Early Vision Optimization: Parametric Models, Parallelization and Curvature

Strandmark, Petter

2010

Early Vision Optimization

Parametric Models, Parallelization and Curvature

Petter Strandmark

Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118
SE-221 00 Lund
Sweden

http://www.maths.lth.se/

Licentiate Theses in Mathematical Sciences 2010:1
ISSN 1404-028X

ISBN 978-91-7473-068-5
LUTFMA-2033-2010

Printed in Sweden by MediaTryck, Lund 2010

Abstract

Early vision is the process occurring before any semantic interpretation of an image takes place. Motion estimation, object segmentation and detection are all parts of early vision, but recognition is not. Many of these tasks are formulated as optimization problems and one of the key factors for the success of recent methods is that they seek to compute globally optimal solutions. This thesis is concerned with improving the efficiency and extending the applicability of the current state of the art. This is achieved by introducing new methods of computing solutions to image segmentation and other problems of early vision. The first part studies parametric problems where model parameters are estimated in addition to an image segmentation. For a small number of parameters these problems can still be solved optimally. In the second part the focus is shifted toward curvature regularization, i.e. when the commonly used length and area regularization is replaced by curvature in two and three dimensions. These problems can be discretized over a mesh and special attention is given to the mesh geometry. Specifically, hexagonal meshes are compared to square ones and a method for generating adaptive methods is introduced and evaluated. The framework is then extended to curvature regularization of surfaces. Thirdly, fast methods for finding minimal graph cuts and solving related problems on modern parallel hardware are developed and extensively evaluated. Finally, the thesis is concluded with two applications to early vision problems: heart segmentation and image registration.

Preface

My studies of image processing and computer vision started with an undergraduate project about wavelets in Lund, but it was the course in image processing lectured by Kalle Åström which really started my interest in the field. My other undergraduate contact with the Mathematical Imaging Group was a course in linear and combinatorial optimization by Fredrik Kahl. Despite these positive experiences I had completely different plans after spending one semester as an exchange student at the University of Illinois. I took a great course in numerical methods for atmospheric sciences, which led me to apply to a PhD position in fluid mechanics at Chalmers University, which I was offered. At that time, however, I had realized that I enjoyed my stay abroad too much and decided to stay for the full year. This meant taking a course in computer vision held by David Forsyth in the spring. After finishing the course, which, ironically, was the course with the lowest grade in my transcript, I took a position as an assistant at Chalmers under the supervision of Irene Gu to write my master's thesis. Ultimately, I ended up back in Lund as a PhD student in the Mathematical Imaging Group.

Most of the work collected in this thesis is the result of collaboration with my colleagues at Lund University. Without them, this thesis could not have come into existence. First, and foremost, I would like to thank my supervisor, Fredrik Kahl, for his help and collaboration on several papers. The chapters on curvature in this thesis has been done in collaboration with Thomas Schoenemann, who co-authored the 2009 paper which became my introduction to the topic. Linus Svärm and I developed the ideas in the very last chapter in this thesis about a year ago. The material about medical imaging, heart segmentation specifically, is more recent, and ongoing with Johannes Ulén as the principal investigator. Niels-Christian Overgaard was of great help for the very first paper. I am grateful to be able to collaborate with such professional people. Everyone at the Centre for Mathematical Sciences has my thanks for providing great company, courses and seminars.

Contents

Chapter 1

Introduction

Eyes have evolved independently in at least forty groups of animals (Coyne, 2009). Despite considerable complexities and evolutionary hurdles, the camera-type eye, with a lens focusing the image onto the retina, has evolved in very different groups: from vertebrates to crustaceans and jellyfish (Nilsson, 1999). This is a testament to the importance of vision for life on earth and a suggestion that artificial vision systems, if constructed properly, will be immensely useful to machines for navigation, interaction and measurement. The construction of such artificial vision systems may therefore be seen as a subfield of artificial intelligence, and is commonly referred to as *computer vision*. It is a relatively recent subfield of artificial intelligence; recent because it usually involves processing large quantity of data – even very small images could exhaust the best computers in the 70s. The history of artificial intelligence itself goes back to at least 1950, when Alan Turing concluded his seminal paper with the following paragraph, suggesting two lines of future research:

> We may hope that machines will eventually compete with men in all purely intellectual fields. But which are the best ones to start with? Even this is a difficult decision. Many people think that a very abstract activity, like the playing of chess would be best. It can also be maintained that it is best to provide the machine with the best sense organs that money can buy, and then teach it to understand and speak English. This process could follow the normal teaching of a child. Things would be pointed out and named, etc. Again I do not know what the right answer is, but I think both approaches should be tried.

Half a century later, the first possibility has turned out to be overwhelmingly successful, whereas the second has had little success. "The best sense organs that money can buy" are indeed stunningly powerful today, but any sense organ, visual ones certainly not excluded, need a powerful computational system to process and interpret the data. One of the ultimate goals of computer vision is precisely this.

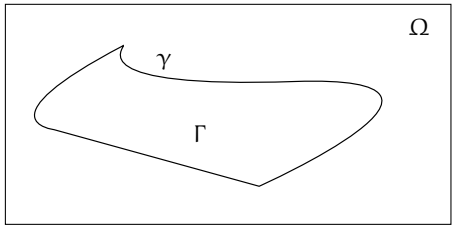


Figure 1.1: Segmentation of an image. The region Γ has the curve γ as its boundary.

Of course, the goal of this thesis is much more modest. I will describe methods for solving certain optimization problems commonly arising in computer vision. One of the goals will be *globality*, that is, the solution obtained should be guaranteed to be the best possible. Two other goals will be *parallelization* to make use of modern microprocessors and *memory efficiency*, since solving computer vision problems can be prohibitively expensive in terms of memory requirements.

The state of the art of computer vision highly depends on what applications one has in mind. Computers are able to estimate the scene depth from a pair of images with great accuracy and absolute precision and to recover the 3D structure of an entire city just from arbitrarily downloaded images from the Internet (Agarwal et al., 2009). On the other hand, computers are still unable to perform tasks which humans perform effortlessly, such as determining whether an image contains a chair or not. Recognition is an example of high-level vision and is very difficult for computers. In contrast, my thesis will focus on low-level vision – tasks which humans perform subconsciously in every waking second, but still provide challenges for designers of artificial vision systems. These tasks are also commonly referred to as tasks of *early vision*.

1.1 Overview of the Thesis

A major part of my thesis is devoted to image segmentation. A segment of an image is described by a simple closed curve γ enclosing an area $\Gamma \subseteq \Omega$ in such a way that the area represents something significant about the image, e.g. foreground or an interesting object, see Fig. 1.1. The segmentation task is to find the best (closed) curve γ given an image I, or more precisely,

$$\underset{\gamma}{\text{maximize}} \quad \mathrm{P}(\gamma \,|\, I), \tag{1.1}$$

where $\mathrm{P}(\gamma \,|\, I)$ is the posterior probability of γ given the image $I : \Omega \to \mathbf{R}^3$ (or $\mathbf{R}$ if the image is black and white). This thesis will discuss various aspects of solving (1.1). Using Bayes' rule, this expression becomes $\mathrm{P}(\gamma \,|\, I) = \mathrm{P}(\gamma)\,\mathrm{P}(I \,|\, \gamma)/\mathrm{P}(I)$. If the log-likelihood is denoted $\ell(\gamma) = \log \mathrm{P}(\gamma)$,

$$\ell(\gamma \,|\, I) = \ell(\gamma) + \ell(I \,|\, \gamma) \underbrace{-\ell(I)}_{\substack{\text{independent} \\ \text{of } \gamma}} . \tag{1.2}$$

In computer vision and image analysis the probabilistic interpretation is sometimes dropped and $\ell(\gamma \,|\, I)$ is simply referred to as the energy of γ. The reason for this is the influence from the calculus of variations used in physics. Consequently, image segmentation and other tasks are ofter referred to by computer vision researchers as energy minimization problems. The prior $\ell(\gamma)$ is often called the regularizing term, or smoothness term and $\ell(I \,|\, \gamma)$ is called the data term. I will use this vocabulary throughout the thesis. Letting $E_{\text{smooth}}(\gamma) = -\ell(\gamma)$ and $E_{\text{data}}(\gamma) = -\ell(I \,|\, \gamma)$, we have finally arrived at the optimization problem

$$\underset{\gamma}{\text{minimize}} \quad E_{\text{smooth}}(\gamma) + E_{\text{data}}(\gamma). \tag{1.3}$$

It is equivalent to (1.1), but more similar to how these problems are commonly presented in the computer vision community. The data term is the likelihood of an image given a particular segmentation and is commonly written as

$$E_{\text{data}}(\gamma) = \int_{\Gamma} -\ell(\boldsymbol{x} \text{ is foreground} \,|\, I(\boldsymbol{x}))d\boldsymbol{x} + \int_{\Omega \setminus \Gamma} -\ell(\boldsymbol{x} \text{ is background} \,|\, I(\boldsymbol{x}))d\boldsymbol{x}. \tag{1.4}$$

Chapter 3 describes briefly how smoothness and regularizing terms arise in the context of conditional independence between image pixels (the Markov property). The data term require likelihoods $\ell(\boldsymbol{x} \text{ is foreground} \,|\, I(\boldsymbol{x}))$ and $\ell(\boldsymbol{x} \text{ is background} \,|\, I(\boldsymbol{x}))$ for each position $\boldsymbol{x}$ in the image. These are highly application-dependent. A typical simple application uses some known or estimated color histogram to model the two regions. Mumford and Shah (1989) describe various forms $\ell(\gamma)$ and $\ell(I \,|\, \gamma)$ can take.

Maximizing (1.3) with data term (1.4) becomes very easy under the assumption that all possible curves γ are equally likely *a priori*, i.e. $\ell(\gamma)$ is constant. Regrettably,

the resulting model is often not good enough. Each point in the image can under this assumption be classified independently of the others, which is unrealistic. This assumption can be removed and a more complicated prior is then imposed on the curve γ. Several chapters in this thesis will discuss the aspects of the optimization problems arising for different classes of regularizing terms.

Chapter 4 This chapter considers the case where the prior of the curve γ is a linear function of its length:

$$\ell(\gamma) = -\rho \operatorname{length}(\gamma), \qquad \rho \geq 0. \tag{1.5}$$

The theory developed can also handle slight variations thereof, such as anisotropic length (Olsson et al., 2009). This optimization problem is well-known to be solvable and I will describe methods to simultaneously find the best possible *data term* under some conditions. As a simple example, one might wish to model an image as having foreground and background pixels drawn from two different Gaussian distributions with (unknown) means but equal variance. The techniques developed in Chapter 4 enable the efficient computation of the optimally estimated means simultaneously with the optimal segmentation.

Chapter 5 After covering length-based regularization, the focus is shifted to more general functionals involving curvature:

$$\ell(\gamma) = -\rho \operatorname{length}(\gamma) - \sigma \int_\gamma |\kappa(s)|^p ds, \qquad \rho, \sigma \geq 0, \tag{1.6}$$

where the curvature $\kappa(s) = ||\boldsymbol{\gamma}''(s)||$ for a curve γ parametrized by its arc length s. Functionals with such terms can be discretized with a grid and subsequently solved with linear programming, as was shown recently by Schoenemann, Kahl and Cremers (2009). Chapter 5 will build upon their work and extend the method in several ways. A new set of constraints will be introduced which fixes an issue in the original formulation. The framework is then extended in two directions: It is shown that the square grid naturally arising from the image pixels is not as efficient as hexagonal grids. Complementary to this, a method for adaptive grid generation is discussed and tested experimentally.

Chapter 6 Whereas the previous chapter expanded upon the existing framework for functionals involving the curvature of plane curves, Chapter 6 will introduce

a new framework for minimizing the curvature of surfaces. Applications include three-dimensional segmentation and surface completion, for example within stereo vision.

Chapter 7 Image segmentation with length regularity may also be converted into the graph theoretical problem of finding the minimum cut in a directed graph. The introductory third chapter will describe how this is done for Markov random fields. Many other types of problems can also be formulated as minimum cut problems. Fast algorithms for solving these problems are therefore of utmost importance in computer vision. Chapter 7 will introduce a method for solving the minimum cut problem in parallel, solving small problems faster and making larger problems tractable by distributing them across multiple computers. This will be achieved by dual decomposition, a technique which is summarized in Chapter 2.

Chapter 8 I could walk down to the store today and, for a modest amount of money, buy a graphics card capable of performing more than a trillion arithmetic operations per second. The highly parallel graphical processing units (GPUs) have in the last decade found their way into computing, with manufacturers like Nvidia diverting more and more die space from gaming to computing. This chapter continues the search for fast algorithms for graph cuts and other more general labeling problems by exploring possible GPU algorithms. An algorithm based on dual decomposition and dynamic programming is introduced, similar to one presented by Komodakis, Paragios and Tziritas (2007).

Chapter 9 The penultimate chapter considers an application of dual decomposition to the 3D segmentation of the human heart. While this problem has many interesting aspects, I will focus on describing a multi-region model of the heart and how dual decomposition can be used to avoid the costly (w.r.t. memory) use of quadratic pseudo-Boolean optimization (QPBO).

Chapter 10 Finally, the last chapter considers an application of multi-label energy minimization to image registration. With both the number of variables and the number of labels ranging in the millions, the size of the search spaceis too large for any guarantee of globality or approximations thereof.

1.2 Previous Publications

The material presented in this thesis is largely based on previous publications:

- Petter Strandmark, Fredrik Kahl and Niels Chr. Overgaard, *Optimizing Parametric Total Variation Models*, International Conference on Computer Vision (ICCV), Kyoto, 2009.
- Petter Strandmark and Fredrik Kahl, *Parallel and Distributed Graph Cuts by Dual Decomposition*, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), San Francisco, 2010.
- Linus Svärm and Petter Strandmark, *Shift-map Image Registration*, International Conference on Pattern Recognition (ICPR), Istanbul, 2010.
- Petter Strandmark, Fredrik Kahl and Thomas Schoenemann, *Parallel and Distributed Vision Algorithms Using Dual Decomposition*, submitted 2010.
- Petter Strandmark and Fredrik Kahl *Curvature Regularization for Curves and Surfaces in a Global Optimization Framework*, submitted 2010.

Various parts of this work have also been presented at the Swedish Symposium on Image Analysis (SSBA) 2009 and 2010.

Chapter 2

Optimization Methods

This chapter will provide an overview of optimization techniques used throughout this thesis. My view of the field of convex optimization comes from the books by Boyd and Vandenberghe (2004), Bertsekas (1999) and Nesterov (2004). Dual decomposition is a technique for splitting up an optimization problem into smaller and, hopefully, easier subproblems. These subproblems are solved iteratively to obtain a solution to the original problem. Another widely used method is branch and bound, which solves hard optimization problems by searching the spaces of feasible solutions in a clever way, thereby eliminating the need for an exhaustive search. The second half of this chapter will cover methods from combinatorial optimization, which are of utmost importance for modern low-level vision.

2.1 Convex Functions

The minimization of a differentiable function $f \in \mathcal{C}^1(\mathbf{R}^n)$ is in general an intractable task. The class of differentiable functions is too large and has too little structure for there to exist any method to minimize them all in any reasonable amount of time. The best one can expect from gradient descent schemes is the convergence to a local minima in general, not the global minimum. One might then ask for a class of functions which are possible to minimize globally in a reasonable amount of time. What would be a suitable definition of such a class $\mathcal{F} \subseteq \mathcal{C}^1(\mathbf{R}^n)$?

We must be able to tell whether we have reached the global optimum. Furthermore, since we cannot search the entire domain, such a certificate must be possible to compute locally. Therefore, since the functions in $\mathcal{F}$ are only assumed to be differentiable it is natural to require that

$$f \in \mathcal{F}, \quad \nabla f(\boldsymbol{x}^*) = \mathbf{0} \qquad \Longrightarrow \qquad f(\boldsymbol{x}^*) \text{ is the global minimum.} \qquad \text{(A)}$$

The class $\mathcal{F}$ consists of the functions that are "easy" to minimize. We should expect that if f and g are easy to minimize, so is the function $f + g$:

$$f, g \in \mathcal{F} \qquad \Longrightarrow \qquad f + g \in \mathcal{F}. \tag{B}$$

Lastly, we require that the affine functions are members of $\mathcal{F}$. They are certainly easy to minimize over $\mathbf{R}^n$.

$$\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x} + \boldsymbol{b} \in \mathcal{F}. \tag{C}$$

It turns out that the requirements A–C are enough to derive a simple characterization of the functions in $\mathcal{F}$. Let $f \in \mathcal{F}$ and $\boldsymbol{x}_0 \in \mathbf{R}^n$ be fixed. Consider the function g defined by

$$g(\boldsymbol{x}) = f(\boldsymbol{x}) - \nabla f(\boldsymbol{x}_0)^{\mathrm{T}}\boldsymbol{x}. \tag{2.1}$$

Assumptions B and C tells us that $g \in \mathcal{F}$. We have that $\nabla g(\boldsymbol{x}_0) = \nabla f(\boldsymbol{x}_0) - \nabla f(\boldsymbol{x}_0) = \mathbf{0}$. Therefore, $\boldsymbol{x}_0$ is the global minimizer to g according to assumption A which means that for any $\boldsymbol{x} \in \mathbf{R}^n$,

$$g(\boldsymbol{x}) \geq g(\boldsymbol{x}_0) = f(\boldsymbol{x}_0) - \nabla f(\boldsymbol{x}_0)^{\mathrm{T}}\boldsymbol{x}_0 \tag{2.2}$$

and thus,

$$f(\boldsymbol{x}) \geq f(\boldsymbol{x}_0) + \nabla f(\boldsymbol{x}_0)^{\mathrm{T}}(\boldsymbol{x} - \boldsymbol{x}_0). \tag{2.3}$$

This is the definition of **differentiable convex functions** on $\mathbf{R}^n$. Conversely, it is easily seen that A and B hold for convex functions. This argument (Nesterov, 2004) shows the importance of convex functions in optimization. In general, convex functions are defined by the inequality

$$f(t\boldsymbol{x} + (1-t)\boldsymbol{y}) \leq tf(\boldsymbol{x}) + (1-t)f(\boldsymbol{y}), \qquad t \in [0, 1]. \tag{2.4}$$

If instead the reverse inequality holds for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbf{R}^n$, the function is said to be **concave**.

Convex functions need not be differentiable. Still, it is always possible to find a tangent plane completely below the function graph at any point. More precisely, for any $\boldsymbol{x}_0$ it is always possible to find a vector $\boldsymbol{g} \in \mathbf{R}^n$ such that

$$f(\boldsymbol{x}) \geq f(\boldsymbol{x}_0) + \boldsymbol{g}^{\mathrm{T}}(\boldsymbol{x} - \boldsymbol{x}_0), \qquad \text{for all } \boldsymbol{x}. \tag{2.5}$$

$\boldsymbol{g}$ is then called a **subgradient** of f at $\boldsymbol{x}_0$. The analogue for concave functions with a reversed inequality are **supergradients**, shown in Figure 2.1. If f is differentiable, $\boldsymbol{g} = \nabla f(\boldsymbol{x}_0)$ and is unique. If not, the set of all $\boldsymbol{g}$ satisfying (2.5) is denoted $\nabla f(\boldsymbol{x}_0)$. This set is convex. One can observe that $\mathbf{0} \in \nabla f(\boldsymbol{x}_0)$ if and only if $\boldsymbol{x}_0$ is the global optimum.

2.2 Dual Decomposition

This section introduces dual decomposition (Bertsekas, 1999), a general technique in optimization to split a large problem into two or more smaller, better manageable ones. The technique of decomposing problems with dual variables was introduced by Everett (1963) and it has been used in many different contexts, e.g. in control (Rantzer, 2009). The application within computer vision that bears the most resemblance to that of this thesis is the work of Komodakis, Paragios and Tziritas (2007) where dual decomposition is used for computing approximate solutions to general Markov random field (MRF) problems, see Chapter 3. Consider the following optimization problem:

$$\inf_{\boldsymbol{x} \in X} E(\boldsymbol{x}), \tag{P}$$

where X and E are arbitrary. As mentioned in the preceding section, this problem is in general very hard to solve. Sometimes the function E can be split up into two parts: $E(\boldsymbol{x}) = E_1(\boldsymbol{x}) + E_2(\boldsymbol{x})$, where the functions E_1 and E_2 are much easier to minimize. We will see later that this is the case with many energy functions associated with MRFs. Now let us consider the equivalent optimization problem

$$\begin{aligned} &\inf_{\boldsymbol{x}, \boldsymbol{y} \in X} E_1(\boldsymbol{x}) + E_2(\boldsymbol{y}) \\ &\text{subject to} \quad \boldsymbol{x} = \boldsymbol{y}. \end{aligned} \tag{2.6}$$

The dual function (Bertsekas, 1999; Boyd and Vandenberghe, 2004) of this optimization problem is

$$\begin{aligned} d(\boldsymbol{\lambda}) &= \inf_{\boldsymbol{x}, \boldsymbol{y} \in X} \Big(E_1(\boldsymbol{x}) + E_2(\boldsymbol{y}) + \boldsymbol{\lambda}^{\mathrm{T}}(\boldsymbol{x} - \boldsymbol{y}) \Big) \\ &= \inf_{\boldsymbol{x} \in X} \Big(E_1(\boldsymbol{x}) + \boldsymbol{\lambda}^{\mathrm{T}} \boldsymbol{x} \Big) \\ &+ \inf_{\boldsymbol{y} \in X} \Big(E_2(\boldsymbol{y}) - \boldsymbol{\lambda}^{\mathrm{T}} \boldsymbol{y} \Big). \end{aligned} \tag{2.7}$$

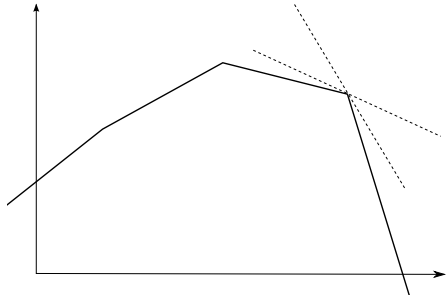


Figure 2.1: Two different supergradients of a concave function represented as two tangent planes.

From the last two rows we see that evaluating the dual function is equivalent to solving two independent minimization problems. If minimizing E_1 and E_2 is much easier than minimizing E, the dual function can be evaluated quite efficiently. Since the value of the dual function $d(\boldsymbol{\lambda})$ is a lower bound on the solution to (P) for every $\boldsymbol{\lambda}$, it is of great interest to compute the optimal such bound, i.e. solve the dual problem:

$$\sup_{\boldsymbol{\lambda}} d(\boldsymbol{\lambda}). \tag{D}$$

The dual function is concave, since it is the infimum over a set of concave (affine) functions of $\boldsymbol{\lambda}$. Furthermore, it is also easy to find a supergradient to d, as stated by the following lemma:

Lemma 2.1. *Given a $\boldsymbol{\lambda}_0$, let $\boldsymbol{x}^*$ be an optimal solution to*

$$d(\boldsymbol{\lambda}_0) = \min_{\boldsymbol{x}} \Big(f(\boldsymbol{x}) + \boldsymbol{\lambda}_0^{\mathrm{T}} \boldsymbol{g}(\boldsymbol{x}) \Big). \tag{2.8}$$

Then $\boldsymbol{g}(\boldsymbol{x}^*)$ is a supergradient to d at $\boldsymbol{\lambda}_0$.

Proof. The following inequality hold for any $\boldsymbol{\lambda}$:

$$\begin{aligned} d(\boldsymbol{\lambda}) &\leq f(\boldsymbol{x}^*) + \boldsymbol{\lambda}^{\mathrm{T}} \boldsymbol{g}(\boldsymbol{x}^*) \\ &= f(\boldsymbol{x}^*) + \boldsymbol{\lambda}_0^{\mathrm{T}} \boldsymbol{g}(\boldsymbol{x}^*) + (\boldsymbol{\lambda} - \boldsymbol{\lambda}_0)^{\mathrm{T}} \boldsymbol{g}(\boldsymbol{x}^*) \\ &= \min_{\boldsymbol{x}} \Big(f(\boldsymbol{x}) + \boldsymbol{\lambda}_0^{\mathrm{T}} \boldsymbol{g}(\boldsymbol{x}) \Big) + (\boldsymbol{\lambda} - \boldsymbol{\lambda}_0)^{\mathrm{T}} \boldsymbol{g}(\boldsymbol{x}^*) \\ &= d(\boldsymbol{\lambda}_0) + (\boldsymbol{\lambda} - \boldsymbol{\lambda}_0)^{\mathrm{T}} \boldsymbol{g}(\boldsymbol{x}^*), \end{aligned} \tag{2.9}$$

which is the definition of a supergradient. □

Maximizing the dual function d can then be done with subgradient methods as described by Bertsekas (1999). Given a $\boldsymbol{\lambda}$, one takes a step in the direction of a supergradient $\boldsymbol{x} - \boldsymbol{y} \in \nabla d(\boldsymbol{\lambda})$:

Start with $\boldsymbol{\lambda} = \mathbf{0}$
repeat
  Update $\boldsymbol{x}$ and $\boldsymbol{y}$ by evaluating $d(\boldsymbol{\lambda})$
  $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} + \tau(\boldsymbol{x} - \boldsymbol{y})$, for some τ
until $\boldsymbol{x} = \boldsymbol{y}$

During the maximization of the dual function, we have access to a lower bound $d(\boldsymbol{\lambda})$ of the optimal function value, and also to two feasible solutions $\boldsymbol{x}$ and $\boldsymbol{y}$ of the original problem. An upper bound on the optimal function value is then given by $\min(E(\boldsymbol{x}), E(\boldsymbol{y}))$. We can compute the *relative duality gap* which gives an indication of how close we are to the optimum:

$$r = \frac{\min(E(\boldsymbol{x}), E(\boldsymbol{y})) - d(\boldsymbol{\lambda})}{\min(E(\boldsymbol{x}), E(\boldsymbol{y}))}. \tag{2.10}$$

If X is a convex set and E a convex function, the optimal duality gap is generally zero. However, in subsequent chapters I consider non-convex sets consisting of a finite set of points with integral coordinates. In Chapter 7 an optimal gap of zero is nonetheless guaranteed as it corresponds to solving an integer linear program whose linear programming relaxation represents the convex hull of the feasible integral solutions.

2.2.1 Choices of Step Sizes and Their Limitations

A step size τ_k needs to be chosen in each iteration k. One of the simplest ways of doing this and still ensuring convergence is to pick $\tau_k = T/k$, with T a constant (Bertsekas, 1999). In fact, all step sizes chosen such that $\tau_k \to 0$ and $\sum_{k=1}^{\infty} \tau_k = \infty$ guarantee convergence. The step sizes for subgradient minimization can be chosen in more sophisticated ways (Bertsekas, 1999), but minimizing a non-differentiable convex function is still much harder than minimizing a differentiable convex function. This somewhat disappointing result is a consequence of the following theorem which describes a worst-case scenario:

Theorem 2.2. *For any $k > 0$ and $L, R > 0$, there exists a convex function f, Lipschitz continuous with constant L with a minimizer $\boldsymbol{x}^*$ such that*

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}^*) \geq \frac{LR}{2(1+\sqrt{k+1})}, \tag{2.11}$$

for any optimization scheme with $||\boldsymbol{x}_0 - \boldsymbol{x}^*|| \leq R$ generating its points from subgradients $\boldsymbol{g}$ of f in the following way:

$$\boldsymbol{x}_k \in \boldsymbol{x}_0 + \operatorname{span}\{\boldsymbol{g}(\boldsymbol{x}_0), \ldots, \boldsymbol{g}(\boldsymbol{x}_{k-1})\}. \tag{2.12}$$

The proof can be found in the book by Nesterov (2004, p. 138). The consequence of this theorem is that $O(\frac{1}{\varepsilon^2})$ iterations are needed to get within ε of the optimal function value for any optimization method generating its iterates as (2.12). This is much worse than for differentiable convex functions, since there are first-order methods requiring $O(\frac{1}{\sqrt{\varepsilon}})$ iterations.[1]

Despite this, we will sometimes see fast convergence for the applications in this thesis. The problems in Chapter 7 converge quickly, but the convergence in Chapter 8 is much slower.

2.2.2 Projected Supergradient Method

Let us instead consider a minimization problem with an inequality constraint:

$$\begin{aligned} &\inf_{\boldsymbol{x} \in X} E(\boldsymbol{x}) \\ &\text{subject to} \quad \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0}. \end{aligned} \tag{2.13}$$

The dual function of this problem is $d(\boldsymbol{\lambda}) = \inf_{\boldsymbol{x} \in X}(E(\boldsymbol{x}) + \boldsymbol{\lambda}^{\mathrm{T}}\boldsymbol{g}(\boldsymbol{x}))$, but its domain is now $\{\boldsymbol{\lambda} \geq \boldsymbol{0}\}$, which turns the dual problem into a constrained maximization problem. The updating of $\boldsymbol{\lambda}$ in the algorithm on page 11 is changed into:

$$\boldsymbol{\lambda} \leftarrow \left[\boldsymbol{\lambda} + \tau \boldsymbol{g}(\boldsymbol{x})\right]^+, \tag{2.14}$$

where $[\cdot]^+$ is the orthogonal projection onto the feasible set $\{\boldsymbol{\lambda} \geq \boldsymbol{0}\}$.

[1] Close to the optimal point, first-order methods may display linear convergence, which require $O(\log C/\varepsilon)$ iterations. Newton's method require $O(\log\log C/\varepsilon)$ iterations once sufficiently close to the optimal point (Nesterov, 2004, p. 36).

2.3 Branch and Bound

Let us now once again consider the problem of minimizing a function $E(\boldsymbol{x})$, where $\boldsymbol{x}$ takes values in an arbitrary set X. If X is partitioned into $\mathcal{X} = \{S_1, \ldots, S_n\}$, we clearly have:

$$\inf_{\boldsymbol{x} \in X} E(\boldsymbol{x}) = \inf_{i=1\ldots n} \inf_{\boldsymbol{x} \in S_i} E(\boldsymbol{x}). \tag{2.15}$$

Let E_S^* be the optimal value of E on the set S. In general, E_S^* is hard to compute. Now assume that we can compute *bounds* of E_S^* much more efficiently than computing E_S^* itself. With a method to compute $\underline{E_S^*}$ and $\overline{E_S^*}$ such that

$$\underline{E_S^*} \leq E_S^* \leq \overline{E_S^*}, \tag{2.16}$$

we can compare the set S to other parts of the search space X. To see this, assume we have computed these bounds for two sets S and T. If $\overline{E_S^*} < \underline{E_T^*}$, we can disregard the set T from further processing, since it is guaranteed to not contain the global minimum of E. An upper bound is usually easy to compute, since it is possible to choose $\overline{E_S^*} = E(\boldsymbol{x})$ where $\boldsymbol{x}$ is any point in S. Finding good ways to compute lower bounds is harder and completely problem-specific. As an example of a lower bound, consider the minimum cut problem. If there is a path from s to t, the lowest arc cost in that path is a lower bound of the minimum cut value.

The process of minimizing E consists of two steps which are iterated. First, bounds are computed for every set in the partition after which sets known not to contain the global minimum are removed. Second, any remaining sets are further subdivided. This is the *branch* step. Formally, the algorithm starts with $\mathcal{X} = \{X\}$ and proceeds as shown in Figure 2.2.

2.3.1 Example

As an example, consider the following pseudo-Boolean optimization problem:

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{minimize}} \quad & \boldsymbol{x}^{\mathrm{T}} \mathsf{A} \boldsymbol{x} + \boldsymbol{c}^{\mathrm{T}} \boldsymbol{x} \\ \text{subject to} \quad & \boldsymbol{x} \in \{0,1\}^n, \end{aligned} \tag{2.17}$$

where A is a symmetric positive definite matrix. In the terminology of the previous section, $E(\boldsymbol{x}) = \boldsymbol{x}^{\mathrm{T}} \mathsf{A} \boldsymbol{x} + \boldsymbol{c}^{\mathrm{T}} \boldsymbol{x}$ and

$$X = \{0,1\}^n. \tag{2.18}$$

$\mathcal{X} = \{X\}$
repeat
  $\overline{E^*} \leftarrow \min_{S \in \mathcal{X}} \overline{E_S^*}$
  $\underline{E^*} \leftarrow \max_{S \in \mathcal{X}} \underline{E_S^*}$
  foreach $S \in \mathcal{X}$ **do**
    if $\overline{E^*} < \underline{E_S^*}$ **then**
      Remove S from $\mathcal{X}$
    else
      Replace S in $\mathcal{X}$ by a partition of S
    end
  end
until $\overline{E^*} - \underline{E^*}$ *is small enough*

Figure 2.2: Branch and bound algorithm to minimize E over the set X.

One simple way of branching (dividing the search space) is to use the following partition $\{S_0, S_1\}$ of X:

$$\begin{aligned} S_0 &= X \cap \{\boldsymbol{x} \,|\, x_1 = 0\} \\ S_1 &= X \cap \{\boldsymbol{x} \,|\, x_1 = 1\}. \end{aligned} \tag{2.19}$$

This can be continued for more variables:

$$\begin{aligned} S_{00} &= X \cap \{\boldsymbol{x} \,|\, x_1 = 0, x_2 = 0\} \\ S_{01} &= X \cap \{\boldsymbol{x} \,|\, x_1 = 0, x_2 = 1\}. \\ &\vdots \end{aligned} \tag{2.20}$$

Both $\{S_0, S_1\}$ and $\{S_{00}, S_{01}, S_{10}, S_{11}\}$ are partitions of X, and so on for higher numbers of fixed elements of $\boldsymbol{x}$. A lower bound can be computed by replacing the set X with $\tilde{X} = [0, 1]^n$ and so on for $\tilde{S}_0, \tilde{S}_1$ etc. We clearly have $X \subset \tilde{X}$, $S_0 \subset \tilde{S}_0$ etc. which means that minimizing E over these larger sets will give a lower bound of the optimum values of the Boolean sets. For an upper bound, we can round the real-valued solution vector and evaluate $E(\boldsymbol{x})$ for the resulting Boolean vector. The following numerical example will further explain the procedure. Let

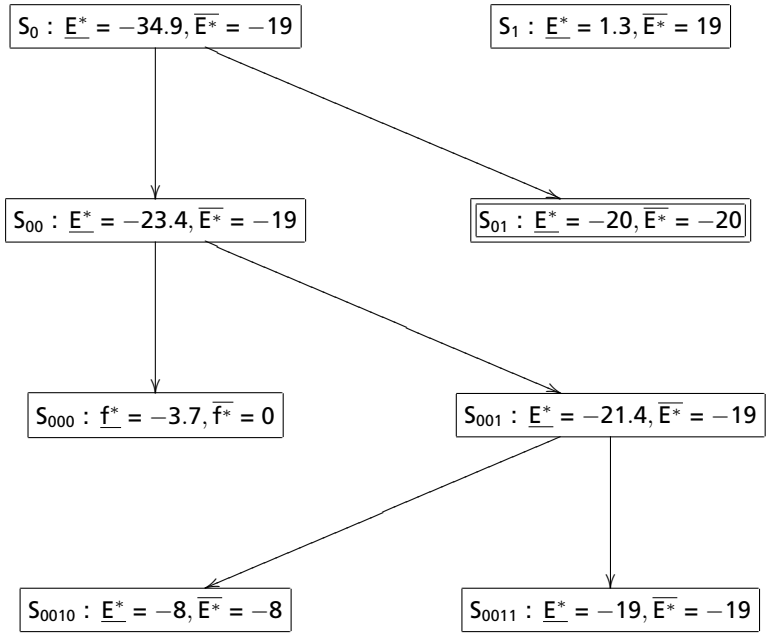


Figure 2.3: The branch and bound tree for the boolean optimization problem (2.17), (2.21). The optimal value is -20. Each line is one outer iteration in Fig. 2.2

us take

$$A = \begin{bmatrix} 46 & 3 & 16 & -15 & -12 \\ 3 & 59 & -10 & -8 & -12 \\ 16 & -10 & 39 & -10 & 16 \\ -15 & -8 & -10 & 27 & 8 \\ -12 & -12 & 16 & 8 & 46 \end{bmatrix} \quad \boldsymbol{c} = \begin{bmatrix} -6 \\ -24 \\ -47 \\ -18 \\ -16 \end{bmatrix}. \tag{2.21}$$

The complete optimization procedure is shown as a tree in Figure 2.3. It starts by computing bounds for S_0 and S_1. This immediately results in S_1 to be discarded. S_0 is split into S_{00} and S_{01}. For S_{01} the optimal value is known since the relaxed and the rounded values are equal. Further branching down the tree shows that it is in fact globally optimal for X.

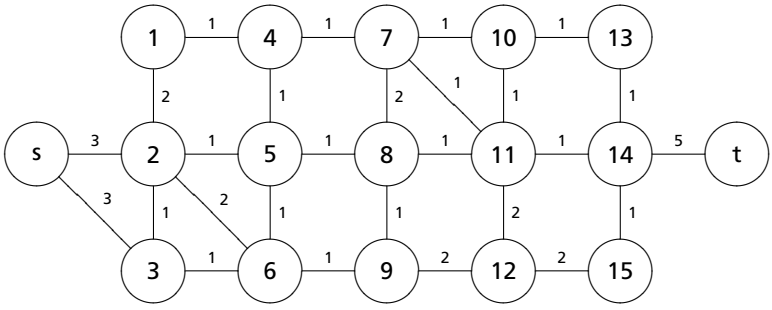


Figure 2.4: Example of an undirected graph. One minimum cut in this graph is S = $\{s, 1 \dots 6\}$ and T = $\{t, 7 \dots 15\}$. Another is S′ = $\{s, 1 \dots 12, 15\}$ and T′ = $\{t, 13, 14\}$. Both have the value 3.

Branch and bound is a very general method and its efficiency highly depends on the quality of the computed bounds. I will use branch and bound in chapter 4 to compute optimal solutions to a special class of segmentation problems.

2.4 Minimum Cut of a Graph

Let G be a graph with n nodes (or vertices) in addition to two special nodes s and t. A non-negative weight $w_{i,j}$ is associated to each pair (i, j) of nodes, which is greater than 0 if there is an edge between i and j. Because no self-loops are allowed in the graph, $w_{i,i} = 0$ is true for all i.

A **cut** is a partition of the nodes of G into two sets S and T, such that $s \in S$ and $t \in T$. The **value** of a cut is the sum of all weights of edges leading from S to T. If all these edges are removed, there is no path from s to t, hence the name "cut". The **minimum cut** of G is the cut with the smallest possible value. Figure 2.4 shows an example of an undirected graph ($w_{i,j} = w_{j,i}$) and its minimum cuts.

Now introduce indicator variables $\boldsymbol{x} = (x_1, \dots x_n)$ where x_i is equal to 0 if node i is in S and 1 otherwise. For convenience, let $\mathcal{N}_i$ denote the neighborhood of i, that is, the set of all nodes $j \neq s, t$ such that there is an edge between i and j. For example, $\mathcal{N}_5 = \{2, 4, 6, 8\}$ in Figure 2.4. With this notation the problem of finding the minimum cut is to

$$\underset{\boldsymbol{x} \in \{0,1\}^n}{\text{minimize}} \quad \sum_{i=1}^{n} \sum_{j \in \mathcal{N}_i} w_{i,j}(1 - x_i)x_j + \sum_{i=1}^{n} w_{s,i}(1 - x_i) + \sum_{i=1}^{n} w_{i,t}x_i. \quad (2.22)$$

There are efficient methods for solving this optimization problem if $w_{i,j} \geq 0$ for all $i, j \in \{1 \ldots n\}$. The algorithm by Boykov and Kolmogorov (2004) is used extensively by the computer vision community.

2.5 Pseudo-Boolean Optimization

The optimization problems (2.17) and (2.22) are examples of pseudo-Boolean optimization problems, i.e. problems where a function $f : \{0,1\}^n \to \mathbf{R}$ is minimized. These optimization problems are thoroughly treated by Boros and Hammer (2002). With the introduction of auxiliary variables, minimizing f is always reduced to minimizing a polynomial g of degree 2.

Some pseudo-Boolean functions can be minimized by computing the minimal cut of a graph. Consider a function of two Boolean variables;[2] it can be written as a polynomial of degree two:[3]

$$\begin{aligned} E(x_1, x_2) = {} & E_{11} x_1 x_2 + E_{10} x_1 (1 - x_2) \\ & + E_{01}(1 - x_1) x_2 + E_{00}(1 - x_1)(1 - x_2), \end{aligned} \tag{2.23}$$

or, ignoring constant terms:

$$\begin{aligned} E(x_1, x_2) = {} & \big(E_{11} - E_{10} - E_{01} + E_{00}\big) x_1 x_2 \\ & + E_{10} x_1 + E_{01} x_2 - E_{00}(x_1 + x_2). \end{aligned} \tag{2.24}$$

This expression can be written on the form (2.22) if and only if the coefficient in front of $x_1 x_2$ is non-positive (since $w_{i,j}$ is required to be non-negative). Written out, this requirement is

$$E_{00} + E_{11} \leq E_{01} + E_{10}. \tag{2.25}$$

This property of the function E is called **submodularity**. The fact that submodular functions can be efficiently minimized by computing a minimum cut was introduced by Ivănescu (1965) for quadratic functions and Billionnet and Minoux (1985) for cubic functions. Kolmogorov and Zabih (2002, 2004) brought this work to the attention of the computer vision community.

[2] Such functions are often called **cliques** of size 2.

[3] $E(0,0)$ is abbreviated as E_{00} etc. E is defined by the four values in its range: $E_{00}, \ldots E_{11}$.

2.5.1 General Quadratic Pseudo-Boolean Functions

The problem of minimizing a quadratic pseudo-Boolean function covers, as mentioned in the previous section, all pseudo-Boolean functions by possible introduction of auxiliary variables. Therefore, it is enough to consider

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{minimize}} \quad & c_0 + \sum_{i=1}^{n} c_i x_i + \sum_{1 \le i < j \le n} c_{i,j} x_i x_j \\ \text{subject to} \quad & \boldsymbol{x} \in \{0,1\}^n. \end{aligned} \tag{2.26}$$

This is known as a **quadratic pseudo-Boolean optimization** problem or **QPBO**. By introducing new variables $y_{i,j}$ to represent the product $x_i x_j$, (2.26) can be reformulated as an integer linear program:

$$\begin{aligned} \underset{\boldsymbol{x},\boldsymbol{y}}{\text{minimize}} \quad & c_0 + \sum_{i=1}^{n} c_i x_i + \sum_{1 \le i < j \le n} c_{i,j} y_{i,j} \\ \text{subject to} \quad & \\ & \left.\begin{aligned} y_{i,j} &\ge x_i + x_j - 1, \\ y_{i,j} &\ge 0 \end{aligned}\right\}, \quad c_{i,j} > 0, \\ & \left.\begin{aligned} y_{i,j} &\le x_i, \\ y_{i,j} &\le x_j \end{aligned}\right\}, \quad c_{i,j} < 0, \\ & \boldsymbol{x} \in \{0,1\}^n. \end{aligned} \tag{2.27}$$

Relaxing the constraint $\boldsymbol{x} \in \{0,1\}^n$ to $\boldsymbol{x} \in [0,1]^n$ results in a linear program. It can be shown by looking at the determinant of all submatrices of the linear constrains and employing Cramer's rule that there always exists an optimal solution $\hat{\boldsymbol{x}} \in \left\{0, 1, \frac{1}{2}\right\}^n$ to the relaxation of (2.27).

Given the solution of the linear programming relaxation, one can often say a lot about the solution to the original problem (2.26). This is due to the following **persistency** result:

Theorem 2.3. *Let $\hat{\boldsymbol{x}} \in \left\{0, 1, \frac{1}{2}\right\}^n$ be a solution to* (2.27) *and $L = \left\{i \,|\, x_i \neq \frac{1}{2}\right\}$. Then there exists a solution $\boldsymbol{x}^*$ to* (2.26) *with $x_i^* = \hat{x}_i$ for all $i \in L$.*

This motivates the following terminology: If $i \neq \frac{1}{2}$ the linear programming relaxation is said to have **labeled** variable i, otherwise x_i is said to be **unlabeled**, which is sometimes written $i = \varnothing$. Thus, the solution of the relaxation is in

$\{0, 1, \varnothing\}^n$. Linear programs can be solved in polynomial time, but there are even more efficient algorithms. A solution satisfying Theorem 2.3 with the same lower bound of the optimal value as in (2.27) can be computed as a minimum cut in a graph with $2n$ nodes (Boros and Hammer, 2002).

If the solution contains several unlabeled variables, it can sometimes be improved with a technique called **probing**. The idea is to pick an index i with $i = \varnothing$. Then two problems are solved, with x_i fixed to 0 and 1, respectively. If a previously unlabeled variable x_j is equal to the *same* value (not $\varnothing$) in both of the new solutions, then x_j is permanently fixed to this value, as it is known to be optimal. The process is then repeated. Probing is sometimes very effective in finding globally optimal solutions.

The persistency results of the linear programming relaxation were introduced by Hammer, Hansen and Simeone (1984). Later, Boros, Hammer and Sun (1991) solved the linear program efficiently as a minimum cut in a graph. The probing technique is due to Boros, Hammer and Tavares (2006). Their work was made popular in the computer vision community due to an efficient implementation by Rother et al. (2007*a*; 2007*b*), which is available for download. In this context, "QPBO" often refers to solving the linear programming relaxation with a minimum cut.

2.6 Beyond Boolean Variables

In image processing and early vision, for example segmentation, two labels for each pixel are often not enough. While optimization problems with a finite number of of possible values for each variable can be converted into Boolean problems by binary encoding, this is normally not the preferred method as the resulting optimization problems tend to be very hard to solve. Instead, an approximate minimization method can be employed, known as **$\boldsymbol{\alpha}$-expansion**. It is a method in which each iteration, given an estimate $\boldsymbol{x}^{(t)} \in \{1 \dots K\}^n$ for a problem with K labels, gives a better estimate $\boldsymbol{x}^{(t+1)}$. The improvement is computed by picking one of the possible pixel values $\alpha \in \{1 \dots K\}$ and computing the optimal *expansion* of that value.

Definition 2.4. *An image $\boldsymbol{x}^{(t+1)}$ is an **$\boldsymbol{\alpha}$-expansion** of an image $\boldsymbol{x}^{(t)}$ if for every pixel i:*

$$x_i^{(t+1)} = \alpha \qquad \textit{or} \qquad x_i^{(t+1)} = x_i^{(t)}. \tag{2.28}$$

The complete algorithm is to perform expansion moves iteratively for different pixel values:

$\boldsymbol{x}^{(0)} \leftarrow \mathbf{1}$
$t \leftarrow 0$
while *function value is improved* **do**
  foreach $\alpha \in \{1 \dots K\}$ **do**
    Set $\boldsymbol{x}^{(t+1)}$ to the optimal α-expansion of $\boldsymbol{x}^{(t)}$
    $t \leftarrow t + 1$
  end
end

An expansion for every pixel value is performed until the function value does not decrease for any value of α. This usually happens within a few loops through the set of pixel values. While reaching the global optimum is not guaranteed, it is possible to prove (Boykov, Veksler and Zabih, 2001, Theorem 6.1) that the α-expansion algorithm will end up within a constant of the optimal function value. The constant depends on the type of pairwise interactions between the variables and is under favorable circumstances equal to 2.
Computing the optimal α-expansion can often be done exactly. Let $\boldsymbol{z}$ be a Boolean vector and let

$$x_i^{(t+1)} = \begin{cases} x_i^{(t)}, & \text{if } z_i = 0 \\ \alpha, & \text{if } z_i = 1. \end{cases} \tag{2.29}$$

The problem of finding the optimal expansion is with this notation a Boolean optimization problem over $\boldsymbol{z}$. With the correct assumptions on the interactions between the variables, all pseudo-Boolean problems will be submodular and can be solved readily as minimum cut problems. The graph construction by Boykov, Veksler and Zabih (2001, Section 5) uses additional nodes where $\alpha \neq x_i^{(t)} \neq x_j^{(t)} \neq \alpha$ and $j \in \mathcal{N}_i$, although the construction can be done without extra nodes (Kolmogorov and Zabih, 2004).

In addition to α-expansion the related **$\alpha\beta$-swap** (Boykov, Veksler and Zabih, 2001) may also be used. Instead of considering a single value α and expanding it as much as possible, the swap considers two possible labels $\alpha, \beta \in \{1 \dots K\}$ and computes the optimal swap between the two. It can handle more general functions than expansion while still ensuring submodularity. The downsides are the lack of

any guarantee to end up close to the optimum and the fact that each full iteration now consists of solving $\binom{K}{2} = (K^2 - K)/2$ Boolean problems instead of K.

2.6.1 Geometric Constraints

Often $\boldsymbol{x}$ represents the individual pixels of an image and the set $\mathcal{L} = \{1 \dots K\}$ contains the **labels** assigned to each pixel. The function f is in this setting often called the **energy** of the labeling. Several recent papers have extended the framework for multi-label energy minimization for which global solutions are tractable. If there is a geometric relationship between the labels (e.g. the image region number 2 is contained in region number 1) the energy can sometimes still be minimized exactly (Delong and Boykov, 2009). This framework will be put to use in Chapter 9.

Chapter 3

Markov Random Fields

One of the main topics in this thesis is graph theoretical optimization problems. I will spend this chapter describing how these optimization problem arise in vision when solving inference problems with Markov random fields (MRFs). This can be done by reformulating the optimization problems into graph theoretical problems which can be solved very fast. The reader is assumed to be familiar with MRFs, see for example the book by Lindgren (2006). These methods are not new and I am in this chapter summarizing methods with which most researchers of computer vision are intimately familiar. Influential work in this field has been conducted by several authors and groups and I have found the work by Greig, Porteous and Seheult (1989); Kolmogorov and Zabih (2004) and Boykov, Veksler and Zabih (2001) particularly useful.

3.1 Maximum *A Posteriori* Estimation

Consider a random field $\boldsymbol{x} \sim p(\boldsymbol{x})$ with n points that each take K different values.[1] In many situations we do not observe $\boldsymbol{x}$ directly. Instead we observe another field $\boldsymbol{y}$ of the same size. A typical assumption is that all components of $\boldsymbol{y}$ are conditionally independent given $\boldsymbol{x}$ with known probability density functions $p_i(y_i|\boldsymbol{x}) = p_i(y_i|x_i)$. One possible model is $\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{\varepsilon}$, where $\boldsymbol{\varepsilon}$ is independent noise.

Given our measurements, we are interested in the maximum *a posteriori* (MAP) estimate of the "true" field $\boldsymbol{x}$. That is, we want to find an estimate $\hat{\boldsymbol{x}}$ such that $p(\hat{\boldsymbol{x}}|\boldsymbol{y})$ is maximized. Following Bayes' rule, $p(\boldsymbol{x}|\boldsymbol{y}) = p(\boldsymbol{x})p(\boldsymbol{y}|\boldsymbol{x})/p(\boldsymbol{y})$, which means that we want to solve the following optimization problem:

$$\underset{\hat{\boldsymbol{x}} \in \{1 \ldots K\}^n}{\text{maximize}} \quad p(\hat{\boldsymbol{x}}) \cdot p(\boldsymbol{y}|\hat{\boldsymbol{x}}). \tag{3.1}$$

[1] $\boldsymbol{x} \sim p(\boldsymbol{x})$ means that $\boldsymbol{x}$ is sampled from the distribution with probability density p.

Using our previous assumption about the independence of $\boldsymbol{y}$, we take the negative logarithm and rewrite (3.1) as a minimization problem:

$$\underset{\hat{\boldsymbol{x}} \in \{1 \dots K\}^n}{\text{minimize}} \quad -\log p(\hat{\boldsymbol{x}}) - \sum_{i=1}^{n} \log p_i(y_i | \hat{x}_i). \tag{3.2}$$

At first glance this seems like a hard optimization problem. Even for a small 100×100 binary image, the search space has $2^{10000} \approx 10^{3010}$ discrete points. This chapter will use the methods described in the previous chapter to solve the binary case exactly within milliseconds even for large images and approximate the solution strongly when the number of pixel labels $K > 2$.

3.2 Binary Images

The work by Greig, Porteous and Seheult (1989), upon which this section is based, seems to have passed largely unnoticed for many years until graph methods were reintroduced to the vision community in the mid-90s. Since then, graph-based methods have had an enormous success in solving problems in low-level computer vision, much thanks to fast minimum cut solvers, e.g. Boykov and Kolmogorov's (2004).

We follow the presentation and notation of MRFs of Lindgren (2006, Section 4.3) and restrict the presentation in this chapter to cliques of size 2. The graph construction becomes more complicated with larger cliques, as additional nodes may have to be added (Section 2.5). To make the presentation easier to follow I also omit cliques of order 1, but the optimization method works just as well with them included. We can now write the probability function for $\boldsymbol{x}$ as

$$p(\boldsymbol{x}) \propto \exp\left(\frac{1}{2} \sum_{i=1}^{n} \sum_{j \in \mathcal{N}_i} V_{i,j}(x_i, x_j) \right). \tag{3.3}$$

We use a homogeneous and isotropic model such that

$$V_{i,j}(x_i, x_j) = \begin{cases} \rho, & x_i = x_j, \qquad \rho > 0 \\ 0, & x_i \neq x_j. \end{cases} \tag{3.4}$$

The argument will also hold without homogeneity and isotropy. We cannot, however, allow any potential and still use graph cuts for optimization. The

condition $\rho > 0$ is required. Let us write down the optimization problem (3.2) explicitly:

$$\begin{aligned}\underset{\boldsymbol{x}\in\{0,1\}^n}{\text{minimize}} \quad & -\frac{1}{2}\sum_{i=1}^{n}\sum_{j\in\mathcal{N}_i}\mathbb{I}(x_i = x_j)\cdot\rho \\ & -\sum_{i=1}^{n}\mathbb{I}(x_i=1)\cdot\log p_i(y_i|x_i=1) - \sum_{i=1}^{n}\mathbb{I}(x_i=0)\cdot\log p_i(y_i|x_i=0).\end{aligned} \tag{3.5}$$

Here $\mathbb{I}$ is the indicator function equal to 1 whenever its argument is true and 0 otherwise. We may in the above expression rewrite $-\mathbb{I}(x_i = x_j) = \mathbb{I}(x_i \neq x_j) - 1$ and the -1 can be ignored as it does not depend on $\boldsymbol{x}$. This means that solving (3.5) is equivalent to problem (2.22), because in this case $j \in \mathcal{N}_i \iff i \in \mathcal{N}_j$ i.e. the graph is undirected. We create a graph with edge weights

$$\begin{aligned} w_{i,j} &= \rho, && j \in \mathcal{N}_i \\ w_{s,i} &= -\log p(y_i|x_i=1) \\ w_{i,t} &= -\log p(y_i|x_i=0). \end{aligned} \tag{3.6}$$

The minimum cut in this graph is the MAP estimate of the Markov random field.

3.2.1 Example

We adopt the following model for our observation: $p(y_i \neq x_i) = e$ and $p(y_i = x_i) = 1 - e$. The constant e should be interpreted as the error rate – the probability that we will observe the opposite of the true value x_i. This observation model is equivalent to

$$\begin{aligned} p(y_i|x_i=1) &= e(1-y_i) + (1-e)y_i \\ p(y_i|x_i=0) &= ey_i + (1-e)(1-y_i). \end{aligned} \tag{3.7}$$

Figure 3.1 shows an experiment with this model. The image used is similar to one of the examples by Greig, Porteous and Seheult (1989) and of the same size. It is interesting to note that the optimization required hundreds of seconds in 1989, while hundreds of microseconds is enough in 2010. The algorithm scales well with image size; Fig. 3.2 shows an example with a 2000×2000 pixel image. The resulting optimization problem has 4 million binary variables, but was

(a) original **(b)** observation **(c)** estimate **(d)** ICM estimate

Figure 3.1: MAP estimation for a 64 $\times$ 64 binary image. A 4-connected neighborhood was used with $\rho = 0.7$ and e = 25%. Computation time ranged between 923 and 1043 µs. The non-optimal ICM estimate is shown for comparison.

nevertheless solved exactly in less than 3 seconds. For comparison, an estimate was also produced with iterated conditional modes (ICM) (Lindgren, 2006, 4.4.1). The ICM solution (Fig. 3.1d) is far from optimal, which is in accordance with Greig, Porteous and Seheult's (1989) results.

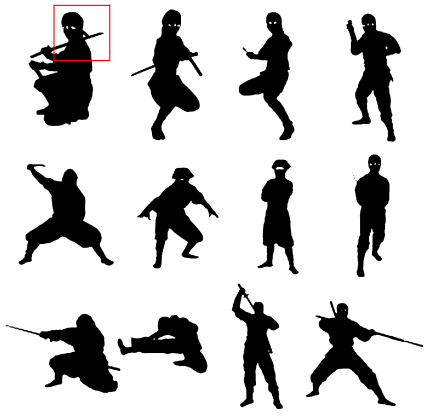
3.2.2 General Case

While graph cuts can be used to minimize a large class of pseudo-Boolean functions, they cannot be used for any function. In the previous example ρ was required to be positive. Without this requirement, the graph is not guaranteed to have positive weights. If negative weights are allowed, the resulting optimization problem is NP-hard. This is easily seen by observing that a maximum-cut problem, known to be NP-hard, can be transformed into a minimum-cut problem by replacing all weights with their negatives. Section 2.5 briefly describes methods to deal with such optimization problems.

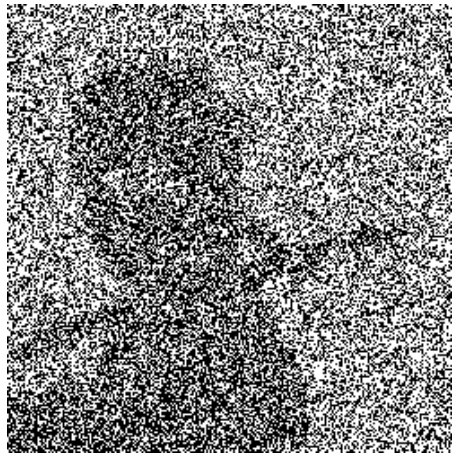
3.3 General Discrete Fields

In this section we will look beyond binary MRFs. Unfortunately, the optimization problems we then need to solve are NP-hard, but there are efficient approximation methods that work well in practice, which were briefly described in Section 2.6.

The optimization problem for MAP estimation is derived in the same way as in section 3.2, with the difference that we now allow each pixel to take K values



(a) original



(b) observation (only red frame in (a) shown)

(c) $\rho = 0.3$ estimate

(d) $\rho = 0.7$ estimate

Figure 3.2: MAP estimation for a 2000×2000 binary image. A 4-connected neighborhood was used with e = 40%. Computation times ranged between 2.41 and 2.99 s. The optimization problem with 4 million binary variables was solved exactly.

instead of 2:

$$\underset{\boldsymbol{x}\in\{1\ldots K\}^n}{\text{minimize}} \quad -\frac{1}{2}\sum_{i=1}^{n}\sum_{j\in\mathcal{N}_i}\mathbb{I}(x_i = x_j)\cdot\rho - \sum_{i=1}^{n}\sum_{k=1}^{K}\mathbb{I}(x_i = k)\cdot\log p(y_i|x_i = k). \quad (3.8)$$

In general, ρ may depend on i, j, x_i and x_k, but the above formulation will be sufficient for our purposes.

3.3.1 Example

Let $\boldsymbol{x}$ be a random field with 3 possible values: $\{1, 2, 3\}$. The observation $\boldsymbol{y}$ has its pixels independently Gaussian distributed with mean $\boldsymbol{x}$ and variance 2. The field $\boldsymbol{x}$ can be seen in Fig. 3.3a and the observation in Fig. 3.3b. Applying α-expansion to minimize (3.8) converged in 3 cycles. A good solution was obtained already after the first cycle. For this experiment, I used an 8-neighborhood and $\rho = 4$. Larger neighborhoods give smoother borders.

3.3.2 Dense Stereo Estimation

Without going into any details, we can study another common application of α-expansion: stereo estimation. The problem is to estimate the image depth at every pixel given two images of the same scene, see Fig. 3.4. The images have been captured form slightly different positions and can therefore be used to recover the depth of the scene. The depth map is assumed to be an MRF with 16 possible depth values and the MAP estimate is computed using α-expansion. The probability that a pixel has a specific depth is calculated from the photo-consistency between the images. If a pixel in the left image e.g. has depth 13, its position in the right image can be calculated and the pixel values compared. The paper by Kolmogorov and Zabih (2006) contains a more thorough description of how the interaction potential can be computed and better results than the simple model (3.8) I use here.

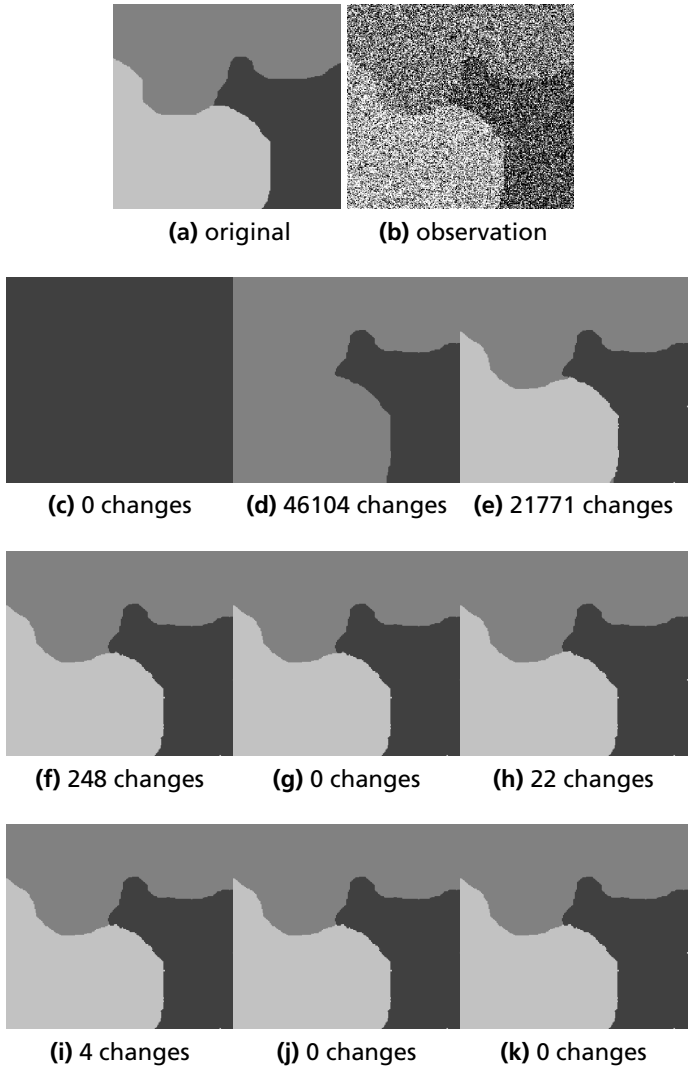
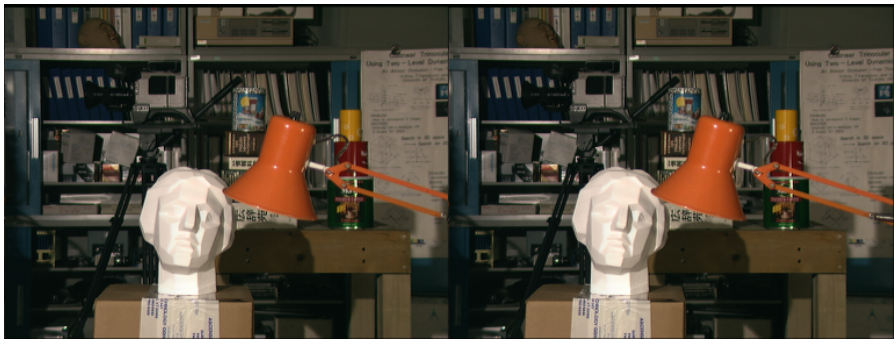
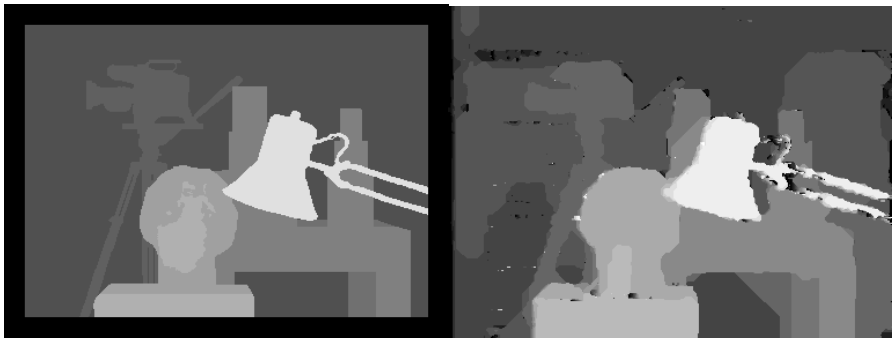


Figure 3.3: Demonstration of α-expansion for a 248×250 image. The algorithm proceeded in alphabetical order (c)–(k). Each row is one complete cycle over the pixel values 1,2,3. Column 1 shows 1-expansions, etc. After the third complete cycle there were no more changes.



(a) left image **(b)** right image



(c) left stereo ground truth **(d)** left stereo estimation

Figure 3.4: Dense stereo estimation from two photos using a MRF with 16 possible values and α-expansion.

Part I

Parametric Models

Chapter 4

Optimizing Parametric Total Variation Models

In the introduction, I presented the segmentation problem as finding an optimal curve γ enclosing a region $\Gamma \subset \Omega$. Alternatively, this can be formulated as finding a function $\theta : \Omega \to \{0, 1\}$, such that $\theta(\boldsymbol{x}) = 1 \iff \boldsymbol{x} \in \Gamma$. The optimal function then defines the region of interest directly.

The data term (1.4) is constructed from an observation model, which normally contains one or several unknown parameters that either must be known *a priori* or estimated together with the segmentation itself. This estimation is a minimization problem, where the optimal parameters depend on the proposed segmentation $\hat{\theta}$. The work upon which this chapter is based has been done in collaboration with Fredrik Kahl and Niels Christian Overgaard (Strandmark, Kahl and Overgaard, 2009) and we have investigated special cases of these functionals, where the image to be estimated is binary and the number of unknown parameters is relatively small.

4.1 The Mumford-Shah Functional

The functional introduced by Mumford and Shah (1989) is a widely used functional for image segmentation. As a special case, Chan and Vese (2001) proposed a segmentation method where an image is approximated with a function taking only two values. By minimizing an energy consisting of a smoothness term added to the squared distance between the original and the approximation, a large variety of images can be segmented correctly. However, the exact minimization of this energy functional is a difficult problem and this chapter will describe new results on this topic obtained by generalizing recent results by Chambolle (2005) and Chan, Esedoglu and Nikolova (2006).

We consider optimization problems over images, where without loss of generality the image $I : \mathbf{R}^2 \supset \Omega \to \mathbf{R}$ is assumed to take values on $[0, 1]$.
Our main contribution is that we show how one can evaluate real-valued functions of the following type:

$$m(\boldsymbol{t}) = \min_{\theta, s} E(\theta, s, \boldsymbol{t}), \tag{4.1}$$

where E is a functional depending on the binary valued function $\theta : \Omega \to \{0, 1\}$, the one-dimensional parameter $s \in \mathbf{R}$ as well as some additional vector of real parameters $\boldsymbol{t}$. The ability to evaluate such functions allows us to efficiently optimize parametric, binary total variation models including several variants of the Mumford-Shah functional. The standard way of solving such problems is by alternating optimization:

1. Keep the real parameters fixed and solve for θ.

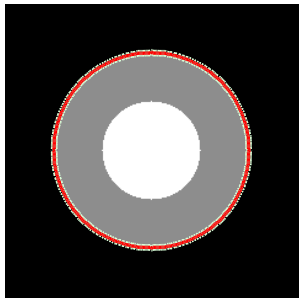
2. Keep θ fixed and solve for the real parameters.

By including one additional parameter in the first step, the neighborhood search is enlarged and the risk of getting trapped in local minima is reduced.

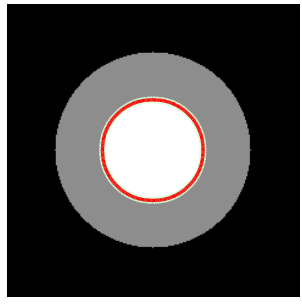
Another consequence of (4.1) is that we can obtain globally optimal solutions to low-order parametric total variation models. We analyze in more detail one of the main problems in this class of segmentation models, namely the Chan-Vese model. We want to approximate the image with a function taking only two values μ_0 and μ_1, by solving the following optimization problem:

$$\begin{aligned} \underset{\theta, \mu_0, \mu_1}{\text{minimize}} \quad & \rho J(\theta) \\ & + \int_\Omega (1 - \theta(\boldsymbol{x}))(I(\boldsymbol{x}) - \mu_0)^2 + \theta(\boldsymbol{x})(I(\boldsymbol{x}) - \mu_1)^2 \, d\boldsymbol{x} \qquad \text{(4.2a)} \\ \text{subject to} \quad & \theta(\boldsymbol{x}) \text{ binary} \\ & 0 \leq \mu_0 < \mu_1 \leq 1. \end{aligned}$$

Here $J(\theta)$ is the total variation of θ, $J(\theta) = \int_\Omega |\nabla \theta| \, d\boldsymbol{x}$. When θ is binary, the total variation is the length of the boundary between the two regions defined by θ. The weight $\rho > 0$ controls how important a short boundary is. The assumption that $\mu_1 > \mu_0$ is without loss of generality and it prevents (4.2a) from inherently having two optima. We show how to find the optimal segmentation as well as the optimal values of the two parameters μ_0 and μ_1 by a simple branch and bound search over a single dimension.



(a) $\mu_1 = 0.662$, $\mu_0 = 0.005$, E = 614.81



(b) $\mu_1 = 1.00$, $\mu_0 = 0.158$, E = 600.05

Figure 4.1: Segmenting a simple image. The result shown in (a) was obtained after setting $\mu_0 = 0$, $\mu_1 = 1$ and alternating between minimizing θ and updating μ_0, μ_1. In (b), the global minimum is shown.

4.1.1 Related Work

If we keep μ_0 and μ_1 fixed and only optimize over θ, problem (4.2a) becomes equivalent to

$$\underset{\theta(\boldsymbol{x}) \text{ binary}}{\text{minimize}} \quad \rho J(\theta) + \int_\Omega \theta(\boldsymbol{x}) \big[(I(\boldsymbol{x}) - \mu_1)^2 - (I(\boldsymbol{x}) - \mu_0)^2 \big] \, d\boldsymbol{x} \,. \tag{4.2b}$$

This problem is still non-convex, because the discrete set $\{0, 1\}$ is non-convex. Chan, Esedoglu and Nikolova (2006) showed that globally optimal solutions can still be obtained by relaxing θ to the interval $[0, 1]$, solving the resulting *convex* problem and then thresholding the result. Several algorithms have been developed to solve this convex minimization problem, e.g. by Bresson et al. (2007). If the image is discretized, optimal solutions can also be obtained via graph-cuts, with a suitable J.

On the other hand, if one wants to also optimize over μ_0 and μ_1 simultaneously the problem is no longer convex. In practice, this is solved by alternating between minimizing over θ with μ_0, μ_1 fixed and minimizing μ_0, μ_1 with θ fixed (Bresson et al., 2007; Chan, Esedoglu and Nikolova, 2006; Chan and Vese, 2001). The latter step is very simple, it just consists of taking the means of the two regions defined by θ (Mumford and Shah, 1989). This procedure does not guarantee that the final solution obtained is globally optimal. Indeed, Fig. 4.1 shows an image

where this procedure fails. The result with initial values of $\mu_0 = 0$ and $\mu_1 = 1$ is shown in Fig. 4.1a, which is only a local optimum, because the segmentation in Fig. 4.1b has a lower energy.

Another method is of course to perform an exhaustive search over the parameters μ_0 and μ_1, solving (4.2b) for each possible pair. This is done by Darbon (2007), where a maximum-flow formulation of (4.2b) is solved for every pair of the two levels. The size of the graphs are reduced with a method that bears some resemblance to the one described here. An alternative approach was pursued by Lempitsky, Blake and Rother (2008) where branch and bound was applied over μ_0 and μ_1 for a discretized version of (4.2a). The main advantage with our approach is that we reduce the problem with one dimension, and branch-and-bound is only necessary in the remaining dimension. Another possible advantage is that our work is based on continuous optimization methods and hence metrication errors are smaller which Klodt et al. (2008) have demonstrated. Moreover, our algorithm is amenable to GPU acceleration as described by Pock et al. (2008*a*).

Our work is also applicable to other variants of the Mumford-Shah family of segmentation methods. Alternating minimization is used, for example, by Grady and Alvino (2008) for a discretized version of Mumford-Shah and by Cremers and Soatto (2005) for motion segmentation of a parametric optical flow model. Kolmogorov, Boykov and Rother (2007) give applications of parametric max-flow problems, including ratio minimization and incorporation of global constraints in the optimization. Instead of solving a series of max-flow problems, we show how the same applications can be solved via a single convex variational problem.

4.1.2 Example: Existence of Local Minima

We now explicitly construct an image for which (4.2a) has a non-optimal local minimum. Let $\Omega = [0, 4] \times [0, 1]$. Let $I(\boldsymbol{x})$ consist of three regions separated by the two vertical lines described by $x = 1$ and $x = 2$, respectively. The three regions have gray values 1, 0.5 and 0. The areas of the three regions are:

$$\begin{aligned} A_1 &= 1 \\ A_2 &= 1 \\ A_3 &= 2. \end{aligned} \tag{4.3}$$

For symmetry reasons, the optimal segmentation will always be a vertical line. We consider the simplest case where $\rho = 0$. If the line $x = 1$ is chosen the optimal

values of the mean values are $\mu_0 = \frac{0.5A_2+0A_3}{A_2+A_3} = \frac{1}{6}$ and $\mu_1 = 1$, for a total energy of

$$A_1(1-1)^2 + A_2\left(0.5 - \frac{1}{6}\right)^2 + A_3\left(0 - \frac{1}{6}\right)^2 = \frac{1}{6}. \tag{4.4}$$

And for these values of μ_0 and μ_1 the segmentation is optimal since $|0.5 - \mu_0| < |0.5 - \mu_1|$ makes the pixels valued 0.5 be optimally assigned to the background. If the outer boundary is chosen the optimal values of the mean values are $\mu_0 = 0$ and $\mu_1 = \frac{1A_1+0.5A_2}{A_1+A_2} = \frac{3}{4}$, for a total energy of

$$A_1\left(1 - \frac{3}{4}\right)^2 + A_2\left(0.5 - \frac{3}{4}\right)^2 + A_3(0-0)^2 = \frac{1}{8}. \tag{4.5}$$

Because $|0.5 - \mu_0| > |0.5 - \mu_1|$ the gray pixels are optimally assigned to the foreground (μ_1). Since $1/8 < 1/6$, the first segmentation is a non-optimal local minimum with respect to alternating minimization. Figure 4.1 shows how this can occur in practice.

4.2 Parametric Binary Problems

For any function v, let $v^{(t)}$ denote the function thresholded at t, i.e. $v^{(t)}(\boldsymbol{x}) = 1$ if $v(\boldsymbol{x}) > t$ and 0 otherwise. From now on, we require that our smoothness function J satisfies the following requirements:

1. $J(v)$ is convex and $J(v) \geq 0$.
2. $J(tv) = tJ(v)$ for every $t > 0$.
3. $J(v) = \int_{-\infty}^{\infty} J(v^{(t)})dt$ (general co-area formula).

For example, the total variation $\int_\Omega |\nabla v| \, d\boldsymbol{x}$ satisfies these three conditions.

We will now define two optimization problems and show that by thresholding the solution to one, we get a solution to the other. Let $f(\boldsymbol{x}, s)$ be a real-valued function such that $f(\boldsymbol{x}, \cdot)$ is continuously strictly increasing for each fixed $\boldsymbol{x} \in \Omega$ and $f(\boldsymbol{x}, z(\boldsymbol{x})) = 0$ for all $\boldsymbol{x}$ and some bounded function z. Let F be any function such that $\partial F/\partial s(\boldsymbol{x}, s) = f(\boldsymbol{x}, s)$ for all $(\boldsymbol{x}, s)$. Consider the following discrete problem, defined for a real parameter s:

$$\underset{\theta(\boldsymbol{x}) \text{ Boolean}}{\text{minimize}} \quad \rho J(\theta) + \int_\Omega \theta(\boldsymbol{x}) f(\boldsymbol{x}, s) \, d\boldsymbol{x}. \tag{P_s}$$

We will need the following property of the solutions to (P_s):

Lemma 4.1. *Let $s_1 > s_2$. Then the solutions θ_1 and θ_2 to (P_{s_1}) and (P_{s_2}), respectively, satisfy $\theta_1(\boldsymbol{x}) \leq \theta_2(\boldsymbol{x})$ (a.e.).*

Proof. Define operators $\wedge$ and $\vee$ by:

$$\begin{aligned}(\eta \wedge \theta)(\boldsymbol{x}) &= \min(\eta(\boldsymbol{x}), \theta(\boldsymbol{x})) \\ (\eta \vee \theta)(\boldsymbol{x}) &= \max(\eta(\boldsymbol{x}), \theta(\boldsymbol{x})).\end{aligned} \tag{4.6}$$

Let $E_s(\theta)$ denote the functional to be minimized in (P_s). Since θ_1 and θ_2 are optimal, $E_{s_1}(\theta_1) \leq E_{s_1}(\theta_1 \wedge \theta_2)$ and $E_{s_2}(\theta_2) \leq E_{s_2}(\theta_1 \vee \theta_2)$. Summing these inequalities and using the fact that $J(\theta_1 \wedge \theta_2) + J(\theta_1 \vee \theta_2) \leq J(\theta_1) + J(\theta_2)$ (Chambolle, 2005, Lemma 2.3) results in

$$\int_\Omega \Big(f(\boldsymbol{x}, s_1)(\theta_1(\boldsymbol{x}) - (\theta_1 \wedge \theta_2)(\boldsymbol{x})) + f(\boldsymbol{x}, s_2)(\theta_2(\boldsymbol{x}) - (\theta_1 \vee \theta_2)(\boldsymbol{x}))\Big)\, d\boldsymbol{x} \leq 0 \tag{4.7}$$

But $\theta_1(\boldsymbol{x}) - (\theta_1 \wedge \theta_2)(\boldsymbol{x}) = -(\theta_2(\boldsymbol{x}) - (\theta_1 \vee \theta_2)(\boldsymbol{x}))$, so this is equivalent to

$$\int_\Omega (f(\boldsymbol{x}, s_1) - f(\boldsymbol{x}, s_2))(\theta_1(\boldsymbol{x}) - (\theta_1 \wedge \theta_2)(\boldsymbol{x}))d\boldsymbol{x} \leq 0. \tag{4.8}$$

Since $f(\boldsymbol{x}, s_1) - f(\boldsymbol{x}, s_2) > 0$, $\theta_1(\boldsymbol{x}) - (\theta_1 \wedge \theta_2)(\boldsymbol{x})$ must be equal to 0. $\theta_1(\boldsymbol{x}) \leq \theta_2(\boldsymbol{x})$ a.e. follows. □

The corresponding convex variational problem to (P_s) is:

$$\underset{w(\boldsymbol{x}) \in \mathbf{R}}{\text{minimize}} \quad \rho J(w) + \int_\Omega F(\boldsymbol{x}, w(\boldsymbol{x}))\, d\boldsymbol{x}\,. \tag{Q}$$

Problems (P_s) and (Q) are related, as stated by the following theorem:

Theorem 4.2. *A function w solves (Q) if and only if $w^{(s)}$ solves (P_s) for any $s \in \mathbf{R}$.*

Proof. Define w, for every $\boldsymbol{x} \in \Omega$, as follows:

$$w(\boldsymbol{x}) = \sup\{s \mid \exists \theta \text{ solving } (P_s) \text{ with } \theta(\boldsymbol{x}) = 1\}.$$

The first task is to show that w is a well-defined real-valued function. If $f(\boldsymbol{x}, s) \leq 0$ for all $\boldsymbol{x}$ it follows that $\theta \equiv 1$ solves (P$_s$). Similarly, $f(\boldsymbol{x}, s) \geq 0$ implies that $\theta \equiv 0$ is a solution. From this we see that w is bounded, more precisely that $\inf_{\boldsymbol{y}} z(\boldsymbol{y}) \leq w(\boldsymbol{x}) \leq \sup_{\boldsymbol{y}} z(\boldsymbol{y})$. To see this, choose $s' < \inf_{\boldsymbol{y}} z(\boldsymbol{y})$. By definition we have $w(\boldsymbol{x}) \geq s'$ for all $\boldsymbol{x}$.

Next, we prove that $w^{(s)}$ solves (P$_s$). Let s be fixed. If $s < w(\boldsymbol{x})$, any solution θ of (P$_s$) must satisfy $\theta(\boldsymbol{x}) = 1$, while if $s > w(\boldsymbol{x})$ we must have $\theta(\boldsymbol{x}) = 0$ (Lemma 4.1). Since $w^{(s)}$ satisfies these requirements, we see that $w^{(s)}$ is a solution to (P$_s$). If the set where $w(\boldsymbol{x}) = s$ is not a null set, we can consider s' arbitrary close to s.

Finally, to show that w is the solution to (Q), we start out by letting $s_* < \inf_{\boldsymbol{y}} v(\boldsymbol{y})$ for some v and observing that

$$\int_{s_*}^{\infty} v^{(s)}(\boldsymbol{x}) f(\boldsymbol{x}, s)\, ds \;=\; \int_{s_*}^{v(\boldsymbol{x})} f(\boldsymbol{x}, s)\, ds = F(\boldsymbol{x}, v(\boldsymbol{x})) - F(\boldsymbol{x}, s_*). \tag{4.9}$$

Now we integrate over problem (P$_s$) for all s:

$$\int_{s_*}^{\infty} \left(\rho J(v^{(s)}) + \int_{\Omega} v^{(s)}(\boldsymbol{x}) f(\boldsymbol{x}, s)\, d\boldsymbol{x} \right) ds$$
$$= \rho J(v) + \int_{\Omega} F(\boldsymbol{x}, v(\boldsymbol{x}))\, d\boldsymbol{x} - \int_{\Omega} F(\boldsymbol{x}, s_*)\, d\boldsymbol{x}\,.$$

We have already shown that $w^{(s)}$ minimizes the integrand for every s. This means that for any function v,

$$\rho J(v) + \int_{\Omega} F(\boldsymbol{x}, v(\boldsymbol{x}))\, d\boldsymbol{x} \;\geq\; \rho J(w) + \int_{\Omega} F(\boldsymbol{x}, w(\boldsymbol{x}))\, d\boldsymbol{x}\,.$$

This shows that w is the unique solution of the strictly convex functional in (Q). □

Remark 4.3 Problem (Q) with $F(\boldsymbol{x}, w(\boldsymbol{x})) = \frac{1}{2}(w(\boldsymbol{x}) - I(\boldsymbol{x}))^2$ is a well-studied convex functional for image restoration, (Rudin, Osher and Fatemi, 1992). We will refer to it as a ROF problem.

Remark 4.4 Chambolle (2005) proved Theorem 4.2 for the special case $f(\boldsymbol{x}, s) = s - G(\boldsymbol{x})$ and used the result to approximate the solution to problem (Q) with

a series of discrete solutions to (P_s). We will use the result in the other direction, for solving a one-parameter family of discrete problems by thresholding a single ROF solution.

Remark 4.5 If $f(\boldsymbol{x}, s) = H(\boldsymbol{x})s - G(\boldsymbol{x})$ with $H(\boldsymbol{x}) > 0$, then

$$F(\boldsymbol{x}, w(\boldsymbol{x})) = \frac{1}{2}\left(\sqrt{H(\boldsymbol{x})}w(\boldsymbol{x}) - \frac{G(\boldsymbol{x})}{\sqrt{H(\boldsymbol{x})}}\right)^2 . \tag{4.10}$$

This is the result we will use for the rest of this chapter. We call Problem (Q) with this data term a weighted ROF problem.

4.2.1 Numerical Method

In our numerical experiments, we will use the following smoothness function:

$$J(\theta) = \sum_{i=0}^{M}\sum_{j=0}^{N}\sqrt{(\theta_{i+1,j} - \theta_{i,j})^2 + (\theta_{i,j+1} - \theta_{i,j})^2}, \tag{4.11}$$

which can be seen as a discretization of the "true" length of the boundary. Problem (Q) with (4.10) can be written as

$$\underset{w}{\text{minimize}} \qquad J(w) + \frac{1}{2\rho}\int_\Omega \left(D(\boldsymbol{x})w(\boldsymbol{x}) - B(\boldsymbol{x})\right)^2 \, d\boldsymbol{x}\,, \tag{4.12}$$

and we solve it by adapting a method in (Chambolle, 2004, 2005). Introduce a dual field ξ, with which a solution can be found by iterating the following scheme:

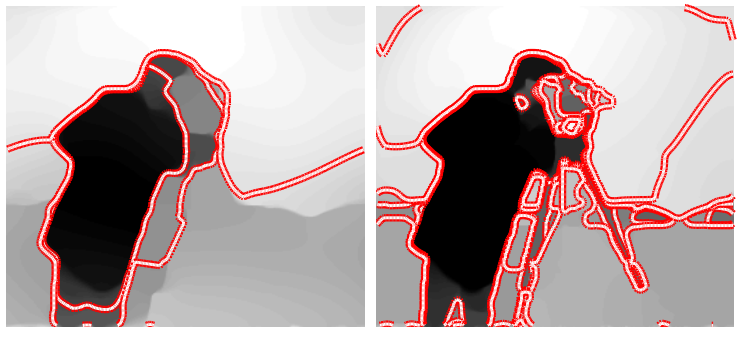
$$\begin{cases} w_{i,j}^{(n)} = \left(B_{i,j} + \rho\left(\operatorname{div}\xi^{(n)}\right)_{i,j}/D_{i,j}\right)/D_{i,j} \\ \xi_{i,j}^{(n+1)} = \dfrac{\xi_{i,j}^{(n)} + (\tau/\rho)(\nabla w^{(n)})_{i,j}}{\max\{1, |\xi_{i,j}^{(n)} + (\tau/\rho)(\nabla w^{(n)})_{i,j}|\}} \end{cases}, \tag{4.13}$$

where τ is the step-length. The initial condition can be set to $\xi^{(0)} = \mathbf{0}$ and $w^{(0)} = B$. A suitable stopping criterion when $D \equiv 1$ is also derived by Chambolle (2005); if $\bar{w}$ is the true solution we have the following error bound:

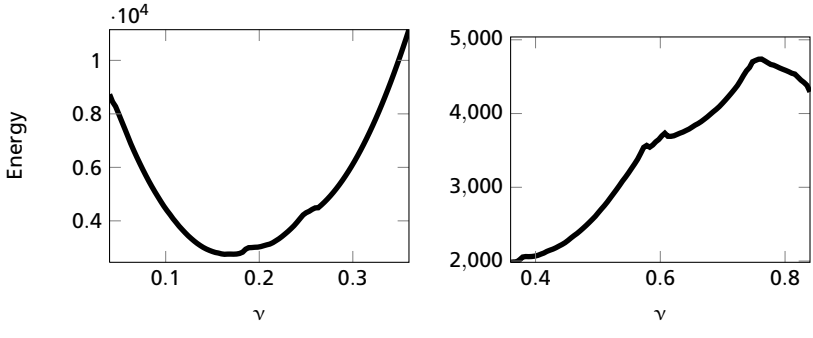
$$||w^n - \bar{w}||^2 \leq \rho J(w) - \rho\sum_{i,j}\xi_{i,j}^n \cdot (\nabla w^n)_{i,j}\,. \tag{4.14}$$

We divide this error bound with the number of pixels in order to get a bound independent of the size of the image being processed.

(a) Solutions to (Q)



(b) Thresholded solutions



(c) Energies as functions of ν

Figure 4.2: Finding the globally optimal energy for a fixed δ by thresholding the solution to problem (Q). The two columns correspond to different values of δ: 0.2 (left column) and 0.6 (right column). Notice the completely different non-convex energy profiles for the two different values of δ.

4.3 Two-Phase Mumford-Shah Functional

We now return to the original problem (4.2a). To formulate our optimization problem, we perform the following change of variables:

$$\begin{cases} \delta = \mu_1 - \mu_0 \\ \nu = \mu_1^2 - \mu_0^2 \end{cases} \quad \Longleftrightarrow \quad \begin{cases} \mu_1 = \dfrac{\nu + \delta^2}{2\delta} \\ \mu_0 = \dfrac{\nu - \delta^2}{2\delta} \end{cases}. \tag{4.15}$$

We can now rewrite the energy in (4.2a) as

$$E(\theta, \nu, \delta) = \rho J(\theta) + \int_\Omega \theta(\boldsymbol{x})(\nu - 2\delta I(\boldsymbol{x})) + \left(\frac{\nu - \delta^2}{2\delta} - I(\boldsymbol{x}) \right)^2 d\boldsymbol{x}\,. \tag{4.16}$$

Let the function $m(\delta)$ previously introduced in (4.1) denote the minimum energy possible given a fixed δ, $m(\delta) = \min_{\theta,\nu} E(\theta, \nu, \delta)$. The set of parameters (μ_0, μ_1) has two degrees of freedom and when we evaluate $m(\delta)$ we optimize over one degree of freedom while keeping the other one fixed.

4.3.1 Optimization with Fixed Difference

The result in Theorem 4.2 implies that $m(\delta)$ can be evaluated by thresholding the solution to the real-valued problem (Q) for all ν and evaluating the energy. This is because evaluating $m(\delta)$ amounts to solving

$$\min_\nu \left(\min_\theta \; E(\theta, \nu, \delta) \right). \tag{4.17}$$

The inner problem is a special case of (P_s) with $f(\boldsymbol{x}, s) = s - 2\delta I(\boldsymbol{x})$. To see this, recall that the last term of $E(\theta, \nu, \delta)$ does not depend on θ. From Remark 4.5 we see that it can be solved with (4.13). After the solution to (Q) is obtained, the solution is thresholded and E evaluated for all ν. To summarize, computing $m(\delta)$ consists of the following steps:

1. Solve problem (Q) with $F(\boldsymbol{x}, s) = \frac{1}{2}(s - 2\delta I(\boldsymbol{x}))^2$.
2. For each ν, threshold the solution w at ν and evaluate the resulting energy (4.16).

3. The pair (ν^*, θ^*) with the lowest energy is the global solution to problem (4.2a) with δ fixed.

Step one is a standard ROF problem, for which there exist fast minimization methods, see (Pock, 2008) for an overview and (Pock et al., 2008*a*) for a GPU implementation. Our simple MATLAB implementation performed one (4.13)-iteration in about 27 ms for $\rho = \delta = 0.5$. The number of iterations required until convergence is strongly dependent on ρ and δ. The second step does not need as much attention as it is a very fast procedure and can trivially be parallelized.

Figure 4.2 shows an example where δ has been fixed to 0.2 and 0.6, respectively. The graphs show that the energy has a lot of local minima as ν varies. The thresholding process finds the global minimum quickly. It is also interesting to note that the graph of the energy looks entirely different for different δ, which suggest that the minimum energy is a complicated function with respect to (δ, ν), and therefore nontrivial to minimize.

Figure 4.3 shows $m(\delta)$ evaluated on the entire interval $[0, 1]$ for ten images. Note that $m(\delta)$ is often very flat around the global optimum, which has two consequences: (i) it will be difficult to find the optimum δ^* with certainty, but (ii) one evaluation of $m(0.5)$ is often enough to find a good solution, close to the global solution.

4.3.2 Optimization with Varying Difference

It is also possible to solve problem (4.2a) along another degree of freedom. We can define another function

$$\hat{m}(\nu) = \min_{\theta,\delta} E(\theta, \nu, \delta), \quad \theta(\boldsymbol{x}) \text{ Boolean.} \tag{4.18}$$

We set $s = -\delta$ and see that computing this function means solving

$$\min_{s} \left(\min_{\theta(\boldsymbol{x})} \rho J(\theta) + \int_{\Omega} \theta(\boldsymbol{x})(2I(\boldsymbol{x})s + \nu)\, d\boldsymbol{x} \right). \tag{4.19}$$

The procedure for calculating $\hat{m}(\nu)$ is the same as the one described in the previous section, with the first step replaced by:

1'. Solve problem (Q) with $F(\boldsymbol{x}, s) = \frac{1}{2}(2I(\boldsymbol{x})s + \nu)^2$.

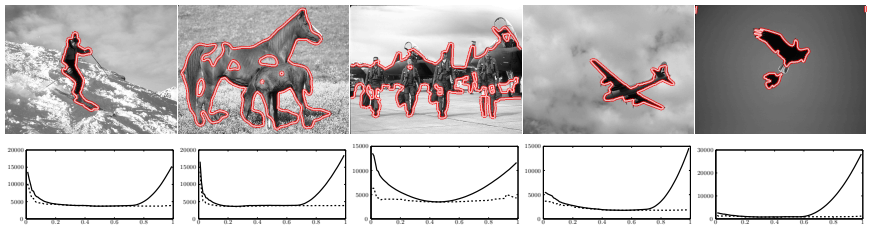


Figure 4.3: The function m(δ) for 5 images. Note that m is very flat near the optimum. The dashed line shows the energy after subsequent optimization of μ_0, μ_1. The weight ρ = 0.5 was used for these images from (Martin et al., 2001).

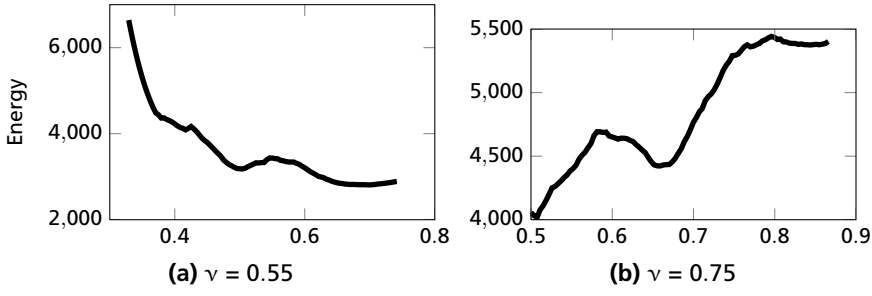


Figure 4.4: "Camera man" image. Energies (y-axis) for problem (4.2a) on the curve $\mu_1^2 - \mu_0^2 = \nu$, each obtained by thresholding all possible δ (x-axis).

The resulting minimization problem can be written on the form (4.12). Therefore, this step can be performed with the method described in Section 4.2.1.

Figure 4.4 shows an example with the "camera man" image. In the experiment ν was fixed to 0.55 and 0.75. This resulted in two very different energy curves for the same image.

4.3.3 Obtaining a Lower Bound

We have a method to compute $m(\delta)$; the next logical step is to minimize it. To be able to prove a lower bound, we need a way to obtain a lower bound for m on an interval $[\delta_1, \delta_2]$. A good lower bound is an essential part of the branch and bound paradigm.

Finding a lower bound for $m(\delta)$ amounts to finding a lower bound for the energy $E(\theta, \nu, \delta)$ defined in (4.16) for any $\delta \in [\delta_1, \delta_2]$. This energy can be

bounded from below on the interval by:

$$E^{\text{bound}}_{\delta_1,\delta_2}(\theta,\nu) = \rho J(\theta) + \int_\Omega \theta(\boldsymbol{x})(\nu - 2\delta_2 I(\boldsymbol{x}))\,d\boldsymbol{x} + \min_{\delta\in[\delta_1,\delta_2]} \int_\Omega \left(\frac{\nu-\delta^2}{2\delta} - I(\boldsymbol{x})\right)^2 d\boldsymbol{x}\,. \quad (4.20)$$

It follows that the minimum of $E^{\text{bound}}_{\delta_1,\delta_2}$ is a lower bound to the minimum of m on $[\delta_1,\delta_2]$. The last term does not depend on θ and can be computed by choosing δ such that $\frac{\nu-\delta^2}{2\delta}$ is as close to the mean of the image as possible. Finding the lower bound therefore amounts to solving (P_s) with $f(\boldsymbol{x},s) = s - 2\delta_2 I(\boldsymbol{x})$ for every s and computing the minimum of the resulting energies. Just like before, every solution can be obtained by thresholding the solution to (Q). Denote the obtained lower bound $m_{\text{bound}}(\delta_1,\delta_2)$.

4.3.4 Global Optimization

The lower bound can be used to perform a branch and bound search on the interval $[0,1]$, splitting each subinterval until it can either be discarded or contains the optimum. However, obtaining a useful bound for even moderately large intervals is hard because m is flat (see Fig. 4.3). Since every calculated bound and every evaluation of m require a solution to (Q), it is essential that previous solutions can be reused. The number of (4.13)-iterations can then be kept to a minimum. For memory reasons, we also want to keep the number of cached solutions as low as possible.

For these reasons, we propose the following method to search for the optimum δ^*: A *feasible region* $[\delta_L,\delta_H]$ is maintained, known to contain the optimal value. This region is initially set to $[0,1]$. The goal is to shrink the feasible region from both ends, i.e. to provide new regions $[\delta_L^{(n+1)},\delta_H^{(n+1)}] \subset [\delta_L^{(n)},\delta_H^{(n)}]$ containing δ^*, with the limit of the lengths equal to 0. The algorithm consists of three main steps: two for shrinking the interval from both endpoints using lower bounds and one to search the remaining feasible interval after good candidates to the optimal energy E^*. Good candidates are necessary for the bounds to be useful; fortunately, good candidates are found very quickly in practice.

The algorithm iterates three main steps, each associated with a cached dual field ξ for speeding up the (4.13)-iterations. The two bounding steps also store

step lengths t_L, t_H which controls the size of the interval to be removed. The steps are detailed in the following list:

1. Try to shrink the interval from above
   - Using the cached dual field ξ_H, solve problem (4.10) with $G(\boldsymbol{x}) = 2(\delta_H + t_H)I(\boldsymbol{x})$.
   - Evaluate $m_{\text{bound}}(\delta_H, \delta_H + t_H)$ by thresholding the solution.
   - If the bound is greater than the currently best energy, discard the interval by setting $\delta_H \leftarrow \delta_H + t_H$. Otherwise, replace t_H by a smaller step; we used $0.8t_H$.
2. Similarly, try to shrink the interval from below.
3. Choose δ inside the feasible interval from $\left\langle \frac{1}{2}, \frac{1}{4}, \frac{3}{4}, \frac{7}{8}, \frac{5}{8}, \frac{3}{8}, \frac{1}{8}, \frac{1}{16}, \ldots \right\rangle$ and evaluate $m(\delta)$.

Because the sequence of evaluated δ is dense in $[0, 1]$, m will eventually be evaluated arbitrarily close to the optimal value. We also have $m_{\text{bound}}(\delta, \delta + t) \to m(\delta)$ as $t \to 0$. From these observations, it is not hard to show that the algorithm is convergent.

4.3.5 Results

Because $m(\delta)$ typically is very flat (Fig. 4.3), the interval cannot be made very small without substantial computational effort. But an approximate localization of the global optimum can be computed and proved in reasonable time. Figure 4.5 shows the cumulative number of (4.13)-iterations required to localize the global optimum for the "camera man" image. The computation of the first bound for $m(\delta)$ required 401 iterations, while the total number of iterations required to compute the bounds for every subinterval was 1151. The search for the optimal point within the feasible interval required 302 iterations.

It should be noted that even a single evaluation of m at e.g. $\delta = 0.5$ is enough for most images in practice, due to the flatness of m and the fact that the solution will be optimal in the s-direction, which typically has lots of local minima as shown in Fig. 4.2. Also, after the optimal solution for a particular δ is obtained, μ_0 and μ_1 are updated before evaluating the energy. In fact, the first evaluation of $m(0.5)$ during the test in Fig. 4.5 resulted in a solution that could not be further improved.

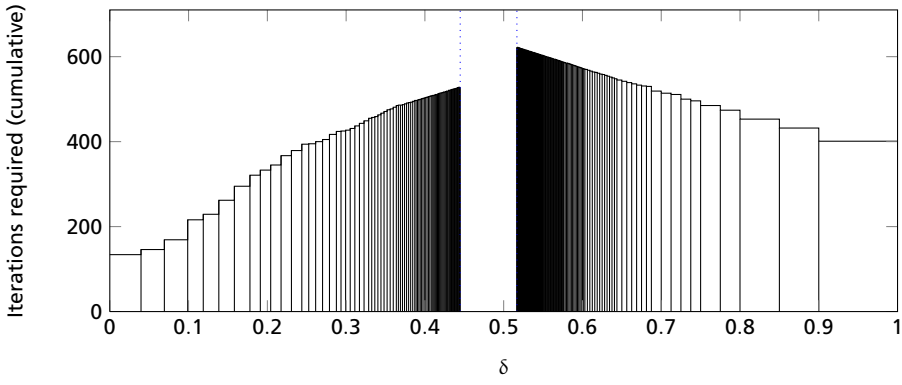


Figure 4.5: Iterations required to shrink the interval for the "camera man" image. A precision of 0.001 was used for the ROF problems. As the intervals grow small, the cached dual field ξ can be reused, allowing the total number of iterations to stay reasonable.

4.4 Ratio Minimization

Problem (P_s) appears in (Kolmogorov, Boykov and Rother, 2007) as "parametric max-flow", where it is used, among other things, to minimize a ratio of two functionals. A similar method is used in (Kolev and Cremers, 2009), where instead a sequence of convex problems is solved. We shall see how one can solve some problems of the same type by solving a single convex minimization problem. The ratio of two functionals P and Q with $Q(\theta) > 0$ is considered:

$$R(\theta) = \frac{P(\theta)}{Q(\theta)}. \tag{4.21}$$

Let $s^* = R(\theta^*)$ be the optimal value of $R(\theta)$. We see that $P(\theta^*) - s^*Q(\theta^*) = 0$ and

$$\begin{aligned} \min_\theta P(\theta) - sQ(\theta) \geq 0 \quad &\iff \quad s \leq s^* \\ \min_\theta P(\theta) - sQ(\theta) \leq 0 \quad &\iff \quad s \geq s^*. \end{aligned} \tag{4.22}$$

This means that we can solve problems for different values of s until we have come arbitrarily close to the optimal value s^* using bisections. This is done in (Kolmogorov, Boykov and Rother, 2007) with repeated max-flow problems.

With the result in Theorem 4.2, we are able to minimize functionals of the following form:

$$\frac{P(\theta)}{Q(\theta)} = \frac{\rho J(\theta) + \int_\Omega \theta(\boldsymbol{x}) g(\boldsymbol{x})\, d\boldsymbol{x}}{\int_\Omega \theta(\boldsymbol{x}) h(\boldsymbol{x})\, d\boldsymbol{x} + K}, \quad h(x) > 0. \tag{4.23}$$

To compute the minimizer of $P(\theta)/Q(\theta)$, we formulate the problem of finding the minimizer of $P(\theta) - sQ(\theta)$, which amounts to solving

$$\underset{\theta(\boldsymbol{x}) \text{ Boolean}}{\text{minimize}} \quad \rho J(\theta) + \int_\Omega \theta(\boldsymbol{x}) \left(h(\boldsymbol{x})(-s) + g(\boldsymbol{x}) \right). \tag{4.24}$$

By solving (Q) once and thresholding the result at $-s$ we find minimizers to $P(\theta) - sQ(\theta)$. Searching for s^* is now reduced to thresholding the solution w at different levels and evaluating an energy, which can be performed very fast. Define $E_s(\theta) = P(\theta) - sQ(\theta)$ and

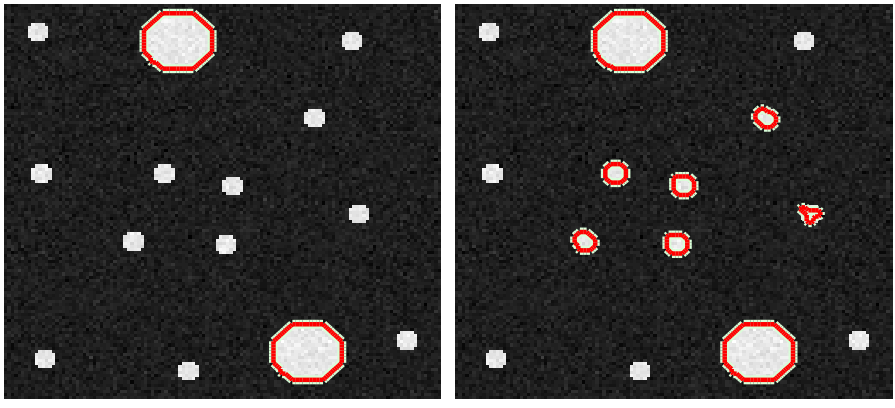
1. Start with $s_{\text{min}}, s_{\text{max}}$
2. $s \leftarrow (s_{\text{min}} + s_{\text{max}})/2$.
3. $\theta \leftarrow w^{(-s)}$.
4. If $E_s(\theta) > 0$ set $s_{\text{max}} \leftarrow s$. Otherwise, set $s_{\text{min}} \leftarrow s$.
5. Repeat from step 2.

This scheme will rapidly converge to s^*. Figure 4.6 shows an example. More examples of ratio minimization are found in (Kolmogorov, Boykov and Rother, 2007).

4.4.1 Constrained Optimization

It is interesting to note that the minimizing a ratio of two functionals bears some similarities to constrained minimization. Consider the following problem, where in addition to an energy functional, the area of the resulting foreground (where $\theta(\boldsymbol{x}) = 1$) is also required to be larger than a predetermined minimum value:

$$\begin{aligned} \underset{\theta}{\text{minimize}} \quad & E(\theta, \mu_0, \mu_1) \\ \text{subject to} \quad & \theta(\boldsymbol{x}) \text{ Boolean} \\ & \int_\Omega \theta(\boldsymbol{x})\, d\boldsymbol{x} \geq A\,. \end{aligned} \tag{4.25}$$



(a) Minimizing P(θ) **(b)** Minimizing P(θ)/Q(θ)

Figure 4.6: Minimizing a ratio of two functions. Q(θ) = $\int_\Omega \theta(x)h(x)\,dx$, where h(x) is chosen to be larger in the center of the image domain, which makes θ(x) = 1 more favorable.

The dual function $d(s)$ (Boyd and Vandenberghe, 2004) of this problem is:

$$\min_{\theta} \left(E(\theta, \mu_0, \mu_1) + s \left(A - \int_\Omega \theta(\boldsymbol{x})\, d\boldsymbol{x} \right) \right). \tag{4.26}$$

For any $s \geq 0$ we have $d(s) \leq E^*$, where E^* is the optimal energy for (4.25). The best lower bound is given by $d^* = \max_{s \geq 0} d(s)$, which can be computed by thresholding a solution to (Q), since computing $d(s)$ is equivalent to solving

$$\underset{\theta}{\text{minimize}} \quad E(\theta, \mu_0, \mu_1) + \int_\Omega \theta(\boldsymbol{x})\, s\, d\boldsymbol{x}\,, \tag{4.27}$$

followed by an evaluation of (4.26). However, since θ is constrained to be Boolean, strong duality does not generally hold (Boyd and Vandenberghe, 2004), that is, $d^* < E^*$ in general.

4.5 Gaussian Distributions

Many variants of the two-phase Mumford-Shah functional have been used for image segmentation. For example, ultrasound images can be segmented using a

maximum-likelihood formulation with the assumption that the image pixels are Rayleigh distributed (Sarti et al., 2004). To emphasize the difference to (4.2a), we will instead use a model where all pixels have equal expected values. This problem has previously been treated in (Rousson and Deriche, 2002) with local methods. Consider the following observation model, were the image pixels comes from two Gaussian distributions with zero mean and different variance:

$$I(\boldsymbol{x}) \sim \begin{cases} \mathrm{N}(0, \sigma_1^2), & \theta(\boldsymbol{x}) = 1 \\ \mathrm{N}(0, \sigma_0^2), & \theta(\boldsymbol{x}) = 0\,. \end{cases} \tag{4.28}$$

Given that $\theta(\boldsymbol{x}) = i$, the log-likelihood for the image pixel is

$$\ell_i(I(\boldsymbol{x})) = \log\left(\frac{1}{\sigma_i\sqrt{2\pi}}\exp\left(\frac{I(\boldsymbol{x})^2}{2\sigma_i^2}\right)\right)\,. \tag{4.29}$$

Given an observed image, we want to recover θ, so we want to solve the following minimization problem:

$$\begin{aligned}\underset{\theta,\sigma_0,\sigma_1}{\text{minimize}} \qquad & \rho J(\theta) + \int_\Omega \theta(\boldsymbol{x})\left[-\ell_1(I(\boldsymbol{x}))\right] \\ & + \big(1-\theta(\boldsymbol{x})\big)\big[-\ell_0(I(\boldsymbol{x}))\big]\,d\boldsymbol{x}\,. \end{aligned} \tag{4.30}$$

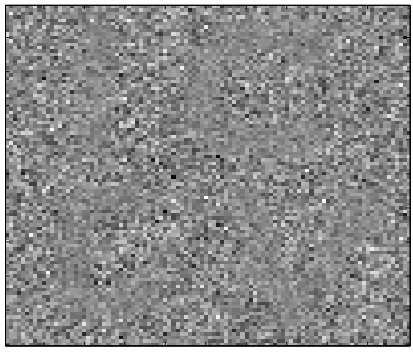
Following the same approach as in Section 4.3.1, we remove the term which does not depend on θ. After rearranging the factors inside the logarithms of the functional, we obtain:

$$\rho J(\theta) + \int_\Omega \theta(\boldsymbol{x})\left(\log\frac{\sigma_1}{\sigma_0} + I(\boldsymbol{x})^2\left(\frac{1}{2\sigma_1^2} - \frac{1}{2\sigma_0^2}\right)\right)\,d\boldsymbol{x}\,.$$

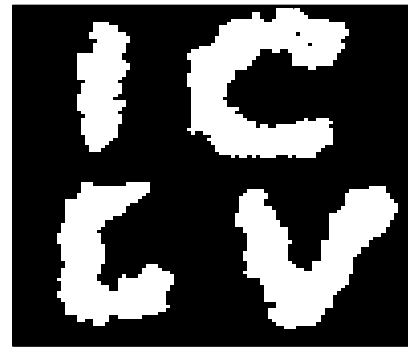
This suggests the following change of variables:

$$\begin{cases} r = \log\frac{\sigma_1}{\sigma_0} \\ t = \frac{1}{2\sigma_1^2} - \frac{1}{2\sigma_0^2} \end{cases} \quad\Longleftrightarrow\quad \begin{cases} \sigma_1 = \frac{\sqrt{-2t(e^{2r}-1)}}{2t} \\ \sigma_0 = \frac{\sqrt{-2t(e^{2r}-1)}}{2te^r}\,. \end{cases}$$

We can now solve problem (4.30) for $t =$ constant. First, we solve a ROF problem with $(w(\boldsymbol{x}) + I(\boldsymbol{x})^2 t)^2$ as data term. Then we threshold the solution at all possible levels r, evaluating the energy in (4.30) and choosing the lowest energy. A result with this segmentation model can be seen in Fig. 4.7.



(a) Original image with σ_1 = 6 and σ_0 = 10



(b) Resulting segmentation. Recovered σ: 6.07 and 10.35

Figure 4.7: Segmentation of an image composed of two Gaussian distributions with zero mean.

4.6 Conclusion

We have shown that the two-phase, binary Mumford-Shah functional can be effectively optimized by solving a continuous problem followed by thresholding. The method works if the difference between the two levels is fixed. To solve the general case, we give a branch and bound-like method which is shown to perform quite efficiently. It is interesting to note that a single evaluation of $m(\delta)$, which is optimal along one dimension, often seems to be enough to find the global optimum. The two main contributions of this chapter are:

1. We have defined $m(\delta)$ in (4.1) and shown how it can be computed efficiently.

2. This allows us to solve the problem in (4.2a) using branch and bound in a single dimension.

In addition, we have briefly discussed some connections to the parametric max-flow problems (Kolmogorov, Boykov and Rother, 2007) and optimization under constraints. We have also extended the numerical method given by Chambolle (2004; 2005) for the weighted ROF problem (Q).

Part II

Curvature Regularization

Chapter 5

Curvature Regularization in the Plane

The previous chapter involved finding the optimal curve with respect to its contents and its length. This chapter adds a curvature term, which requires radically different methods. The discussions I have had on this topic with Thomas Schoenemann have been very useful, and his source code have been essential for my work in this area. As this chapter builds upon the framework by Schoenemann, Kahl and Cremers (2009), familiarity with their paper will be useful.

5.1 Background

The problem we are interested in solving amounts to minimizing the following energy functional:

$$E(\Gamma) = \int_{\Gamma} g(\boldsymbol{x})\, d\boldsymbol{x} + \int_{\gamma} (\rho + \sigma|\kappa(s)|^p)\, ds, \tag{5.1}$$

where γ is the boundary curve of Γ, parametrized by its arc length s. Here $g(\boldsymbol{x})$ is the data term, which may take many different forms depending on the application, ρ is a positive weighting factor for length regularization, and σ controls the amount of curvature regularization.

Second order priors like curvature are important for many vision applications, for example, stereo (Woodford et al., 2009). In image segmentation, experiments have shown that curvature regularization is able to capture thin, elongated structures (El-Zehiry and Grady, 2010; Schoenemann, Kahl and Cremers, 2009) where standard length-based regulators would fail. Curvature has also been identified as a key factor in human perception based on psychophysical experiments on contour completion (Kanizsa, 1971). Still, most segmentation based approaches

in computer vision do not use curvature information. This is contrast to length or area regularity which do play an important role. One of the reasons for this fact is that curvature regularity is harder to incorporate in a global optimization framework. Note that curvature regularity is fundamentally different from length or area regularity. While, for example, length regularization prefers shorter boundaries, there is no such bias in curvature regularization. In fact, due to a famous theorem of Werner Fenchel, the integral of the absolute curvature for any closed convex plane curve is equal to 2π.

This chapter proposes several improvements to the current state-of-the-art of curvature regularization. This gives both faster running times and smaller memory requirements, and hence important steps are taken to make curvature regularization more practical. More specifically, we solve an identified issue with extraneous arcs while keeping the relaxation tight. Dual decomposition is applied to curvature problems to reduce memory. We also show how to obtain better suited tessellations of the domain, and higher resolution by using adaptive meshes.

There are number of application problems that can be modeled by the energy functional in (5.1). This chapter concentrates on improving the methodology for optimizing the functional and we compare with current state-of-the-art methods (Schoenemann, Kahl and Cremers, 2009; El-Zehiry and Grady, 2010). Experimental results are given for segmentation, but other applications include surface completion (Kawai, Sato and Yokoya, 2009), which will be discussed in the next chapter, and inpainting (Schoenemann, Kahl and Cremers, 2009; Masnou, 2002).

5.2 Length-Based Regularization

The basis for this work is the discrete differential geometry framework developed by Sullivan (1990) and Grady (2010) for computing minimal surfaces and shortest paths. The goal is to compute a discrete approximation of the continuous functional in (5.1). We will recast the problem as an integer linear program and solve it via LP relaxation. In this section we limit the exposition to the standard case without the curvature term (corresponding to $\sigma = 0$). Interestingly, this integer linear program can be shown to be totally unimodular and hence the LP relaxation will be tight.

The method is based on tessellating the domain of interest into a so-called cell complex, a collection of non-overlapping basic regions whose union gives the original domain. Several different kinds of tessellations are possible. Some

two-dimensional examples are given in Fig. 5.3. Typical choices are square meshes (2D), resulting in 4-connectivity. To mimic 8-connectivity, pixels are subdivided into four triangular regions each. This issue will be elaborated upon in Section 5.4.

The boundaries of 2D regions are called edges. It is necessary to consider both possible orientations of each edge and facet. In the integer linear program, there are two sets of Boolean variables, one reflecting regions and the other related to boundaries. For each basic region, a binary variable reflects whether the region belongs to the foreground or the background. Let x_i, $i = 1, \ldots, m$ denote these binary variables, where m is the number of basic regions. The region integral in (5.1) is now easily approximated by a linear objective function of the form $\sum_{i=1}^{m} g_i x_i$.

In this chapter we let x_i denote region variables and y_i boundary variables. The length term in (5.1) is then represented with $\rho \sum_i \ell_i y_i$, where ℓ_i denotes the length of edge i. To enforce consistency between the region and boundary variables, surface continuation constraints (Schoenemann, Kahl and Cremers, 2009) are used:

Surface continuation constraints. Assume we know that a basic region is part of the foreground. Then, for each of its edges we can have two valid configurations: either the associated basic region on the other side of the edge is also part of the foreground — or the foreground regions terminates here in an appropriately oriented boundary element. These constraints, together with the cases where the considered basic region is background, can be phrased as a linear equation system, with one constraint for each edge k:

$$\sum_i b_{k,i} x_i + \sum_i b_{k,i} y_i = 0, \tag{5.2}$$

where $b_{k,i}$ indicates whether region i is positive (1), negative (-1) or not incident (0) to edge k. $b_{k,i}$ indicates the incidence (± 1 or 0) of y_i to the edge k. Two terms in each sum will be nonzero.

5.3 Incorporating Curvature

To be able to handle curvature regularization, *pairs* of boundary variables are introduced. We denote these pairs by $y_{i,j}$. Schoenemann et al. (Schoenemann, Kahl and Cremers, 2009) described how to introduce boundary continuation

constraints to ensure that an actual boundary curve is formed. Without this constraint, only straight line pairs would be used.

Boundary continuation constraints. If a pair of line segments (l_1, l_2) is part of the boundary line, there must be another pair of line segments (l_2, l_3) that is also part of the boundary line. Furthermore, there must be a pair (l_0, l_1) that likewise belongs to the boundary line. Again, these constraints can be phrased as a linear equation system for each oriented edge ℓ:

$$\sum_{i,j} c_{\ell,i,j} y_{i,j} = 0, \tag{5.3}$$

where

$$c_{\ell,i,j} = \begin{cases} 1 & \text{if } \ell = i \\ -1 & \text{if } \ell = j \\ 0 & \text{otherwise.} \end{cases} \tag{5.4}$$

Having introduced the line pair variables, the last term in (5.1) may also be represented as a linear function: $\sigma \sum_{i,j} b_{i,j} y_{i,j}$. The coefficients $b_{i,j}$ used by Bruckstein, Netravali and Richardson (2001) and Schoenemann et al. were

$$b_{i,j} = \min\{\ell_i, \ell_j\} \left(\frac{\alpha}{\min\{\ell_i, \ell_j\}} \right)^p, \tag{5.5}$$

where α is the angle difference between the two lines. This chapter uses $p = 2$ exclusively.

5.3.1 Avoiding Extraneous Arcs

The constraints introduced by Schoenemann, Kahl and Cremers admit too many feasible solutions. This is illustrated in Fig. 5.2a, where sharp corners are avoided by introducing large, extra curves which due to the nonexistent length penalty have low cost. This solution is integral and optimal in the original formulation, because along the spurious large arcs both $y_{i,j}$ and $y_{j,i}$ are active.

The solution seems simple: to add constraints $y_{i,j} + y_{j,i} \leq 1$. This would indeed solve the problem if the variables could be restricted to be integral, but we have found these constraints in practice gives a fractional solution with even more spurious arcs (Fig. 5.2b). Therefore, the additional linear constraints we propose are of a different type:

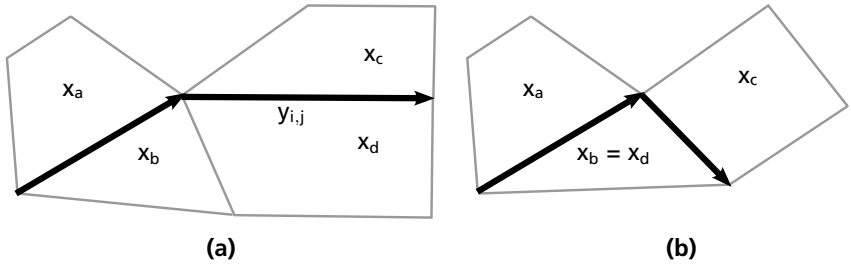
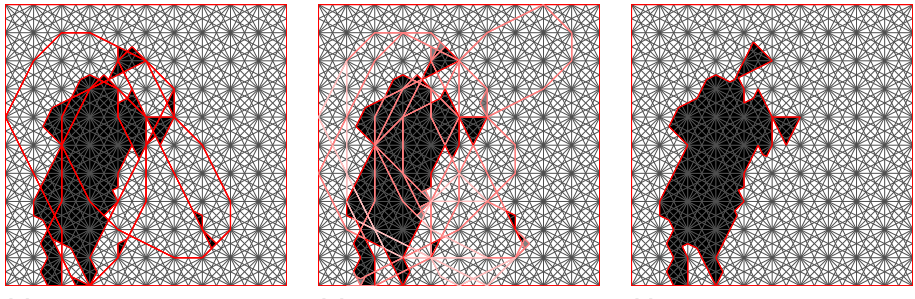


Figure 5.1: The line pair variable $y_{i,j}$ and its four incident region variables x_a, x_b, x_c and x_d. The four region variables may coincide for some edge pairs.



(a) Using the original constraints from (Schoenemann, Kahl and Cremers, 2009). The small black regions to the right have their boundary costs reduced by the large arcs of extra boundaries.

(b) Simply requiring that $y_{i,j} + y_{j,i} \leq 1$ would work if the variables were integers, but causes the LP relaxation to output a fractional solution.

(c) Result using the additional constraints (5.7). The LP-relaxation output an integral solution.

Figure 5.2: Segmentation with and without region consistency constraints. A very crude mesh was used to make the visualization clearer. Gray scale polygons indicate region variables and red lines indicate edge pair variables. Gray lines show the mesh used. Parameters: $\rho = 0$, $\sigma = 300000$. The time to solve the problem decreased by adding the extra constraints: 0.251s vs. 3.402s for the original problem.

Region consistency constraints. Consider a line pair variable $y_{i,j}$ and call its four incident regions x_a, x_b, x_c and x_d, located as shown in Fig. 5.1a. If $x_a = x_b = 1$ or $x_a = x_b = 0$, the region pair should not be active. Similarly for x_c and x_d. This can be linearly encoded as

$$\begin{aligned} x_a + x_b + y_{i,j} + y_{j,i} &\leq 2 \\ -x_a - x_b + y_{i,j} + y_{j,i} &\leq 0. \end{aligned} \tag{5.6}$$

Similar constraints hold for x_c and x_d. All in all, four new constraints are introduced for each pair of edges. It might be the case that x_a and x_c or x_b and x_d coincide, c.f. Fig. 5.1b. The constraint (5.6) still looks the same. To reduce the total number of constraints, the constraints can be combined into four constraints per edge k:

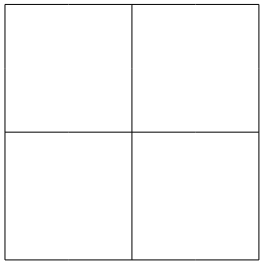
$$\begin{aligned} x_{k_1} + x_{k_2} + \sum_{(k,j) \text{ a pair}} y_{k,j} &\leq 2 \\ -x_{k_1} - x_{k_2} + \sum_{(k,j) \text{ a pair}} y_{k,j} &\leq 0 \\ x_{k_1} + x_{k_2} + \sum_{(j,k) \text{ a pair}} y_{j,k} &\leq 2 \\ -x_{k_1} - x_{k_2} + \sum_{(j,k) \text{ a pair}} y_{j,k} &\leq 0. \end{aligned} \tag{5.7}$$

Here x_{k_1} and x_{k_2} denote the two regions adjacent to edge k. The first two constraints sum over all line pairs starting with edge k and the last two sum over all pairs ending with edge k.

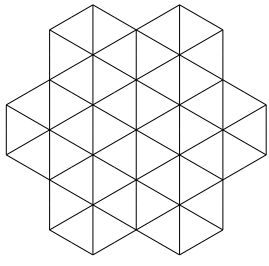
Fig. 5.2c show the result with these additional constraints where the boundary now is consistent with the region variables. As a bonus, the new constraints reduced the time required to solve the problem to about 7%. We can also see from Fig. 5.2 that both before and after the additional constrains, the optimal solution has its region variables equal or very close to 0 or 1.

5.3.2 QPBO Optimization

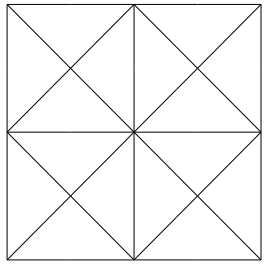
Solving the discrete optimization problem does not have to be done using a linear program. It is also possible to use discrete optimization methods such as QPBO.



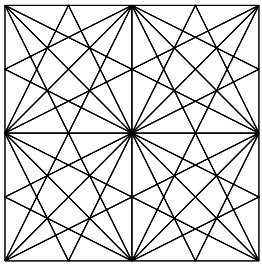
(a) Square mesh with 4-connectivity. Each cell has 1 region and 2 lines (on average).



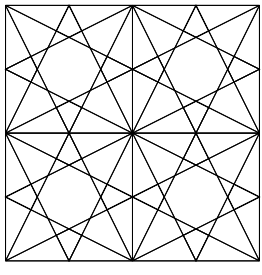
(b) Hexagonal mesh with 6-connectivity. Each cell has 6 regions and 9 lines.



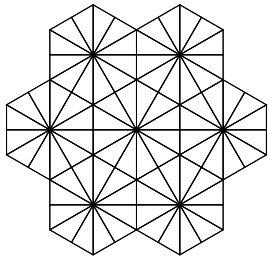
(c) Square mesh with 8-connectivity. Each cell has 4 regions and 6 lines.



(d) Square mesh with 16-connectivity. Each cell has 31 regions and 52 lines.



(e) Square mesh with 12-connectivity. Each cell has 21 regions and 44 lines.



(f) Hexagonal mesh with 12-connectivity. Each cell has 12 regions and 18 lines.

Figure 5.3: Different types of grids. The maximum angle between the possible straight lines is 90° in (a), 60° in (b) and 45° in (c). The last three (d), (e) and (f) all have 30° as their maximum angle.

Each edge pair is represented as a 3- or 4-clique in the energy minimization. This formulation has the advantage that it readily carries over to three dimensions. El-Zehiry and Grady (El-Zehiry and Grady, 2010) used 3-cliques for minimizing curvature functionals and their formulation is equivalent to (Schoenemann, Kahl and Cremers, 2009) for 4-connected grids. This is because in a 4-connected grid, only configurations of the type in Fig. 5.1b are present. If one wants to go to higher connectivities, configurations as shown in Fig. 5.1a are present and 4-cliques are required. We will now see why.

3-cliques. An edge pair connected to three region variables is shown in Fig. 5.1b. The edge pair adds a cost to the total energy of the segmentation if $x_a = x_c = 1$ and $x_b = 0$ or the opposite: $x_a = x_c = 0$ and $x_b = 1$. If ω is the cost of the edge pair (length and curvature), the terms added to the total energy are:

$$\omega\Big(x_a x_c(1 - x_b) + (1 - x_a)(1 - x_c)x_b\Big). \tag{5.8}$$

It is noted in (El-Zehiry and Grady, 2010) that this expression, although technically a 3-clique, does not need an extra node to be handled correctly. This is because it is actually equal to

$$\omega\Big(x_a x_c - x_a x_b - x_b x_c + x_b\Big), \tag{5.9}$$

and this can be represented with only pairwise interactions between the three variables.

4-cliques. For all connectivities higher than 4, many edge pairs are adjacent to 4 regions, see Fig. 5.1a. Just as before, this can be represented as

$$\omega\Big(x_a x_c(1 - x_b)(1 - x_d) + (1 - x_a)(1 - x_c)x_b x_d\Big). \tag{5.10}$$

Representing this, however, requires extra nodes to be added, because the above expression simplifies to

$$\begin{aligned}\omega\Big(&2x_a x_b x_c x_d + x_a x_c - x_a x_b x_c - \\ &x_a x_c x_b + x_b x_d - x_a x_b x_d - x_b x_c x_d\Big),\end{aligned} \tag{5.11}$$

where the high-arity factors have not canceled each other out. How to handle energy terms like this is described by Ishikawa (2009) and we mention that representing the above 4-clique requires 10 extra nodes and 24 edges. This is much worse than the three clique which required no extra nodes and only 3 edges.

5.4 Types of Meshes

The mesh used for the segmentation can be created in a number of ways. The quality of the approximation depend on how many different possible straight lines that can be represented by the mesh, since a larger possible choice of line slopes allows the mesh to approximate a continuous curve more closely. Fig. 5.3 shows some possible meshes and the straight lines they admit. If a mesh allows n possible straight line directions, it is referred to as *n-connected*.

5.4.1 Hexagonal Meshes

Hexagonal meshes have long been studied for image processing (Middleton and Sivaswamy, 2005). One characterizing fact of hexagons is that they are the optimal way of subdividing a surface into regions of equal area while minimizing the sum of the boundary lengths (Hales, 2001). The fact that is more important to us is the neighborhood structure. In a hexagonal lattice every region has 6 equidistant neighbors. When approximating curvature we would like to represent as many different straight lines as possible and we would like the maximum angle between them to be small, as that gives us a better approximation of a smooth curve (Bruckstein, Netravali and Richardson, 2001). The neighborhood structure of the hexagonal mesh allows for similar performance (number of lines and angle between them) while using fewer regions. This is illustrated in Fig. 5.3, where three crude meshes and three finer meshes are shown. All three fine meshes have the same maximal angle between the possible straight lines, but the hexagonal mesh achieves this with fewer regions due to the favorable intersection pattern of the lines. This suggests that hexagonal meshes can achieve the same accuracy as the mesh (d) used by Schoenemann, Kahl and Cremers (2009) with a significantly smaller linear program.

Calculating the data term. With the introduction of the hexagonal grid, every region is no longer contained within a single pixel. Some regions will partly overlap

more than one pixel. The data term for the region R_k is the integral of g over that region:

$$g_k = \int_{R_k} g(\boldsymbol{x}) d\boldsymbol{x}. \tag{5.12}$$

The data term is a function on a continuous domain. However, because it arises from a measured image, it will be piecewise constant on the $n \times m$ square image pixels. If each pixel is subdivided into regions, the data term for each region is simply the area of the region multiplied with the data term for the pixel. In the general case, the data term for R_k is computed as:

$$g_k = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} g\left(i + \frac{1}{2}, j + \frac{1}{2}\right) \cdot \text{area}(R_k \cap p_{i,j}), \tag{5.13}$$

where $p_{i,j}$ is the square representing pixel (i, j):

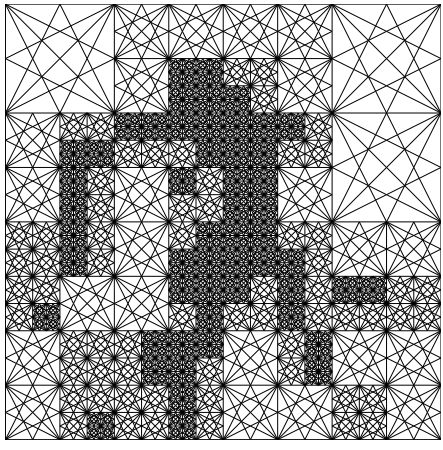
$$p_{i,j} = \{(x, y) \,|\, i \leq x \leq i + 1,\ j \leq y \leq j + 1\}. \tag{5.14}$$

Calculating this sum requires a large number of polygon intersections to be computed. For this we used the General Polygon Clipper library (GPC) from the The University of Manchester.

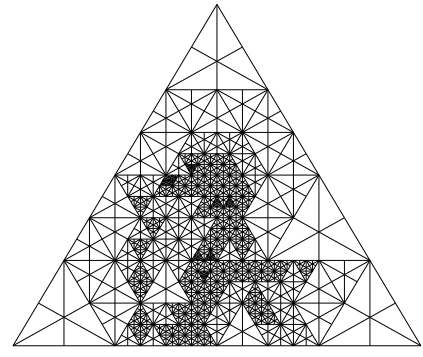
5.4.2 Adaptive Meshes

The memory requirements for solving the linear programs arising from the discretizations are very large. Each pair of connected edges introduce two variables. Linear programs are typically solved using the simplex method or interior point methods, both of which require a substantial amount of memory for our problems. As one example, a problem with 131,072 regions and 1,173,136 edge pairs required about 2.5 GB of memory to solve using the Clp solver.

For this reason, it is desirable to keep the size of the mesh small. However, a fine mesh is needed to be able to approximate every possible curve. The solution to this conflict of interest is to generate the mesh adaptively, to only give it high resolution where the segmentation boundary is likely to pass through. Adaptive meshes have previously been considered for image segmentation in the level-set framework (Xu, Thompson and Toga, 2004) and in combinatorial optimization of continuous functionals (Kirsanov and Gortler, 2004).



(a) 16-connectivity by subdividing rectangles



(b) 12-connectivity by subdividing triangles

Figure 5.4: Adaptive meshes can be constructed by recursively subdividing basic shapes into several similar shapes and finally adding the extra connectivity.

The mesh is refined using an iterative process. First, a single region is put into a priority queue. Then regions are removed from the priority queue and subdivided into smaller regions which are put back into the queue. The region which most urgently needs to be split is removed first from the priority queue. This is determined by a score which is computed for each region as follows.

Start with q an empty priority queue
$R \leftarrow (0, 0, w, h)$
Add R to q with priority $= \mathrm{score}(R)$
while $\mathrm{size}(q) < L$ **do**
  Remove R from q
  Split R into $R_1 \ldots R_k$
  Add $R_1 \ldots R_k$ to q with $\mathrm{score}(R_1) \ldots \mathrm{score}(R_k)$
end while

Both square and triangular basic shapes can be split up into four identical shapes similar to the original one. Thus, for all adaptive meshes we use $k = 4$. The score function can be chosen in many different ways. One way is to use the

squared deviation from the mean of each region, i.e.:

$$\text{score}(R) = \int_R (I(\boldsymbol{x}) - \mu(R))^2 d\boldsymbol{x}, \tag{5.15}$$

where $\mu(R) = \frac{1}{|R|} \int_R I(\boldsymbol{x}) d\boldsymbol{x}$. This way, regions where the data term vary a lot will be split before regions which have a uniform data term. The score is not normalized, because otherwise many very small regions would tend to have a big score. The integrals may be computed in the same manner as (5.13) for images with square pixels, resulting in the computation of polygon intersections.

5.5 Experimental Results

This chapter does not focus on how to model the data term and we will use a simple, two-phase version throughout all our experiments:

$$g(\boldsymbol{x}) = (I(\boldsymbol{x}) - \mu_1)^2 - (I(\boldsymbol{x}) - \mu_0)^2, \tag{5.16}$$

where μ_0 and μ_1 are two fixed mean values and I is the image.

5.5.1 Hexagonal Meshes

In our first experiment we evaluate hexagonal vs. square meshes. We are comparing three types of meshes, the 8- and 16-connected square mesh and the 12-connected hexagonal mesh, shown in Fig. 5.3 (c), (d) and (f). We fixed a data term of a 256×256 image (cameraman) and lay meshes of various types and sizes on top of it and calculated the optimal energy.

The result is shown in Fig. 5.5, where the optimal energy is plotted as a function of the number of regions used. This is reasonable, since the number of regions is a good indicator of the total size of the linear program. The analogous plots using the number of line pairs or edges look the same. We see that the 8-connected grid converges quickly, but to a suboptimal energy. The hexagonal mesh consistently outperforms the 16-connected grid. If we were to let the number of regions grow very large, the 16-connected grid would probably achieve a lower energy than the hexagonal, due to it having 2 more possible straight lines. We have not been able to observe this in practice, though, due to the memory requirements.

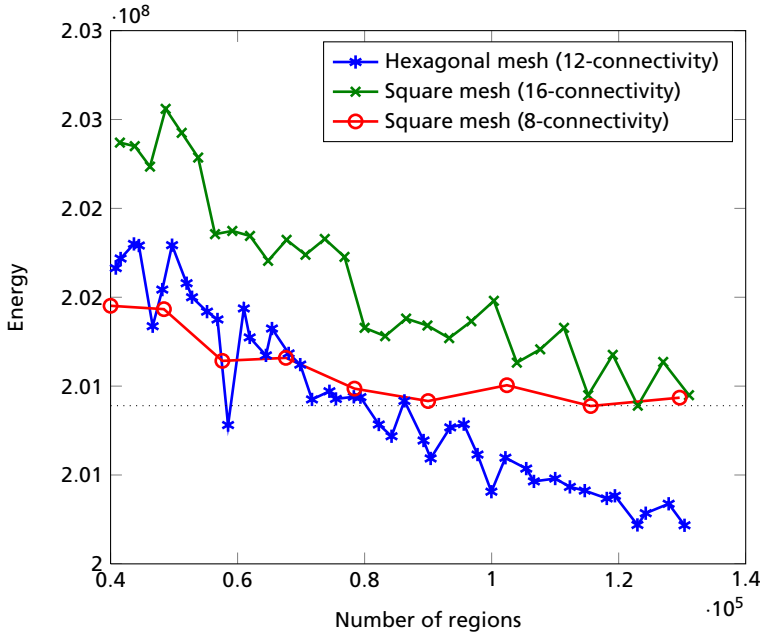


Figure 5.5: Optimal energy vs. the total number of regions. The best accuracy obtained by the square mesh was achieved by the hexagonal mesh with about half the number of regions. This experiment used $\rho = \sigma = 10000$. The energy difference might seem small, but differences of these magnitudes often correspond to significant changes in segmentation, cf. Fig. 5.6.

(a) Regular grid **(b)** Adaptive grid

Figure 5.6: Results with regular and adaptive 16-connected grids. The number of regions used were 32,768 in both cases and the number of edge pairs were 291,664 and 285,056, respectively. The adaptive mesh gives a smoother curve and correctly includes the hand of the camera man. The optimal energy for the regular mesh was $2.470 \cdot 10^8$ and $2.458 \cdot 10^8$ for the adaptive. This experiment used $\rho = 30000$ and $\sigma = 1000$.

5.5.2 Adaptive Meshes

To evaluate the effect of adaptive meshes, we performed a number of experiments. Firstly, we evaluated the visual quality of the segmentation for regular and adaptive 16-connected meshes with the same number of regions. The result can be seen in Fig. 5.6. There is a significant visual difference. The fact that the adaptive mesh achieved a smoother curve is also reflected in the optimal energy, which is lower. The results for 8-connected meshes are shown in Fig. 5.7 and yield the same conclusion.

To evaluate the performance more quantitatively, we solve the same segmentation problem a large number of times for different number of regions. The adaptive mesh converged to what probably is the optimal energy for that connectivity, while the regular mesh did not. The regular mesh would have required more than 20 times more regions to achieve the same energy. Fig. 5.8 shows the optimal energies for the different number of regions and the two types of meshes.

(a) Regular grid **(b)** Adaptive grid

Figure 5.7: Results with (a) regular and (b) adaptive 8-connected grids. The optimal energy for the regular mesh was $2.517 \cdot 10^8$ and $2.481 \cdot 10^8$ for the adaptive. This experiment used $\rho = 30000$ and $\sigma = 1000$.

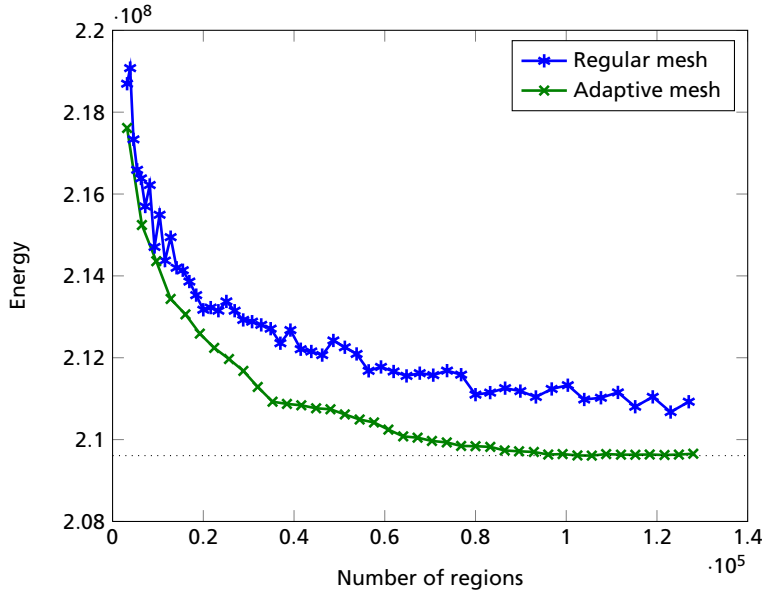


Figure 5.8: Optimal energy vs. the total number of regions for a square 16-connected mesh. To get the same accuracy as the finest parts of the adaptive mesh, the regular mesh would need $21 \cdot 10^5$ regions. In contrast, the adaptive mesh converged using about $1 \cdot 10^5$ regions. This experiment used $\rho = \sigma = 10000$.

	Peak memory usage (MB)
Original problem	468
Dual decomposition (2 parts)	279

Table 5.1: Peak memory usage for a 128×128 homogenous 8-connected mesh. The reduction not as good as 50% because the solvers cache the solutions and system matrices between iterations.

5.5.3 QPBO Experiments

Our QPBO experiments were disappointing. It seems that the formulation with higher-order cliques is weaker than the previously discussed linear programming formulations. In all cases except with a non-negligible value of σ we obtained 75%–95% unlabeled nodes, with no hope of recovering a good solution with e.g. QPBO-P (Rother et al., 2007*a*). No 4-cliques are required if the very simple cubic mesh in Fig. 5.3a is used, and we were in this case able to solve the optimization problem with QPBO, hence reproducing the results in (El-Zehiry and Grady, 2010). However, such meshes are too coarse to be of any practical use.

5.5.4 Dual decomposition

Chapter 7 will describe a dual decomposition scheme, where the domain is split into two pieces which are solved separately and constrained to be equal on an overlap, see Fig. 7.1 on page 85. The same method can be used to split curvature linear programs to reduce the memory requirements of the linear solver. While splitting an adaptive mesh into two or more parts takes some care, splitting a regular square mesh is relatively straightforward. A specialized solver for the original problem must be able to solve the subproblems as well. Therefore, some care has to be taken to make sure all edge pairs are counted the correct number of times in and around the overlap. It is easy to test if the split has been made correctly. The sum of the minimal energies in the left and right part has to match the minimal energy of the original problem exactly. The solutions to the two problems are constrained to be equal for all overlapping regions. We use the exact same updating scheme as will be described in depth for graph cuts in Section 7.2.3.

To be able to compare the difference in peak memory usage, we ran experiments using a 128×128 mesh. The memory reduction with dual decomposition was significant, see Table 5.1. The achieved memory reduction was slightly less than

50% since the intermediate states of the solvers are stored between iterations to increase execution speed. The total time required by dual decomposition (single-threaded) is usually slightly less than solving the original problem, although this can vary. The main benefit is the reduced memory consumption.

5.6 Conclusions

The purpose of this chapter has been to discuss the problem of segmentation with curvature regularization and to enhance the methodology in numerous ways.

First of all, new region consistency constraints (5.7) have been introduced, which are essential for the method in (Schoenemann, Kahl and Cremers, 2009) to work well (Fig. 5.2c). These new constraints also reduced the computation time to less than one tenth of the original time.

I have argued, by studying the angles between straight lines and the number of regions in different meshes that hexagonal meshes are more suitable than square meshes with the same number of regions. Experiments have confirmed this conclusion as well (Fig. 5.5).

Another way of reducing the memory requirements is to allow an adaptive generation of the mesh. Generating the mesh adaptively by examining the changes of the data term results in far better segmentations, both quantitatively (Fig. 5.8) and qualitatively (Figs. 5.6 and 5.7). Finally, dual decomposition may be used to split the linear program in smaller, more manageable pieces.

I have included a discussion of Boolean optimization with higher-order terms. Unfortunately, experiments have shown that this formulation is not an attractive alternative (except for 4-connectivity which does not have 4-cliques and was introduced by El-Zehiry and Grady (2010)). This is unfortunate, since this formulation easily generalizes to three dimensions. A three-dimensional framework for surface completion and segmentation will be the topic of the next chapter.

Chapter 6

Surface Completion and Segmentation with Curvature

We have seen that it in many cases is possible to minimizes functionals involving the curvatures of plane curves. This chapter will describe the work I conducted together with Fredrik Kahl on extending the framework to surfaces in three dimensions. The problem we are interested in solving amounts to minimizing the following energy functional:

$$E(R) = \int_R g(\boldsymbol{x})\, d\boldsymbol{x} + \int_{\partial R} \left(\rho + \sigma\kappa(\boldsymbol{x})^2\right) dA(\boldsymbol{x}), \tag{6.1}$$

where R is a volume with boundary ∂R. Here $\kappa(x)$ is the mean curvature of the surface ∂R at $\boldsymbol{x}$. $g(\boldsymbol{x})$ is the data term, which may take many different forms depending on the application. It is typically present in segmentation problems and not present in surface completion problems. ρ and σ controls the amount of area and curvature regularization, respectively.

In differential geometry, energy functionals of the type in (5.1) have been studied for a long time. The functional is known as the Willmore energy (Willmore, 1965). It gives a quantitative measure of how much a given surface deviates from a round sphere. Local descent techniques have been derived for minimizing (5.1) (Hsu, Kusner and Sullivan, 1992), but they are very dependent on a good initialization. This chapter will introduce a framework for minimizing these functionals *globally*.

6.1 Curvature of Surfaces

Each facet in our 3D mesh is associated with a variable $\boldsymbol{y} = (y_1, \ldots, y_{2n})$ of areas $\boldsymbol{a} = (a_1, \ldots, a_{2n})$. There are twice as many variables as facets, because

each facet is associated with two variables, one for each orientation. The two are distinguished by (arbitrarily) assigning a normal to each face in the mesh. The optimization problem for surface completion with area regularization is

$$\begin{aligned} \underset{\boldsymbol{y}}{\text{minimize}} \quad & \rho \boldsymbol{a}^{\mathrm{T}} \boldsymbol{y} \\ \text{subject to} \quad & \mathsf{B}\boldsymbol{y} = 0 \\ & \boldsymbol{y} \in \{0,1\}^{2n} \\ & y_k = 1,\ k \in K. \end{aligned} \tag{6.2}$$

K is the set of facets that are supposed to be part of the minimal surface a priori. The matrix B is defined by Grady (2010) as:

$$\mathsf{B}_{e,y_i} = \begin{cases} +1, & \text{if edge e borders } y_i \text{ with coherent orientation} \\ -1, & \text{if edge e borders } y_i \text{ with coherent orientation} \\ 0, & \text{otherwise.} \end{cases} \tag{6.3}$$

We now extend this formulation to support curvature by introducing face pairs. Each pair of facets in the mesh with an edge in common are associated with two variables $\{y_{i,j}\}$ (one for each orientation). Enforcing consistency between the face variables and the variables corresponding to the pairs of faces can be done with linear constraints:

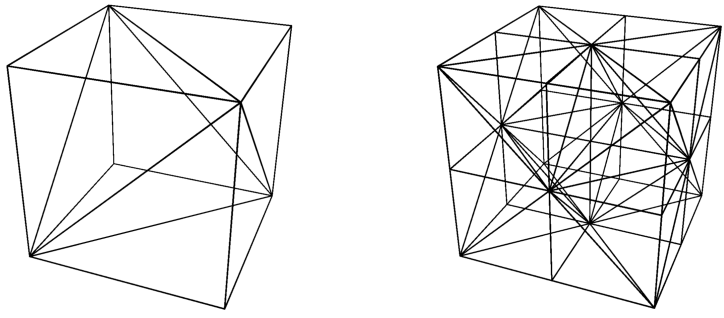
Surface continuation constraints. For each oriented facet k and each one of its edges e we add the following constraint:

$$y_k = \sum_{(i,j) \text{ with edge } \boldsymbol{e}} d_{k,i,j} y_{i,j}. \tag{6.4}$$

The sum is over all pairs ij with edge $\boldsymbol{e}$ in common. The indicator $d_{k,i,j}$ is 1 if facet k is part of the pair (i,j).

Having introduced the facet pairs, we follow Wardetzky et al. (2007) and associate them with a cost $b_{i,j}$, approximating the mean curvature (compare with (5.5) on page 58):

$$b_{ij} = \frac{3||\boldsymbol{e}_{ij}||^2}{2(a_i + a_j)} \left(2 \cos \frac{\theta_{ij}}{2}\right)^2, \tag{6.5}$$



(a) One unit cube.

(b) Eight unit cubes in a $2 \times 2 \times 2$ mesh.

Figure 6.1: Each unit cube is split into 5 tetrahedrons. This is the type of mesh used for our experiments in 3D. When stacking several, every other cube has to be mirrored in order to fit.

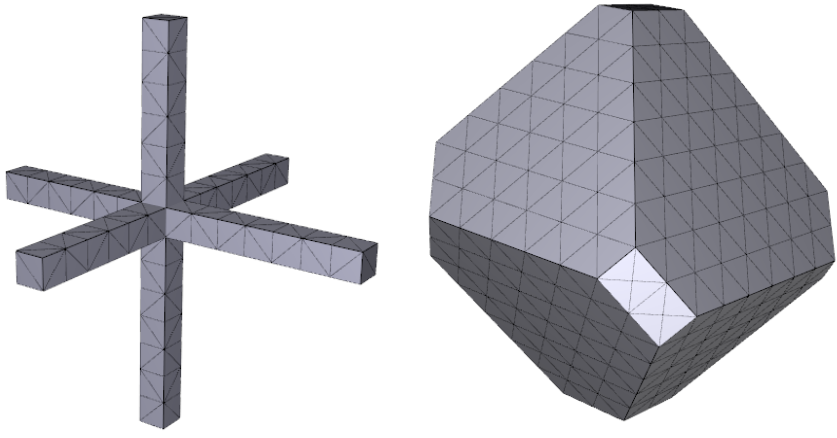
where θ_{ij} is the dihedral angle between the two facets in the pair. $||\boldsymbol{e}_{ij}||$ is the length of their common edge. The objective function we are minimizing is then $\rho \sum_i a_i y_i + \sigma \sum_{i,j} b_{i,j} y_{i,j}$, subject to the constraints in (6.2) and (6.4). This approximation is far from perfect; for example, it will not give the correct approximation for saddle points. However, it measures how much the surface bends and fulfills a couple of requirements listed by Wardetzky et al. (2007).

Segmentation, as opposed to surface completion, require variables for each volume element in order to incorporate the data term. Let x_i be variables associated with the volume elements. Additional consistency constraints are then required:

Volume continuation constraints. For each facet k,

$$\sum_i b_{k,i} y_i + \sum_i g_{k,i} x_i = 0, \tag{6.6}$$

where b_k indicates whether the facet y_k is positively or negatively incident w.r.t. the chosen face normal. $g_{k,i}$ is 1 if the volume element x_i is positive incident (the face normal points towards its center), -1 if it is negative incident and 0 otherwise. Both sums have two non-zero terms.



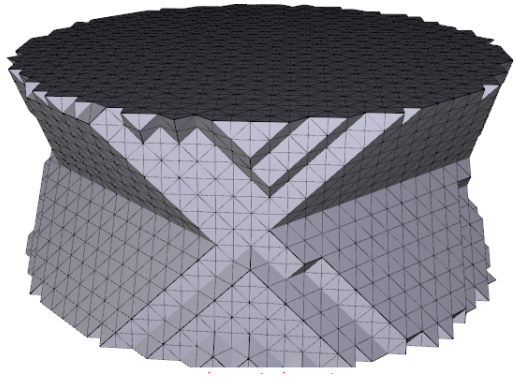
(a) Area regularization **(b)** Curvature regularization

Figure 6.2: Surface completion on a $16 \times 16 \times 16$ mesh with area and curvature regularization and volume element variables. The data term and the optimal surface using area regularization coincide. The radius of the volume in (b) is constrained by the mesh size. Otherwise, a minimal surface would not exist for the continuous problem.

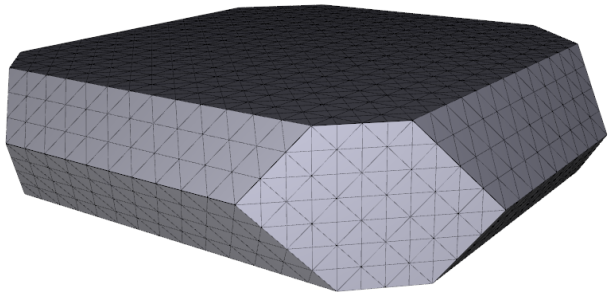
6.2 Experiments

For our experiments in three dimensions we generated a mesh where each unit cube was split into 5 tetrahedrons, see Fig. 6.1. We then create the set K as two circular surfaces at $z = 0$ and $z = z_{\max}$, with nothing in between. The analytic solution with area penalty is the catenoid, one of the first minimal surfaces found. Fig. 6.3a show the discrete version obtained with $\rho = 1$ and $\sigma = 0$. If instead the mean curvature is chosen as the regularizer, the optimal surface instead bends outwards. The solution to this problem is shown in Fig. 6.3b and is the global optimum, since all variables ended up integral in the LP relaxation of (6.2). We used Clp as our LP solver.

In another experiment we also used variables for the volume elements. The data term was a 3D 'cross' where the volume elements were forced to be equal to 1, whereas the volume elements at the boundary were forced to be 0. The optimal segmentation when the area was minimized coincided with the data term and is shown in Fig. 6.2a. When instead minimizing the curvature the optimal



(a) Area regularization on a $40 \times 40 \times 15$ mesh (491k variables, 398k constraints and 447 seconds). A $25 \times 25 \times 7$ mesh required 91k variables and 5 seconds.



(b) Curvature regularization on a $25 \times 25 \times 7$ mesh (637k variables, 441k constraints and 178 seconds).

Figure 6.3: Surface completion with area and curvature regularization. Two flat, circular surfaces at the top and bottom were fixed to 1. The surface in (a) bends inwards to approximate a catenoid and in (b) it correctly bends outwards to minimize the curvature.

segmentation should resemble a sphere, which is observed in Fig. 6.2b.

This chapter has introduced constraints for 3D surface completion and segmentation. Experiments are encouraging with exclusively globally optimal solutions. To my knowledge, this is the first time the mean curvature of surfaces has been optimized globally. The next step would be to apply this method to e.g. the partial surfaces obtained by stereo estimation algorithms. Another line of further research is how to be able to cope with finer discretizations of the 3D volume.

Part III

Parallel and Distributed Optimization

Chapter 7

Parallel and Distributed Graph Cuts

Many problems in low-level computer vision can be formulated as labeling problems using Markov Random Fields (MRFs). Among the examples are image segmentation, image restoration, dense stereo estimation and shape estimation (Greig, Porteous and Seheult, 1989; Boykov and Kolmogorov, 2004; Boykov, Veksler and Zabih, 1998; Rother, Kolmogorov and Blake, 2004; Lempitsky and Boykov, 2007). I showed in chapter 3 that when the number of labels is equal to 2, the problem can sometimes be formulated as a maximum flow or a minimum cut problem (Kolmogorov and Zabih, 2004). For problems with three or more labels α-expansion and other approximation methods (Komodakis, Paragios and Tziritas, 2007; Komodakis and Tziritas, 2007; Carr and Hartley, 2009) are available.

In this and the next chapter I will describe how the technique of dual decomposition from chapter 2 applies to these and related optimization methods. The focus for this chapter will be graph cuts. We will see that it is a valuable tool to parallelize existing algorithms to make them run faster on modern multi-core processors. We will also see that prohibitively large memory requirements can be made tractable on both desktop computers and supercomputer clusters. Examples of optimization problems with huge memory requirements are segmentation problems in three or more dimensions and curvature regularization problems. Finally, we will see that dual decomposition also provides an interesting way of implementing algorithms on parallel hardware, such as graphical processing units (GPUs).

The next section reviews other approaches to parallel graph cuts. I will begin with an overview of the contributions of this chapter, which is based on a paper previously published at CVPR (Strandmark and Kahl, 2010).

Decomposition of graph cuts The contribution of this chapter is a parallel implementation of the graph cuts method by Boykov and Kolmogorov (2004), which henceforth is referred to as "BK". The approach has a number of advantages compared to other parallelization methods:

1. The BK-method has been shown to have superior performance compared to competing methods for a number of applications (Boykov and Kolmogorov, 2004), most notably sparse 2D graphs and moderately sized 3D graphs.

2. It is possible to reuse the search trees in the BK-method (Kohli and Torr, 2005, 2007) which makes the dual decomposition approach attractive. Further, we show that the dual function can be optimized using integer arithmetic. This makes the dual optimization problem easier to solve.

3. Perhaps most importantly, we demonstrate good empirical performance with significant speed-ups compared to single-threaded computations, both on multi-core platforms and multi-computer networks.

Naturally, there are also some disadvantages:

1. There is no theoretical guarantee that the parallelization will be faster for every problem instance. Already for the BK-method no polynomial time guarantee is known, and we cannot give one for the number of iterations, either. In practice this matters little.

2. Our current implementation is only effective for graphs for which the BK-method is effective. The underlying dual decomposition principle can however be applied in combination with any graph cut algorithm.

7.1 Previous Approaches to Graph Cuts in Vision

Our work builds on the following two trends: the ubiquity of maximum flow computations in computer vision and the tendency of modern microprocessor manufacturers to increase the number of cores in mass-market processors. This implies that an efficient way of parallelizing maximum flow algorithms would be of great use to the community. Due to a result from Goldschlager et al. (Goldschlager, Shaw and Staples, 1982), there is little hope in finding a general algorithm for parallel maximum flow with guaranteed performance gains. However, the graphs encountered in computer vision problems are often sparse with much fewer edges

than the maximum $n^2 - n$ in a graph with n vertices. The susceptibility to parallelization depends on the structure and costs of the graph.

There are essentially three types of approaches used in computer vision for solving the maximum flow/minimum cut problem:

Augmenting paths The most popular method due to its computational efficiency for 2D problems and moderately sized 3D problems with low connectivity (i.e., sparse graphs) is the BK-method using augmenting paths (Boykov and Kolmogorov, 2004). However, as augmenting path algorithms use non-local operations, they have not been considered as a viable candidate for parallelization. One way of making multiple threads cooperate is to divide the graph into disjoint parts. This is the approach taken by Liu, Sun and Shum (2009), in which the graph is split, solved and then split differently in an iterative fashion until no augmenting paths can be found. The key observation is that the search trees of the subgraphs can be merged relatively fast. The more recent work by Liu and Sun (2010) splits the graph into many pieces which, in turn, multiple threads solve and merge until only one remains and all augmenting paths have been found. In this chapter the graph is also split into multiple pieces, but our approach differs in that we do not require a shared-memory model, which makes distributed computation possible.

Push-relabel The push-relabel algorithm (Goldberg and Tarjan, 1986) is an algorithm suitable for parallelization. The implementation by Delong and Boykov (2008) has been tested for up to 8 processors with good results. There have been attempts to implement this method on a GPU, the latest being CUDA cuts by Vineet and Narayanan (2008; 2009), but our tests of the (freely available) implementation only gave the correct result for graphs with low regularization. Another attempt was made by Hussein, Varshney and Davis (2007), which performed all experiments on generated images with a very low amount of regularization. Solving such graphs essentially reduces to trivial thresholding of the data term. The earliest reference I was able to find was the paper by Dixit, Keriven and Paragios (2005) which does not report any speed-up compared to sequential push-relabel.

Convex optimization Another approach to parallel graph cuts is to formulate the problem as a linear program. Under the assumption that all edges are bidirectional, the problem can then be reformulated as an ℓ_1 minimization problem. The work by Bhusnurmath and Taylor (2008) attempts to solve this problem with

Newton iterations using the conjugate gradient method with a suitable preconditioner. Matrix-vector multiplications can be highly parallelized, but this approach has not proven to be significantly faster than the single-threaded BK algorithm for any type of graph, even though Bhusnurmath and Taylor used a GPU in their implementation.

Convex optimization based on a GPU has also been used to solve continuous versions of graph cuts, e.g. (Klodt et al., 2008). However, the primary advantage of continuous cuts has been to reduce metrication errors due to discretization.

Graph cuts is also a popular method for multi-label problems using, e.g., iterated α-expansion moves. Such local optimization methods can naturally be parallelized by performing two different moves in parallel and then trying to fuse the solutions, as done in (Lempitsky et al., 2009).

7.2 Decomposition of Graphs

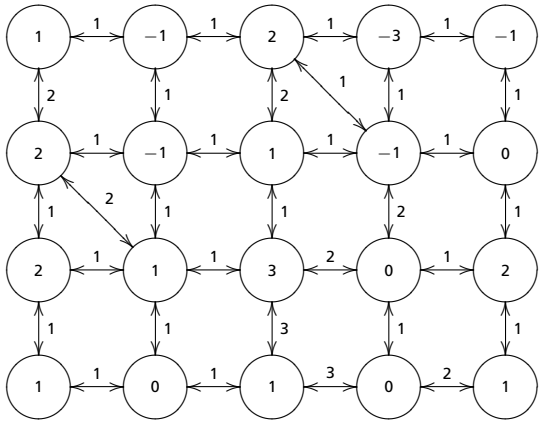
Section 2.2 on page 9 gave an introduction to dual decomposition. This section describes how the graph is split and how the dual variables enter the two subgraphs. The next two sections provide extensive experiments for graphs in 2, 3 and 4 dimensions, both multi-threaded and distributed across many computational nodes in a supercomputer.

7.2.1 Graph Cuts as a Linear Program

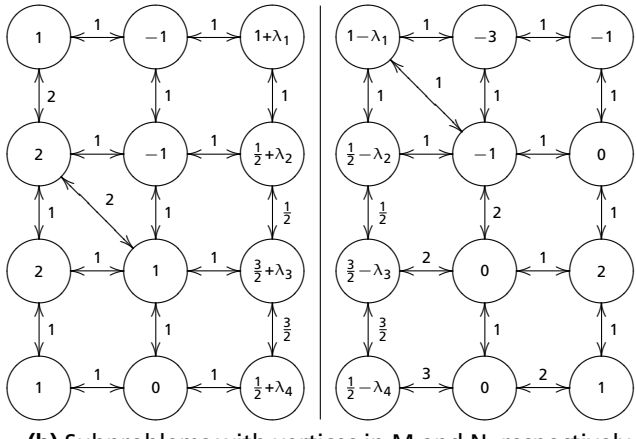
Finding the maximum flow, or, by duality, the minimum cut in a graph can be formulated as a linear program. Let $G = (V, \boldsymbol{c})$ be a graph where $V = \{s, t\} \cup \{1, 2, \ldots, n\}$ are the source, sink and vertices, respectively, and $\boldsymbol{c}$ the edge costs. A cut is a partition S, T of V such that $s \in S$ and $t \in T$. The minimum cut problem is finding the partition where the sum of all costs of edges between the two sets is minimal. It can be formulated as

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{minimize}} \quad & \sum_{i,j \in V} \boldsymbol{c}_{i,j} x_{i,j} \\ \text{subject to} \quad & x_{i,j} + x_i - x_j \geq 0, \quad i, j \in V \\ & x_s = 0, \quad x_t = 1, \quad \boldsymbol{x} \geq 0. \end{aligned} \tag{7.1}$$

The variable x_i indicates whether vertex i is part of S or T ($x_i = 0$ or 1, respectively) and $x_{i,j}$ indicates whether the edge (i, j) is cut or not. The variables are



(a) Original graph.



(b) Subproblems with vertices in M and N, respectively.

Figure 7.1: The graph decomposition into sets M and N. The pairwise energies in $M \cap N$ are part of both E_M and E_N and has to be weighted by 1/2. Four dual variables $\lambda_1 \ldots \lambda_4$ are introduced as s/t connections.

not constrained to be 0 or 1, but there always exists one such solution, according to the duality between maximum flow and minimum cut. Let $\mathcal{D}_V$ denote the convex set defined by the constraints in (7.1).

7.2.2 Splitting the Graph

Now pick two sets M and N such that $M \cup N = V$ and $\{s, t\} \subset M \cap N$. We assume that when $i \in M \setminus N$ and $j \in N \setminus M$, $\boldsymbol{c}_{i,j} = \boldsymbol{c}_{j,i} = 0$. That is, every edge is either within M or N, or within both. See Fig. 7.1.

We now observe that the objective function in (7.1) can be rewritten as:

$$\sum_{i,j \in V} \boldsymbol{c}_{i,j} x_{i,j} = \sum_{i,j \in M} \boldsymbol{c}_{i,j} x_{i,j} + \sum_{i,j \in N} \boldsymbol{c}_{i,j} x_{i,j} - \sum_{i,j \in M \cap N} \boldsymbol{c}_{i,j} x_{i,j}. \tag{7.2}$$

Define

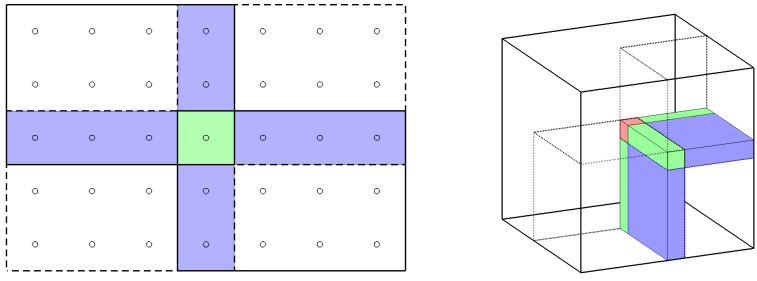
$$\begin{aligned} E_M(\boldsymbol{x}) &= \sum_{i,j \in M} \boldsymbol{c}_{i,j} x_{i,j} - \frac{1}{2} \sum_{i,j \in M \cap N} \boldsymbol{c}_{i,j} x_{i,j} \\ E_N(\boldsymbol{y}) &= \sum_{i,j \in N} \boldsymbol{c}_{i,j} y_{i,j} - \frac{1}{2} \sum_{i,j \in M \cap N} \boldsymbol{c}_{i,j} y_{i,j}. \end{aligned} \tag{7.3}$$

This leads to the following equivalent linear program:

$$\begin{aligned} \underset{\substack{\boldsymbol{x} \in \mathcal{D}_M \\ \boldsymbol{y} \in \mathcal{D}_N}}{\text{minimize}} \quad & E_M(\boldsymbol{x}) + E_N(\boldsymbol{y}) \\ \text{subject to} \quad & x_i = y_i, \quad i \in M \cap N. \end{aligned} \tag{7.4}$$

Here $\boldsymbol{x}$ is the variable belonging to the set M (left in Fig. 7.1b) and $\boldsymbol{y}$ belongs to N. The two variables $\boldsymbol{x}$ and $\boldsymbol{y}$ are constrained to be equal in the overlap. The dual function of this optimization problem is:

$$\begin{aligned} d(\boldsymbol{\lambda}) &= \min_{\substack{\boldsymbol{x} \in \mathcal{D}_M \\ \boldsymbol{y} \in \mathcal{D}_N}} \left(E_M(\boldsymbol{x}) + E_N(\boldsymbol{y}) + \sum_{i \in M \cap N} \lambda_i (x_i - y_i) \right) \\ &= \min_{\boldsymbol{x} \in \mathcal{D}_M} \left(E_M(\boldsymbol{x}) + \sum_{i \in M \cap N} \lambda_i x_i \right) \\ &+ \min_{\boldsymbol{y} \in \mathcal{D}_N} \left(E_N(\boldsymbol{y}) - \sum_{i \in M \cap N} \lambda_i y_i \right). \end{aligned} \tag{7.5}$$



(a) 2×2 **(b)** $2 \times 2 \times 2$

Figure 7.2: Splitting a graph into several components. The blue, green and red parts are weighted by 1/2, 1/4 and 1/8, respectively.

We now see that evaluating the dual function d amounts to solving two independent minimum cut problems. The extra unary terms $\lambda_i x_i$ are shown in Fig. 7.1b. Let $\boldsymbol{x}^*, \boldsymbol{y}^*$ be the solution to (7.4) and let $\boldsymbol{\lambda}^*$ maximize d. Because strong duality holds, we have $d(\boldsymbol{\lambda}^*) = E_M(\boldsymbol{x}^*) + E_N(\boldsymbol{y}^*)$ (Bertsekas, 1999). Each subproblem may in general have multiple solutions, so to obtain a unique solution we always set our optimal $\boldsymbol{x}^*$ and $\boldsymbol{y}^*$ equal to 1, wherever possible.

Splitting a graph into more than two components can be achieved with the same approach. The energy functions analogous to (7.3) might then contain terms weighted by 1/4 and 1/8, depending on the geometry of the split. See Fig. 7.2.

7.2.3 Implementation

Solving the original problem (7.4) amounts to finding the maximum value of the dual function. It follows from Lemma 2.1 that $x_i - y_i$, for $i \in M \cap N$, is a supergradient to g. In order to maximize d, the iterative scheme described in Section 2.2 can be used. This scheme requires the dual function to be evaluated many times. To make this efficient we reuse the search trees as described by Kohli and Torr (2007). Only a small part of the cost coefficients is changed between iterations and our experiments show that the subsequent max-flow computations can be completed within microseconds, see Table 7.1.

The step size τ needs to be chosen in each iteration. One possible choice is $\tau = 1/k$, where k is the current iteration number. For this particular application, we have found that this scheme and others appearing in the literature (Bertsekas, 1999; Komodakis, Paragios and Tziritas, 2007) are a bit too conservative for our

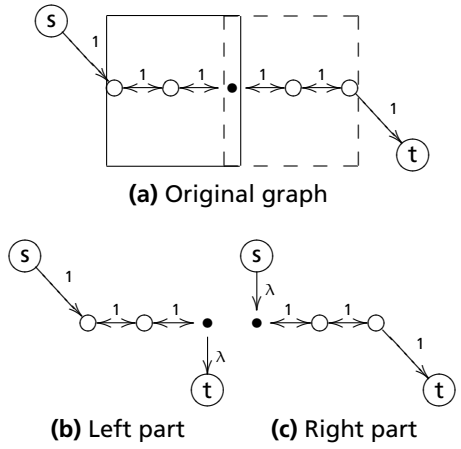


Figure 7.3: Convergence problem. The original graph (a) has multiple solutions, i.e., multiple minimum cuts. If $\lambda \neq 1$, the solutions for the two graphs (b) and (c) will not agree.

purposes. Instead of using a single step length τ, we associate each vertex in the overlap with its own step length τ_i. This is because different parts of the graph behave in different ways. In each iteration, we ideally want to choose λ_i so that $x_i = y_i$; therefore, if $x_i - y_i$ changed sign, then step length was too large and we should move in the opposite direction with a reduced step length.

foreach $i \in M \cap N$ **do**
  if $x_i - y_i \neq 0$ **then**
    $\lambda_i \leftarrow \lambda_i + \tau_i(x_i - y_i)$;
    if $x_i - y_i \neq$ *previous difference* **then**
      $\tau_i \leftarrow \tau_i/2$;
    end
  end
end

To handle cases like the one shown in Fig. 7.8, we also increase the step length if nothing happens between iterations. Empirical tests show that keeping an individual step length improves convergence speed for all graphs we tried. The extra memory requirements are insignificant.

Convergence Some graphs may have problems converging to the optimal solution. This can occur for graphs admitting multiple solutions. Fig. 7.3 shows an illustrative example. While the proposed scheme will converge toward $\lambda = 1$ for this graph, it will not reach it in a finite number of iterations. If $\lambda \neq 1$, the two partial solutions will not agree for the vertex marked black. In practice, we observed this phenomenon for a few pixels when processing large graphs with integer costs.

One possible solution is to add small, positive, random numbers to the edge costs of the graph. If the graph only has integer costs, this is not a problem. Increasing edge costs only increases the maximum flow, so the global maximum flow in the original graph is the integer part of the flow in the modified graph provided the sum of all values added is less than 1. However, there is an alternative way of handling graphs with integer costs.

Theorem 7.1. *If all the edge costs* $\boldsymbol{c}_{i,j}$ *are* even *integers, then there is an integer vector* $\boldsymbol{\lambda}$ *maximizing the dual function* (7.5).

Proof. The constraint sets $\mathcal{D}_M$ and $\mathcal{D}_N$ in (7.4) can be described by $\mathsf{A}\boldsymbol{z} \geq \mathbf{b}$, where A is an (integer) matrix, $\boldsymbol{z} = (\boldsymbol{x}, \boldsymbol{y})$ and $\mathbf{b}$ is an (integer) vector. Therefore, the optimization problem (7.4) can be written as:

$$\begin{aligned} \underset{\boldsymbol{z}}{\text{minimize}} \quad & \mathbf{a}^{\mathrm{T}}\mathbf{z} \\ \text{subject to} \quad & \mathsf{A}\mathbf{z} \geq \mathbf{b} \\ & x_i - y_i = 0, \quad i \in M \cap N. \\ & \boldsymbol{z} \geq 0. \end{aligned} \tag{7.6}$$

Here, $\mathbf{a}$ is an integral vector describing the objective function in (7.4). We also write $\mathsf{B}\boldsymbol{z} = 0$ for the equality constraints. Forming the dual function, we get:

$$\tilde{d}(\boldsymbol{w}, \boldsymbol{\lambda}) = \min_{\boldsymbol{z}} \left\{ \mathbf{a}^{\mathrm{T}}\boldsymbol{z} + \boldsymbol{w}^{\mathrm{T}}(\mathbf{b} - \mathsf{A}\mathbf{z}) + \boldsymbol{\lambda}^{\mathrm{T}}\mathsf{B}\boldsymbol{z} \right\}.$$

The dual is also a linear program,

$$\begin{aligned} \underset{\boldsymbol{w},\boldsymbol{\lambda}}{\text{maximize}} \quad & \mathbf{b}^{\mathrm{T}}\boldsymbol{w} \\ \text{subject to} \quad & \mathsf{A}^{\mathrm{T}}\boldsymbol{w} + \mathsf{B}^{\mathrm{T}}\boldsymbol{\lambda} \leq \mathbf{a} \\ & \boldsymbol{w} \geq 0. \end{aligned} \tag{7.7}$$

Since A is totally unimodular (TUM) (c.f. (Papadimitriou and Steiglitz, 1998, Section 13.2)), so is A^{T}. Also B is TUM and $\mathbf{a}$ is integral, and thus the dual function $\tilde{g}$ has an integer optimum $(\boldsymbol{w}^*, \boldsymbol{\lambda}^*)$. Since we have $d(\boldsymbol{\lambda}) = \max_{\boldsymbol{w} \geq 0} \tilde{d}(\boldsymbol{w}, \boldsymbol{\lambda})$, we are done. □

Remark. The theorem does not necessarily hold if the costs are integers. The graph $s \xrightarrow{\infty} \circ \xrightarrow{1} \circ \xrightarrow{1} t$, split at the second node, provides a counterexample. The subproblems are $s \xrightarrow{\infty} \circ \xrightarrow{1} \circ \xrightarrow{1/2+\lambda} t$ and $s \xrightarrow{\lambda} \circ \xrightarrow{1/2} t$. We have $d(0) = d(1) = 1/2$, but $d(1/2) = 1$.

For a general graph with integer costs split into two pieces, we multiply each edge by 2 and obtain an equivalent problem with an integer maximizer $\boldsymbol{\lambda}$. The graph may be split in more than two pieces in such a way that smaller costs than $1/2$ are used, as in Fig. 7.2. When setting up these problems we multiply every cost by 4 and 8, respectively, to ensure integer maximizers.

7.3 Experiments on a Single Machine

In this section we describe experiments performed in parallel on a single machine executed across multiple threads. We used the BK-method (v3.0 available for download) both for the single-threaded experiments and for solving the subproblems. All timings of the multi-threaded algorithm include any overhead associated with starting and stopping threads, allocation of extra memory etc. In all experiments we have only considered the time for actual maximum flow calculations, which means that the time required to construct the graphs is not taken into account. We note, however, that graph construction trivially benefits from parallel processing.

Image segmentation We applied our parallel method for the 301 images in the Berkeley segmentation database (Martin et al., 2001), see Fig. 7.4 for examples. The segmentation model used was a piecewise constant model (the pixel values were normalized to $[0, 1]$) with the boundary length as a regulating term:

$$E(\boldsymbol{x}) = \rho \sum_i \sum_{j \in \mathcal{N}(i)} w_{ij} |x_i - x_j| + \sum_i x_i \Big((I_i - c_1)^2 - (I_i - c_0)^2 \Big). \quad (7.8)$$

The boundary length is here approximated using a neighborhood $\mathcal{N}(i)$ of edges around each pixel, usually of sizes 4, 8 or 16 in the two-dimensional case. See

Cameraman (256×256)

Iteration	1	2	3	4	5
Number of Differences	36	17	3	3	0
Time (ms)	6.8	0.141	0.078	0.071	0.447

Tree (1152× 1536, shown in Fig. 7.7)

Iteration	1	2	3	4	5	···	8	9	10	11
Differences	108	105	30	33	16	···	16	9	9	0
Time (ms)	245	1.5	1.2	0.1	0.08	···	0.15	0.06	0.07	0.47

Table 7.1: The processing time for each iteration for two example images. The number of overlapping pixels (M ∩ N) was 256 and 1536, respectively (one column). Deallocating memory and terminating threads is the cause of the processing time increase in the last iteration. Note the short processing times after the first iteration due to the reuse of search trees.

(Boykov and Kolmogorov, 2003; Kolmogorov and Boykov, 2005) for details on how to choose w_{ij}. The overall influence of the boundary length is specified with the parameter ρ, where larger values usually correspond to harder optimization problems (longer s/t paths). The two constants c_1 and c_0 specify the intensity of the foreground and background. They were estimated in an alternating EM fashion for each image separately before the experiments.

The relative times ($t_{\text{multi-thread}}/t_{\text{single}}$) using two computational threads are shown in Figs. 7.5 and 7.6. Since the images in the database are quite small, the total processing time for a single image is around 10 milliseconds. Even with the overhead of creating threads and iterating to find the global minimum, we were able to get a significant speed improvement for almost all images.

Table 7.1 shows how the processing time varies with each iteration. In the last steps, very few vertices change and solving the maximum flow problems can therefore be done very quickly within microseconds.

It is very important to note that the problem complexity depends heavily on the amount of regularization used. That is, segmentation problems with low penalties on boundary length are easy to solve and parallelize. In the extreme case where no regularization is used, the problem reduces to simple thresholding, which of course is trivial to parallelize. Therefore, it is relevant to investigate how the algorithm performs with different amounts of regularization. We have done this and can conclude that our graph decomposition scheme performs well for a wide range of different settings, see Fig. 7.7. We see that the relative improvement in

Figure 7.4: Examples from the Berkeley database (Martin et al., 2001).

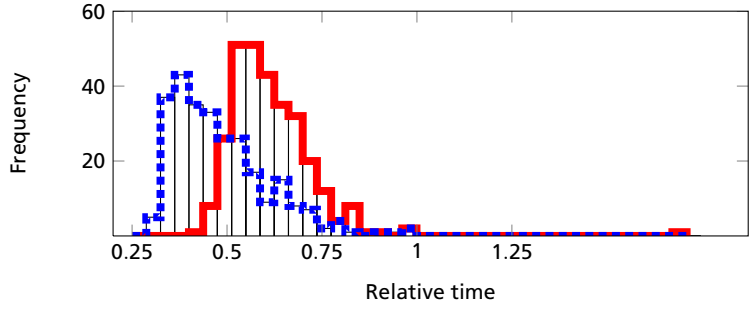


Figure 7.5: Relative times with 2 (red) and 4 (dotted blue) computational threads for the 301 images in the Berkeley segmentation database, using 4-connectivity. The medians are 0.596 and 0.455.

speed remains roughly constant over a large interval of different regularizations, whereas the absolute processing times vary by an order of magnitude.

When the number of computational threads increase, the computation times decrease as shown in Fig. 7.5.

Stereo problems The "Tsukuba" data set, which we obtained from University of Western Ontario (UWO), consists of a sequence of max-flow instances corresponding to the first iteration of α-expansions (Boykov, Veksler and Zabih, 1998). First, we solved the 16 problems without any parallelization and then, with two computational threads. The relative times ranged from 0.51 to 0.72, with the average being 0.61.

Three-dimensional graphs We used the graph construction described by Lempitsky and Boykov (2007) with data downloaded from UWO to evaluate the algorithm in three dimensions. For the "bunny" data set from UWO the relative time was 0.67 with two computational threads.

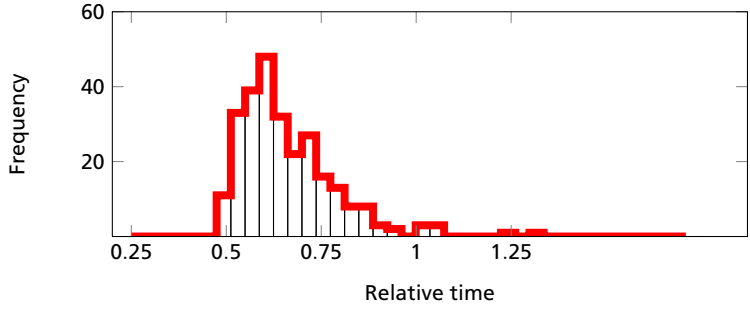


Figure 7.6: Relative times using 8-connectivity and 2 computational threads. The median is 0.628.

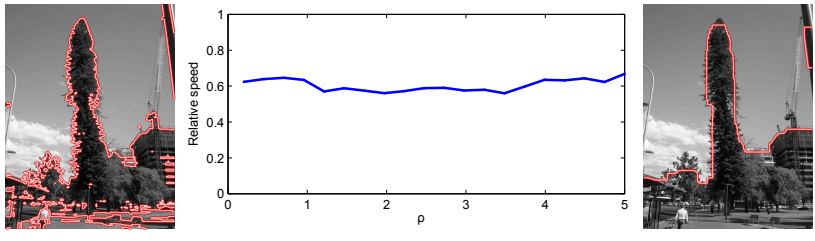


Figure 7.7: Relative improvement in speed with two computational threads when the regularization parameter changes. Although the processing time ranged from 230 ms to 4 seconds, the relative improvement was not affected.

Figure 7.8: "Worst-case" test. The left and right side of the image is connected to the source and sink, respectively. The edge costs are determined by the image gradient. All flow must be communicated between the two computational threads when splitting the graph vertically.

Limitations We have tried to explore the limitations of the method by examining cases with poor splits and poor speed-ups.

To see how our algorithm performs when the choice of split is very poor, we took a familiar image and split it in half from top to bottom as depicted in Fig. 7.8. We attached the leftmost pixel column to the source and the rightmost to the sink. Splitting horizontally would have been much more preferable, since splitting vertically severs every possible s-t path and all flow has to be communicated between the threads. Still, the parallel approach finished processing the graph 30% *faster* than the single-threaded approach. This is a good indication that the choice of the split is not crucial.

Figs. 7.5 and 7.6 contain a few examples ($< 1\%$) where the multi-threaded algorithm actually performs slower or almost the same as the single-threaded algorithm. The single example in Fig. 7.5 is interesting, because solving one of the subgraphs *once* takes significantly *longer* than solving the entire original graph. This can happen for the BK algorithm, but is very uncommon in practice. We have noted that slightly perturbing any of the problem parameters (regularization, image model, split position etc.) makes the multi-threaded algorithm faster also for this example.

The other slow examples have a simpler explanation: there is simply nothing

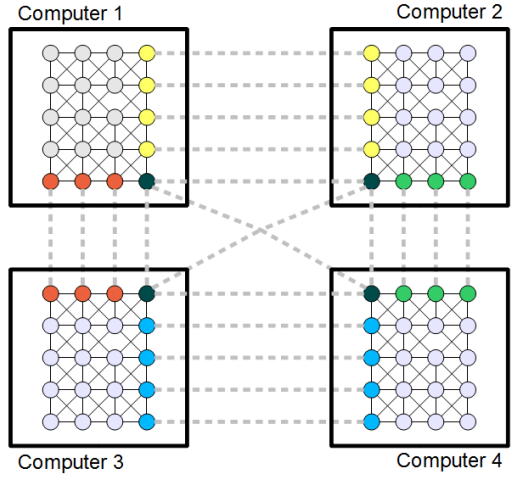


Figure 7.9: Splitting a graph across many computers. Colors indicate nodes that should be equal.

interesting going on in one of the two halves of the graph, see e.g. the first image in Fig. 7.4. Therefore, the overhead of creating and deallocating the threads and extra memory gives the multi-threaded algorithm a slight disadvantage. The approach by Liu and Sun (2010) (using smaller pieces) is better suitable for these graphs.

7.4 Splitting across Different Machines

We now turn to another application of graph decomposition. Instead of assigning each part of the graph to a computational thread, one may assign each subgraph to a different *machine* and let the machines communicate the flow over a network. Fig. 7.9 shows a diagram of the setup.

Memory is often a limiting factor for maximum flow calculations. Using splitting we were able to segment 4-dimensional (space+time) MRI heart data with $95 \times 98 \times 30 \times 19 = 5.3$M voxels. The connectivity used was 80, requiring 12.3 GB memory for the graph representation. By dividing this graph among 4 (2-by-2) different machines and using MPI (Snir and Otto, 1998) for communication, we were able to solve this graph in 1980 seconds. Since only a small amount of data (54 kB in this case) needs to be transmitted between machines each iteration,

this is an efficient way of processing large graphs. On the system we used[1], the communication time was about 7-10 ms per iteration, for a total of 68 iterations until convergence.

We also evaluated the algorithms for some of the big problems available from the University of Western Ontario. The largest version of the "bunny" data set is $401 \times 396 \times 312 = 50$M with 300M edges was solved in 7 seconds across 4 machines. As a reference, a slightly larger version of the same data set (not publicly available) was solved in over a minute with an (iterative) touch-and-expand approach in (Lempitsky and Boykov, 2007).

The largest data set we used was a $512 \times 512 \times 2317 = 607$M voxel CT scan with 6-connectivity. Storing this graph required 131 GB of memory divided among 36 ($3 \times 3 \times 4$) machines. We are not aware of any previous methods designed to handle graphs of this magnitude. The regularization used was low, which ensured convergence in 38 seconds with fairly even load distribution. Even with low regularization, the computation required 327 iterations.

Splitting graphs across multiple machines also saves computation time, even though the MPI introduces some overhead. For the small version of the "bunny" data set, a single machine solved the problem in 268 milliseconds, while two machines used 152 ms. Four machines (2-by-2) required 105 ms. For the medium sized version the elapsed times were 2.3, 1.34 and 0.84 seconds, respectively.

It should be noted that in many cases the BK algorithm is not the fastest possible choice, especially for graphs with higher dimensionality than 2 and connectivity greater than the minimum (Boykov and Kolmogorov, 2004). However, the method described in this chapter could just as easily be combined with a push-relabel algorithm better suited for graphs with 3 or 4 dimensions. Using a method optimized for grid graphs with fixed connectivity instead of the general BK would also reduce memory requirements significantly.

7.5 Conclusions

We have shown that it is possible to split a graph and obtain the global maximum flow by iteratively solving subproblems in parallel. Two applications of this technique were demonstrated:

- Faster maximum flow computations when multiple CPU cores are available

[1] LUNARC Iris, http://www.lunarc.lu.se/Systems/IrisDetails

(Section 7.3).

- The ability to handle graphs which are too big to fit in the computer's RAM, by splitting the graph across multiple machines (Section 7.4). This is in contrast to other approaches (Liu and Sun, 2010; Liu, Sun and Shum, 2009) where shared memory is required.

Good results were demonstrated even if the split severs many, or even all s-t paths of the graph (Fig. 7.8). Experiments with different amounts of regularization suggest that the speed-up is relatively insensitive to regularization (Fig. 7.7). The technique was applied to different graphs arising in computer vision. Our experiments included surface reconstruction, stereo estimation and image segmentation with two, three and four dimensional data. Methods based on push-relabel generally perform better than BK for large, high dimensional and highly connected graphs as discussed in (Boykov and Kolmogorov, 2004). Therefore, using our approach with push-relabel should be investigated in the future.

Chapter 8

Parallel Labeling on a GPU

I will now shift the focus from two-label problems to multi-label segmentation problems. The goal is to use dual decomposition for a highly parallel algorithm which can be run on hardware such as a field-programmable gate array (FPGA) or a graphics processing unit (GPU). The energy function to be minimized is

$$E(\boldsymbol{x}) = \sum_{i=1}^{n} T_i(x_i) + \sum_{i=1}^{n} \sum_{j \in \mathcal{N}_i} E_{ij}(x_i, x_j), \tag{8.1}$$

where $\boldsymbol{x} \in \mathcal{L} = \{1 \dots L\}^n$ and the functions T_i and E_{ij} are arbitrary. $T_i(y)$ denotes the cost of assigning label y to node i and $E_{ij}(y, z)$ is the cost of jointly assigning labels y and z to nodes i and j. The set $\mathcal{N}_i$ is the neighborhood of i, that is, all nodes that are directly dependent on i. This chapter focuses on 4-connectivity, but the ideas readily carry over to higher connectivities.

While this chapter does not offer any new theoretical insights, I will offer a practical way to parallelize previous approaches with good performance. The method me and my coauthors developed uses dual decomposition and dynamic programming similar to Komodakis, Paragios and Tziritas (2007), but our subproblems are simpler and therefore easily solved on massively parallel architectures. The potential for parallelization was not explored by Komodakis et al. and we show that these methods are useful to greatly increase the speed of algorithms for which other fast methods already are available, such as submodular minimum cut problems (Boykov and Kolmogorov, 2004).

GPU optimization Graphical processing units (GPUs) have been used extensively to facilitate the often very computationally expensive tasks in low-level vision. Segmentation with total variation models (e.g. problem (Q) considered in chapter 4 on page 38) has successively been implemented and can be performed in real

time (Pock et al., 2008*b*; Unger et al., 2008). Computing the optical flow between two images (Werlberger et al., 2009) and stereo estimation have also benefited greatly from the parallelization offered by a multi-processor GPU.

Solving discrete labeling problems such as minimum cuts and its generalizations on a GPU has proven to be harder. Often the speed-up is not great compared to algorithms that run on a CPU or the method might work well for only a restricted amount of regularization. In my experience, existing approaches do not work well, as mentioned in the previous chapter on page 83 regarding push-relabel. Fig 8.3 shows one of my experiments.

8.1 Splitting the Graph

The first thing to do is to split the energy function E in (8.1). The approach is simple: instead of considering a single graph, we consider two graphs, one of which contains all vertical connections and one contains all horizontal connections. The two new energy functions are

$$\begin{aligned} E_1(\boldsymbol{x}) &= \frac{1}{2}\sum_{i=1}^{n} T_i(x_i) + \sum_{i=1}^{n}\sum_{j\in\mathcal{H}_i} E_{ij}(x_i, x_j), \\ E_2(\boldsymbol{x}) &= \frac{1}{2}\sum_{i=1}^{n} T_i(x_i) + \sum_{i=1}^{n}\sum_{j\in\mathcal{V}_i} E_{ij}(x_i, x_j), \end{aligned} \tag{8.2}$$

where $\mathcal{H}_i$ and $\mathcal{V}_i$ denotes the horizontal and vertical neighbors of i, respectively. The factor $1/2$ is needed to have $E(\boldsymbol{x}) = E_1(\boldsymbol{x}) + E_2(\boldsymbol{x})$. Figure 8.1 illustrates this splitting of the energy function.

Having split the graph, we may solve the dual problem (2.7) on page 9. Not only are the two subproblems in (2.7) independent, but each also consists of many one-dimensional subproblems, completely independent of each other. Such (acyclic) one-dimensional problems can always be solved exactly in polynomial time using dynamic programming.

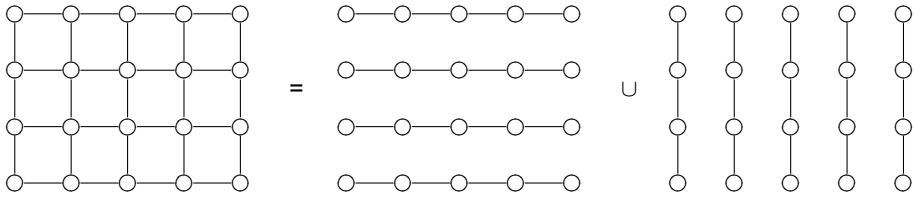


Figure 8.1: The graph decomposition. An edge between two nodes i and j indicate that j $\in \mathcal{N}_\text{i}$ so that the two nodes are directly dependent upon each other. Note that the two subproblems each consists of many independent one-dimensional problems.

8.2 Dynamic Programming

It is well-known that one-dimensional energy functions of the type (8.1) can be minimized exactly. We want to minimize

$$E(\boldsymbol{x}) = \sum_{i=1}^{n} T_i(x_i) + \sum_{i=1}^{n-1} E_i(x_i, x_{i+1}), \tag{8.3}$$

where $\boldsymbol{x} \in \mathcal{L} = \{1 \dots L\}^n$ and the functions T_i and E_i are arbitrary. $T_i(y)$ denotes the cost of assigning label y to node i and $E_i(y, z)$ is the cost of jointly assigning labels y and z to nodes i and $i + 1$. To solve this problem, we define $C_k(x)$ to mean be the lowest energy we are able to get if we assign $x_k = x$ and only count all nodes up to k, i.e.

$$C_k(y) \equiv \min_{x_1 \dots x_k = y} \left(\sum_{i=1}^{k} T_i(x_i) + \sum_{i=1}^{k-1} E_i(x_i, x_{i+1}) \right). \tag{8.4}$$

The following lemma describes how to compute $C_k(y)$ recursively over k:

Lemma 8.2.

$$C_k(y) = \begin{cases} T_1(y) & k = 1 \\ T_k(y) + \min_{z \in \mathcal{L}} \left(C_{k-1}(z) + E_{k-1}(z, y) \right) & k > 1. \end{cases}$$

Proof. We can compute $C_1(y) = T_1(y)$ directly from the definition. For any $k > 1$ we have

$$\begin{aligned} C_k(y) &= \min_{x_1 \dots x_k = y} \left(T_k(y) + E_{k-1}(x_{k-1}, y) + \sum_{i=1}^{k-1} T_i(x_i) + \sum_{i=1}^{k-2} E_i(x_i, x_{i+1}) \right) \\ &= T_k(y) + \min_{x_1 \dots x_{k-1} = z} \left(E_{k-1}(z, y) + \sum_{i=1}^{k-1} T_i(x_i) + \sum_{i=1}^{k-2} E_i(x_i, x_{i+1}) \right) \\ &= T_k(y) + \min_{z \in \mathcal{L}} \Big(C_{k-1}(z) + E_{k-1}(z, y) \Big). \end{aligned} \tag{8.5}$$

□

Lemma 8.2 gives an algorithm to efficiently compute $C_k(y)$ for all nodes k and labels y. An optimal labeling $\boldsymbol{x}^*$ can be extracted from this information. Assigning the optimal label to x_k can be done given $\boldsymbol{x}^*_{k+1}$:

Lemma 8.3.

$$\boldsymbol{x}^*_k = \begin{cases} \arg\min\limits_{y \in \mathcal{L}} \; C_n(y) & k = n \\ \arg\min\limits_{y \in \mathcal{L}} \; C_k(y) + E_k(y, \boldsymbol{x}^*_{k+1}). & k < n \end{cases}$$

Proof. The case $k = n$ is the definition of $C_n(y)$. If $k < n$ we observe that

$$\begin{aligned} \boldsymbol{x}^*_k &= \arg\min_{y \in \mathcal{L}} \; \min_{\boldsymbol{x} \in \mathcal{L},\, x_k = y} E(\boldsymbol{x}) \\ &= \arg\min_{y \in \mathcal{L}} \; \min_{\boldsymbol{x} \in \mathcal{L},\, x_k = y} \left(\sum_{i=1}^{n} T_i(x_i) + \sum_{i=1}^{n-1} E_i(x_i, x_{i+1}) \right) \\ &= \arg\min_{y \in \mathcal{L}} \; \min_{\boldsymbol{x} \in \mathcal{L},\, x_k = y} \Bigg(\sum_{i=1}^{k} T_i(x_i) + \sum_{i=1}^{k-1} E_i(x_i, x_{i+1}) \\ &\quad + \sum_{i=k+1}^{n} T_i(\boldsymbol{x}^*_i) + \sum_{i=k}^{n-1} E_i(\boldsymbol{x}^*_i, \boldsymbol{x}^*_{i+1}) \Bigg) \\ &= \arg\min_{y \in \mathcal{L}} \; \left(C_k(y) + \sum_{i=k+1}^{n} T_i(\boldsymbol{x}^*_i) + E_k(y, \boldsymbol{x}^*_{k+1}) + \sum_{i=k+1}^{n-1} E_i(\boldsymbol{x}^*_i, \boldsymbol{x}^*_{i+1}) \right) \\ &= \arg\min_{y \in \mathcal{L}} \; C_k(y) + E_k(y, \boldsymbol{x}^*_{k+1}). \end{aligned} \tag{8.6}$$

□

Lemmas 8.2 and 8.3 together give an efficient method of minimizing $E(\boldsymbol{x})$. The time required for arbitrary energies will be quadratic in the number of labels, which makes the method described in this chapter prohibitively slow for problems with a huge number of labels, such as problems arising in (Pritch, Kav-Venaki and Peleg, 2009).

8.3 Boolean Formulation and Updating of Weights

The problem formulation of minimizing (8.1) such that $\boldsymbol{x} \in \mathcal{L}$ imposes an ordering on the labels. This is because a supergradient to d is $\boldsymbol{x}^* - \boldsymbol{y}^*$ and this difference depends on the numerical values of the labels. To avoid this, we reformulate the optimization problem as a Boolean problem:

$$\bar{E}(\bar{\boldsymbol{x}}) = \sum_{i=1}^{n} \sum_{\ell \in \mathcal{L}} \bar{x}_{\ell,i} \cdot T_i(\ell) + \sum_{i=1}^{n} \sum_{j \in \mathcal{N}_i} \sum_{\ell_1 \in \mathcal{L}} \sum_{\ell_2 \in \mathcal{L}} \bar{x}_{\ell_1,i} \cdot \bar{x}_{\ell_2,j} \cdot E_{ij}(\ell_1, \ell_2), \tag{8.7}$$

where now $\bar{x}_{\ell,i} \in \{0,1\}$ and $\bar{x}_{\ell,i} = 1 \iff x_i = \ell$. Minimizing $\bar{E}$ is clearly seen to be equivalent to minimizing E. The constraint $x = y$ is reformulated as $\bar{x}_\ell = \bar{y}_\ell$, $\ell \in \mathcal{L}$ and each of these is associated with a dual variable $\bar{\lambda}_\ell$. The supergradient is according to Lemma 2.1

$$(\nabla \bar{d}(\bar{\lambda}_1, \ldots, \bar{\lambda}_\ell))_{\ell,i} = \bar{x}_{\ell,i} - \bar{y}_{\ell,i}. \tag{8.8}$$

If the two solutions disagree for node i, the dual variables will then have to be updated accordingly:

$$\bar{\lambda}_{\ell,i} \leftarrow \bar{\lambda}_{\ell,i} + \tau \cdot \begin{cases} 1 & \bar{x}_{\ell,i} = 1 \\ -1 & \bar{y}_{\ell,i} = 1 \\ 0 & \text{otherwise,} \end{cases} \tag{8.9}$$

where τ is the step length for the current iteration. Since the dual variables enter as multiplicative constants in front of $\{0,1\}$-variables, adding a number c to $\lambda_{\ell,i}$ is equivalent to adding c to $T_i(\ell)$. Therefore, the dual variables need not be stored explicitly and the data terms are instead modified directly.

8.3.1 Step Lengths

The step length rule we found to work best in practice was $\tau = C/k$, where k is the iteration number. To allow for data terms of different magnitudes, we normalized the step lengths to $\tau = m/(3k)$, where m is the maximum value of the data terms. If the data terms contain hard constraints with infinite cost weights, these will have to be excluded. We have also tested the primal-dual based rule used by Komodakis, Paragios and Tziritas (2007), but in this context it was slightly inferior. Different step size schemes were discussed in section 2.2.1 on page 11 and it was mentioned that no method working with supergradients alone can be very effective in general, and the problems in this chapter are very general indeed.

8.4 Linear Programming Relaxation

A very relevant question is whether the decomposed problem will solve the original problem. In general, this is too much to hope for, but how good will the solution be? This is answered by Komodakis, Paragios and Tziritas (in press). The combinatorial minimization problem of minimizing E can be relaxed to a linear programming problem. Problem (2.27) on page 18 is such a relaxation for the case $\mathcal{L} = \{0, 1\}$. For the one-dimensional subproblems the relaxation is always tight, and this fact can be used to show that the optimal value of the decomposed problem is equal to the value of the relaxed linear program. This is very similar to what we used to prove that dual decomposition always converges to the global optimum in the submodular Boolean case in the previous chapter.

8.5 Experiments

All experiments on the CPU were performed with an Intel Core2 Quad 2.5GHz processor (using a single core only) and for the GPU experiments we used an nVidia Tesla 2050. Results are given for the standard Potts model for which the regularizer is simply

$$E_{ij}(\ell_1, \ell_2) = \begin{cases} \rho & \ell_1 \neq \ell_2 \\ 0 & \ell_1 = \ell_2 \end{cases}. \tag{8.10}$$

If the number of labels $L = 2$, the exact solution can be computed as the minimum cut in an appropriate graph. If $L \geq 3$, we compare our method to two approximate

ρ	Min-cut time	CPU time	GPU time	rel. duality gap	iterations
10^4	0.0108s	0.0691s	0.042s	0.00024	71
10^5	0.0204s	0.0166s	0.01s	0.00017	16
10^6	0.4973s	0.0330s	0.017s	0.00046	26

Table 8.1: Boolean segmentation results for the 'cameraman' image, Figure 8.2. We compared the minimum cut algorithm (Boykov and Kolmogorov, 2004) (v.3.01) to a CPU and GPU implementation of the algorithm described in this chapter. We note that the dual decomposition algorithm seems to perform best using high regularizations.

ρ	Number of images	CPU Relative speed			CPU Absolute time (s)
		Min.	Median	Max.	Median
10^4	300	0.45	3.99	9.13	0.0120
$5 \cdot 10^4$	293	0.43	2.50	8.69	0.0240
10^5	279	0.35	1.55	4.18	0.0373
$5 \cdot 10^5$	163	0.07	0.31	1.77	0.1548
10^6	95	0.04	0.13	0.77	0.3483

Table 8.2: Binary segmentation results using (7.8) on page 90 for the images in the Berkeley segmentation data set (CPU, single thread). If the result was a trivial (constant) segmentation, we excluded that image, hence the lower number of images for higher regularization. Two facts can be seen from these figures: (i) that higher regularization results in far more difficult problems and that (ii) the row/column decomposition performs much better for those problems and those problems only.

methods: α-expansion (Boykov, Veksler and Zabih, 2001; Kolmogorov and Zabih, 2004; Boykov and Kolmogorov, 2004) and FastPD (Komodakis and Tziritas, 2007). The results are given in Tables 8.1–8.3 as well as Figures 8.2–8.5. We tried comparing to CUDA Cuts (Vineet and Narayanan, 2008), but the publicly available algorithm did not give the correct solution for any of the problems in Fig. 8.2. For $\rho = 10^5$ and 10^6 the algorithm produced a constant image.

We note that unlike the previous section (Fig. 7.7), we only observe a speed-up for problems where the amount of regularization is high.

When the number of labels increases, the time until convergence increases drastically. This is illustrated in Fig. 8.4. Furthermore, for general energies, the processing time required for a single iteration increases quadratically with the number of labels. These two facts make the method unattractive for a large number of labels and inferior to e.g. α-expansion.

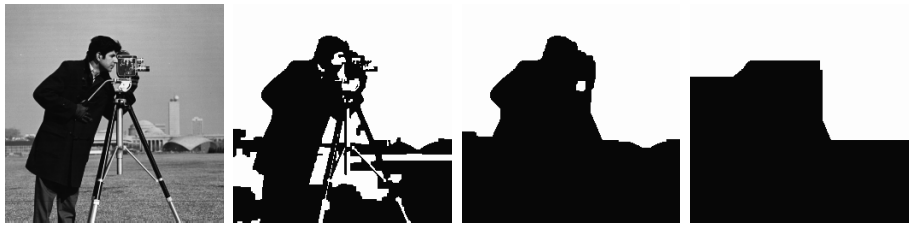


Figure 8.2: Boolean segmentation with different regularizations $\rho \in \{10^4, 10^5, 10^6\}$. See Table 8.1.

Figure 8.3: Result using CUDA Cuts (Vineet and Narayanan, 2008) with $\rho = 10^4$. The correct solution is shown in Fig 8.2. For $\rho = 10^5$ and 10^6 the algorithm produces a constant image.

Figure 8.4: Result using the Potts model with 64 labels. To the left the solution obtained by α-expansion is shown, followed by the row and column solutions of Fig. 8.1 after 500 iterations. Running 10000 iterations did not improve the solutions significantly.

ρ	α-expansion	FastPD	CPU time	GPU time	rel. gap	iters.
$5 \cdot 10^4$	0.348s	0.123s	0.149s	0.069s	0.00092	121
10^5	0.223s	0.148s	0.138s	0.064s	0.00054	111
10^6	0.500s	0.318s	0.086s	0.027s	0.000035	46

Table 8.3: Multi-label segmentation with Potts model, Figure 8.5. We used the implementation of α-expansion and FastPD from their authors, respectively.

Figure 8.5: Three class segmentation with the Potts model for different regularizations $\rho \in \{5 \cdot 10^4, 10^5, 10^6\}$, respectively. See Table 8.3.

8.6 Coordinate Ascent

I will end this chapter with a comment on the shape of the dual function d in the Boolean case where $\boldsymbol{x} \in \{0,1\}^n$. It will be useful to consider the dual function as a function of only one component of $\boldsymbol{\lambda}$, say corresponding to x_j, and keep all other components fixed: $\lambda_i = \boldsymbol{\mu}_i$ if $i \neq j$ and we write $\lambda_j = \lambda$. It is natural to consider the two parts of the dual function separately:

$$\begin{aligned} d_1(\boldsymbol{x}, \lambda) &= E_1(\boldsymbol{x}) + \sum_{i \neq j} \boldsymbol{\mu}_i x_i + \lambda x_j \\ d_2(\boldsymbol{y}, \lambda) &= E_2(\boldsymbol{y}) - \sum_{i \neq j} \boldsymbol{\mu}_i y_i - \lambda y_j \end{aligned} \tag{8.11}$$

and

$$\begin{aligned} d_1(\lambda) &= \min_{\boldsymbol{x} \in \{0,1\}^n} d_1(\boldsymbol{x}, \lambda) \\ d_2(\lambda) &= \min_{\boldsymbol{y} \in \{0,1\}^n} d_2(\boldsymbol{y}, \lambda). \end{aligned} \tag{8.12}$$

What do d_1 and d_2 look like? In fact, they are quite simple: Figures 8.6a and 8.6b show the shape of these functions and the following lemmas prove that this is the case.

Lemma 8.4. *d_1 is increasing.*

Proof. Let $\lambda' \geq \lambda$. We have that $d_1(\lambda) \leq d_1(\boldsymbol{x}', \lambda) \leq d_1(\boldsymbol{x}', \lambda')$ for any $\boldsymbol{x}'$. Taking the minimum gives $d_1(\lambda) \leq d_1(\lambda')$. □

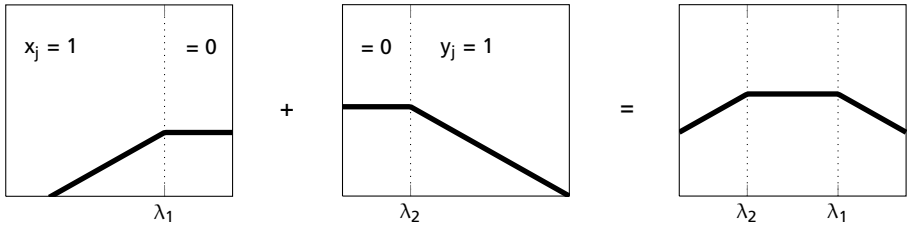


Figure 8.6: The dual function value as a function of one of the components in λ.

Lemma 8.5. *Let* $\lambda' > \lambda$. *If* $x_j = 0$ *for* some *solution* $\boldsymbol{x}$ *to the minimization problem* $d_1(\lambda)$*, then*

(a) $x'_j = 0$ *for* all *solutions* $\boldsymbol{x}'$ *to the problem* $d_1(\lambda')$ *and*

(b) $d_1(\lambda) = d_1(\lambda')$.

Proof. Let $\boldsymbol{x}$ be a solution to $d_1(\lambda)$ with $x_j = 0$. To prove statement (a), we assume that there is a solution $\boldsymbol{x}'$ to $d_1(\lambda')$ with $x'_j = 1$. Then $d_1(\boldsymbol{x}, \lambda) = d_1(\boldsymbol{x}, \lambda') \geq d_1(\boldsymbol{x}', \lambda') > d_1(\boldsymbol{x}', \lambda)$, which is a contradiction to the optimality of $\boldsymbol{x}$.

To prove (b), we observe $d_1(\lambda) = d_1(\boldsymbol{x}, \lambda) = d_1(\boldsymbol{x}, \lambda') \geq d_1(\lambda')$. Since d_1 is increasing this establishes that $d_1(\lambda) = d_1(\lambda')$. □

Lemma 8.6. *Let* $\lambda' < \lambda$. *If* $x_j = 1$ *for* some *solution* $\boldsymbol{x}$ *to* $d_1(\lambda)$*, then*

(a) $x'_j = 1$ *for* all *solutions* $\boldsymbol{x}'$ *to* $d_1(\lambda')$ *and*

(b) $d_1(\lambda') = d_1(\lambda) - (\lambda - \lambda')$.

Proof. We proceed in the same manner as in the previous lemma. Let the solutions to $d_1(\lambda')$ and $d_1(\lambda')$ be $\boldsymbol{x}$ and $\boldsymbol{x}'$, with $x_j = 1$ and $x'_j = 0$. Then $d_1(\boldsymbol{x}', \lambda) = d_1(\boldsymbol{x}', \lambda') \leq d_1(\boldsymbol{x}, \lambda') < d_1(\boldsymbol{x}, \lambda)$, which is a contradiction to the optimality of $\boldsymbol{x}$.

For (b), we have $d_1(\lambda) - (\lambda - \lambda') = d_1(\boldsymbol{x}, \lambda) - (\lambda - \lambda') = d_1(\boldsymbol{x}, \lambda') \geq d_1(\lambda')$. On the other hand, $d_1(\lambda) - (\lambda - \lambda') \leq d_1(\boldsymbol{x}', \lambda) - (\lambda - \lambda') \leq d_1(\boldsymbol{x}', \lambda')$ for any $\boldsymbol{x}' \in \{0, 1\}^n$. Taking the minimum gives the reverse inequality. □

The previous lemmas show that d_1 has at most one breakpoint and is constant for values larger than the breakpoint and increasing with slope 1 for values smaller. It is now easy to see that there must be exactly one breakpoint, by letting $\lambda \to \pm\infty$. Figure 8.6a shows the function $d_1(\lambda)$. The function d_2 is of course analogous, as shown in Fig. 8.6b. We have

Lemma 8.7. *d_2 is decreasing.*

Lemma 8.8. *Let $\lambda' < \lambda$. If $y_j = 0$ for* some *solution to $d_2(\lambda)$, then $y'_j = 0$ for* all *solutions to $d_2(\lambda')$.*

Lemma 8.9. *Let $\lambda' > \lambda$. If $y_j = 1$ for* some *solution to $d_2(\lambda)$, then $y'_j = 1$ for* all *solutions to $d_2(\lambda')$.*

Lemma 8.10. *Let $\lambda' \leq \lambda$. If $\boldsymbol{y}$ is a solution to $d_2(\lambda)$ with $y_j = 0$, then $d_2(\lambda) = d_2(\lambda')$.*

Lemma 8.11. *Let $\lambda' \geq \lambda$. If $\boldsymbol{y}$ is a solution to $d_2(\lambda)$ with $y_j = 1$, then $d_2(\lambda') = d_2(\lambda) - (\lambda' - \lambda)$.*

If we now consider the full dual function $d(\lambda) = d_1(\lambda) + d_2(\lambda)$ we know that it must be constant between the two breakpoints λ_1 and λ_2 and have slope ± 1 outside this interval, c.f. Fig. 8.6c. This holds both if $\lambda_1 > \lambda_2$ or the other way around. This gives an ascent method for the dual function. The breakpoints of d are easily obtained from the dynamic programming scheme seen previously in this chapter.

8.7 Conclusion

The method of splitting the graph used in the previous chapter is not suitable for massive parallelization. If a lot of parallelism is desired, the graph needs to be split in another way, preserving some of the long paths of the original graph. This chapter has proposed such a decomposition, with the additional benefit of being able to find approximate solutions for arbitrary energies of number of labels. While the idea of tree decompositions is not new, the possibilities for parallelism and GPU implementation have not been previously investigated. However, we conclude that in general, GPU parallelization of minimum cut algorithms for vision is still an unsolved problem. Implementing the dynamic programming schemes on a massively parallel architecture is non-trivial.

Part IV

Applications

Chapter 9

Multiple Regions for Heart Segmentation

This chapter will describe a method to simultaneously segment the myocardium and the left and the right ventricles of the human heart imaged by MRI. By segmenting all regions at the same time the different regions can influence each other to give a better result. A framework with geometric interactions (Delong and Boykov, 2009) is used, which for our model gives rise to non-submodular energies. We will use the now familiar subgradient methods to solve these optimization problems, instead of using QPBO, allowing us to save a substantial amount of memory. The purpose of this chapter is to give an overview of a current research project in medical image analysis and how the techniques used in previous chapters of this thesis applies to this problem. The principal researcher in this project is Johannes Ulén and I am grateful for our collaboration.

The heart below the atrioventricular plane is modeled to consist of five different regions as shown in Figure 9.1. Region 1 is the myocardium surrounding the two ventricles. Region 2 is the left ventricle and region 3 is the right ventricle. Region 4 is the papillary muscles inside the left ventricle along with other dark regions, and region 5 is the same for the right ventricle. We will be using the fact that both the left and right ventricle is contained inside the myocardium to our advantage.

9.1 Multi-Region Segmentation

The five regions of the heart have geometric relationships between each other. The framework introduced by Delong and Boykov (2009) is ideal for this type of model. The segmentation is found by finding the minimal value for a constructed energy function.

Let $\mathcal{L}$ be a set of region indices and let $\mathcal{P}$ be the set of voxels in an image. Each

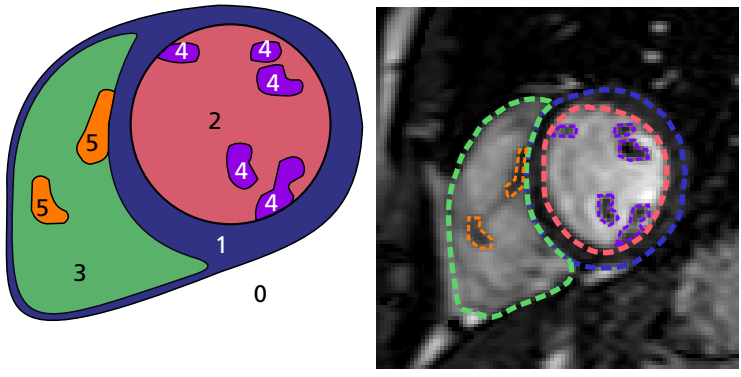


Figure 9.1: Left: A constructed short-axis view showing how the heart is modeled. Each region is numbered. Region 0 is the background, region 1 the myocardium. Region 2 is the left ventricle and region 3 the right ventricle. Region 4 is papillary muscles and other darker areas inside the left ventricle, and region 5 ditto for the right ventricle. **Right:** An example of a slice from a short-axis image acquired with MRI where all five regions have been delineated

voxel should be assigned a region $r_p \in \mathcal{L}$. We introduce $\boldsymbol{x} \in \{0,1\}^{|\mathcal{L}|\times|\mathcal{P}|}$. Here $\boldsymbol{x}$ is indexed as x_p^i with $i \in \mathcal{L}$ and $p \in \mathcal{P}$; $\boldsymbol{x}^i$ represents every Boolean variable for region i and $\boldsymbol{x}_p$ represents every Boolean variable for voxel p. In this way each voxel in the image is represented by $|\mathcal{L}|$ Boolean variables, making it possible to let the different regions influence each other. Table 9.1 shows the correspondence between $\boldsymbol{r}$ and $\boldsymbol{x}$.

When using standard graph-cuts the energy function is built the data term containing unary terms and the regularizing term containing pairwise interactions.

Region r_p	Boolean representation x_p
0	(0, 0, 0, 0, 0)
1	(1, 0, 0, 0, 0)
2	(1, 1, 0, 0, 0)
3	(1, 0, 1, 0, 0)
4	(1, 1, 0, 1, 0)
5	(1, 0, 1, 0, 1)

Table 9.1: Boolean representation of the 5 regions and background. All other possible values for the 5 Boolean variables are prevented by inclusion and exclusion constraints (Table 9.2).

i contains j		
x_p^i	x_q^j	W_{pq}^{ij}
0	0	0
0	1	∞
1	0	0
1	1	0

i excludes j		
x_p^i	x_q^j	W_{pq}^{ij}
0	0	0
0	1	0
1	0	0
1	1	∞

Table 9.2: Energy terms for different geometric interactions.

Delong and Boykov (2009) introduced a third kind of interaction – the so-called geometric interaction which associates a cost between Boolean variables representing the same voxel but different regions. The energy function to be minimized is:

$$E(\boldsymbol{x}) = \underbrace{D(\boldsymbol{x})}_{\text{data}} + \underbrace{V(\boldsymbol{x})}_{\text{regularizer}} + \underbrace{W(\boldsymbol{x})}_{\text{geometric interaction}} . \tag{9.1}$$

The data term associates a cost for each individual pixel to be part of any given region:

$$D(\boldsymbol{x}) = \sum_{p \in \mathcal{P}} \sum_{i \in \mathcal{L}} D_p^i(\boldsymbol{x}_p) . \tag{9.2}$$

The regularizing term associate a cost for two pixel p and q connected by some connectivity $\mathcal{N}$ to be part of the *same* region:

$$V(\boldsymbol{x}) = \sum_{i \in \mathcal{L}} \sum_{(p,q) \in \mathcal{N}} V_{p,q}^i\left(x_p^i, x_q^i\right) . \tag{9.3}$$

The geometric interaction is used to either repel or attract different regions to each other:

$$W(\boldsymbol{x}) = \sum_{\substack{i,j \in \mathcal{L} \\ i \neq j}} \sum_{p,q \in \mathcal{N}} W_{p,q}^{i,j}\left(x_p^i, x_q^j\right) . \tag{9.4}$$

In our model region 1 should contain both region 2 and region 3. This can be modeled with the use of geometric interaction terms, see Table 9.2.

The energy in (9.1) can be minimized exactly with graph-cuts as long as all energy terms are submodular. In our model we would like region 1 to contain

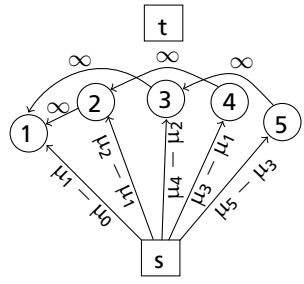


Figure 9.2: Graph construction for one voxel. The circled number corresponds to a vertex associated with the region number. The directed arrows are the directed edges in the graph. If either of the edges has a negative weight they are simply flipped to point to the sink with reversed cost. μ_i is the cost for the voxel to belong to region i.

region 2 and 3 and at the same time region 2 and 3 to be separated. Unfortunately this leads to a frustrated cycle and cannot be modeled by a submodular energy function (Delong and Boykov, 2009). One solution to this is to use the supermodular energy function and hope that the QPBO relaxation will be able to label all voxels. In our approach we chose another strategy explained in the next section. All interactions except the separation of region 2 and 3 can be introduced into a submodular graph, shown for one voxel in Figure 9.2.

9.2 Solving by Duality

A max-flow/min-cut problem can be reformulated into a Linear Program (LP):

$$\underset{\boldsymbol{x}\in\mathcal{D}}{\text{minimize}} \quad \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}, \tag{9.5}$$

where $\mathcal{D}$ is the set of points which fulfill the restrictions of the min-cut problem, $\boldsymbol{x}$ consists of the edges and vertices of the $s-t$ graph, and $\boldsymbol{c}$ is a cost matrix associated with the min-cut instance.

By our modeling there exists five different kinds of vertices $(\boldsymbol{x}^1,\ldots,\boldsymbol{x}^5)=\boldsymbol{x}$, where the super-index denote which label they model. The constraint we would like to enforce is that region 2 and regions 3 should never overlap. This is a linear

constraint, resulting in the the following LP:

$$\begin{aligned} \underset{\boldsymbol{x}\in\mathcal{D}}{\text{minimize}} \quad & \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} \\ \text{subject to} \quad & \boldsymbol{x}^2 + \boldsymbol{x}^3 \leq 1. \end{aligned} \tag{9.6}$$

Solving (9.6) with a standard LP solver is not tractable since the number of variables and constraints is very large. A better approach is to dualize the linear constraints in (9.6):

$$\underset{\boldsymbol{\lambda}\geq 0}{\text{maximize}} \quad d\left(\boldsymbol{\lambda}\right) \tag{9.7}$$

$$\text{where} \quad d\left(\boldsymbol{\lambda}\right) = \min_{\boldsymbol{x}\in\mathcal{D}} \Big(\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} + \boldsymbol{\lambda}^{\mathrm{T}} \left(\boldsymbol{x}^2 + \boldsymbol{x}^3 - 1 \right) \Big).$$

Let $d^\star$ denote the optimal solution to (9.7) and $p^\star$ the optimal solution to (9.6), by weak duality we then have that $d^\star \leq p^\star$. Problem (9.7) is solved using the projected supergradient method from section 2.2.2 on page 12. Just as in Chapter 7, evaluating d consists of solving a minimum cut problem.

9.3 Data Term

The data term is constructed from the probability of each voxel to be any of the five regions or the background. For voxel p and region i it is calculated as

$$\mu_r(p) = -\log \mathrm{P}\left(\boldsymbol{r}_p = r \,|\, \boldsymbol{v}_p\right). \tag{9.8}$$

The probability is a function of the voxel location p and the observed image intensity $\boldsymbol{v}_p$ at p. It needs to be estimated from training data. Having estimated μ_r, the costs D_p^i in (9.2) need to be constructed to reflect Table 9.1:

$$\begin{aligned} D_p^1(1) &= \mu_1(p) - \mu_0(p), \\ D_p^2(1) &= \mu_2(p) - \mu_1(p), \\ D_p^3(1) &= \mu_3(p) - \mu_1(p), \\ D_p^4(1) &= \mu_4(p) - \mu_2(p), \\ D_p^5(1) &= \mu_5(p) - \mu_3(p) \end{aligned} \tag{9.9}$$

and $D_p^i(0) = 0$ for all i and p. For example, region 4 is according to Table 9.1 represented as $\boldsymbol{x}_p = (1, 1, 0, 1, 0)$. The cost of this $\boldsymbol{x}_p$ is $(\mu_1(p) - \mu_0(p)) + (\mu_2(p) - \mu_1(p)) + (\mu_4(p) - \mu_2(p)) = \mu_4(p) - \mu_0(p)$.

9.4 Regularization

The regularization between two voxels p and q with intensity I_p and I_q is chosen as described by Boykov and Jolly (2001):

$$V_{p,q}^i(p,q) = \frac{\rho}{\text{dist}(p,q)} \exp\left(-\frac{(I_p - I_q)^2}{2\sigma^2}\right), \tag{9.10}$$

where ρ and σ can be used to tweak the regularization and should be estimated from training data. The neighborhood $\mathcal{N}$ is 18-connectivity (8 in-plane, 5+5 off-plane). Since MRI images have anisotropic resolution it is very important to take that into consideration when calculating the distance between voxels.

Distance prior The thickness of the myocardium surrounding left ventricle is proportional to the radius of the left ventricle in each short-axis slice. This can be included in the model by adding more edges of ∞ cost from region 2 to region 1 (Delong and Boykov, 2009). But adding all these edges will require too much memory. Instead, the algorithm is run once without the distance prior and a rough estimate of the left ventricle is obtained. From this estimate the radius is approximated and ∞-cost edges between region 2 and region 1 are added in a small circle around the initial approximation of the left ventricle.

Anisotropic regularization The rough estimate of region 2 is also used as an approximation of the center of the left ventricle in each slice. From the approximate center the regularization is modified to punish segmentation which do not result in a rounded edge of the ventricle. Another anisotropic punishment used is weight the regularization differently dependent on whether voxel p is brighter than q. In this way the segmentation can prefer to have a border between the left ventricle and the myocardium where the left ventricle is brighter than the myocardium.

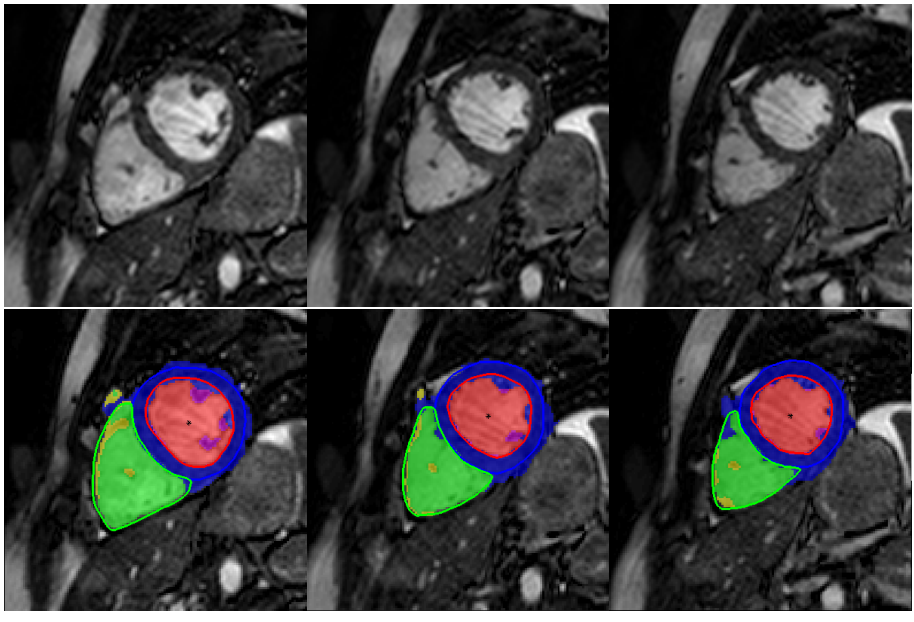


Figure 9.3: Three slices from a segmentation experiment. Top row shows the three original slices. The colored regions in the bottom row are the segmentation results and the colored lines are the ground-truth. The projected supergradient method required five iterations.

9.5 Experiments

Our preliminary experiments have been encouraging. The projected supergradient method often converges relatively fast. For the data set shown in Fig. 9.3, five iterations were enough to reach a relative duality gap of 0.0000705. The times required to solve the minimum cut problems were 15, 2, 1, 1 and 1 seconds, respectively. Subsequent problems require less time because flow can be reused between iterations (Kohli and Torr, 2007). QPBO found a solution to this problem in 31 seconds, with 144 unlabeled nodes. Even more important is the memory consumption, which for QPBO is approximately twice as large. This is a large issue in medical imaging, as previously discussed in Chapter 7.

Chapter 10

Shift-Map Image Registration

This chapter will describe an image processing application to multi-label optimization. It is based on previously published material (Svärm and Strandmark, 2010). I have added a comparison to the similar work by Liu et al. (2008), with which we were unfamiliar when we wrote the original paper.

Shift-map image processing was recently introduced by Pritch, Kav-Venaki and Peleg (2009) who applied their framework to image inpainting, content aware resizing, texture synthesis and image rearrangement. This chapter will extend the range of applications to image registration.

10.1 Problem Formulation

Registration can be performed using a parametric model, e.g. an affine or a projective transformation estimated from point correspondences between the two images. In this chapter, we consider a non-parametric model. We have a base image $B(i,j)$ and an input image $I(i,j)$. These two imaged need not have the same size. The goal is to register the pixels of the input image onto the base image using a shift-map $\boldsymbol{T}(i,j) = \Big(t_i(i,j) \;\; , \;\; t_j(i,j)\Big)$. The pixel $I(i,j)$ is registered onto $B\Big(i + t_i(i,j) \;\; , \;\; j + t_j(i,j)\Big)$. Figure 10.2 shows the input and base images and

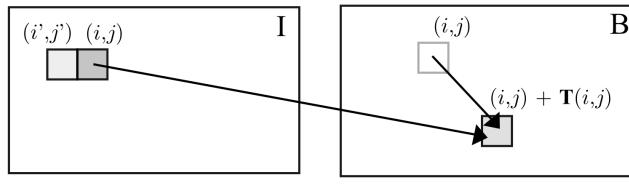
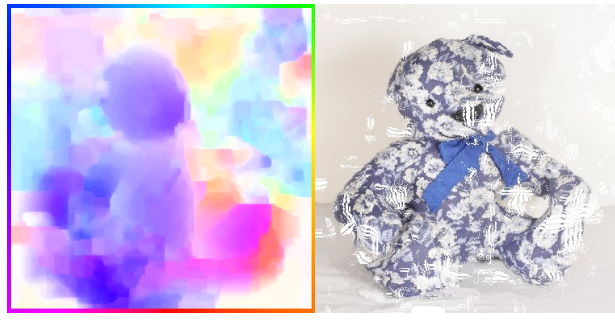
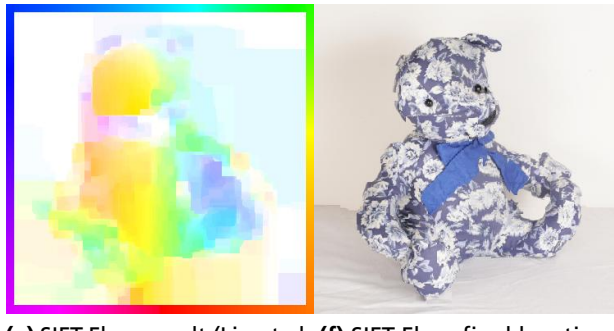


Figure 10.1: Shift-map between two images

(a) Input image I **(b)** Base image B



(c) Shift-map result **(d)** Final locations of pixels



(e) SIFT Flow result (Liu et al., 2008) **(f)** SIFT Flow final locations

Figure 10.2: Registration of two images using a shift-map. Each pixel in the input image is placed on the base image as described by the shift-map.

the resulting image obtained by moving all pixels in the input image as specified by the computed shift-map.

Each possible shift-map is assigned an energy, based on *a priori* assumptions on what a good shift-map typically looks like and how well the two images match each other. The goal is then to find the optimal shift-map, that is, the shift-map with the lowest energy:

$$E(\boldsymbol{T}) = \rho \sum_{\substack{1\leq i\leq m\\ 1\leq j\leq n}} E_d^{ij}(\boldsymbol{T}(i,j)) + \sum_{\substack{1\leq i\leq m\\ 1\leq j\leq n}} \sum_{(i',j')} E_s^{ij}(\boldsymbol{T}(i,j), \boldsymbol{T}(i',j')), \tag{10.1}$$

where the last summation refers to summations over all (i', j') in a neighborhood $\mathcal{N}(i, j)$ of (i, j). Figure 10.1 one such neighbor. We have used 4-connectivity of adjacent pixels throughout this chapter. E_d^{ij} and E_s^{ij} are the data terms and smoothness terms, respectively. They will be described in separate sections below.

10.2 Registration Energy Terms

The framework by Pritch, Kav-Venaki and Peleg (2009) deal with constructing a new image from an old one and the registration problem is about finding a map between two existing images. Hence the energy previously used for finding shift-maps is not suitable for registration and new energy terms must be constructed.

Comparison of pixels A related problem to image registration is dense depth estimation from two images of the same object with known camera positions. This problem has been studied extensively, see for example the work by Kolmogorov and Zabih (2006). Recently a new descriptor, DAISY, was proposed by Tola, Lepetit and Fua (2009), tailored to dense stereo estimation where the position of the two cameras differ by a large amount. This descriptor is shown to outperform other approaches (e.g. SIFT, SURF and pixel differences) in extensive experiments. Therefore, it seems relevant to try and apply this descriptor to the related problem of estimating a dense image registration.

Not unlike SIFT (Lowe, 2004), a DAISY descriptor samples the image derivative in different directions. Eight different directions and three different scales are used.

By sampling these fields at different points around the feature location, a descriptor of dimensionality 200 is obtained. Since the same fields are used for all image locations, a dense field of descriptors can be computed in a couple of seconds. The main goal of the DAISY descriptor was efficient dense computation. In order to choose relevant parameters, we found the work by Winder, Hua and Brown (2009) helpful.

Data terms The data terms E_d^{ij} were previously used by Pritch, Kav-Venaki and Peleg to enforce hard constraints on the shift-map. When inpainting an image, the data term makes sure no pixels in the "hole" are used in the output image by assigning such shifts a cost of ∞.

In this chapter, where image registration is considered, we need to develop more complex data terms to incorporate the fact that we want to find a mapping between two images such that similar pixels are mapped to similar pixels The data terms dictates that similar parts of the images should end up on top of each other. To measure similarity, dense DAISY is used.

It might only be possible to register parts of the input image, so shifting pixels outside the base image is permitted, at a constant cost P per pixel. The data terms are then given by

$$E_d^{ij}(\boldsymbol{T}) = \begin{cases} \left|\left|\hat{I}(i,j) - \hat{B}\Big((i,j) + \boldsymbol{T}(i,j)\Big)\right|\right|_2 \\ P \text{ when } (i,j) + \boldsymbol{T}(i,j) \text{ is outside B,} \end{cases} \tag{10.2}$$

where $\hat{I}(i,j)$ is the DAISY descriptor describing the image I at pixel location (i,j). If the shift takes pixel (i,j) outside the bounds of the base image, a constant cost is issued. Otherwise, dissimilarity of the pixels determines the cost of the assignment. Figure 10.6f shows a heat map of the distance from the circled feature in the first row to all locations in the image in row 2.

Smoothness terms The smoothness terms are used to enforce global consistency to the shift-map, while allowing discontinuities at a limited number of places. Pritch, Kav-Venaki and Peleg's (2009) smoothness terms compared the color and gradient pixel-wise. Where a discontinuity in the shift-map occurs, the penalty is computed as the difference in color and gradients. Our smoothness function takes the form of the Euclidean distance between the endpoints of the two shifts:

$$E_s^{ij}(\boldsymbol{T}(i,j), \boldsymbol{T}(i',j')) = ||(i',j') + \boldsymbol{T}(i',j') - (i,j) - \boldsymbol{T}(i,j)||_2. \tag{10.3}$$

Here, (i, j) and (i', j') are neighboring pixels, see (10.1). Using the shift difference $||\boldsymbol{T}(i', j') - \boldsymbol{T}(i, j)||_2$ will penalize smoothly varying shift-maps too much, and hence it is important to compare the end points (as in (10.3)).

Color information The DAISY descriptor does not use color information, yet intuitively it makes little sense to match pixels of very different colors. Because of this, we have also made experiments where the color information of the images are incorporated into the above data terms. The color model used assigned a cost of P to pixels with large difference in hue, given that the intensity and saturation allowed a reliable value of the hue. This model improved the result of the registration in Fig. 10.6. We did not use color information in the experiment shown in Fig. 10.2.

10.3 Experiments

To minimize the energy (10.1), we used α-expansion[1] (section 2.6 on page 19). Each possible shift value $\boldsymbol{T}(i, j) \in \{-m \dots m\} \times \{-n \dots n\}$ is mapped to a 1D label space. The number of labels for even moderately sized images then becomes very large. In order to make it tractable, a Gaussian pyramid was used. For the images in Fig. 10.6, an initial size of 128×23 was used. The size was doubled 3 times until the final resolution of 1024×179 was reached. Each doubling of the image size is followed by a linear interpolation of the shift-map. This shift-map was used as a starting guess for the optimization at the larger level. At each level after the first, only 9 possible shifts then need to be considered: $\{-1, 0, 1\}$ in each direction.

To verify our implementation, we inpainted an example image used by Pritch, Kav-Venaki and Peleg (2009), see Fig. 10.4. We tried to follow their implementation as closely as possible and got different, but qualitatively similar results. We did not allow the pixels outside the area to be removed to move at all, which is in contrast to (Pritch, Kav-Venaki and Peleg, 2009), where all pixels except the border of the image were allowed to be shifted.

Figures 10.2 and 10.3 show shift-map registration results. The bear image in Fig. 10.2 shows the same object from two different views and is from (Kushal and Ponce, 2006). The building images in 10.3 register correctly, except for the light pole, which is very thin and does not have a large enough data term.

[1] α-expansion requires every pairwise energy terms to be a metric. Pritch, Kav-Venaki and Peleg (2009) truncated every term not satisfying this constraint. We follow this approach.

We have also conducted an experiment where we used shift-maps to recover a known image deformation. The results are displayed in Fig. 10.5.

During large-scale reconstruction of a city using images taken with a cylindrical camera (Hitta.se street view), we have encountered many difficult image pairs where SIFT is unable to provide useful correspondences. The top two rows in Fig. 10.6 show one of the hardest. Computed SIFT features for the two images (794 and 1019 feature points, respectively) only yielded 3 correct matches. The main reason for this was the image geometry and large, repetitive patterns. Using shift-map we obtained a dense, mostly correct map between the images. This was then used as an aid to compute SIFT correspondences. We then obtained 28 matches, of which 12 were correct. The run time for this image was about 2 minutes.

We compared shift-map registration to the recent SIFT Flow algorithm (Liu et al., 2008). Both algorithms worked well for simple distortions of small magnitudes, which can be seen in Fig. 10.5. However, for our other experiments, we were not able to get any satisfactory results using SIFT Flow. An example is shown in Fig. 10.2

10.4 Conclusion and Further Work

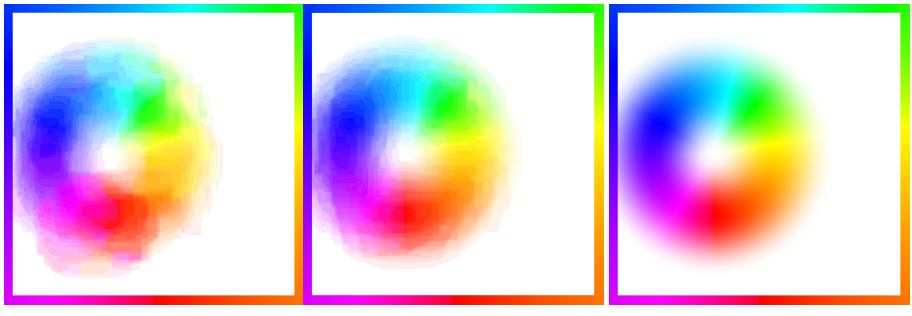
We have studied the application of shift-maps to image registration. Computing the smoothness term with color and gradient differences as done by Pritch, Kav-Venaki and Peleg (2009) did not give satisfactory results when extended to image registration, but we found a great improvement with the dense DAISY descriptor. For relatively easy cases (Figs. 10.2 and 10.3), we obtained very good results. For very hard cases (Fig. 10.6) we obtained results which proved very useful for obtaining correspondences between the images. One interesting future line of work would be to investigate whether shift-map inpainting can be improved by the DAISY descriptor as well. We have also not investigated large rotations in this chapter, which would require additional considerations.

(a) input image I **(b)** base image B **(c)** final location of pixels

Figure 10.3: Registration of two images of a building.

Figure 10.4: Our reimplementation of the inpainting algorithm by Pritch, Kav-Venaki and Peleg (2009). The complete running time for this example was 3.1415 seconds (pure coincidence!).

(a) Base image **(b)** Input image



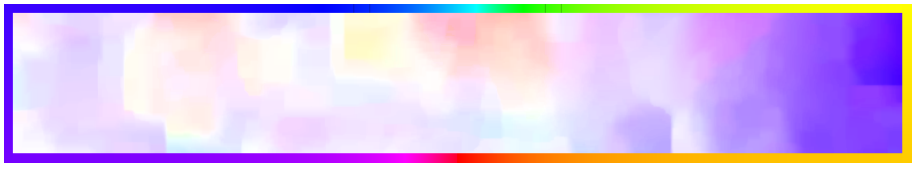
(c) Shift-map result **(d)** SIFT Flow result **(e)** Ground truth

Figure 10.5: Recovering a known image distortion. The maximum and mean error for the shift-map estimation was 7.3 and 0.7 pixels, respectively. Photo by Tristan Savatier obtained through Flickr.

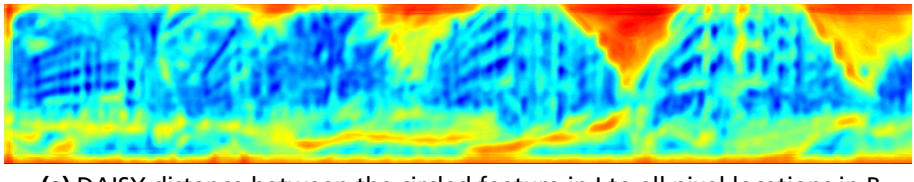
(a) Input image I

(b) Base image B

(c) Final locations of the pixels in I



(d) Resulting shift-map



(e) DAISY distance between the circled feature in I to all pixel locations in B

Figure 10.6: Registration of 1024×179 Hitta images. We note that we achieved a dense, highly nonlinear registration. This shift-map allowed us to obtain useful point-correspondences between the images, which was not possible using SIFT alone.

Bibliography

Agarwal, S., N. Snavely, I. Simon, S. M. Seitz and R. Szeliski. 2009. Building Rome in a Day. In *Int. Conf. Computer Vision*. Cited on page 2.

Bertsekas, D. P. 1999. *Nonlinear programming*. Athena Scientific. Cited on pages 7, 9, 11 and 87.

Bhusnurmath, A. and C. J. Taylor. 2008. "Graph Cuts via ℓ_1 Norm Minimization." *IEEE Trans. Pattern Analysis and Machine Intelligence* 30(10):1866–1871. Cited on pages 83 and 84.

Billionnet, A. and M. Minoux. 1985. "Maximizing a supermodular pseudoboolean function: A polynomial algorithm for supermodular cubic functions." *Discrete Applied Mathematics* 12(1):1 – 11. Cited on page 17.

Boros, E. and P. L. Hammer. 2002. "Pseudo-boolean optimization." *Discrete Applied Mathematics* 123:155–225. Cited on pages 17 and 19.

Boros, E., P. L. Hammer and G. Tavares. 2006. Preprocessing of unconstrained quadratic binary optimization. Technical report RUTCOR RRR 10-2006. Cited on page 19.

Boros, E., P. L. Hammer and X. Sun. 1991. Network flows and minimization of quadratic pseudo-boolean functions. Technical report RUTCOR RRR 17-1991. Cited on page 19.

Boyd, S. and L. Vandenberghe. 2004. *Convex Optimization*. Cambridge University Press. Cited on pages 7, 9 and 49.

Boykov, Y. and M. Jolly. 2001. Interactive graph cuts for optimal boundary and region segmentation of objects in ND images. In *Int. Conf. Computer Vision*. Vol. 1 pp. 105–112. Cited on page 118.

Boykov, Y., O. Veksler and R. Zabih. 1998. Markov Random Fields with Efficient Approximations. In *Conf. Computer Vision and Pattern Recognition*. Cited on pages 81 and 92.

Boykov, Y., O. Veksler and R. Zabih. 2001. "Fast approximate energy minimization via graph cuts." *IEEE Trans. Pattern Analysis and Machine Intelligence* 23(11):1222–1239. Cited on pages 20, 23 and 105.

Boykov, Y. and V. Kolmogorov. 2003. Computing Geodesics and Minimal Surfaces via Graph Cuts. In *Int. Conf. Computer Vision*. Cited on page 91.

Boykov, Y. and V. Kolmogorov. 2004. "An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision." *IEEE Trans. Pattern Analysis and Machine Intelligence* 26(9):1124–1137. Cited on pages 17, 24, 81, 82, 83, 96, 97, 99 and 105.

Bresson, X., S. Esedoḡlu, P. Vandergheynst, J. Thiran and S. Osher. 2007. "Fast Global Minimization of the Active Contour/Snake Model." *Journal of Mathematical Imaging and Vision* 28(2):151–167. Cited on page 35.

Bruckstein, A. M., A. N. Netravali and T. J. Richardson. 2001. "Epi-convergence of discrete elastica." *Applicable Analysis, Bob Caroll Special Issue* 79:137–171. Cited on pages 58 and 63.

Carr, P. and R. Hartley. 2009. Minimizing energy functions on 4-connected lattices using elimination. In *Int. Conf. Computer Vision*. Cited on page 81.

Chambolle, A. 2004. "An Algorithm for Total Variation Minimization and Applications." *Journal of Mathematical Imaging and Vision* 20:89–97. Cited on pages 40 and 51.

Chambolle, A. 2005. Total Variation Minimization and a Class of Binary MRF Models. In *EMMCVPR05*. pp. 136–152. Cited on pages 33, 38, 39, 40 and 51.

Chan, T. and L. Vese. 2001. Active contours without edges. In *IEEE Transactions on Image Processing*. Vol. 10 pp. 266–277. Cited on pages 33 and 35.

Chan, T., S. Esedoglu and M. Nikolova. 2006. "Algorithms for Finding Global Minimizers of Image Segmentation and Denoising Models." *SIAM Journal on Applied Mathematics* 66(5):1632–1648. Cited on pages 33 and 35.

Coyne, J. A. 2009. *Why evolution is true*. Oxford University Press. Cited on page 1.

Cremers, D. and S. Soatto. 2005. "Motion Competition: A variational framework for piecewise parametric motion segmentation." *Int. Journal Computer Vision* 63(3):249–265. Cited on page 36.

Darbon, J. 2007. A Note on the Discrete Binary Mumford-Shah Model. In *MIRAGE*, ed. André Gagalowicz and Wilfried Philips. Vol. 4418 of *Lecture Notes in Computer Science* Springer pp. 283–294. Cited on page 36.

Delong, A. and Y. Boykov. 2008. A Scalable graph-cut algorithm for N-D grids. In *Conf. Computer Vision and Pattern Recognition*. Cited on page 83.

Delong, A. and Y. Boykov. 2009. Globally Optimal Segmentation of Multi-Region Objects. In *Int. Conf. Computer Vision*. Cited on pages 21, 113, 115, 116 and 118.

Dixit, N., R. Keriven and N. Paragios. 2005. GPU-Cuts: Combinatorial Optimisation, Graphic Processing Units and Adaptive Object Extraction. Technical Report 05-07 CERTIS. Cited on page 83.

El-Zehiry, N. and L. Grady. 2010. Fast Global Optimization of Curvature. In *Conf. Computer Vision and Pattern Recognition*. Cited on pages 55, 56, 62, 70 and 71.

Everett, H. 1963. "Generalized Lagrange Multiplier Method for Solving Problems of Optimum Allocation of Resources." *Operations Research* 11:399–417. Cited on page 9.

Goldberg, A V and R E Tarjan. 1986. A new approach to the maximum flow problem. In *STOC '86: Proceedings of the eighteenth annual ACM symposium on Theory of computing*. New York, NY, USA: ACM pp. 136–146. Cited on page 83.

Goldschlager, L. M., R. A. Shaw and J. Staples. 1982. "The Maximum Flow Problem is Log Space Complete for P." *Theoretical Computer Science* 21(1):105–111. Cited on page 82.

Grady, L. 2010. "Minimal Surfaces Extend Shortest Path Segmentation Methods to 3D." *IEEE Trans. on Pattern Analysis and Machine Intelligence* 32(2):321–334. Cited on pages 56 and 74.

Grady, L. and C. Alvino. 2008. Reformulating and Optimizing the Mumford-Shah Functional on a Graph - A Faster, Lower Energy Solution. In *European Conf. Computer Vision*. Marseille, France: pp. 248–261. Cited on page 36.

Greig, D. M., B. T. Porteous and A. H. Seheult. 1989. "Exact Maximum A Posteriori Estimation for Binary Images." *Journal of the Royal Statistical Society* . Cited on pages 23, 24, 25, 26 and 81.

Hales, T. C. 2001. "The Honeycomb Conjecture." *Discrete & Computational Geometry* 25(1):1–22. Cited on page 63.

Hammer, P. L., P. Hansen and Simeone. 1984. "Roof duality, complementation and persistency in quadratic 0-1 optimization." *Mathematical programming* 28:121–155. Cited on page 19.

Hsu, Lucas, Rob Kusner and John Sullivan. 1992. "Minimizing the squared mean curvature integral for surfaces in space forms." *Experimental Mathematics* 1:191–207. Cited on page 73.

Hussein, M., A. Varshney and L. Davis. 2007. On Implementing Graph Cuts on CUDA. In *Workshop on General Purpose Processing on Graphics Processing Units*. Cited on page 83.

IEEE 11th International Conference on Computer Vision, ICCV 2007, Rio de Janeiro, Brazil, October 14-20, 2007. 2007. IEEE. Cited on page 136.

Ishikawa, H. 2009. Higher-Order Clique Reduction in Binary Graph Cut. In *Conf. Computer Vision and Pattern Recognition*. Cited on page 63.

Ivănescu, P. L. (Hammer). 1965. "Some Network Flow Problems Solved with Pseudo-Boolean Programming." *Operations Research* 13(3):388–399. Cited on page 17.

Kanizsa, G. 1971. "Contours without gradients or cognitive contours." *Italian Jour. Psych.* 1:93–112. Cited on page 55.

Kawai, N., T. Sato and N. Yokoya. 2009. Efficient surface completion using principal curvature and its evaluation. In *Int. Conf. Image Processing*. pp. 521 –524. Cited on page 56.

Kirsanov, D. and S. J. Gortler. 2004. A Discrete Global Minimization Algorithm for Continuous Variational Problems. Technical Report TR-14-04 Harvard. Cited on page 64.

Klodt, M., T. Schoenemann, K. Kolev, M. Schikora and D. Cremers. 2008. An Experimental Comparison of Discrete and Continuous Shape Optimization Methods. In *European Conf. Computer Vision*. Cited on pages 36 and 84.

Kohli, P. and P Torr. 2007. "Dynamic graph cuts for efficient inference in markov random fields." *IEEE Trans. Pattern Analysis and Machine Intelligence* 29(12):2079–2088. Cited on pages 82, 87 and 120.

Kohli, Pushmeet and Philip H. S. Torr. 2005. Efficiently Solving Dynamic Markov Random Fields Using Graph Cuts. In *Int. Conf. Computer Vision*. Cited on page 82.

Kolev, K. and D. Cremers. 2009. Continuous Ratio Optimization via Convex Relaxation with Applications to Multiview 3D Reconstruction. In *Conf. Computer Vision and Pattern Recognition*. Cited on page 47.

Kolmogorov, V. and R. Zabih. 2002. What energy functions can be minimized via graph cuts? In *European Conf. Computer Vision*. Cited on page 17.

Kolmogorov, V. and R. Zabih. 2004. "What energy functions can be minimized via graph cuts?" *IEEE Trans. Pattern Analysis and Machine Intelligence* 26(2):147–159. Cited on pages 17, 20, 23, 81 and 105.

Kolmogorov, V. and R. Zabih. 2006. Graph Cut Algorithms for Binocular Stereo with Occlusions. In *Handbook of Mathematical Models in Computer Vision*. Springer. Cited on pages 28 and 123.

Kolmogorov, V. and Y. Boykov. 2005. What Metrics Can Be Approximated by Geo-Cuts, Or Global Optimization of Length/Area and Flux. In *Int. Conf. Computer Vision*. Cited on page 91.

Kolmogorov, V., Y. Boykov and C. Rother. 2007. Applications of parametric maxflow in computer vision. In *Int. Conf. Computer Vision*. Cited on pages 36, 47, 48 and 51.

Komodakis, N. and G. Tziritas. 2007. "Approximate Labeling via Graph Cuts Based on Linear Programming." *IEEE Trans. Pattern Analysis and Machine Intelligence* 29(8):1436–1453. Cited on pages 81 and 105.

Komodakis, N., N. Paragios and G. Tziritas. 2007. MRF Optimization via Dual Decomposition: Message-Passing Revisited. in *Int. Conf. Computer Vision* (*IEEE 11th International Conference on Computer Vision, ICCV 2007, Rio de Janeiro, Brazil, October 14-20, 2007*, 2007). Cited on pages 5, 9, 81, 87, 99 and 104.

Komodakis, N., N. Paragios and G. Tziritas. in press. "MRF Energy Minimization and Beyond via Dual Decomposition." *IEEE Trans. Pattern Analysis and Machine Intelligence* . Cited on page 104.

Kushal, A. and J. Ponce. 2006. Modeling 3D objects from stereo views and recognizing them in photographs. In *European Conf. Computer Vision*. Cited on page 125.

Lempitsky, V., A. Blake and C. Rother. 2008. Image Segmentation by Branch-and-Mincut. In *European Conf. Computer Vision*. Marseille, France: pp. 15–29. Cited on page 36.

Lempitsky, V., C. Rother, S. Roth and A. Blake. 2009. "Fusion Moves for Markov Random Field Optimization." *IEEE Trans. Pattern Analysis and Machine Intelligence* . To appear. Cited on page 84.

Lempitsky, V. and Y. Boykov. 2007. Global Optimization for Shape Fitting. In *Conf. Computer Vision and Pattern Recognition*. Minneapolis, USA: . Cited on pages 81, 92 and 96.

Lindgren, F. 2006. *Image Modelling and Estimation*. Lund University. Available at `http://www.maths.lth.se/matstat/kurser/fms150mas228/book/`. Cited on pages 23, 24 and 26.

Liu, C., J. Yuen, A. Torralba, J. Sivic and W. T. Freeman. 2008. SIFT Flow: Dense Correspondence across Different Scenes. In *European Conf. Computer Vision*. Cited on pages 121, 122 and 126.

Liu, J. and J. Sun. 2010. Parallel Graph-cuts by Adaptive Bottom-up Merging. In *Conf. Computer Vision and Pattern Recognition*. San Francisco, USA: . Cited on pages 83, 95 and 97.

Liu, J., J. Sun and H.-Y. Shum. 2009. "Paint selection." *ACM Transactions on Graphics* 28(3):1–7. Cited on pages 83 and 97.

Lowe, D. 2004. "Distinctive image features from scale-invariant keypoints." *Int. Journal Computer Vision* 20:91–110. Cited on page 123.

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 *Int. Conf. Computer Vision*. Cited on pages 44, 90 and 92.

Masnou, S. 2002. "Disocclusion: A variational approach using level lines." *IEEE Transactions on Image Processing* 11:68–76. Cited on page 56.

Middleton, L. and J. Sivaswamy. 2005. *Hexagonal Image Processing: A Practical Approach*. Springer-Verlag New York, Inc. Cited on page 63.

Mumford, D. and T. Shah. 1989. Optimal Approximation by Piecewise Smooth Functions and Associated Variational Problems. In *Comm. on Pure and Applied Mathematics*. Cited on pages 3, 33 and 35.

Nesterov, Y. 2004. *Introductory lectures on convex optimization: A basic course*. Kluwer Academic Publishers. Cited on pages 7, 8 and 12.

Nilsson, D.-E. 1999. "Vision Optics and Evolution." *BioScience* 39(5):298–307. Cited on page 1.

Olsson, C., M. Byröd, N. C. Overgaard and F. Kahl. 2009. Extending Continuous Cuts: Anisotropic Metrics and Expansion Moves. In *International Conference on Computer Vision*. Cited on page 4.

Papadimitriou, C. H. and K. Steiglitz. 1998. *Combinatorial Optimization; Algorithms and Complexity*. Dover Publications. Cited on page 90.

Pock, T. 2008. Fast Total Variation for Computer Vision PhD thesis Graz University of Technology. Cited on page 43.

Pock, T., M. Unger, D. Cremers and H. Bischof. 2008*a*. Fast and Exact Solution of Total Variation Models on the GPU. In *CVPR Workshop on Visual Computer Vision on GPUs*. Cited on pages 36 and 43.

Pock, T., M. Unger, D. Cremers and H. Bischof. 2008*b*. Fast and Exact Solution of Total Variation Models on the GPU. In *CVPR Workshop on Visual Computer Vision on GPUs*. Cited on page 100.

Pritch, Y., E. Kav-Venaki and S. Peleg. 2009. Shift-Map Image Editing. In *Int. Conf. Computer Vision*. Cited on pages 103, 121, 123, 124, 125, 126 and 127.

Rantzer, A. 2009. Dynamic Dual Decomposition for Distributed Control. In *American Control Conference*. Cited on page 9.

Rother, C., V. Kolmogorov and A. Blake. 2004. "GrabCut – Interactive Foreground Extraction using Iterated Graph Cuts." *ACM Transactions on Graphics* 23(3):309–314. Cited on page 81.

Rother, C., V. Kolmogorov, V. Lempitsky and M. Szummer. 2007*a*. Optimizing Binary MRFs via Extended Roof Duality. In *Conf. Computer Vision and Pattern Recognition*. Cited on pages 19 and 70.

Rother, C., V. Kolmogorov, V. Lempitsky and M. Szummer. 2007*b*. Optimizing Binary MRFs via Extended Roof Duality. Technical Report MSR-TR-2007-46 Microsoft. Cited on page 19.

Rousson, M. and R. Deriche. 2002. A variational framework for active and adaptative segmentation of vector valued images. In *In Proc. IEEE Workshop on Motion and Video Computing*. pp. 56–62. Cited on page 50.

Rudin, L.I., S. Osher and E. Fatemi. 1992. "Nonlinear Total Variation Based Noise Removal Algorithms." *Physica D* 60:259–268. Cited on page 39.

Sarti, A., C. Corsi, E. Mazzini and C. Lamberti. 2004. "Maximum likelihood segmentation with Rayleigh distribution of ultrasound images." *Computers in Cardiology, 2004* pp. 329–332. Cited on page 50.

Schoenemann, T., F. Kahl and D. Cremers. 2009. Curvature Regularity for Region-based Image Segmentation and Inpainting: A Linear Programming Relaxation. In *Int. Conf. Computer Vision*. Cited on pages 4, 55, 56, 57, 58, 59, 62, 63 and 71.

Snir, M. and S. Otto. 1998. *MPI — The Complete Reference: The MPI Core*. Cambridge, MA, USA: MIT Press. Cited on page 95.

Strandmark, P. and F. Kahl. 2010. Parallel and Distributed Graph Cuts by Dual Decomposition. In *Conference on Computer Vision and Pattern Recognition*. Cited on page 81.

Strandmark, P., F. Kahl and N. C. Overgaard. 2009. Optimizing Parametric Total Variation Models. In *Int. Conf. Computer Vision*. Cited on page 33.

Sullivan, J.M. 1990. Crystalline Approximation Theorem for Hypersurfaces PhD thesis Princeton Univ. Cited on page 56.

Svärm, L. and P. Strandmark. 2010. Shift-map Image Registration. In *International Conference on Pattern Recognition (ICPR)*. Cited on page 121.

Tola, E., V. Lepetit and P. Fua. 2009. "DAISY: An Efficient Dense Descriptor Applied to Wide Baseline Stereo." *IEEE Trans. Pattern Analysis and Machine Intelligence* . Cited on page 123.

Turing, A. 1950. "Computing Machinery and Intelligence." *Mind* LIX(236):433–460. Cited on page 1.

Unger, M., T. Pock, D. Cremers and H. Bischof. 2008. TVSeg - Interactive Total Variation Based Image Segmentation. In *British Machine Vision Conf.* Cited on page 100.

Vineet, V. and P. J. Narayanan. 2009. Solving Multi-label MRFs using Incremental alpha-expansion move on the GPUs. In *Asian Conference on Computer Vision*. Cited on page 83.

Vineet, V. and P.J. Narayanan. 2008. CUDA cuts: Fast graph cuts on the GPU. In *Computer Vision and Pattern Recognition Workshops, CVPRW*. Cited on pages 83, 105 and 106.

Wardetzky, M., M. Bergou, D. Harmon, D. Zorin and E. Grinspun. 2007. "Discrete quadratic curvature energies." *Comput. Aided Geom. Des.* 24(8-9):499–518. Cited on pages 74 and 75.

Werlberger, M., W. Trobin, T. Pock, A. Wedel, D. Cremers and H. Bischof. 2009. Anisotropic Huber-L1 Optical Flow. In *British Machine Vision Conf.* Cited on page 100.

Willmore, T.J. 1965. "Note on Embedded Surfaces." *An. Sti. Univ. "Al. I. Cuza" Iasi Sect. I a Mat. (N.S.)* pp. 493–496. Cited on page 73.

Winder, S., G. Hua and M. Brown. 2009. Picking the best DAISY. In *Conf. Computer Vision and Pattern Recognition*. IEEE pp. 178–185. Cited on page 124.

Woodford, O., P.H.S. Torr, I. Reid and A.W. Fitzgibbon. 2009. "Global Stereo Reconstruction under Second Order Smoothness Priors." *IEEE Trans. Pattern Analysis and Machine Intelligence* 31(12):2115–2128. Cited on page 55.

Xu, M., P. M. Thompson and A. W. Toga. 2004. "An Adaptive Level Set Segmentation on a Triangulated Mesh." *IEEE Trans. on Medical Imaging* 23(2):191–201. Cited on page 64.

Index