation results for the the foreground cut, the isoperimetric cut, and the normalized cut.*

## A.2 Comparisons

| | |
|---|---|
| ● | p-value $\leq 0.05$ |
| ○ | p-value $> 0.05$ |

The following tables compare the mean NPR-indexes of instances of the basic segmentation scheme. Additionally, for each comparison of two instances we also use two statistical tests on the NPR indexes in order to see if the difference are actually significant. For this we use the paired t-test and its nonparametric counterpart, the WILCOXON signed rank tests (see for example [MEINTRUP and SCHÄFFLE 2005]). For the significant level we use the standard value $\alpha = 0.05$.

| | mean | | mean | p.ttest | wilcoxon |
|---|---|---|---|---|---|
| cut | 0.504 | > | 0.444 | ● 0.001 | ● 0.000 |
| mcut | 0.529 | > | 0.469 | ● 0.031 | ● 0.018 |
| fcut | 0.520 | > | 0.463 | ● 0.011 | ● 0.044 |
| icut | 0.502 | > | 0.462 | ● 0.007 | ● 0.045 |
| ncut | 0.510 | > | 0.482 | ○ 0.129 | ○ 0.460 |

(a) averGrad

| | mean | | mean | p.ttest | wilcoxon |
|---|---|---|---|---|---|
| cut | 0.518 | > | 0.493 | ○ 0.094 | ○ 0.143 |
| mcut | 0.526 | < | 0.528 | ○ 0.881 | ○ 0.635 |
| fcut | 0.514 | < | 0.541 | ○ 0.080 | ● 0.047 |
| icut | 0.509 | < | 0.536 | ○ 0.214 | ○ 0.582 |
| ncut | 0.508 | < | 0.537 | ○ 0.078 | ● 0.003 |

(b) colorMean

| | mean | | mean | p.ttest | wilcoxon |
|---|---|---|---|---|---|
| cut | 0.518 | > | 0.504 | ○ 0.368 | ● 0.006 |
| mcut | 0.524 | < | 0.546 | ○ 0.094 | ● 0.005 |
| fcut | 0.513 | < | 0.540 | ○ 0.104 | ○ 0.481 |
| icut | 0.510 | < | 0.517 | ○ 0.782 | ● 0.018 |
| ncut | 0.515 | < | 0.541 | ○ 0.252 | ● 0.004 |

(c) EMD

**Table A.3:** *Comparison of mean NPR scores for two different initial segmentations steps for all cuts with results from the paired t-test and the WILCOXON signed rank test if the differences in the distributions is significant.*

| | mean | p.ttest | wilcoxon | p.ttest | wilcoxon | p.ttest | wilcoxon |
|---|---|---|---|---|---|---|---|
| mcut | 0.529 | ◍ | ◍ | ◍ | ◍ | ● 0.035 | ● 0.000 |
| fcut | 0.520 | ○ 0.681 | ● 0.036 | ○ 0.681 | ● 0.036 | ● 0.000 | ● 0.000 |
| ncut | 0.510 | ○ 0.371 | ● 0.021 | ● 0.012 | ● 0.000 | ● 0.000 | ● 0.000 |
| cut | 0.504 | ○ 0.126 | ● 0.002 | ○ 0.672 | ○ 0.134 | ○ 0.097 | ● 0.000 |
| icut | 0.502 | ○ 0.214 | ● 0.004 | ○ 0.845 | ○ 1.000 | ● 0.001 | ● 0.000 |

(a) *mean shift* + averGrad

| | mean | p.ttest | wilcoxon | p.ttest | wilcoxon | p.ttest | wilcoxon |
|---|---|---|---|---|---|---|---|
| mcut | 0.526 | ◍ | ◍ | ◍ | ◍ | ● 0.000 | ● 0.000 |
| cut | 0.518 | ○ 0.130 | ● 0.000 | ○ 0.130 | ● 0.000 | ● 0.001 | ● 0.000 |
| fcut | 0.514 | ○ 0.289 | ○ 0.764 | ○ 0.773 | ● 0.036 | ● 0.008 | ● 0.000 |
| icut | 0.509 | ○ 0.360 | ● 0.000 | ○ 0.732 | ● 0.000 | ○ 0.158 | ● 0.000 |
| ncut | 0.508 | ○ 0.072 | ● 0.021 | ○ 0.931 | ● 0.000 | ● 0.007 | ● 0.000 |

(b) *mean shift* + colorMean

| | mean | p.ttest | wilcoxon | p.ttest | wilcoxon | p.ttest | wilcoxon |
|---|---|---|---|---|---|---|---|
| mcut | 0.524 | ◍ | ◍ | ◍ | ◍ | ● 0.000 | ● 0.000 |
| cut | 0.518 | ○ 0.608 | ○ 0.920 | ○ 0.608 | ○ 0.920 | ● 0.012 | ● 0.000 |
| ncut | 0.515 | ○ 0.478 | ● 0.002 | ○ 0.841 | ● 0.012 | ● 0.038 | ● 0.000 |
| fcut | 0.513 | ○ 0.427 | ● 0.007 | ○ 0.878 | ● 0.007 | ○ 0.068 | ● 0.000 |
| icut | 0.510 | ○ 0.327 | ● 0.036 | ○ 0.882 | ○ 0.057 | ○ 0.069 | ● 0.000 |

(c) *mean shift* + EMD

| | mean | p.ttest | wilcoxon | p.ttest | wilcoxon | p.ttest | wilcoxon |
|---|---|---|---|---|---|---|---|
| ncut | 0.482 | ◍ | ◍ | ◍ | ◍ | ● 0.000 | ● 0.000 |
| mcut | 0.469 | ○ 0.695 | ○ 1.000 | ○ 0.695 | ○ 1.000 | ● 0.000 | ● 0.000 |
| fcut | 0.463 | ○ 0.250 | ○ 0.920 | ○ 0.837 | ○ 0.764 | ● 0.000 | ● 0.000 |
| icut | 0.462 | ○ 0.117 | ● 0.002 | ○ 0.951 | ● 0.036 | ● 0.000 | ● 0.000 |
| cut | 0.444 | ○ 0.084 | ● 0.001 | ○ 0.283 | ● 0.000 | ● 0.000 | ● 0.000 |

(d) *watersheds* + averGrad

| | mean | p.ttest | wilcoxon | p.ttest | wilcoxon | p.ttest | wilcoxon |
|---|---|---|---|---|---|---|---|
| fcut | 0.541 | ◍ | ◍ | ◍ | ◍ | ● 0.000 | ● 0.000 |
| ncut | 0.537 | ○ 0.693 | ○ 0.089 | ○ 0.693 | ○ 0.089 | ● 0.000 | ● 0.000 |
| icut | 0.536 | ○ 0.522 | ○ 0.920 | ○ 0.956 | ● 0.036 | ● 0.000 | ● 0.000 |
| mcut | 0.528 | ○ 0.241 | ● 0.004 | ○ 0.506 | ● 0.007 | ● 0.000 | ● 0.000 |
| cut | 0.493 | ● 0.001 | ● 0.000 | ● 0.007 | ● 0.000 | ● 0.000 | ● 0.000 |

(e) *watersheds* + colorMean

| | mean | p.ttest | wilcoxon | p.ttest | wilcoxon | p.ttest | wilcoxon |
|---|---|---|---|---|---|---|---|
| mcut | 0.546 | ◍ | ◍ | ◍ | ◍ | ● 0.000 | ● 0.000 |
| ncut | 0.541 | ○ 0.764 | ○ 0.134 | ○ 0.764 | ○ 0.134 | ● 0.000 | ● 0.000 |
| fcut | 0.540 | ○ 0.555 | ○ 1.000 | ○ 0.938 | ● 0.036 | ● 0.000 | ● 0.000 |
| icut | 0.517 | ○ 0.145 | ○ 0.484 | ○ 0.225 | ○ 0.764 | ● 0.000 | ● 0.000 |
| cut | 0.504 | ● 0.000 | ● 0.000 | ○ 0.527 | ● 0.000 | ● 0.000 | ● 0.000 |

(f) *watersheds* + EMD

**Table A.4:** *Ranking of the different cut types for possible combinations of initial segmentation step and edge weight function.*

| | | | mean shift | | | | | watersheds | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | mean | | mean | p.ttest | wilcoxon | mean | | mean | p.ttest | wilcoxon |
| averGrad/colorMean | 0.504 | $<$ | 0.518 | ○ 0.220 | ○ 0.057 | 0.444 | $<$ | 0.493 | ● 0.020 | ● 0.000 |
| averGrad/EMD | 0.504 | $<$ | 0.518 | ○ 0.373 | ○ 0.089 | 0.444 | $<$ | 0.504 | ● 0.001 | ● 0.000 |
| colorMean/EMD | 0.518 | $<$ | 0.518 | ○ 0.969 | ○ 0.134 | 0.493 | $<$ | 0.504 | ○ 0.265 | ○ 0.484 |

(a) *foreground cut* (fut)

| | | | mean shift | | | | | watersheds | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | mean | | mean | p.ttest | wilcoxon | mean | | mean | p.ttest | wilcoxon |
| averGrad/colorMean | 0.529 | $>$ | 0.526 | ○ 0.902 | ● 0.036 | 0.469 | $<$ | 0.528 | ● 0.034 | ○ 0.764 |
| averGrad/EMD | 0.529 | $>$ | 0.524 | ○ 0.833 | ○ 0.089 | 0.469 | $<$ | 0.546 | ● 0.007 | ○ 0.134 |
| colorMean/EMD | 0.526 | $>$ | 0.524 | ○ 0.738 | ● 0.000 | 0.528 | $<$ | 0.546 | ● 0.014 | ● 0.000 |

(b) *foreground cut* (fut)

| | | | mean shift | | | | | watersheds | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | mean | | mean | p.ttest | wilcoxon | mean | | mean | p.ttest | wilcoxon |
| averGrad/colorMean | 0.520 | $>$ | 0.514 | ○ 0.524 | ● 0.000 | 0.463 | $<$ | 0.541 | ● 0.000 | ● 0.000 |
| averGrad/EMD | 0.520 | $>$ | 0.513 | ○ 0.610 | ● 0.021 | 0.463 | $<$ | 0.540 | ● 0.000 | ● 0.000 |
| colorMean/EMD | 0.514 | $>$ | 0.513 | ○ 0.891 | ○ 0.194 | 0.541 | $>$ | 0.540 | ○ 0.835 | ○ 0.617 |

(c) *foreground cut* (fut)

| | | | mean shift | | | | | watersheds | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | mean | | mean | p.ttest | wilcoxon | mean | | mean | p.ttest | wilcoxon |
| averGrad/colorMean | 0.502 | $<$ | 0.509 | ○ 0.660 | ● 0.000 | 0.462 | $<$ | 0.536 | ● 0.000 | ● 0.000 |
| averGrad/EMD | 0.502 | $<$ | 0.510 | ○ 0.530 | ● 0.000 | 0.462 | $<$ | 0.517 | ● 0.001 | ● 0.000 |
| colorMean/EMD | 0.509 | $<$ | 0.510 | ○ 0.952 | ● 0.000 | 0.536 | $>$ | 0.517 | ○ 0.337 | ○ 0.617 |

(d) *isoperimetric cut* (icut)

| | | | mean shift | | | | | watersheds | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | mean | | mean | p.ttest | wilcoxon | mean | | mean | p.ttest | wilcoxon |
| averGrad/colorMean | 0.510 | $>$ | 0.508 | ○ 0.834 | ● 0.000 | 0.482 | $<$ | 0.537 | ● 0.003 | ○ 0.057 |
| averGrad/EMD | 0.510 | $<$ | 0.515 | ○ 0.736 | ● 0.000 | 0.482 | $<$ | 0.541 | ● 0.000 | ● 0.000 |
| colorMean/EMD | 0.508 | $<$ | 0.515 | ○ 0.549 | ● 0.000 | 0.537 | $<$ | 0.541 | ○ 0.834 | ● 0.036 |

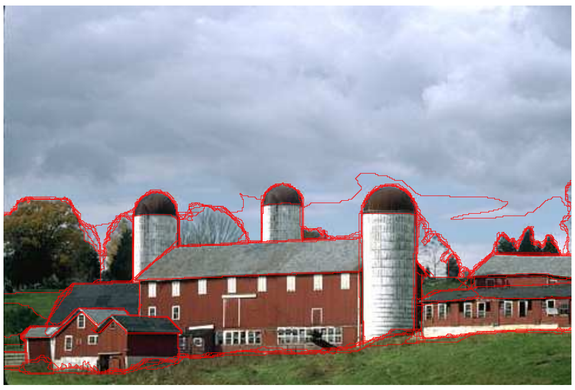
(e) *normalized cut* (ncut)

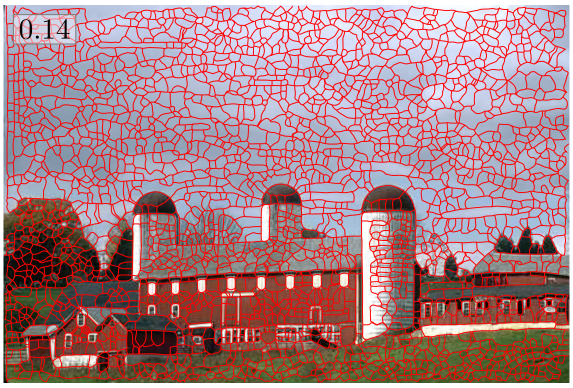
**Table A.5:** *Comparison of mean NPR scores for different edge weight types for the foreground, the isoperimetric and the normalized cut with results from the paired t-test and the* Wilcoxon *signed rank test if the differences in the distributions is significant.*

## A.3 Segmentations

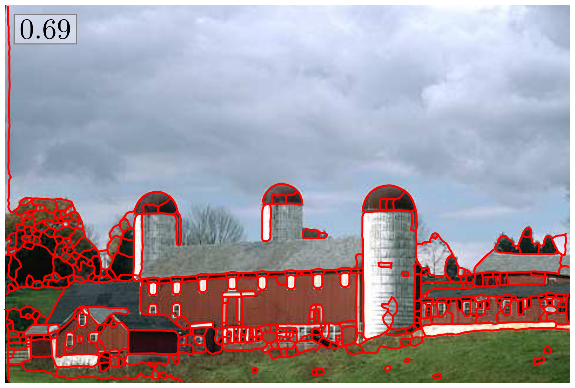
We give on the following pages segmentation results for the instances of the basic segmentation scheme for a hand chosen set of images. We give first three images with the watershed transform as initial segmentation step and then three images with the mean shift method. We also present the ground truth data from the Berkeley Image Database. For this we superimpose the ground truth data from all subjects for one image. The corresponding NPR-indexes are presented as well.

(a) original image

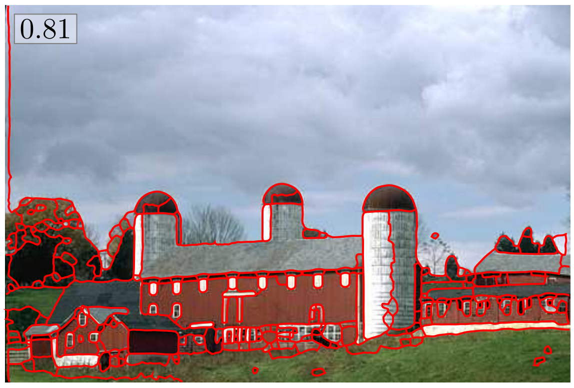

(b) ground truth



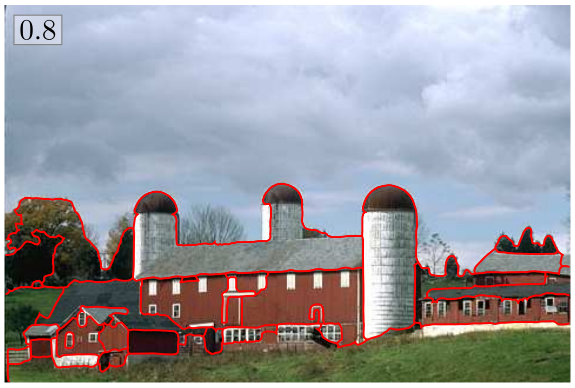
(c) watersheds segmentation



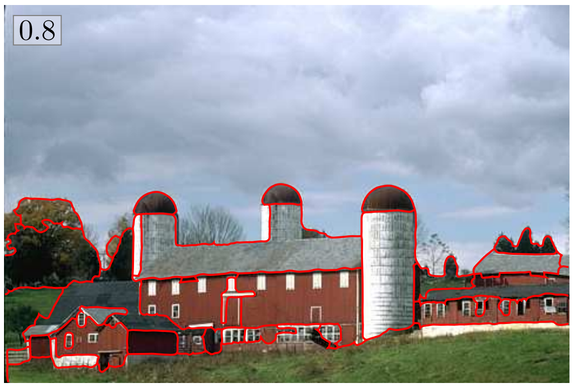
(d) minimum cut



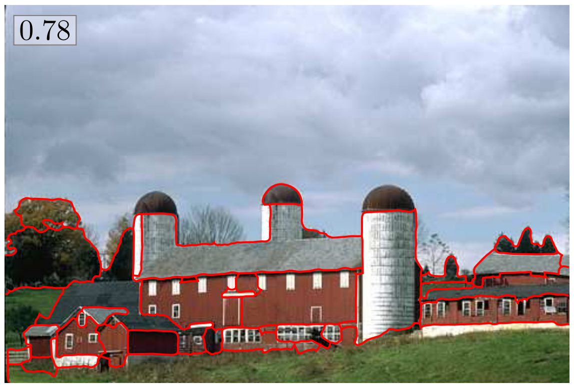
(e) mean cut



(f) foreground cut

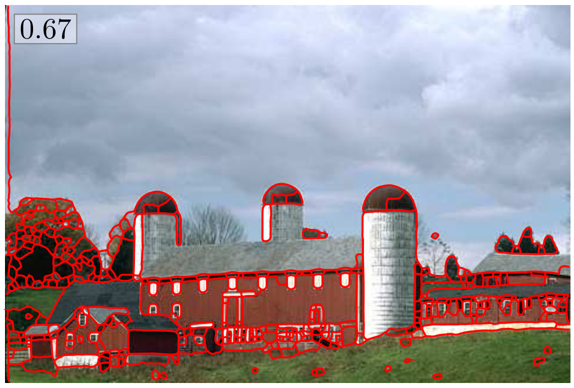


(g) isoperimetric cut

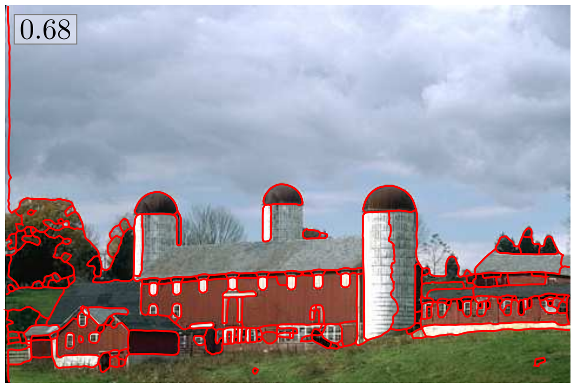


(h) normalized cut

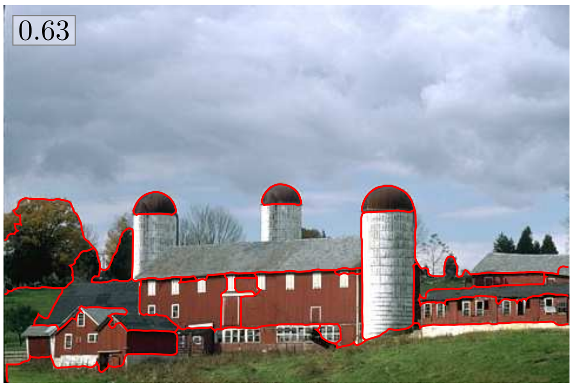
*image-id: 97017; watersheds and* colorMean



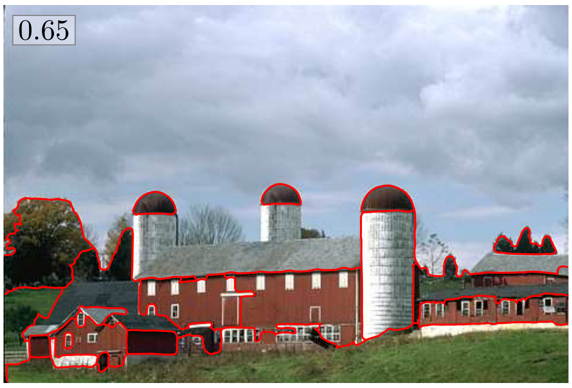
**(d)** *minimum cut*



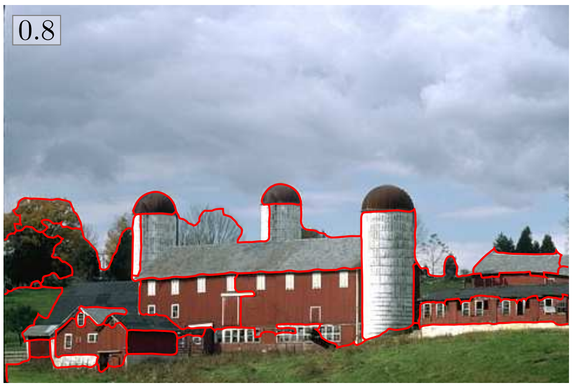
**(e)** *mean cut*



**(f)** *foreground cut*

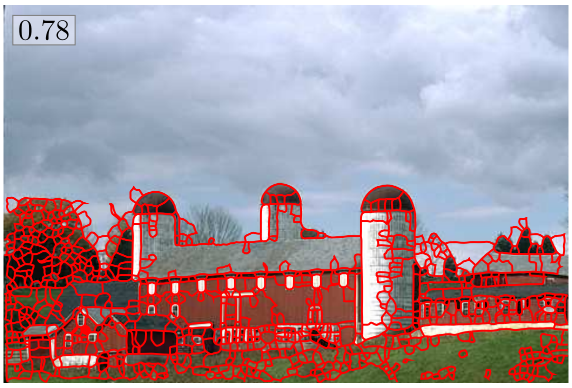


**(g)** *isoperimetric cut*

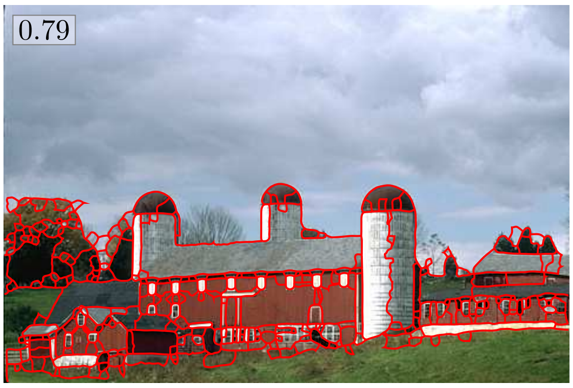


**(h)** *normalized cut*

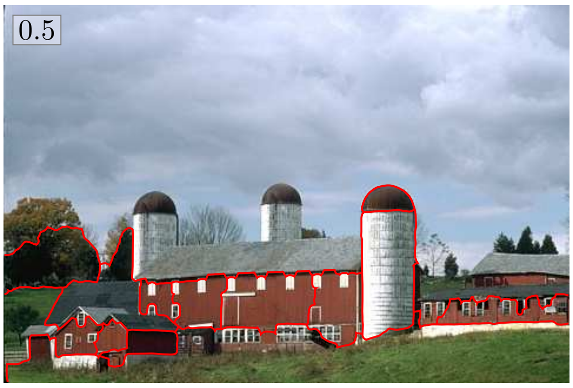
*image-id: 97017; watersheds and* EMD



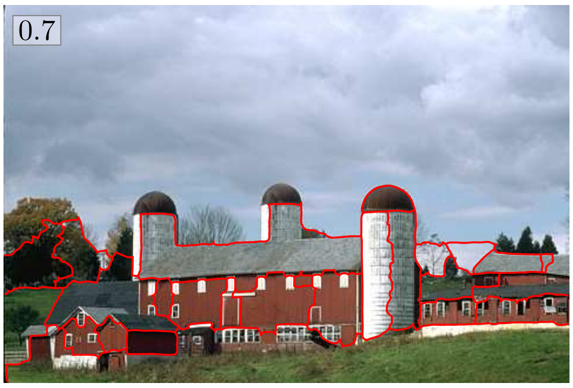
(d) *minimum cut*



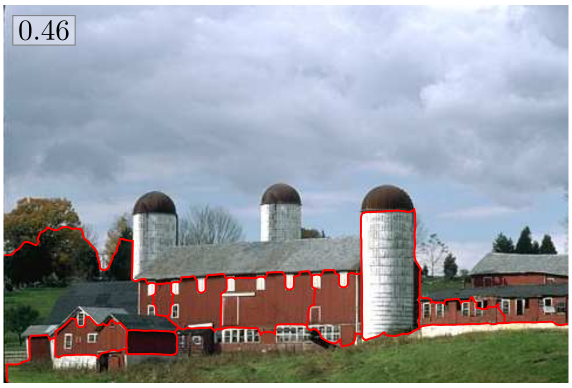
(e) *mean cut*



(f) *foreground cut*



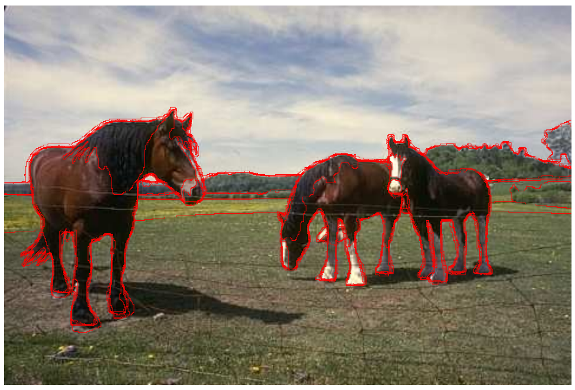
(g) *isoperimetric cut*

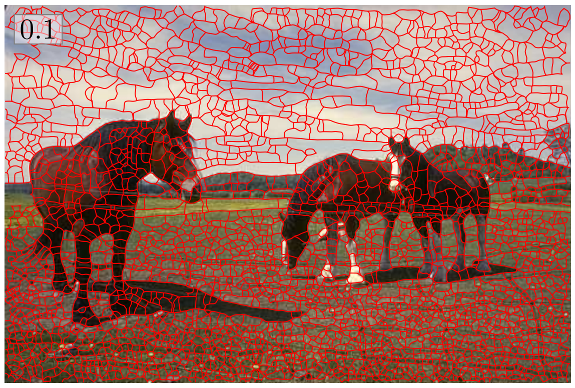


(h) *normalized cut*

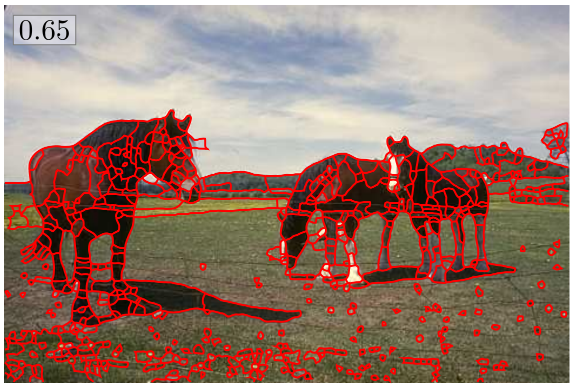
*image-id: 97017; watersheds and* averGrad

**(a)** *original image*

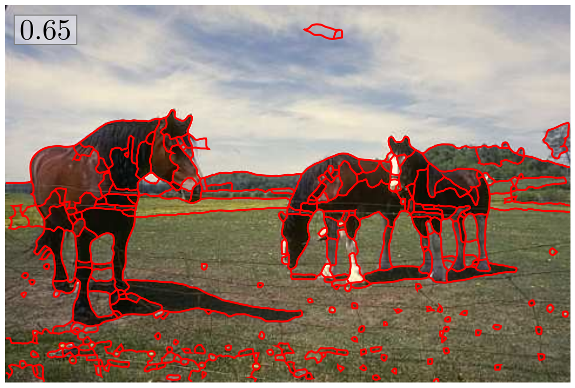

**(b)** *ground truth*



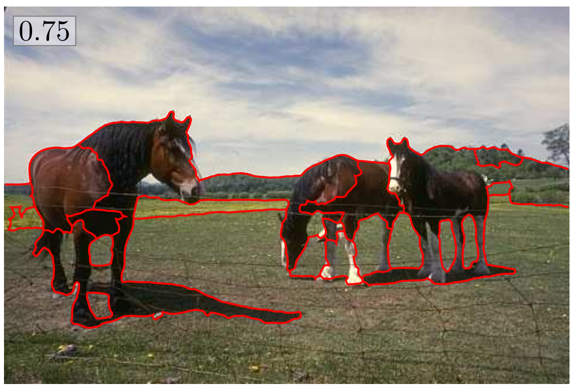
**(c)** *watersheds segmentation*



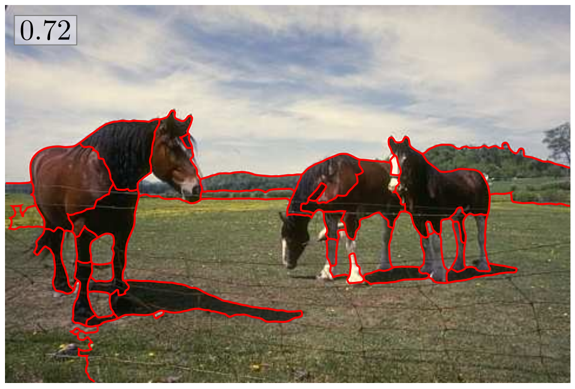
**(d)** *minimum cut*



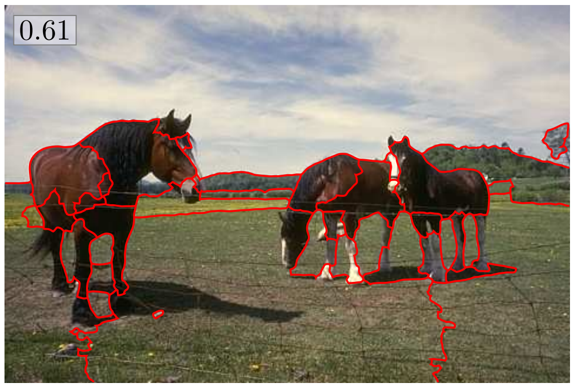
**(e)** *mean cut*



**(f)** *foreground cut*

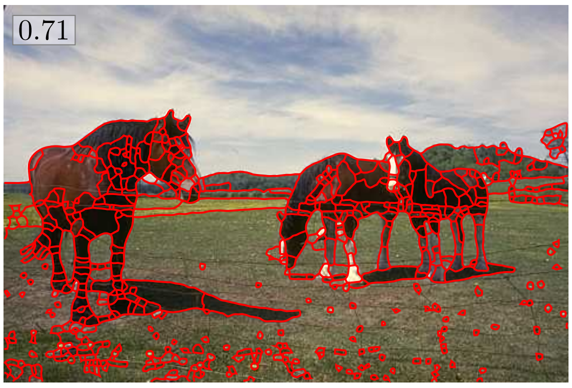


**(g)** *isoperimetric cut*

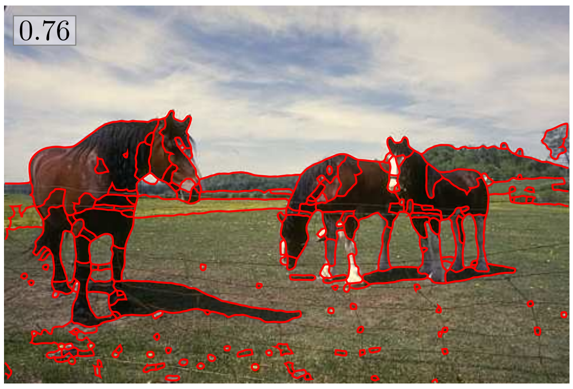


**(h)** *normalized cut*

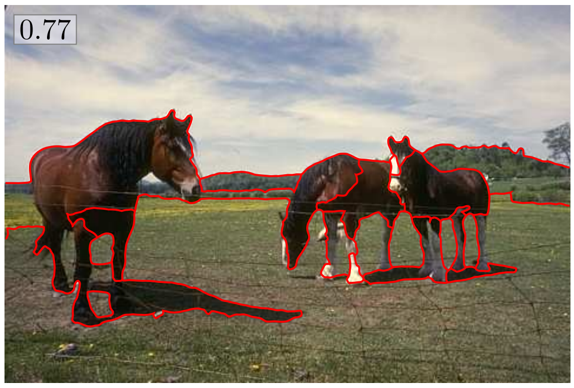
*image-id: 197017; watersheds and* colorMean



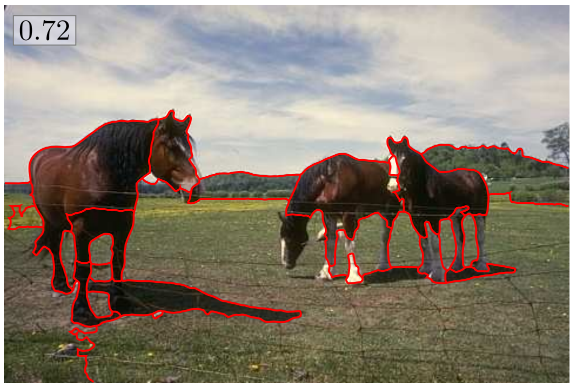
**(d)** *minimum cut*



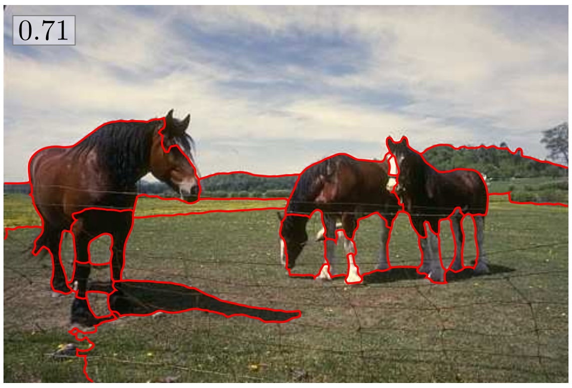
**(e)** *mean cut*



**(f)** *foreground cut*

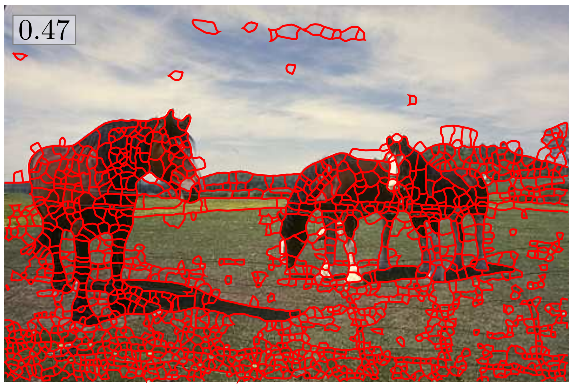


**(g)** *isoperimetric cut*

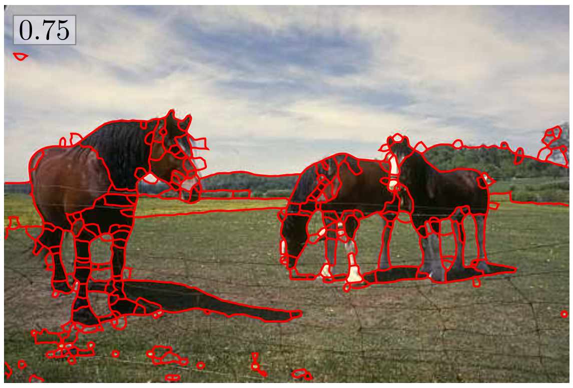


**(h)** *normalized cut*

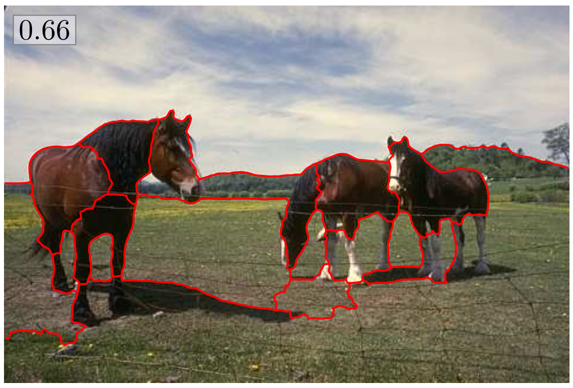
*image-id: 197017; watersheds and* EMD



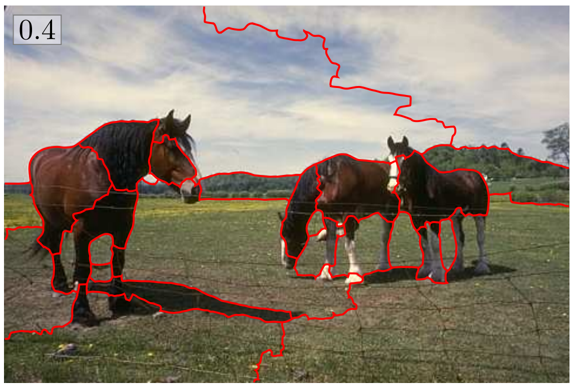
**(d)** *minimum cut*



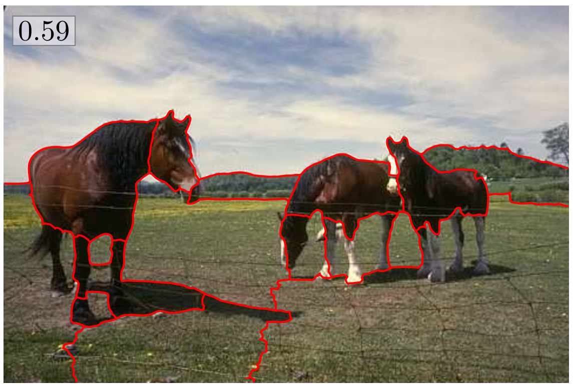
**(e)** *mean cut*



**(f)** *foreground cut*



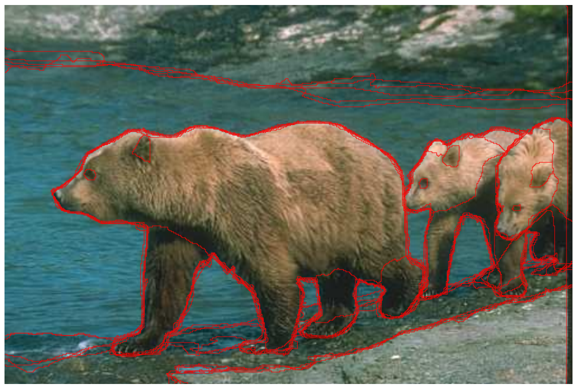
**(g)** *isoperimetric cut*

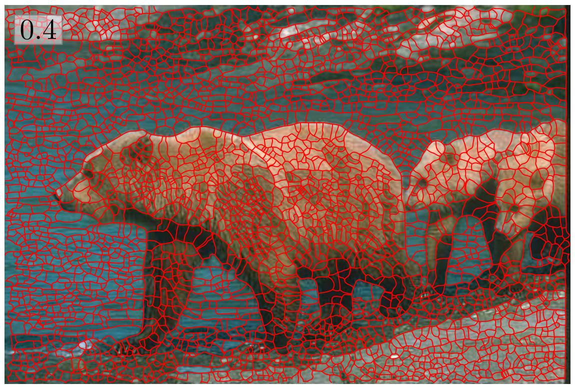


**(h)** *normalized cut*

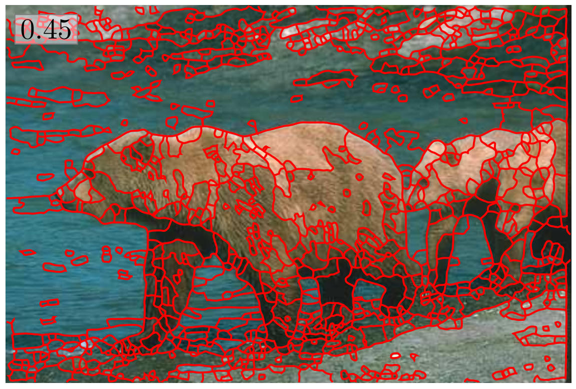
*image-id: 197017; watersheds and* averGrad

**(a)** *original image*

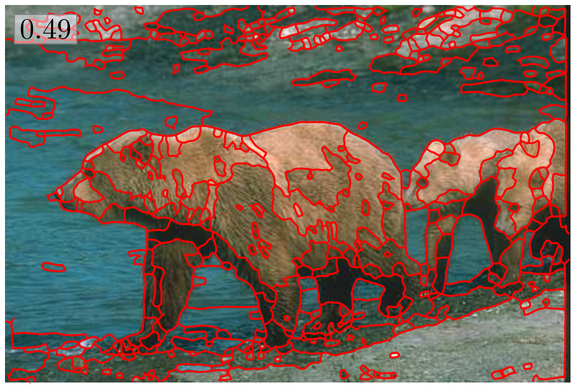

**(b)** *ground truth*



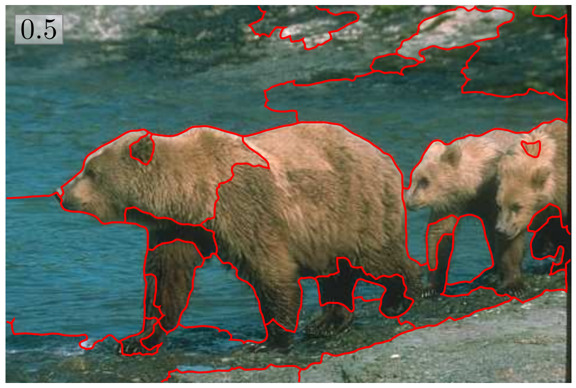
**(c)** *watersheds segmentation*



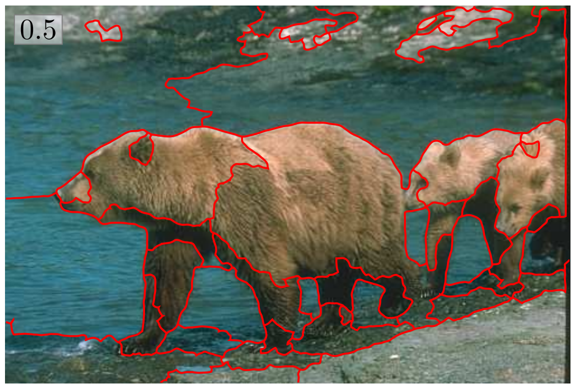
**(d)** *minimum cut*



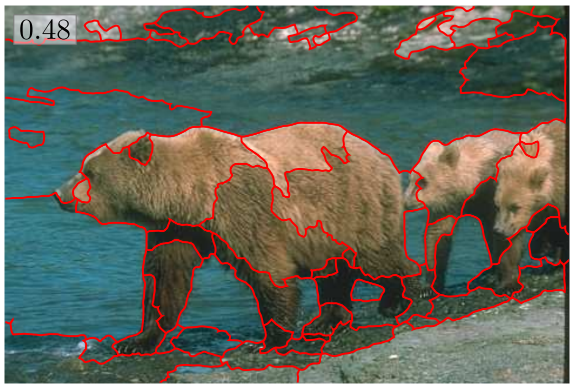
**(e)** *mean cut*



**(f)** *foreground cut*

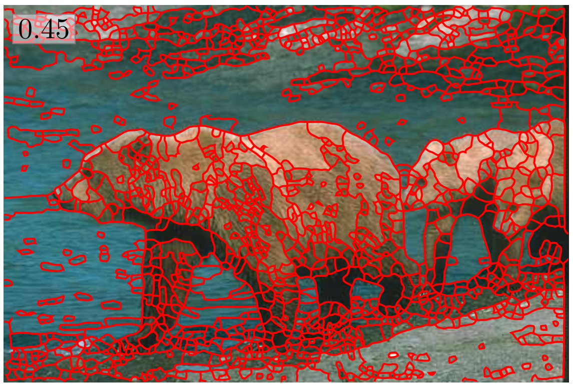


**(g)** *isoperimetric cut*

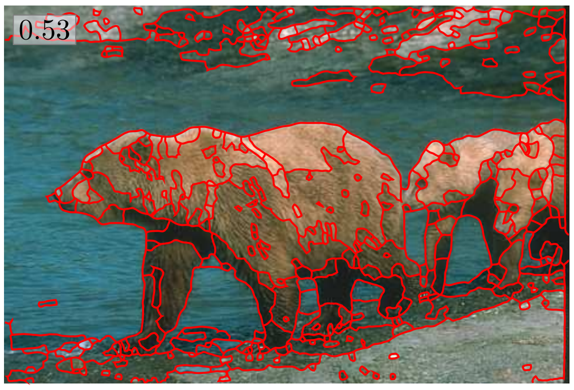


**(h)** *normalized cut*

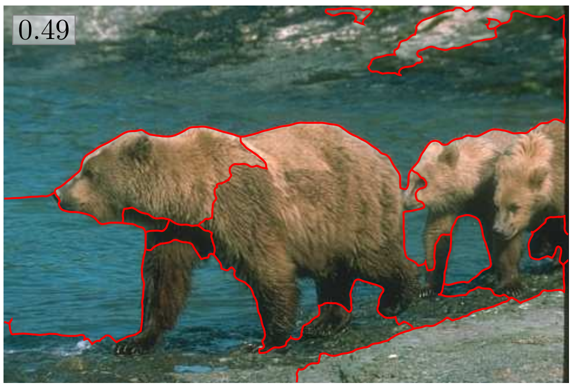
*image-id: 100075; watersheds and* colorMean



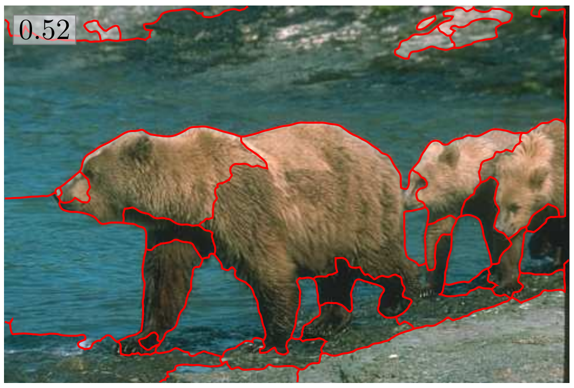
**(d)** *minimum cut*



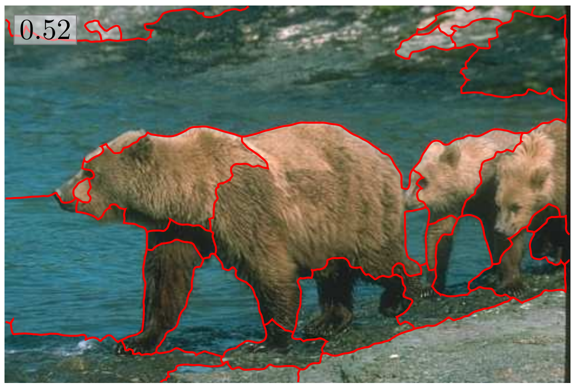
**(e)** *mean cut*



**(f)** *foreground cut*

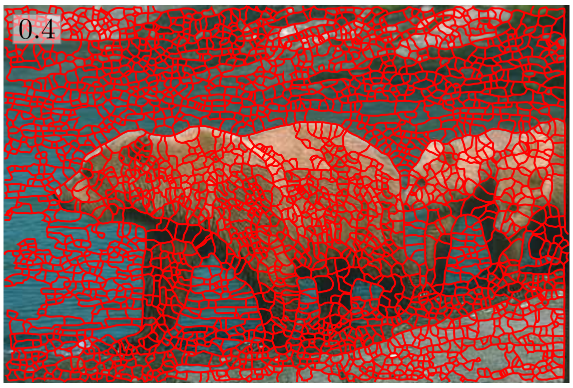


**(g)** *isoperimetric cut*

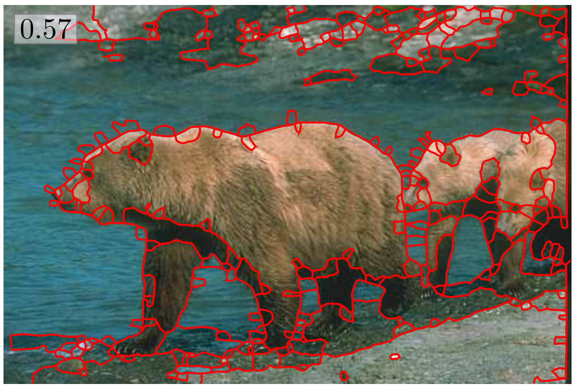


**(h)** *normalized cut*

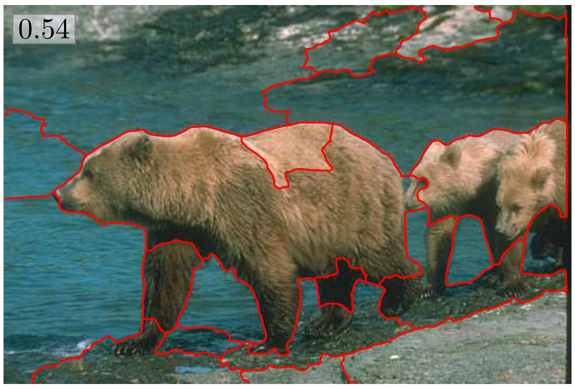
*image-id: 100075; watersheds and* EMD



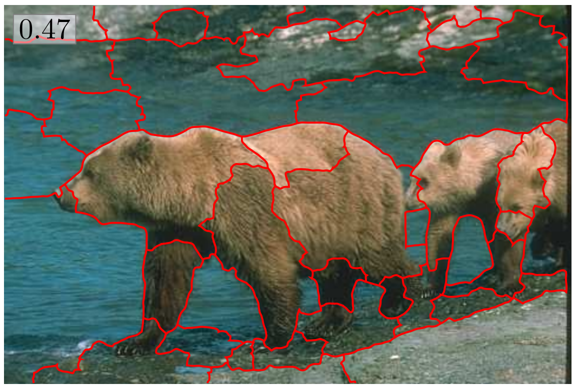
**(d)** *minimum cut*



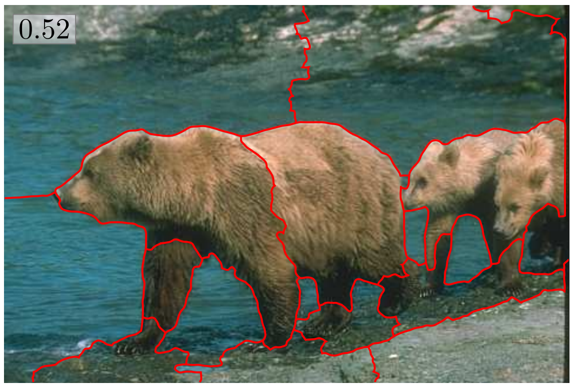
**(e)** *mean cut*



**(f)** *foreground cut*



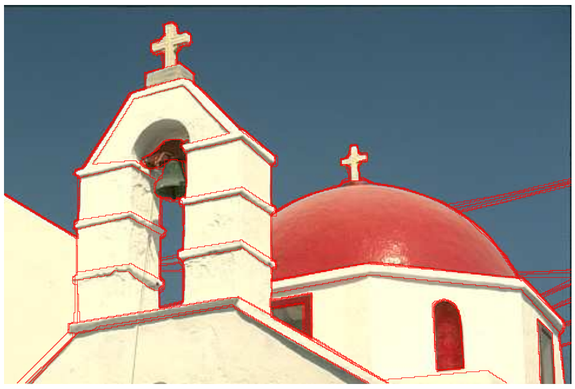
**(g)** *isoperimetric cut*

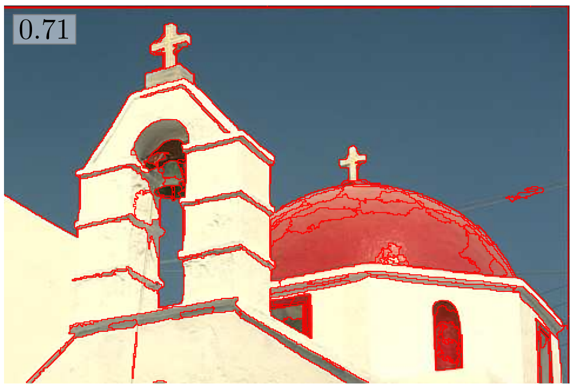


**(h)** *normalized cut*

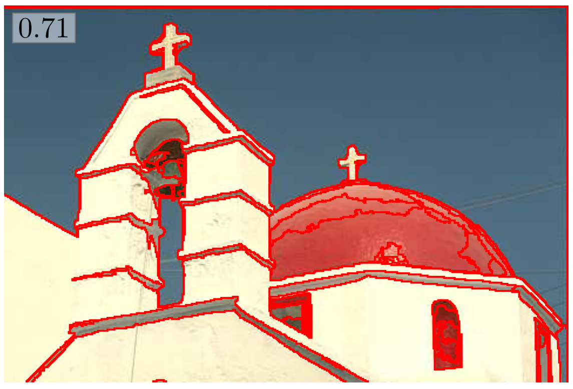
*image-id: 100075; watersheds and* averGrad

**(a)** *original image*

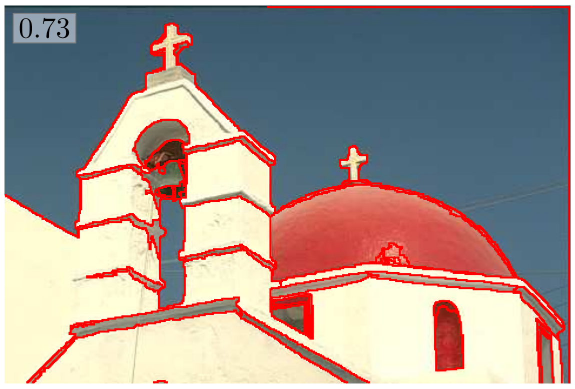

**(b)** *ground truth*



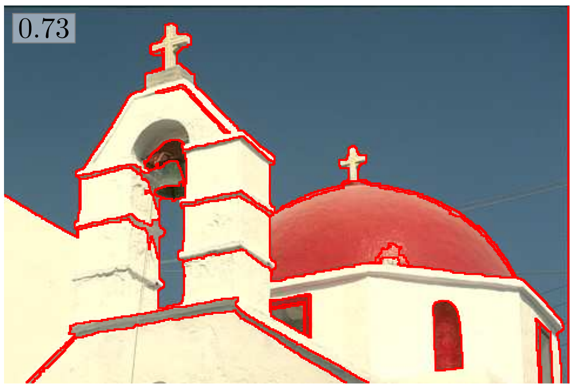
**(c)** *mean shift segmentation*



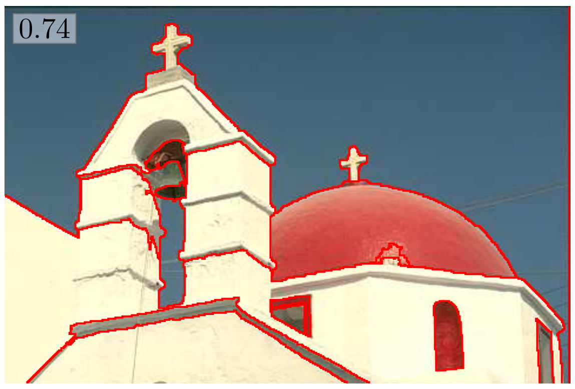
**(d)** *minimum cut*



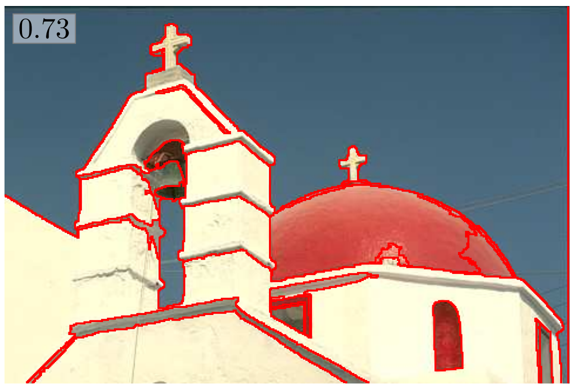
**(e)** *mean cut*



**(f)** *foreground cut*

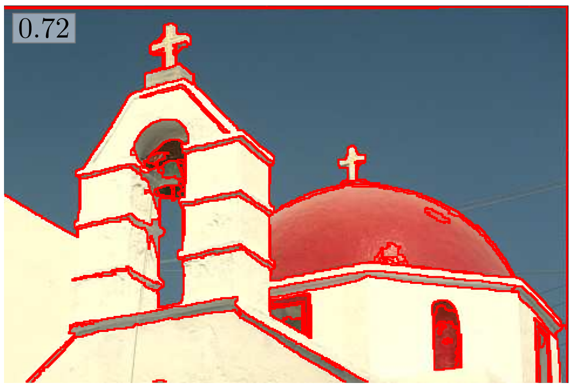


**(g)** *isoperimetric cut*

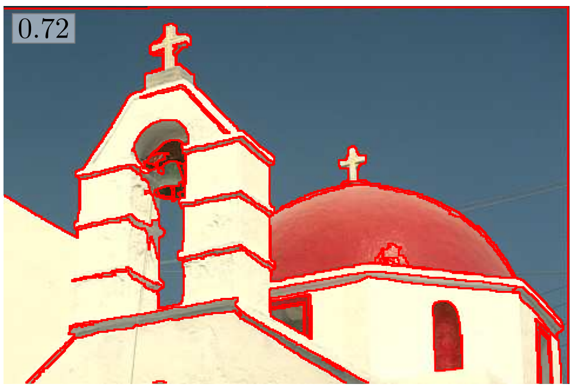


**(h)** *normalized cut*

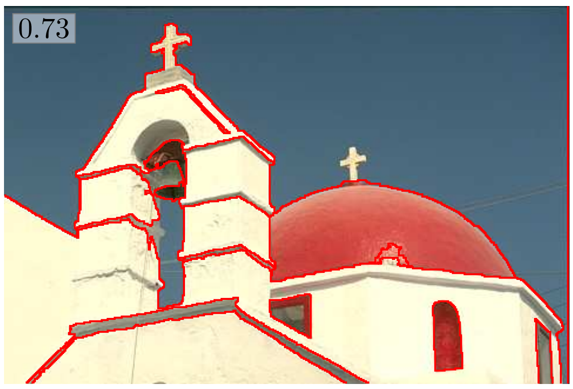
*image-id: 118035; mean shift and* colorMean



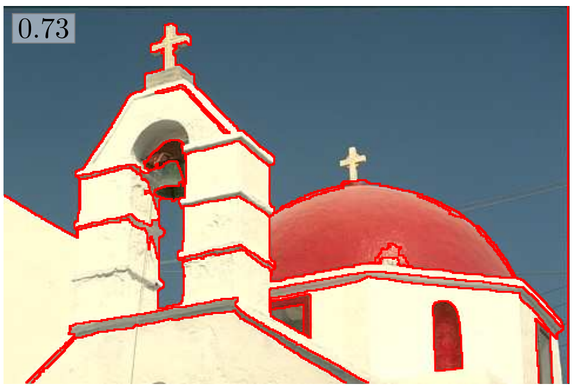
**(d)** *minimum cut*



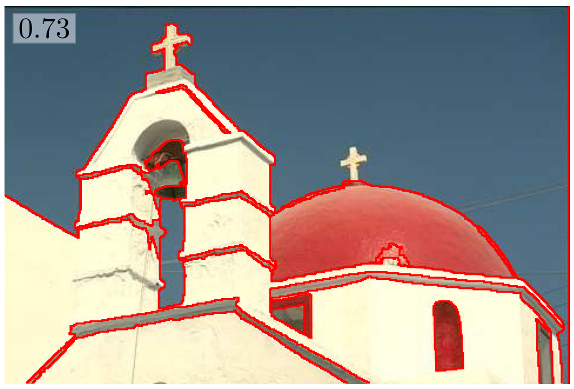
**(e)** *mean cut*



**(f)** *foreground cut*

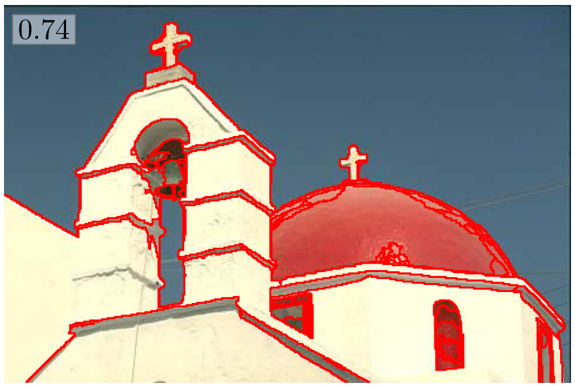


**(g)** *isoperimetric cut*

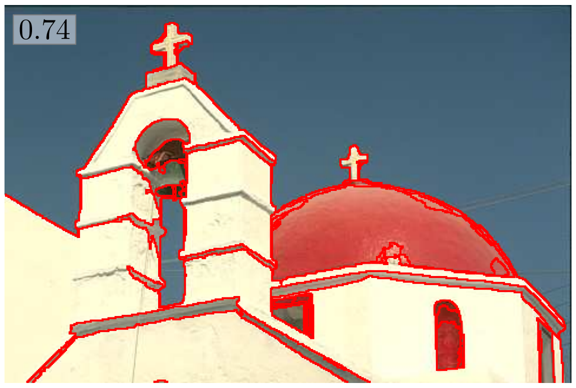


**(h)** *normalized cut*

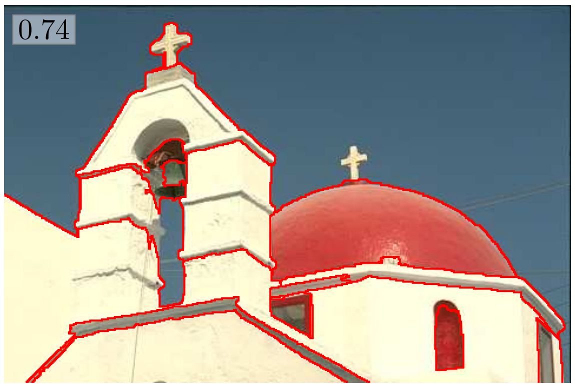
*image-id: 118035; mean shift and* EMD



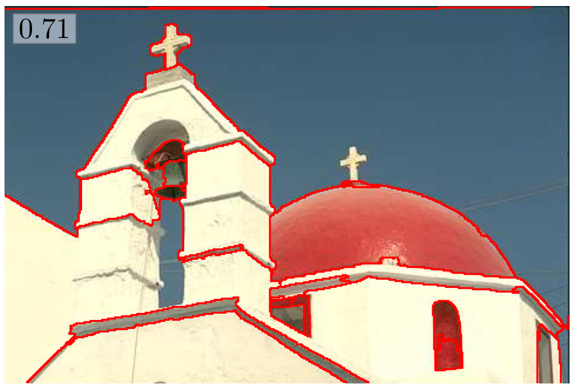
**(d)** *minimum cut*



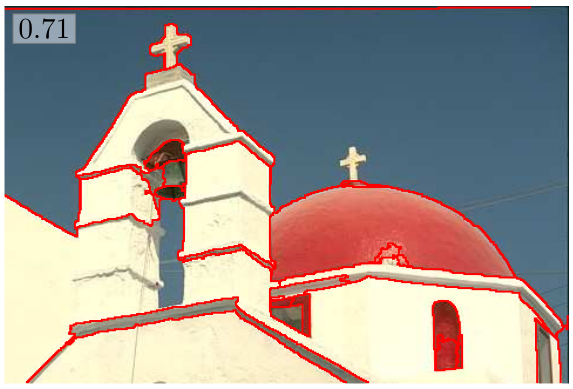
**(e)** *mean cut*



**(f)** *foreground cut*



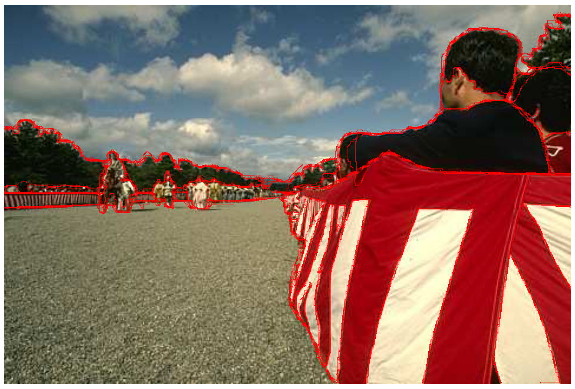
**(g)** *isoperimetric cut*

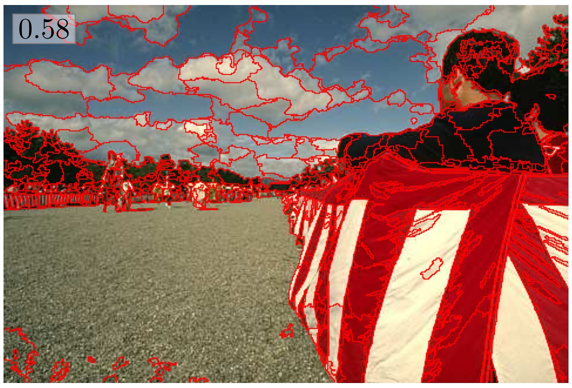


**(h)** *normalized cut*

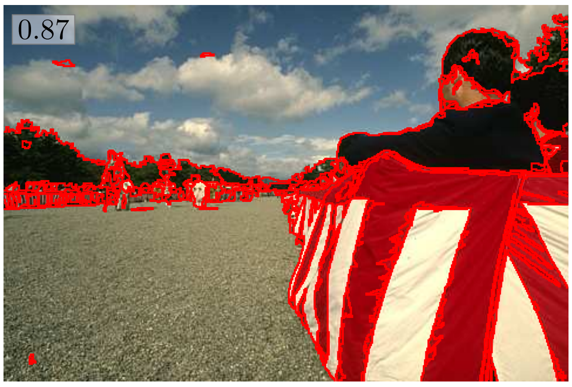
*image-id: 118035; mean shift and* averGrad

**(a)** *original image*

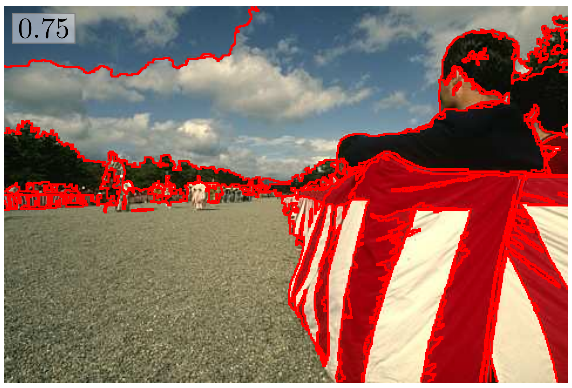

**(b)** *ground truth*



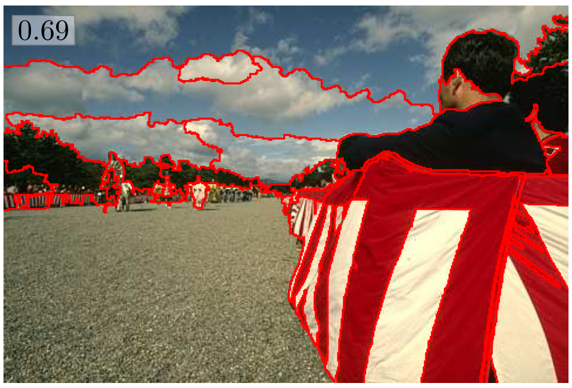
**(c)** *mean shift segmentation*



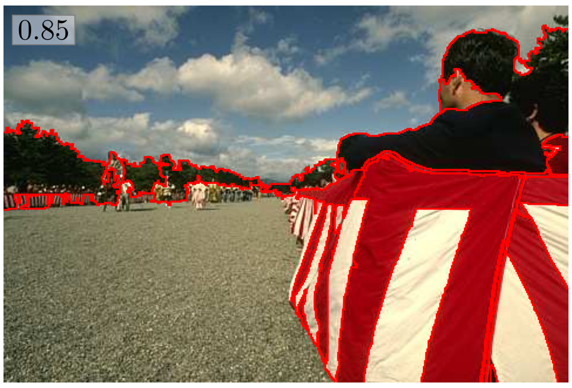
**(d)** *minimum cut*



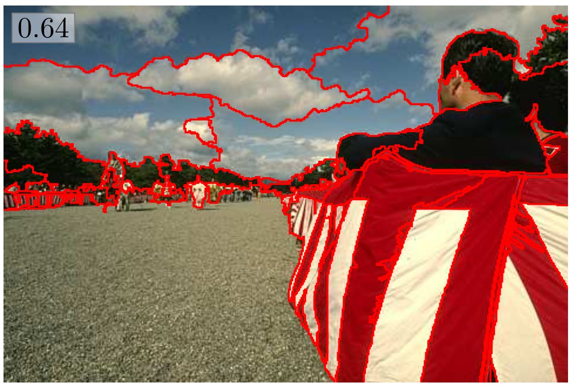
**(e)** *mean cut*



**(f)** *foreground cut*

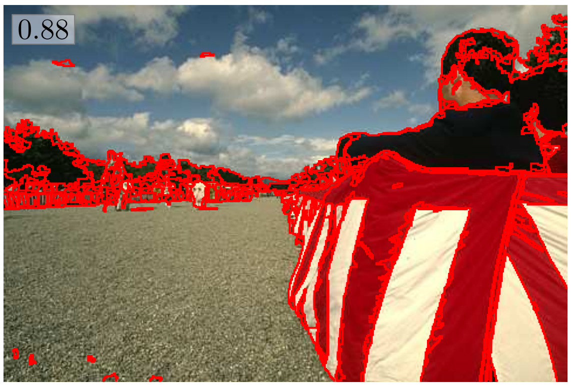


**(g)** *isoperimetric cut*

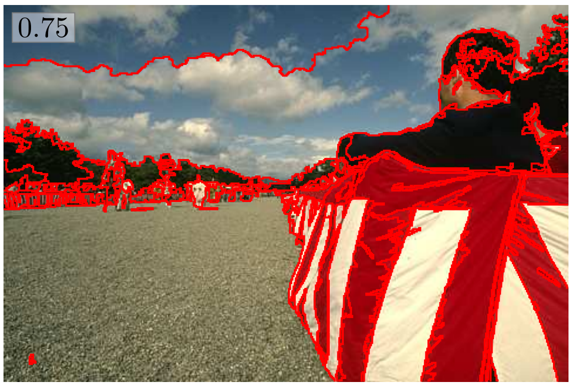


**(h)** *normalized cut*

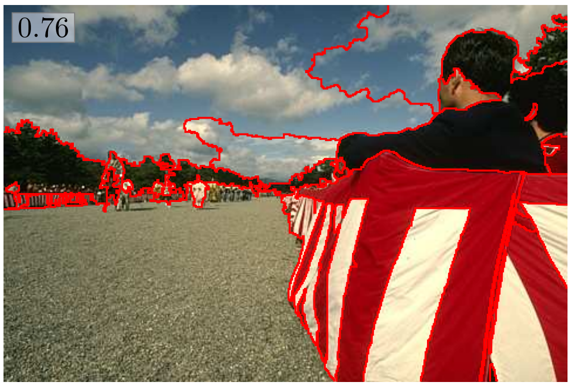
*image-id: 145086; mean shift and* colorMean



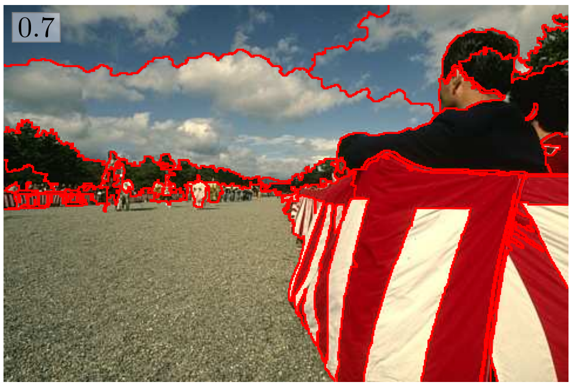
**(d)** *minimum cut*



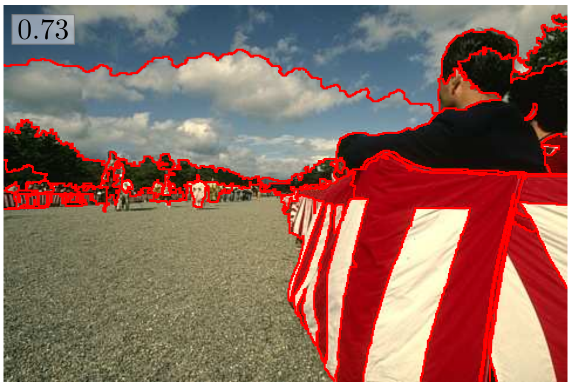
**(e)** *mean cut*



**(f)** *foreground cut*

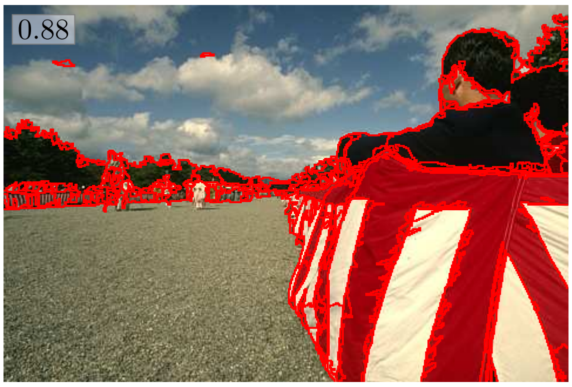


**(g)** *isoperimetric cut*

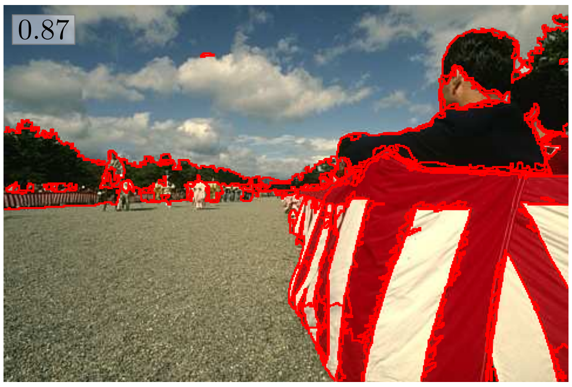


**(h)** *normalized cut*

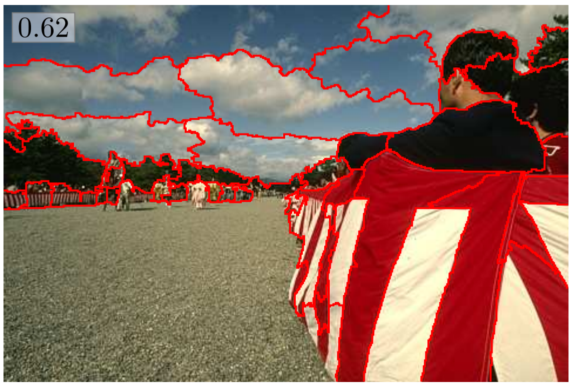
*image-id: 145086; mean shift and* EMD



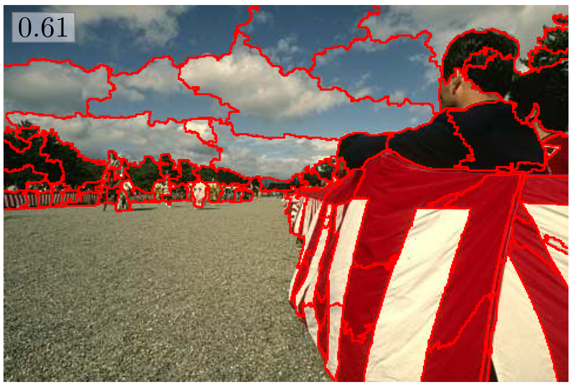
**(d)** *minimum cut*



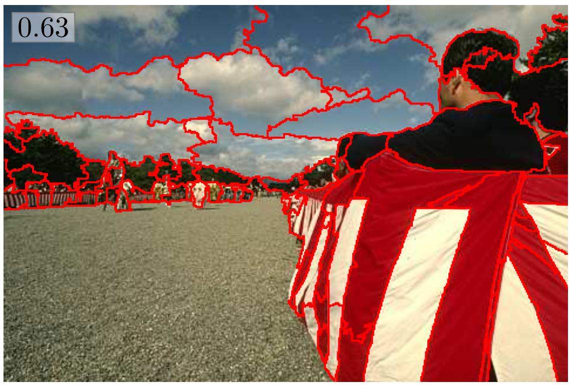
**(e)** *mean cut*



**(f)** *foreground cut*



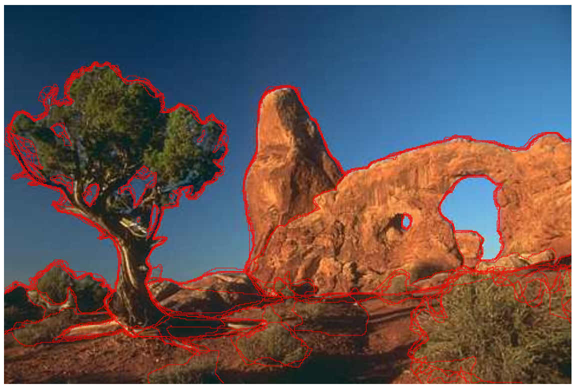
**(g)** *isoperimetric cut*

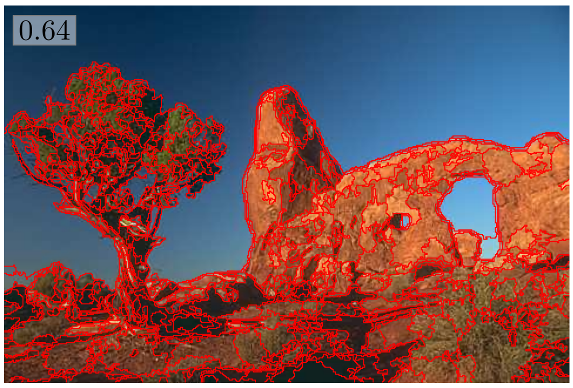


**(h)** *normalized cut*

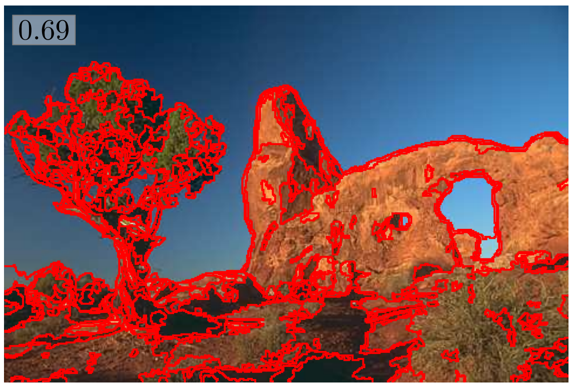
*image-id: 145086; mean shift and* averGrad

**(a)** *original image*

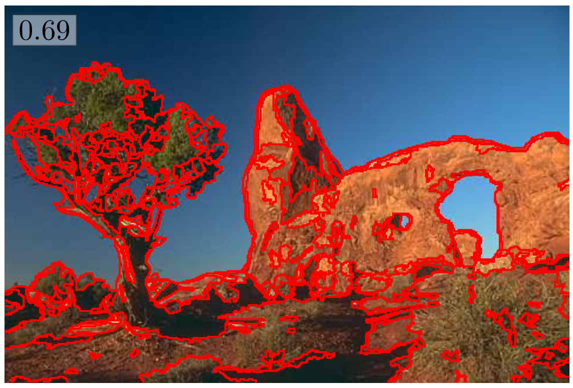

**(b)** *ground truth*



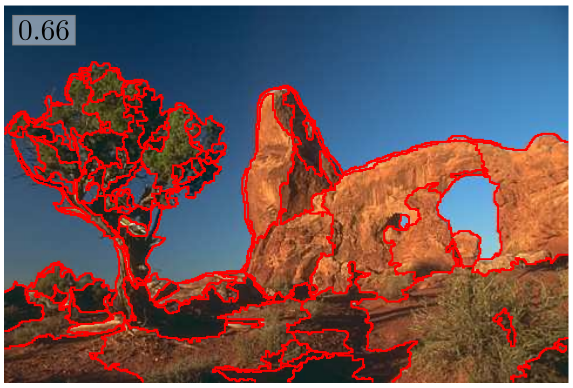
**(c)** *mean shift segmentation*



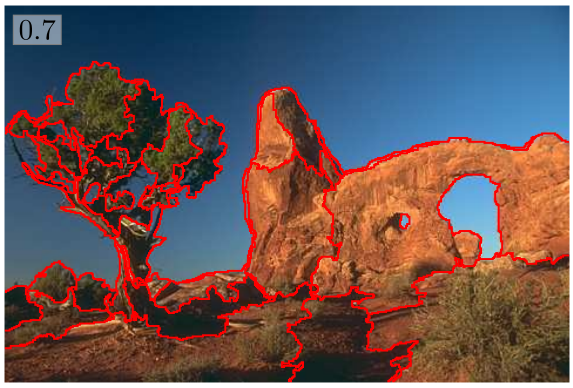
**(d)** *minimum cut*



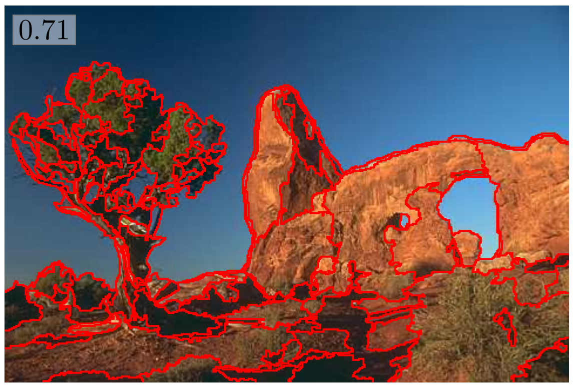
**(e)** *mean cut*



**(f)** *foreground cut*

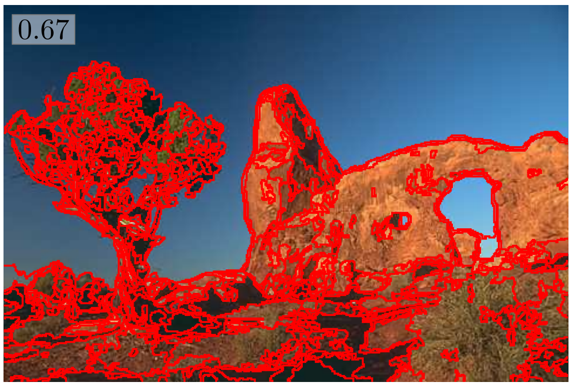


**(g)** *isoperimetric cut*

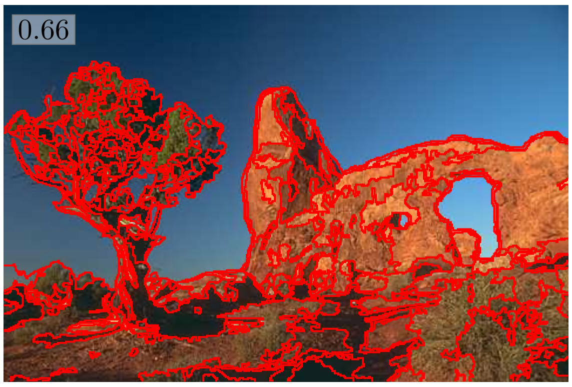


**(h)** *normalized cut*

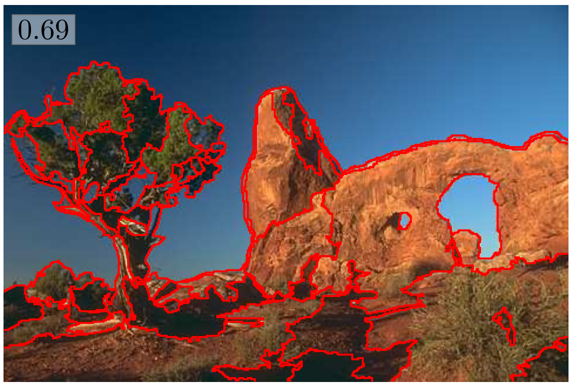
*image-id: 295087; mean shift and* colorMean



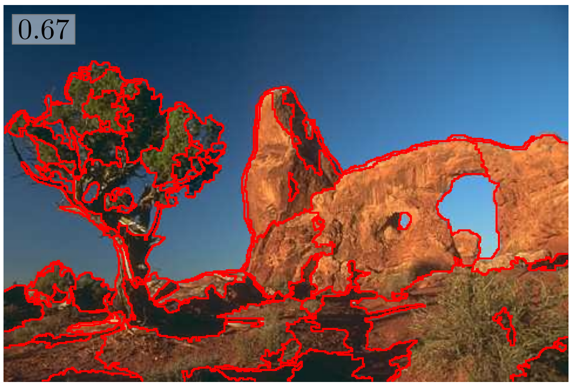
**(d)** *minimum cut*



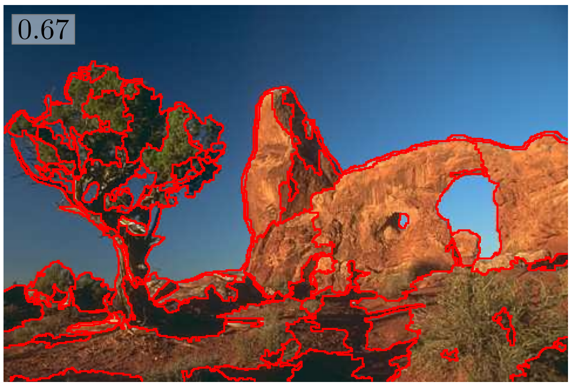
**(e)** *mean cut*



**(f)** *foreground cut*

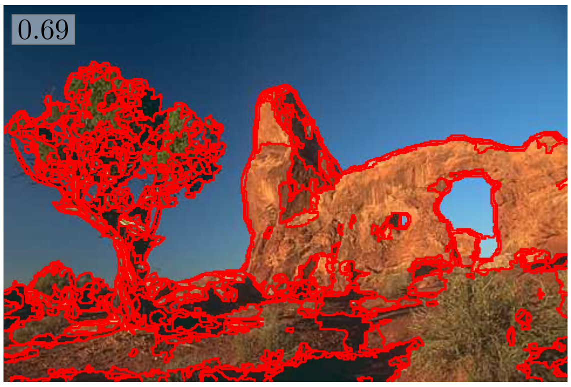


**(g)** *isoperimetric cut*

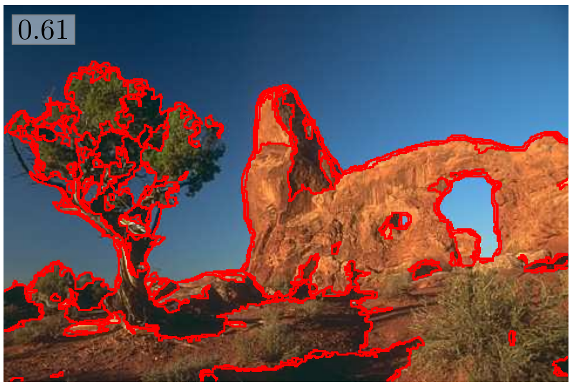


**(h)** *normalized cut*

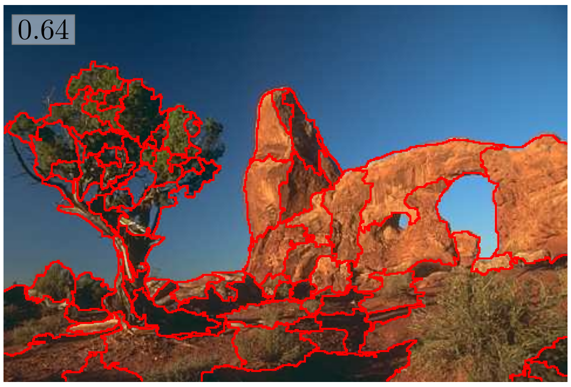
*image-id: 295087; mean shift and* EMD



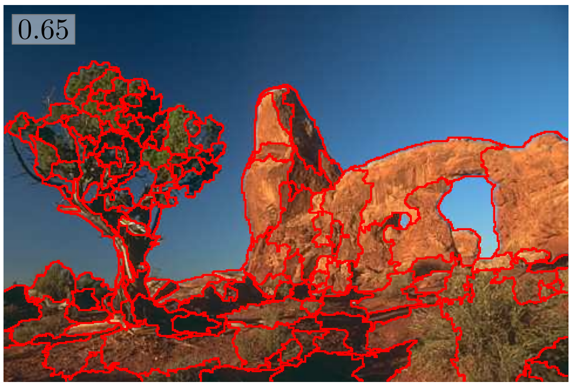
**(d)** *minimum cut*



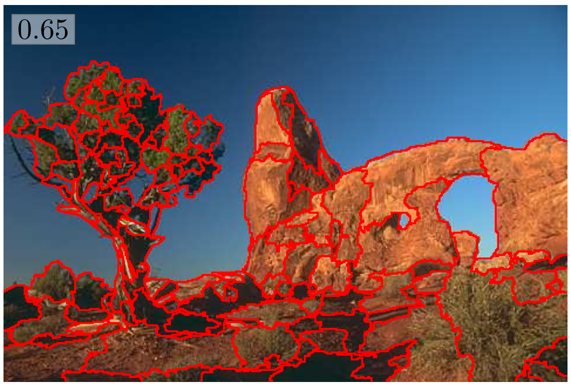
**(e)** *mean cut*



**(f)** *foreground cut*



**(g)** *isoperimetric cut*



**(h)** *normalized cut*

*image-id: 295087; mean shift and* averGrad

# List of Figures

# List of Algorithms

# Bibliography

Ballard, Dana Harry and Christopher M. Brown [1982]. *Computer Vision.* Prentice Hall Professional Technical Reference.

Bentley, Jon Louis [1975]. "Multidimensional binary search trees used for associative searching". In: *Commun. ACM* 18.9. Pp. 509–517.

Beucher, S. and C. Lantuejoul [1979]. "Use of Watersheds in Contour Detection". In: [Oct. 1979].

Boykov Y Veksler, O [2005]. "Handbook of Mathematical Models in Computer Vision". In: ed. by N. Paragios Y. Chen O. Faugeras. Springer-Verlag. Chap. Graph cuts in vision and graphics: Theories and applications, pp. 79,96.

Boykov, Y. Y. and M. P. Jolly [2001]. "Interactive graph cuts for optimal boundary & region segmentation of objects in N-D images". In: *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on* 1. 105–112 vol.1.

Boykov, Y. and V. Kolmogorov [2003]. "Computing geodesics and minimal surfaces via graph cuts". In: *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on* [Oct. 2003]. 26–33 vol.1.

Boykov, Yuri and Gareth Funka-Lea [2006]. "Graph Cuts and Efficient N-D Image Segmentation". In: *Int. J. Comput. Vision* 70.2 [Nov. 2006]. Pp. 109–131.

Boykov, Yuri and Vladimir Kolmogorov [2004]. "An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 26.9 [Oct. 2004]. Pp. 1124–1137.

Boykov, Yuri, Olga Veksler, and Ramin Zabih [1997]. *Markov Random Fields with Efficient Approximations.* Tech. rep. TR97-1658.

Cheng, Yizong [1995]. "Mean shift, mode seeking, and clustering". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 17.8 [Oct. 1995]. Pp. 790–799.

Christofides, Nicos [1975]. *Graph theory: An algorithmic approach (Computer science and applied mathematics).* Academic Press, Inc.

Chung, Fan R. K. [1997]. *Spectral Graph Theory (CBMS Regional Conference Series in Mathematics, No. 92) (Cbms Regional Conference Series in Mathematics).* American Mathematical Society.

Comaniciu, D. and P. Meer [2002]. "Mean shift: a robust approach toward feature space analysis". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 24.5 [Mar. 2002]. Pp. 603–619.

Cook, W. and A. Rohe [1999]. “Computing minimum-weight perfect matchings”. In: *INFORMS Journal on Computing* 11. Pp. 138–148.

Cox, I. J., S. B. Rao, and Yukumar Zhong [1996]. “"Ratio regions": a technique for image segmentation”. In: *Pattern Recognition, 1996., Proceedings of the 13th International Conference on* 2. 557–564 vol.2.

Even, Shimon [1979]. *Graph Algorithms.* New York, NY, USA: W. H. Freeman & Co.

Ford, Adrian and Alan Roberts [1998]. *Colour Space Conversions.* `www.poynton.com/PDFs/coloureq.pdf`. Aug. 1998.

Ford, L.R. and D.R. Fulkerson [1962]. *Flows in Networks.* Princeton University Press, Princeton.

Forsyth, David A. and Jean Ponce [2002]. *Computer Vision: A Modern Approach.* Prentice Hall Professional Technical Reference.

Fukunaga, K. and L. Hostetler [1975]. “The estimation of the gradient of a density function, with applications in pattern recognition”. In: *Information Theory, IEEE Transactions on* 21.1 [Jan. 1975]. Pp. 32–40.

Galil, Zvi, Silvio Micali, and Harold N. Gabow [1986]. “An O(EV log V) Algorithm for Finding a Maximal Weighted Matching in General Graphs”. In: *SIAM J. Comput.* 15.1. Pp. 120–130.

Gallo, G., M. D. Grigoriadis, and R. E. Tarjan [1989]. “A fast parametric maximum flow algorithm and applications”. In: *SIAM J. Comput.* 18.1. Pp. 30–55.

Goldberg, Andrew V. and Kostas Tsioutsiouliklis [2001]. “Cut Tree Algorithms: An Experimental Study”. In: *J. Algorithms* 38.1. Pp. 51–83.

Gomory, R. E. and T. C. Hu [1961]. “Multi-Terminal Network Flows”. In: *Journal of the Society for Industrial and Applied Mathematics* 9.4. Pp. 551–570.

Gonzalez, Rafael C. and Richard E. Woods [2006]. *Digital Image Processing (3rd Edition).* Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Grady, Leo and Eric L. Schwartz [2006*a*]. “Isoperimetric Graph Partitioning for Data Clustering and Image Segmentation”. In: *IEEE Pattern Analysis and Machine Intelligence* 28.3. Pp. 469–475.

– [2006*b*]. “Isoperimetric Graph Partitioning for Image Segmentation”. In: *IEEE Trans. on Pattern Analysis and Machine Intelligence* 28.3 [Mar. 2006]. Pp. 469–475.

Greig, D. M., B. T. Porteous, and A. H. Seheult [1989]. “Exact Maximum A Posteriori Estimation for Binary Images”. In: *Journal of the Royal Statistical Society* 51.2. Pp. 271–279.

Hu, T. C., M. T. Shing, and Y. S. Kuo [1982]. *Combinatorial Algorithms.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Ishikawa, H. and D. Geiger [1998]. “Segmentation by grouping junctions”. In: *Computer Vision and Pattern Recognition, 1998. Proceedings. 1998 IEEE Computer Society Conference on.* Pp. 125–131.

Jüttner, Alpár [2008]. *Lemon Graph Library.* Web Resource: `https://lemon.cs.elte.hu/trac/lemon` [visited September 10, 2008].

Keuchel, Jens [2004]. “Image Partitioning based on Semidefinite Programming”. PhD thesis. Universität Mannheim, Fakultät für Mathematik und Informatik.

Keuchel, Jens, Christoph Schnörr, Christian Schellewald, and Daniel Cremers [2003]. “Binary partitioning, perceptual grouping, and restoration with semidefinite programming”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 25. Pp. 1364–1379.

Kolmogorov, V. and R. Zabin [2004]. “What energy functions can be minimized via graph cuts?” In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 26.2. Pp. 147–159.

Köthe, Ullrich [2000]. “Generische Programmierung für die Bildverarbeitung”. (in german). PhD thesis. Universität Hamburg, Fachbereich Informatik.

– [2002]. “XPMaps and topological segmentation - a unified approach to finite topologies in the plane”. In: *10th Intl. Conference on Discrete Geometry for Computer Imagery (DGCI 2002). Volume 2310 of LNCS.* Springer. Pp. 22–33.

Li, Yin, Jian Sun, Chi-Keung Tang, and Heung-Yeung Shum [2004]. “Lazy snapping”. In: *ACM Trans. Graph.* 23.3 [Aug. 2004]. Pp. 303–308.

Martin, D., C. Fowlkes, D. Tal, and J. Malik [2001]. “A Database of Human Segmented Natural Images and its Application to Evaluating Segmentation Algorithms and Measuring Ecological Statistics”. In: *Proc. 8th Int’l Conf. Computer Vision.* Vol. 2. Pp. 416–423.

Martin, David R. [2003]. “An Empirical Approach to Grouping and Segmentation”. PhD thesis. EECS Department, University of California, Berkeley.

Meine, Hans [2008]. “The GeoMap Representation: On Topologically Correct Sub-pixel Image Analysis”. PhD thesis. Universität Hamburg, Department Informatik der Fkultät für Mathematik, Informatik und Naturwissenschaften.

Meine, Hans and Ullrich Koethe [2005]. *Image Segmentation With The Exact Watershed Transform.* Villanueva, J.J. (Hrsg.): Proceedings of the Fifth IASTED International Conference on Visualization, Imaging, and Image Processing. Benidorm, Spain.

Meine, Hans and Ullrich Köthe [2006]. “A New Sub-pixel Map for Image Analysis”. In: *Proc. 11th Intl. Workshop on Combinatorial Image Analysis (IWCIA0̆6).* Ed. by Ralf Reulke, Ulrich Eckardt, Boris Flach, Uwe Knauer, and Konrad Polthier. Vol. 4040. LNCS. Springer. Pp. 116–130.

Meintrup, David and Stefan Schäffle [2005]. *Stochastik: Theorie und Anwendungen.* Springer.

Park, James K. and Cynthia A. Phillips [1993]. “Finding minimum-quotient cuts in planar graphs”. In: *STOC ’93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing.* New York, NY, USA: ACM Press. Pp. 766–775.

Perona, P. and W. Freeman [1998]. “A Factorization Approach to Grouping”. In: *Lecture Notes in Computer Science* 1406. Pp. 655–670.

Press, William H., Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery [2002]. *Numerical Recipes in C++: The Art of Scientific Computing.* Cambridge University Press.

Ren, Xiaofeng and Jitendra Malik [2003]. “Learning a classification model for segmentation”. In: *In Proc. 9th Int. Conf. Computer Vision*. Pp. 10–17.

Roerdink, J. B. T. M. and Arnold Meijster [2000]. “The Watershed Transform: Definitions, Algorithms and Parallelization Strategies”. In: *Fundamenta Informaticae* 41.1-2. Pp. 187–228.

Rubner, Yossi, Carlo Tomasi, and Leonidas J. Guibas [1998]. “A Metric for Distributions with Applications to Image Databases”. In: *ICCV '98: Proceedings of the Sixth International Conference on Computer Vision*. Washington, DC, USA: IEEE Computer Society. P. 59.

– [2000]. “The Earth Mover's Distance as a Metric for Image Retrieval”. In: *Int. J. Comput. Vision* 40.2. Pp. 99–121.

Sarkar, Sudeep and Padmanabhan Soundararajan [2000]. “Supervised Learning of Large Perceptual Organization: Graph Spectral Partitioning and Learning Automata”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.5. Pp. 504–525.

Shapiro, Linda G. and George C. Stockman [2001]. *Computer Vision*. Prentice Hall.

Shi, Jianbo and Jitendra Malik [1997]. “Normalized Cuts and Image Segmentation”. In: *CVPR '97: Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)*. Washington, DC, USA: IEEE Computer Society. P. 731.

– [2000]. “Normalized Cuts and Image Segmentation”. In:

Sonka, Milan, Vaclav Hlavac, and Roger Boyle [2007]. *Image Processing, Analysis, and Machine Vision*. Thomson-Engineering.

Soundararajan, P. and S. Sarkar [2003]. “An in-depth study of graph partitioning measures for perceptual organization”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 25.6 [June 2003]. Pp. 642–660.

Steger, C. [1999]. “Subpixel-precise extraction of watersheds”. In: *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on* 2. 884–890 vol.2.

Stoer, Mechthild and Frank Wagner [1997]. “A simple min-cut algorithm”. In: *J. ACM* 44.4 [July 1997]. Pp. 585–591.

Unnikrishnan R. Hebert, M. [2005]. “Measures of Similarity”. In: *Application of Computer Vision, 2005. WACV/MOTIONS '05 Volume 1. Seventh IEEE Workshops on*. Vol. 1. Pp. 394–394.

Unnikrishnan, Ranjith, Caroline Pantofaru, and Martial Hebert [2007]. “Toward Objective Evaluation of Image Segmentation Algorithms”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 29.6. Pp. 929–944.

Unser, M., A. Aldroubi, and M. Eden [1993*a*]. “B-Spline Signal Processing: Part I—Theory”. In: *IEEE Transactions on Signal Processing* 41.2 [Feb. 1993]. IEEE Signal Processing Society's 1995 best paper award. Pp. 821–833.

– [1993*b*]. “B-Spline Signal Processing: Part II—Efficient Design and Applications”. In: *IEEE Transactions on Signal Processing* 41.2 [Feb. 1993]. Pp. 834–848.

Veksler, O. [2000]. “Image segmentation by nested cuts”. In: *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on* 1. 339–344 vol.1.

Wang, S. and J. M. Siskind [2001]. “Image segmentation with minimum mean cut”. In: *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on* 1. 517–524 vol.1.

Wang, S. and J. Siskind [2003]. “Image segmentation with ratio cut”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 25. Pp. 675–690.

Wang, S., T. Kubota, J.M. Siskind, and J. Wang [2005]. “Salient closed boundary extraction with ratio contour”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 27.4 [Apr. 2005]. Pp. 546–561.

Wu, Zhenyu and R. Leahy [1990]. “Tissue Classification In MR Images Using Hierarchical Segmentation”. In: *Nuclear Science Symposium, 1990. Conference record : Including Sessions on Nuclear Power Systems and Medical Imaging Conference, 1990 IEEE.* Pp. 1410–1414.

– [1993]. “An optimal graph theoretic approach to data clustering: theory and its application to image segmentation”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 15.11. Pp. 1101–1113.

Zhang, Yu Jin [1996]. “A survey on evaluation methods for image segmentation”. In: *Pattern Recognition* 29.8. Pp. 1335–1346.

– [2001]. “A review of recent evaluation methods for image segmentation”. In: *Signal Processing and its Applications, Sixth International, Symposium on. 2001* 1. 148–151 vol.1.

– [2006]. “An Overview of Image and Video Segmentation in the Last 40 Years”. In: *Advances in Image and Video Segmentation.* Chapter 1 (1–15).

## Erklärung

Ich versichere, daß ich die vorstehende Arbeit selbständig und ohne fremde Hilfe angefertigt und mich nicht anderer als der im beigefügten Verzeichnis angegebenen Hilfsmittel bedient habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich bin mit einer Einstellung in den Bestand der Bibliothek des Departments einverstanden.

Hamburg, den 10. September 2008

(Christian Bähnisch)