Higher-Order Regularization in Computer Vision

Ulén, Johannes

2014

Higher-Order Regularization in Computer Vision

Higher-Order Regularization in Computer Vision

Johannes Ulén

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/

Doctoral Theses in Mathematical Sciences 2014:7
ISSN 1404-0034

ISBN 978-91-7623-163-0 (print)
ISBN 978-91-7623-164-7 (digital)
LUTFMA-1051-2014

Printed in Sweden by MediaTryck, Lund 2014

Preface

At the core of many computer vision models lies the minimization of an objective function consisting of a sum of functions with few arguments. The order of the objective function is defined as the highest number of arguments of any summand. To reduce ambiguity and noise in the solution, regularization terms are included into the objective function, enforcing different properties of the solution. The most commonly used regularization is penalization of boundary length, which requires a second-order objective function. Most of this thesis is devoted to introducing higher-order regularization terms and presenting efficient minimization schemes.

One of the topics of the thesis covers a reformulation of a large class of discrete functions into an equivalent form. The reformulation is shown, both in theory and practical experiments, to be advantageous for higher-order regularization models based on curvature and second-order derivatives. Another topic is the parametric max-flow problem. An analysis is given, showing its inherent limitations for large-scale problems which are common in computer vision. The thesis also introduces a segmentation approach for finding thin and elongated structures in 3D volumes. Using a line-graph formulation, it is shown how to efficiently regularize with respect to higher-order differential geometric properties such as curvature and torsion. Furthermore, an efficient optimization approach for a multi-region model is presented which, in addition to standard regularization, is able to enforce geometric constraints such as inclusion or exclusion of different regions. The final part of the thesis deals with dense stereo estimation. A new regularization model is introduced, penalizing the second-order derivatives of a depth or disparity map. Compared to previous second-order approaches to dense stereo estimation, the new regularization model is shown to be more easily optimized.

During my time in Lund I have had the privilege to work with many colleagues, without them this thesis would not exist. I have had many fruitful discussions with my advisors Fredrik Kahl and Carl Olsson without whom my production would have been scarce. Their meticulous reading of many of my manuscripts have greatly improved their quality, this especially includes this thesis. I would also like to extend my gratitude to all my co-authors and colleagues. I would especially like to thank Petter Strandmark for helping me out a lot during my early years of research and for many collaborations. For everything outside of work I would like to thank my friends, family, and Sofia.

This work has been funded by the Swedish Research Council (grant no. 2012-4215), the Swedish Foundation for Strategic Research (Future Research Leaders), and the European Research Council (GlobalVision grant no. 209480).

Lund, October 2014
Johannes Ulén

Previous publications

This thesis is based on the following papers.

Main papers

- Johannes Ulén, Petter Strandmark and Fredrik Kahl (2011). "Optimization for Multi-Region Segmentation of Cardiac MRI." *MICCAI Workshop on Statistical Atlases and Computational Models of the Heart: Imaging and Modelling Challenges*, Toronto, Canada, Springer.

- Johannes Ulén, Petter Strandmark and Fredrik Kahl (2013), "An Efficient Optimization Framework for Multi-Region Segmentation based on Lagrangian Duality," *IEEE Transactions on Medical Imaging*.

- Carl Olsson, Johannes Ulén and Yuri Boykov (2013). "In Defense of 3D-Label Stereo." *Conference on Computer Vision and Pattern*, Portland, USA, IEEE.

- Johannes Ulén and Carl Olsson (2013). "Simultaneous Fusion Moves for 3D-Label Stereo." *International Conference on Energy Minimization Methods in Computer Vision and Pattern Recognition*, Lund, Sweden, Springer.

- Carl Olsson, Johannes Ulén, Yuri Boykov and Vladimir Kolmogorov (2013). "Partial Enumeration and Curvature Regularization." *International Conference on Computer Vision*, Sydney, Australia, IEEE.

- Petter Strandmark, Johannes Ulén, Fredrik Kahl and Leo Grady (2013). "Shortest Paths with Curvature and Torsion." *International Conference on Computer Vision*, Sydney, Australia, IEEE.

- Carl Olsson, Johannes Ulén and Anders Eriksson (2014). "Local Refinement for Stereo Regularization." *International Conference on Pattern Recognition*, Stockholm, Sweden, IEEE.

- Johannes Ulén, Petter Strandmark and Fredrik Kahl. "Shortest Paths with Higher-Order Regularization." *In submission*.

Subsidiary papers

- Petter Strandmark, Johannes Ulén and Fredrik Kahl (2012). "HEp-2 Staining Pattern Classification." *International Conference on Pattern Recognition*, Tsukuba, Japan, IEEE.

- Zhayida Simayijiang, Sofia Backman, Johannes Ulén, Sverre Wikström and Kalle Åström (2013). "Exploratory study of EEG burst characteristics in preterm infants." *International Conference of the IEEE Engineering in Medicine and Biology Society*, Osaka, Japan, IEEE.

Code

Code which can be used to reproduce many of the experiments in this thesis is available at http://github.com/johannesu.

Contents

Chapter 1

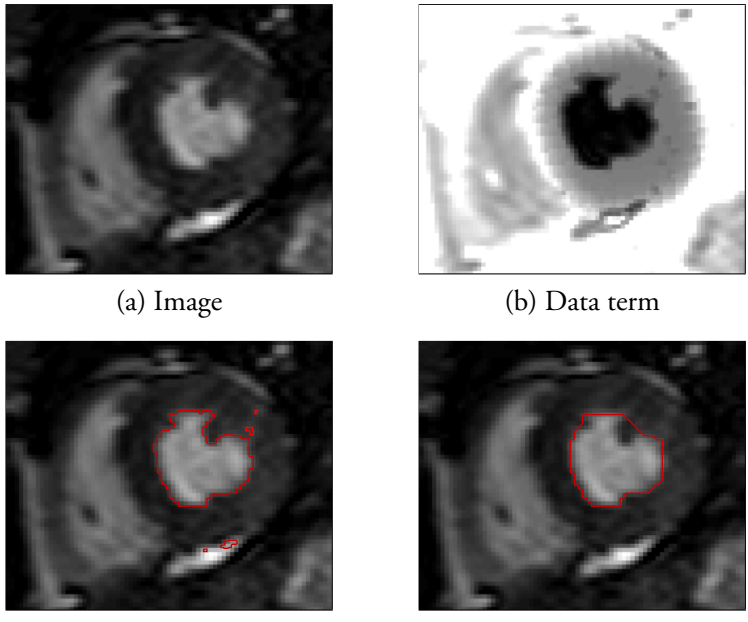
Introduction

Images contain a lot of information which humans are very apt at extracting. Examples include boundaries of objects and distances between objects. The problem of finding such information belongs to the family of *inverse problems* which forms the basis of computer vision. Solving an inverse problem is usually posed as computing the minimizer to an objective function of the form

$$f(\boldsymbol{x}) = \underbrace{D(\boldsymbol{x})}_{\text{data term}} + \underbrace{R(\boldsymbol{x})}_{\text{regularization term}}, \tag{1.1}$$

where $\boldsymbol{x}$ is a vector of variables representing for instance labels for all the pixels in an image. The data term usually models the local appearance of a pixel and could for example be the probability of a pixel being part of an object. In an ideal scenario, the data term would be sufficient to solve the inverse problem. However, in most cases the output of the data term is noisy and far from perfect. To overcome ambiguity and noisy solutions, some kind of regularization term is often used. Usually, the regularization term does not depend on the underlying data. The term penalizes unwanted properties of the solution such as, for example, unsmooth boundaries. Figure 1.1 shows an example of an inverse problem: object segmentation. The example shows the effect of adding a regularization term.

Without regularization, the optimal solution can often be found by thresholding the data term. Adding regularization adds complexity to the model, at the cost of having to minimize a more difficult function. Most of this thesis is devoted to introducing higher-order regularization terms and presenting efficient methods for minimization of the resulting functions.



(a) Image (b) Data term

(c) Solution *without* regularization (d) Solution *with* regularization

Figure 1.1: Example of an inverse problem; segmenting the boundary of the left ventricle in the human heart. In (a) a two-dimensional image of the heart is shown. In (b) a data term, $D(\boldsymbol{x})$, is constructed from the image, based on both spatial and intensity statistics. Darker regions indicate higher probability of the left ventricle. In (c) the solution when $f(\boldsymbol{x}) = D(\boldsymbol{x})$ is shown. The resulting segmentation boundary is incoherent. In (d) the solution when $f(\boldsymbol{x}) = D(\boldsymbol{x}) + R(\boldsymbol{x})$ is shown, where the regularization, $R(\boldsymbol{x})$, penalizes the boundary length. The resulting segmentation is now one coherent object. In reality, the left ventricle in a healthy human heart is round. In this example, the data term alone is not sufficient to model the ventricle, since the ventricular boundary is occluded by the papillary muscle. By penalizing the boundary length, the resulting segmentation becomes more correct as opposed to simply choosing the largest connected component in (c).

Thesis overview

Chapter 2. This chapter introduces necessary background needed for the rest of the thesis.

Chapter 3. Most objective functions used in computer vision can be written on the form

$$f(\boldsymbol{x}) = \sum_{a \in \mathcal{F}} f_a(\boldsymbol{x}_a), \tag{1.2}$$

where each function f_a is only defined for a subset of $\boldsymbol{x}$, denoted $\boldsymbol{x}_a$. Each set $\boldsymbol{x}_a$ usually represents a set of pixels which are spatially close to each other, for instance all pixels in some patch of an image. In this chapter, this special structure is exploited to reformulate the minimization of f into a *weighted constraint satisfaction problem*. The gain is two-fold. Firstly, it allows a larger class of functions to be optimized using popular and efficient decomposition methods such as TRW-s and dual decomposition. Secondly, it is shown both in theory and experiments that even if the original function can be solved using a decomposition method, the reformulated function is preferable as the resulting solutions often give a lower objective value. The chapter also introduces a novel curvature regularization model where costs are defined on overlapping patches.

Author contributions: My contributions to this chapter include improving the implementation, performing experiments and comparing the method to the state of the art.

Chapter 4. This chapter deals with the parametric max-flow problem:
Given a submodular function,

$$f_{\boldsymbol{\gamma}}(\boldsymbol{x}) = \sum_{i=1}^{n} U_i(x_i) + \sum_{i=1}^{n} \sum_{j>i}^{n} B_{ij}(x_i, x_j) + \sum_{i=1}^{n} \sum_{j=1}^{d} \gamma_j c_{ij} x_i, \tag{1.3}$$

with the parameter vector $\boldsymbol{\gamma}$, find all optimal solutions for any choice of $\boldsymbol{\gamma} \in \mathbf{R}^d$.

Working with the one-dimensional case, $d = 1$, has found its uses in computer vision, see Kolmogorov, Boykov, et al. (2007) and Carreira and Sminchisescu (2010). Working with two or more dimensions, $d \geq 2$;

however, has been largely overlooked. In this chapter, an efficient algorithm used to solve the parametric max-flow problem is considered. The algorithm performs multiple intersections of hyperplanes, resulting in numerical issues. An implementation using exact arithmetic is compared to an implementation using floating precision. The result shows that even for moderately sized problems, the floating precision algorithm fails to recover many solutions. Furthermore, it is revealed that the number of solutions, even using only two parameters, can be intractably large even for modestly sized problem instances. To combat this, an approximate algorithm is also presented.

Author contributions: This chapter is based on joint work between me, Fredrik Kahl and Olof Enqvist. We have all worked on the theoretical side of the problem. I have done the implementation of the algorithm and performed all experiments.

Chapter 5. In medical imaging, a common problem is locating elongated structures in 3D volumes, such as for example blood vessels. These structures are often modeled as curves. Standard properties of a curve include length, curvature and torsion. Yet most methods used to extract curves in 3D only regularize length and sometimes curvature. In this chapter, it is shown, using a line graph formulation, how to penalize all of the three aforementioned properties. It is also shown how to more properly measure the length between two points on a given depth map.

Author contributions: The implementation and experiments are joint work between me and Petter Strandmark, our contributions to this chapter are roughly equal.

Chapter 6. In this chapter, it is shown how to efficiently optimize a segmentation model able to segment several regions simultaneously. The model is able to enforce geometric constraints such as inclusion or exclusion of different regions. These priors are especially useful for medical images where anatomy can pose natural constraints on the resulting segmentation. For instance, a constraint could be that the heart chamber is always found inside the heart muscle.

Author contributions: I did most of the work on this chapter. Petter Strandmark helped out with the initial implementation and some experiments.

Chapter 7. Dense stereo estimation is a classical computer vision problem; given two or more input images, estimate the depth of every pixel in one of the images. A common approach is to give each pixel a label indicating its depth or disparity. In this chapter, the label space is increased by associating a tangent plane to each pixel. By penalizing deviations from the tangent planes, it is possible to efficiently regularize a function of the second derivative. Penalizing the second derivative has previously been shown to be successful, see Woodford et al. (2009). The optimization itself is performed by fusing proposed solutions. On this note, the chapter also introduces a method of fusing several proposals at the same time, improving the end results. Furthermore, novel ways of generating proposed solutions are also presented.

Author contributions: Most of my work with this chapter is related to disparity experiments and development of the theory for simultaneous fusion. I also did initial work with ADMM for local refinement.

Chapter 2
Preliminaries

This chapter introduces some necessary background which will benefit newcomers to the field of optimization in computer vision. Firstly, some computer vision basics are covered; the pinhole camera model and stereo vision. Thereafter, a brief introduction to differential geometry is given, followed by a more thorough introduction to optimization, with an emphasis on discrete optimization.

2.1 Cameras and stereo vision

A pair of human eyes can estimate the distance to an object by combining the view of each eye. This is used in modern 3D cinemas to emulate a feeling of depth in a movie. A related problem is to estimate the depth of each pixel in an image, given two images of the same scene taken from slightly different angles. This problem is known as *dense stereo estimation* and is a classical problem in computer vision.

In a camera, points in a scene in $\mathbf{R}^3$ are projected into images in $\mathbf{R}^2$. To model this we use the standard pinhole camera model, see Figure 2.1 and Hartley and Zisserman (2003). Every scene point $\boldsymbol{X} \in \mathbf{R}^3$ along a ray projects onto the same pixel $\boldsymbol{x} \in \mathbf{R}^2$. This projection is described as

$$\lambda \begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix} = \boldsymbol{P} \begin{bmatrix} \boldsymbol{X} \\ 1 \end{bmatrix}, \tag{2.1}$$

where $\boldsymbol{P}$ is the 3×4 *camera matrix* and $\lambda \in \mathbf{R}$. The camera matrix contains information about the spatial location of the camera and its internal parameters.

Now assume that we have a stereo setup; two identical cameras are viewing the same point $\boldsymbol{X} \in \mathbf{R}^3$. The geometry of this setup is called

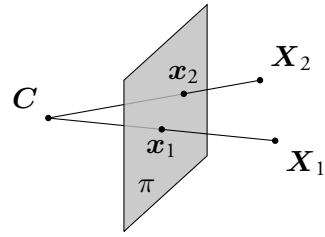


Figure 2.1: Mathematical model of the pinhole camera. $\boldsymbol{C}$ is the camera center and π the image plane. The two scene points $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$ are projected onto $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ in the image plane. In a real pinhole camera, the image plane is located *behind* the camera center. By placing the image plane *in front of* the camera center, we avoid upside down images.

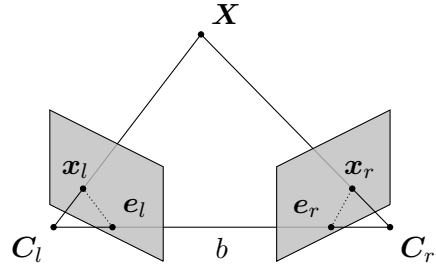


Figure 2.2: Epipolar geometry; two cameras viewing the same scene point $\boldsymbol{X}$. The left and right camera centers are given by $\boldsymbol{C}_l$ and $\boldsymbol{C}_r$, respectively, and the projections are given by $\boldsymbol{x}_l$ and $\boldsymbol{x}_r$. The two camera centers are connected by the baseline b. The intersections of the baseline and the two images give the two epipoles $\boldsymbol{e}_l$ and $\boldsymbol{e}_r$.

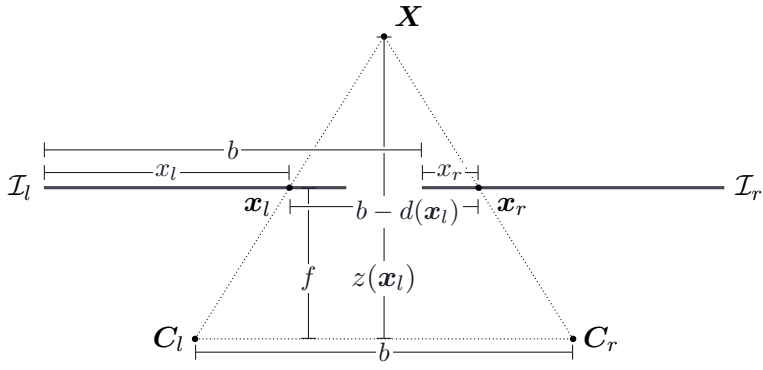


Figure 2.3: A rectified setup viewed from above, showing the two camera centers, $\boldsymbol{C}_l$ and $\boldsymbol{C}_r$, and their respective image planes, $\mathcal{I}_l$ and $\mathcal{I}_r$. The scene point $\boldsymbol{X}$ is projected onto $\boldsymbol{x}_l$ and $\boldsymbol{x}_r$ in the two images. Note that $b + x_r - x_l = b - d(\boldsymbol{x}_l)$.

epipolar geometry and is depicted in Figure 2.2. The ray connecting the left camera center, $\boldsymbol{C}_l$, to $\boldsymbol{X}$ will be projected along a line in the right image, this line is called the *epipolar line* of the point $\boldsymbol{x}_l$. Equivalently, the right epipolar line is defined using the ray connecting $\boldsymbol{C}_r$ and $\boldsymbol{X}$. The geometry is further simplified if the right and left epipolar lines coincide in the image coordinate system. This can be achieved either by aligning the cameras before the images are taken, or by a process known as *image rectification*. Two images are rectified by rotating the camera pair and transforming the images, see Seitz (2001). A rectified setup is shown in Figure 2.3. Once we have a pair of rectified images, matching $\boldsymbol{x}_l$ and $\boldsymbol{x}_r$ is easier. The reason is that $\boldsymbol{x}_r$ must be located on the same row as $\boldsymbol{x}_l$ in their respective images.

Definition 2.1. *Given two rectified cameras and a scene point, $\boldsymbol{X}$. Let $\boldsymbol{x}_l$ and $\boldsymbol{x}_r$ be the projections of $\boldsymbol{X}$ in the left and right image plane. For the two pixels $\boldsymbol{x}_l = (x_l, y_l)$ and $\boldsymbol{x}_r = (x_r, y_r)$ it follows that $y_l = y_r$, since the cameras are rectified. The* disparity *of $\boldsymbol{x}_l$ is defined as*

$$d(\boldsymbol{x}_l) = x_l - x_r. \tag{2.2}$$

Using the notation from Figure 2.3 and similar triangles we get

$$\frac{b - d(\boldsymbol{x}_l)}{z(\boldsymbol{x}_l) - f} = \frac{b}{z(\boldsymbol{x}_l)} \iff z(\boldsymbol{x}_l) = \frac{fb}{d(\boldsymbol{x}_l)}. \tag{2.3}$$

For all pixel pairs in the two images, the focal length, f, and baseline, b, are constant. It follows that the *depth*, $z(\boldsymbol{x}_l)$, is inversely proportional to the disparity, $d(\boldsymbol{x}_l)$.

2.2 Differential geometry

An *arc length parametrized curve* is a $\mathcal{C}^3$ function,

$$\boldsymbol{\gamma} : [a, b] \to \mathbf{R}^3, \tag{2.4}$$

such that $|\boldsymbol{\gamma}'(s)| = 1$ for all $s \in [a, b]$. Assume that $\boldsymbol{\gamma}$ is parametrized by arc length and that $\boldsymbol{\gamma}''(s) \neq 0$ for all $s \in [a, b]$ and define

$$\boldsymbol{t}(s) = \boldsymbol{\gamma}'(s) \tag{2.5}$$

as the *tangent vector* to the curve. Furthermore, define

$$\boldsymbol{n}(s) = \frac{\boldsymbol{\gamma}''(s)}{|\boldsymbol{\gamma}''(s)|} \tag{2.6}$$

as the *normal vector* to the curve. The two vectors are orthogonal, this follows because $\boldsymbol{\gamma}$ is parametrized by arc length,

$$0 = \frac{d}{ds}(1) = \frac{d}{ds}\big(\boldsymbol{\gamma}'(s) \cdot \boldsymbol{\gamma}'(s)\big) = 2\big(\boldsymbol{\gamma}''(s) \cdot \gamma'(s)\big). \tag{2.7}$$

The tangent vector and the normal vector span the *osculating plane* for each point $s \in [a, b]$. Finally, define

$$\boldsymbol{b}(s) = \boldsymbol{t}(s) \times \boldsymbol{n}(s) \tag{2.8}$$

as the *binormal vector* to the curve. We have now defined three orthogonal vectors in $\mathbf{R}^3$ at each point s. These vectors together define the *Frenet-Serret frame* for the point s, see Figure 2.4.

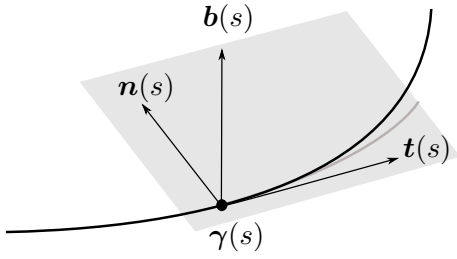


Figure 2.4: The Frenet-Serret frame for a point, s, on the curve $\boldsymbol{\gamma}$. The osculating plane for the same point is indicated in shaded gray.

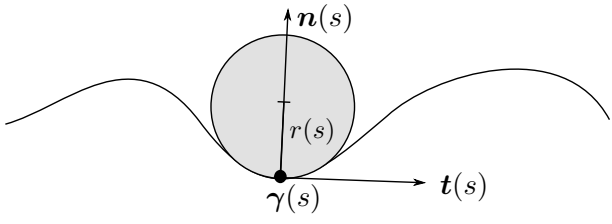


Figure 2.5: Interpretation of curvature in $\mathbf{R}^2$. The osculating circle is the largest circle whose boundary, but not interior, touches the curve. It can be shown that $\kappa(s) = \frac{1}{r(s)}$, where $r(s)$ is the radius of the osculating circle. Intuitively for a straight line, $\kappa(s) \to 0$ will give $r(s) \to \infty$ and for a very curved line, $r(s) \to 0$ will give $\kappa(s) \to \infty$.

At each point of a curve, the *curvature* is a measure of how much the curve bends and it is defined as

$$\kappa(s) = |\boldsymbol{\gamma}''(s)| = |\boldsymbol{t}'(s)|. \tag{2.9}$$

A straight line will always have zero curvature. In $\mathbf{R}^2$, the curvature has a geometrical interpretation in terms of an *osculating circle* as shown in Figure 2.5. Another property of a curve is the *torsion* which tells us how much the curve bends out of the osculating plane. The torsion is defined as

$$\tau(s) = -\boldsymbol{n}(s) \cdot \boldsymbol{b}'(s). \tag{2.10}$$

From this definition it follows that a curve contained inside a plane has zero torsion everywhere.

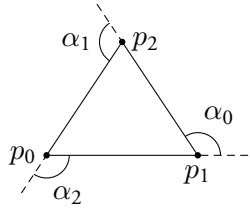


Figure 2.6: The exterior angles of a closed piecewise linear curve, $\boldsymbol{\delta}$, are denoted α_i. The absolute integral of the curvature is given by $P(\boldsymbol{\delta}) = \alpha_0 + \alpha_1 + \alpha_2$.

In computer vision, we deal with discrete data, which means that the curves which we treat tend to be piecewise linear. Let $\boldsymbol{\delta}$ be a piecewise linear curve defined by the points $(p_0, \ldots, p_n)$ and let $\boldsymbol{\alpha}$ denote the set of exterior angles formed by the ordered points (p_{i-1}, p_i, p_{i+1}). The *absolute integral of the curvature* of $\boldsymbol{\delta}$ is then defined as

$$P(\boldsymbol{\delta}) = \sum_{\alpha \in \boldsymbol{\alpha}} \alpha. \tag{2.11}$$

An example of exterior angles is given in Figure 2.6. This definition is possible to extend to closed smooth curves $\boldsymbol{\gamma}$ as

$$\int_a^b |\kappa(s)| ds = \sup \left\{ \int_a^b P(\boldsymbol{\delta}) : \boldsymbol{\delta} \text{ is inscribed in } \boldsymbol{\gamma} \right\}, \tag{2.12}$$

see Bruckstein et al. (2001).

Up until now, we have only considered curves in Euclidean space. Suppose that a curve, $\boldsymbol{\gamma}$, lies on some surface, M, embedded in $\mathbf{R}^3$. Let $T_M(s)$ be the tangent plane at the point s on the curve. Then, a curve on M is a *geodesic* if

$$T_M(s) \perp \boldsymbol{\gamma}''(s) \qquad \text{for all } s \in [a, b]. \tag{2.13}$$

Geodesics can be seen as a generalization of the concept of straight lines in Euclidean space. The shortest path between two points on a surface is guaranteed to be a geodesic, this distance will be referred to as the *geodesic distance*.

2.3 Optimization

All methods discussed in this thesis basically boil down to the same core; a problem is modeled by some function and the minimizer of the function gives the solution to the model. The computational cost to optimize these functions ranges from low to very high depending on the model. Generally, we will consider the optimization problem

$$\underset{\boldsymbol{x}\in\mathcal{X}^n}{\text{minimize}}\, f(\boldsymbol{x}), \tag{2.14}$$

where $\mathcal{X}$ is some space and $f : \mathcal{X}^n \to \mathbf{R}$. The function f is the *objective function* and the function value $f(\boldsymbol{x})$ is the *objective value*.

Remark 2.2. *In the computer vision community, f is often referred to as the* energy function *and $f(\boldsymbol{x})$ as the* energy. *This is most probably due to many functions' mathematical relation to the Ising model in physics.*

We will start this section with some necessary definitions.

Definition 2.3. *A function $f \in \mathcal{C}^n$ if $f', f'', \ldots, f^{(n)}$ all exist and are continuous.*

Definition 2.4. *A set S is said to be* convex *if $\boldsymbol{x}, \boldsymbol{y} \in S$ implies that*

$$(1-t)\boldsymbol{x} + t\boldsymbol{y} \in S, \tag{2.15}$$

for all $t \in [0, 1]$. That is, if every line connecting two points in the set is fully contained in the set.

Definition 2.5. *A function f is* convex *if, given $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathcal{X}^n$,*

$$f(t\boldsymbol{x}_1 + (1-t)\boldsymbol{x}_2) \leq tf(\boldsymbol{x}_1) + (1-t)f(\boldsymbol{x}_2) \tag{2.16}$$

for all $t \in [0, 1]$. Similarly, a function f is concave *if, given $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathcal{X}^n$,*

$$f(t\boldsymbol{x}_1 + (1-t)\boldsymbol{x}_2) \geq tf(\boldsymbol{x}_1) + (1-t)f(\boldsymbol{x}_2) \tag{2.17}$$

for all $t \in [0, 1]$.

For non-differentiable convex functions, there is a generalization of the concept of gradients known as *subgradients*.

Definition 2.6. *A vector $\boldsymbol{g} \in \mathcal{X}^n$ is a* subgradient *to a convex function f at $\boldsymbol{x}_0$ if, for all $\boldsymbol{x} \in \mathcal{X}^n$,*

$$f(\boldsymbol{x}) - f(\boldsymbol{x}_0) \geq \boldsymbol{g}^{\mathrm{T}}(\boldsymbol{x} - \boldsymbol{x}_0). \tag{2.18}$$

Similarly, a vector $\boldsymbol{g} \in \mathcal{X}^n$ is a supergradient *to a concave function f at $\boldsymbol{x}_0$ if, for all $\boldsymbol{x} \in \mathcal{X}^n$,*

$$f(\boldsymbol{x}) - f(\boldsymbol{x}_0) \leq \boldsymbol{g}^{\mathrm{T}}(\boldsymbol{x} - \boldsymbol{x}_0). \tag{2.19}$$

A reason for focusing on convex functions is that they are relatively easy to optimize globally. In this chapter, we will cover several methods which are able to find the global minimum of convex functions. The methods covered are all iterative; starting at some initial solution, $\boldsymbol{x}_1$, the solution at iteration k, $\boldsymbol{x}_k$, is given by some function of $\boldsymbol{x}_{k-1}$. All methods also have some convergence criteria which for instance can be to stop when

$$|f(\boldsymbol{x}_k) - f(\boldsymbol{x}_{k-1})| \leq \varepsilon, \tag{2.20}$$

for some $\varepsilon \in \mathbf{R}$, or when

$$k > M, \tag{2.21}$$

for some $M \in \mathbf{N}$.

2.3.1 Gradient descent

Assume that $f \in \mathcal{C}^1$. The optimization method called *gradient descent* iteratively solves (2.14) by taking steps in the descent direction. The method can be expressed as

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \tau_k \nabla f(\boldsymbol{x}_k), \tag{2.22}$$

where τ_k is a step length usually determined via line search. An example of line search is the exact line search given by solving

$$\tau_k = \underset{\tau}{\operatorname{argmin}}\, f\left(\boldsymbol{x}_k + \tau \nabla f(\boldsymbol{x}_k)\right). \tag{2.23}$$

2.3.2 Newton's method

If we have the slightly stronger requirement $f \in \mathcal{C}^2$, a related optimization method is *Newton's method*, where f is locally approximated by a second-order polynomial. The second-order *Taylor expansion* at $\boldsymbol{x}_k$ is given by

$$t(\boldsymbol{x}) = f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)^T (\boldsymbol{x} - \boldsymbol{x}_k) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_k)^T \boldsymbol{H}_f(\boldsymbol{x}_k)(\boldsymbol{x} - \boldsymbol{x}_k), \tag{2.24}$$

where $\boldsymbol{H}_f$ is the Hessian of f. Differentiation of t gives the following characterization of an optimum of the Taylor expansion:

$$\nabla t(\boldsymbol{x}) = \nabla f(\boldsymbol{x}) + \boldsymbol{H}_f(\boldsymbol{x}_k)(\boldsymbol{x} - \boldsymbol{x}_k) = 0 \iff \tag{2.25}$$

$$\boldsymbol{x} = \boldsymbol{x}_k - \boldsymbol{H}_f(\boldsymbol{x}_k)^{-1} \nabla f(\boldsymbol{x}_k). \tag{2.26}$$

Provided that H_f is invertible, this short derivation gives the Newton method, which can be expressed as

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{H}_f(\boldsymbol{x}_k)^{-1} \nabla f(\boldsymbol{x}_k). \tag{2.27}$$

Newton's method has faster theoretical convergence than the gradient descent method, see Boyd and Vandenberghe (2004). Even though Newton's method has faster convergence, it has its drawbacks. Evaluating

$$\boldsymbol{H}_f(\boldsymbol{x}_k)^{-1} \nabla f(\boldsymbol{x}_k) \tag{2.28}$$

is computationally expensive and $\boldsymbol{H}_f$ may be close to singular, resulting in numerical instabilities.

2.3.3 Levenberg-Marquardt

The *Levenberg-Marquardt* method addresses the numerical stability issues of Newton's method. The method is given by

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \big(\boldsymbol{H}_f(\boldsymbol{x}_k) + \lambda \boldsymbol{I}\big)^{-1} \nabla f(\boldsymbol{x}_k), \tag{2.29}$$

where $\boldsymbol{I}$ is the identity matrix and λ is a non-negative number chosen such that $\boldsymbol{H}_f(\boldsymbol{x}_k) + \lambda \boldsymbol{I}$ is positive definitive. Choosing $\lambda = 0$ gives Newton's method (2.27). Choosing λ as a very large number gives a method similar to the gradient descent method (2.22).

2.3.4 Broyden-Fletcher-Goldfarb-Shanno

The *Broyden-Fletcher-Goldfarb-Shanno* (*BFGS*) method addresses the issue of the computational efficiency of Newton's method. It replaces the Hessian inverse with an approximation,

$$\mathbf{B}_k \approx \boldsymbol{H}_f^{-1}(\boldsymbol{x}_k). \tag{2.30}$$

The approximation $\mathbf{B}_k$ is initialized as the identity matrix and in each iteration, it is efficiently updated by a low-rank approximation. Hence, the first iteration is just a gradient descent step. The method is simply expressed as

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \mathbf{B}_k \nabla f(\boldsymbol{x}_k). \tag{2.31}$$

For an n-dimensional function, storing $\mathbf{B}_k$ requires an $n \times n$ matrix which might be intractable for large problems. This issue is addressed in the *Limited-memory BFGS* (*LBFGS*) method where $\mathbf{B}_k$ is never explicitly stored. Instead, a history of previous updates is stored, making it possible to evaluate (2.31) without actually storing $\mathbf{B}_k$. See Wright and Nocedal (1999) for details.

2.3.5 The subgradient method

Efficient minimization of non-differential functions is also possible. Suppose that $f \in C^0$ and that f is convex. The *subgradient method* is given by

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \tau_k \boldsymbol{g}(\boldsymbol{x}_k), \tag{2.32}$$

where $\boldsymbol{g}(\boldsymbol{x}_k)$ is *any* subgradient of f at $\boldsymbol{x}_k$ and τ_k is the step length at iteration k. There are several ways of choosing the step length. One family of step lengths with favorable theoretical properties is the *non-summable diminishing* step lengths.

Definition 2.7. *A* non-summable diminishing *step length τ_k satisfies*

$$\lim_{k\to\infty} \tau_k = 0, \quad \sum_{k=1}^{\infty} \tau_k = \infty. \tag{2.33}$$

A commonly used non-summable diminishing step length is

$$\tau_k = \frac{1}{k}, \tag{2.34}$$

where k is the iteration number.

Proposition 2.8. *For a non-summable diminishing step length and any $\varepsilon > 0$, we are guaranteed that the iterative solutions, $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k\}$, given by the subgradient method fulfill*

$$\lim_{k\to\infty} \left| \min_{\boldsymbol{x}\in\{\boldsymbol{x}_0,\ldots,\boldsymbol{x}_k\}} f(\boldsymbol{x}) - f(\boldsymbol{x}^\star) \right| < \varepsilon, \tag{2.35}$$

where $\boldsymbol{x}^\star = \operatorname{argmin}_{\boldsymbol{x}} f(\boldsymbol{x})$.

Proof. See Boyd, Xiao, et al. (2003). □

For a non-summable diminishing step length, each step taken in the direction of the optimal solution is shorter than the previous one. Since the sum of all step lengths is infinite, we will eventually reach the optimal solution. It is worth noting that we are not guaranteed to decrease the objective value in each iteration.

There are also some step length schemes which are more experimental. In Strandmark, Kahl, and Schoenemann (2011), each dimension, p, is given its own step length, τ_k^p, which is initialized as $\tau_1^p = 1$ and updated as

$$\tau_{k+1}^p = \begin{cases} \tau_k^p & \text{if } \operatorname{sign}\left(x_k^p\right) = \operatorname{sign}\left(x_{k-1}^p\right), \\ \tau_k^p/2 & \text{if } \operatorname{sign}\left(x_k^p\right) \neq \operatorname{sign}\left(x_{k-1}^p\right). \end{cases} \tag{2.36}$$

This is a more aggressive step length scheme which lacks the theoretical justification of non-summable diminishing step lengths. However, in practice, it converges much faster.

Proposition 2.9. *For any integer $k \geq 0$ and point $\boldsymbol{x}_1$, there exists a convex function f with subgradient $\boldsymbol{g}$ at $\boldsymbol{x}$, such that any optimization scheme, where $\boldsymbol{x}_k$ is chosen as*

$$\boldsymbol{x}_{k+1} \in \boldsymbol{x}_1 + \operatorname{span}\left\{\boldsymbol{g}(\boldsymbol{x}_1), \ldots, \boldsymbol{g}(\boldsymbol{x}_k)\right\}, \tag{2.37}$$

needs $\mathcal{O}(1/\epsilon^2)$ *iterations to achieve*

$$|f(\boldsymbol{x}_k) - f(\boldsymbol{x}^\star)| < \epsilon, \tag{2.38}$$

where $\boldsymbol{x}^\star = \operatorname{argmin}_{\boldsymbol{x}} f(\boldsymbol{x})$ *and* $\varepsilon > 0$.

Proof. Theorems 3.2.1 and 3.2.2 in Nesterov (2004). □

As shown in Proposition 2.9 the worst case convergence of the subgradient method, in (2.32), is very slow. Empirically, the rate of convergence, in the applications in this thesis, is far better than the worst case of Proposition 2.9.

2.4 Constrained optimization

Restrict the domain of (2.14) by considering the *primal problem*

$$\begin{aligned} \underset{\boldsymbol{x}\in\mathcal{X}^n}{\text{minimize}} \quad & f(\boldsymbol{x}) \\ \text{subject to} \quad & \boldsymbol{a}(\boldsymbol{x}) \leq \boldsymbol{0}, \\ & \boldsymbol{b}(\boldsymbol{x}) = \boldsymbol{0}, \end{aligned} \tag{2.39}$$

where $\boldsymbol{a} : \mathcal{X}^n \to \mathbf{R}^m$ and $\boldsymbol{b} : \mathcal{X}^n \to \mathbf{R}^\ell$. The set

$$\mathcal{C} = \{\boldsymbol{x} : \boldsymbol{a}(\boldsymbol{x}) \leq \boldsymbol{0}, \boldsymbol{b}(\boldsymbol{x}) = \boldsymbol{0}\} \tag{2.40}$$

is called the *feasible set*. The *Lagrangian* to (2.39) is defined as

$$L(\boldsymbol{x}, \boldsymbol{\gamma}, \boldsymbol{\mu}) = f(\boldsymbol{x}) + \boldsymbol{\gamma}^{\mathrm{T}}\boldsymbol{a}(\boldsymbol{x}) + \boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{b}(\boldsymbol{x}), \tag{2.41}$$

where $\boldsymbol{\gamma} \in \mathbf{R}^m$ and $\boldsymbol{\mu} \in \mathbf{R}^\ell$. For notational convenience introduce

$$\begin{aligned} \boldsymbol{c}(\boldsymbol{x}) &= \begin{bmatrix} \boldsymbol{a}(\boldsymbol{x})^{\mathrm{T}} & \boldsymbol{b}(\boldsymbol{x})^{\mathrm{T}} \end{bmatrix}^{\mathrm{T}}, \\ \boldsymbol{\lambda} &= \begin{bmatrix} \boldsymbol{\gamma}^{\mathrm{T}} & \boldsymbol{\mu}^{\mathrm{T}} \end{bmatrix}^{\mathrm{T}}. \end{aligned} \tag{2.42}$$

Using this we can rewrite the Lagrangian on a more compact form

$$L(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^{\mathrm{T}}\boldsymbol{c}(\boldsymbol{x}), \tag{2.43}$$

where $\boldsymbol{\lambda} \in \mathbf{R}^{m+\ell}$. Furthermore, the *dual function* is defined as

$$d(\boldsymbol{\lambda}) = \min_{\boldsymbol{x} \in \mathcal{X}^n} L(\boldsymbol{x}, \boldsymbol{\lambda}). \tag{2.44}$$

Given any solution $\boldsymbol{x}$, it follows that $f(\boldsymbol{x}) + \boldsymbol{\lambda}^T \boldsymbol{c}(\boldsymbol{x})$ is an affine function in $\boldsymbol{\lambda}$. The dual function is concave in $\boldsymbol{\lambda}$ since it is defined as the pointwise minimum of a set of affine functions. The domain of the dual function is given by

$$\boldsymbol{\Lambda} = \Big\{ \boldsymbol{\lambda} : \boldsymbol{\gamma} \in \{0 \cup \mathbf{R}_+\}^m, \boldsymbol{\mu} \in \mathbf{R}^\ell \Big\}. \tag{2.45}$$

Since the dual function is concave, the *dual problem*

$$\max_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}} d(\boldsymbol{\lambda}), \tag{2.46}$$

is often easier to optimize than the primal problem defined in (2.39).

Proposition 2.10 (Weak duality)**.** *Given the dual problem* (2.46) *to the primal problem* (2.39)*, the following holds*

$$d^\star = \max_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}} d(\boldsymbol{\lambda}) \leq \min_{\boldsymbol{x} \in \mathcal{C}} f(\boldsymbol{x}) = p^\star, \tag{2.47}$$

where $\mathcal{C}$ is the set of feasible solutions *to* (2.39)*.*

Proof. For any $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ we have

$$L(\boldsymbol{\lambda}, \boldsymbol{x}) - f(\boldsymbol{x}) = \boldsymbol{\lambda}^{\mathrm{T}} \boldsymbol{c}(\boldsymbol{x}) \leq 0 \quad \text{for all } \boldsymbol{x} \in \mathcal{C}, \tag{2.48}$$

from which the result follows directly. □

Given any solution $\boldsymbol{\lambda}_0$ to the dual problem, we can extract a primal feasible solution $\boldsymbol{x}_p$ as

$$\boldsymbol{x}_p = P_{\mathcal{C}} \bigg(\operatorname*{argmin}_{\boldsymbol{x} \in \mathcal{X}^n} L(\boldsymbol{x}, \boldsymbol{\lambda}_0) \bigg), \tag{2.49}$$

where $P_{\mathcal{C}}$ is an orthogonal projection on the feasible set $\mathcal{C}$. The feasible solution $\boldsymbol{x}_p$ does not have to be the global minimizer to the primal problem.

By weak duality we have that the *duality gap* is greater than or equal to zero:

$$f(\boldsymbol{x}_p) - f(\boldsymbol{x}_d) \geq 0. \tag{2.50}$$

The duality gap gives a bound on the difference between the current objective value and the optimal objective value.

All models which will be used in this thesis have an arbitrary scaling; minimizing $2f(\boldsymbol{x})$ or $f(\boldsymbol{x})$ will yield the same result but the former will have twice the duality gap. To alleviate this problem, we define the *relative duality gap* as as

$$\frac{|f(\boldsymbol{x}_p) - f(\boldsymbol{x}_d)|}{|f(\boldsymbol{x}_p)|}. \tag{2.51}$$

Lagrangian duality lays the foundation to several methods able to optimize (2.39). The two methods *dual ascent* and *augmented dual ascent* are both based on the *projected supergradient method* and are suitable for a large number of constraints. The *cutting plane method*, on the other hand, is more suited for fewer constraints.

2.4.1 Projected supergradient method

Consider the problem

$$\begin{aligned} &\text{maximize} && d(\boldsymbol{\lambda}) && (2.52)\\ &\text{subject to} && \boldsymbol{\lambda} \in \boldsymbol{\Lambda}, && (2.53) \end{aligned}$$

for some constraint set $\boldsymbol{\Lambda}$. This constrained problem is possible to iteratively maximize using the *projected supergradient method* given by

$$\boldsymbol{x}_{k+1} = P_{\mathcal{C}}\big(\boldsymbol{x}_k + \tau_k \boldsymbol{g}(\boldsymbol{x}_k)\big), \tag{2.54}$$

where $P_{\mathcal{C}}$ is an orthogonal projection on $\mathcal{C}$, $\boldsymbol{g}$ is a subgradient to f at $\boldsymbol{x}_k$ and τ_k is a step length. The convergence results from Section 2.3.5 for the subgradient method can be extended to the projected supergradient method, see Boyd, Xiao, et al. (2003) for details.

2.4.2 Dual ascent

The method of dual ascent minimizes (2.39) by solving the dual problem (2.46) using the projected supergradient method. The dual function in (2.44) is not differentiable, it is however concave. For a general $\mathcal{C}^0$ function, it is difficult to find a supergradient, fortunately (2.44) is an exception.

Proposition 2.11. *Let $\boldsymbol{\lambda}_0$ be given and consider the dual function in* (2.44) *and the constraints $\boldsymbol{c}(\boldsymbol{x})$ in* (2.42)*. Furthermore, let*

$$\boldsymbol{x}^\star = \operatorname*{argmin}_{\boldsymbol{x}\in\mathcal{X}^n} L(\boldsymbol{x}, \boldsymbol{\lambda}_0) = \min_{x\in\mathcal{X}^n} f(\boldsymbol{x}) + \boldsymbol{\lambda}_0^{\mathrm{T}} \boldsymbol{c}(\boldsymbol{x}). \tag{2.55}$$

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

Proof. Given any $\boldsymbol{\lambda}$ weak duality gives

$$\begin{aligned} d(\boldsymbol{\lambda}) &\leq f(\boldsymbol{x}^\star) + \boldsymbol{\lambda}^{\mathrm{T}} \boldsymbol{c}(\boldsymbol{x}^\star) && (2.56)\\ &= f(\boldsymbol{x}^\star) + \boldsymbol{\lambda}_0^{\mathrm{T}} \boldsymbol{c}(\boldsymbol{x}^\star) + \big(\boldsymbol{\lambda} - \boldsymbol{\lambda}_0\big)^{\mathrm{T}} \boldsymbol{c}(\boldsymbol{x}^\star) && (2.57)\\ &= d(\boldsymbol{\lambda}_0) + \big(\boldsymbol{\lambda} - \boldsymbol{\lambda}_0\big)^{\mathrm{T}} \boldsymbol{c}(\boldsymbol{x}^\star), && (2.58) \end{aligned}$$

which simplifies to

$$d(\boldsymbol{\lambda}) - d(\boldsymbol{\lambda}_0) \leq \big(\boldsymbol{\lambda} - \boldsymbol{\lambda}_0\big)^{\mathrm{T}} \boldsymbol{c}(\boldsymbol{x}^\star). \tag{2.59}$$

□

Using Proposition 2.11 and the projected supergradient method, we get the *dual ascent* method as

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \operatorname*{argmin}_{\boldsymbol{x}\in\mathcal{X}^n} L(\boldsymbol{x}, \boldsymbol{\lambda}_k), && (2.60)\\ \boldsymbol{\lambda}_{k+1} &= P_{\mathcal{C}}\big(\boldsymbol{\lambda}_k + \tau_k \boldsymbol{c}(\boldsymbol{x}_{k+1})\big). && (2.61) \end{aligned}$$

2.4.3 Augmented dual ascent

For some classes of problems, dual ascent may converge slowly. To combat this, the Lagrangian is augmented with an additional penalty term $\rho > 0$. Define the *augmented Lagrangian* to (2.43) as

$$L_\rho(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^T \boldsymbol{c}(\boldsymbol{x}) + \frac{\rho}{2} \left\| \boldsymbol{c}(\boldsymbol{x}) \right\|_2^2, \tag{2.62}$$

and the *augmented dual function* as

$$d_\rho(\boldsymbol{\lambda}) = \min_{\boldsymbol{x}\in\mathcal{X}} L_\rho(\boldsymbol{x}, \boldsymbol{\lambda}). \tag{2.63}$$

The set of supergradients for the augmented dual function is identical to that of the dual function (2.44).

Proposition 2.12. *Consider the augmented Lagrangian* (2.62) *and the augmented dual function* (2.63)*. Let* $\boldsymbol{\lambda}_0$ *be given and*

$$\boldsymbol{x}^\star = \operatorname*{argmin}_{\boldsymbol{x}\in\mathcal{X}^n} L_\rho(\boldsymbol{x}, \boldsymbol{\lambda}_0). \tag{2.64}$$

Then $\boldsymbol{c}(\boldsymbol{x}^\star)$ *is a supergradient to* d_ρ *at* $\boldsymbol{\lambda}_0$*.*

Proof. Identical to the proof of Proposition 2.11. □

This leads to an algorithm almost identical to the projected supergradient method:

$$\boldsymbol{x}_{k+1} = \operatorname*{argmin}_{\boldsymbol{x}\in\mathcal{X}^n} L_\rho(\boldsymbol{x}, \boldsymbol{\lambda}_k), \tag{2.65}$$

$$\boldsymbol{\lambda}_{k+1} = P_{\mathcal{C}}\big(\boldsymbol{\lambda}_k + \rho\boldsymbol{c}(\boldsymbol{x}_{k+1})\big). \tag{2.66}$$

The change in step length is motivated by the following example.

Example 2.13. *Suppose that* f *is differentiable, then for an optimal solution we have*

$$0 = \nabla_{\boldsymbol{x}} L_\rho(\boldsymbol{x}_{k+1}, \boldsymbol{\lambda}_k) \tag{2.67}$$

$$= \nabla_{\boldsymbol{x}} f(\boldsymbol{x}_{k+1}) + \nabla_{\boldsymbol{x}} \boldsymbol{c}(\boldsymbol{x}_{k+1})^{\mathrm{T}} \left(\boldsymbol{\lambda}_k + \rho\boldsymbol{c}(\boldsymbol{x}_k)\right) \tag{2.68}$$

$$= \nabla_{\boldsymbol{x}} f(\boldsymbol{x}_{k+1}) + \nabla_{\boldsymbol{x}} \boldsymbol{c}(\boldsymbol{x}_{k+1})^{\mathrm{T}} \boldsymbol{\lambda}_{k+1} \iff \tag{2.69}$$

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_{k+1}) = -\nabla_{\boldsymbol{x}} \boldsymbol{c}(\boldsymbol{x}_{k+1})^{\mathrm{T}} \boldsymbol{\lambda}_{k+1}, \tag{2.70}$$

under the assumption $\boldsymbol{\lambda}_k + \rho\boldsymbol{c}(\boldsymbol{x}_k) \in \mathcal{C}$*. After updating the dual variables, the gradient of the function and the constraints are parallel, which is required for an optimal solution.*

2.4.4 The cutting plane method

Consider the dual problem (2.46). Given any point $\boldsymbol{\lambda}_k \in \boldsymbol{\Lambda}$, let

$$\boldsymbol{x}_k = \underset{\boldsymbol{x} \in \mathcal{X}^n}{\operatorname{argmin}} \, L(\boldsymbol{x}, \boldsymbol{\lambda}_k). \tag{2.71}$$

Using this, we can define a *hyperplane* as

$$d(\boldsymbol{\lambda}_k) - (\boldsymbol{\lambda}_k - \boldsymbol{\lambda})^{\mathrm{T}} \boldsymbol{c}(\boldsymbol{x}_k) = 0. \tag{2.72}$$

If each point in $\boldsymbol{\Lambda}$ is evaluated, then the lower envelope of all hyperplanes would correspond to $d(\boldsymbol{\lambda})$. The lower envelope is defined as

$$\begin{aligned} C^k(\boldsymbol{\lambda}) = \min\big\{ & d(\boldsymbol{\lambda}_0) + (\boldsymbol{\lambda}_0 - \boldsymbol{\lambda})^{\mathrm{T}} \boldsymbol{c}(\boldsymbol{x}_0), \ldots, \\ & d(\boldsymbol{\lambda}_{k-1}) + (\boldsymbol{\lambda}_{k-1} - \boldsymbol{\lambda})^{\mathrm{T}} \boldsymbol{c}(\boldsymbol{x}_{k-1}) \big\}. \end{aligned} \tag{2.73}$$

Using this, the cutting plane method can be expressed as the following iterative algorithm:

$$\boldsymbol{x}_{k+1} = \underset{\boldsymbol{x} \in \mathcal{X}^n}{\operatorname{argmin}} \, L(\boldsymbol{x}, \boldsymbol{\lambda}_k), \tag{2.74}$$

$$\boldsymbol{\lambda}_{k+1} = \underset{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}}{\operatorname{argmax}} \, C^k(\boldsymbol{\lambda}). \tag{2.75}$$

Proposition 2.14. *Assume that the supergradients $\boldsymbol{c}(\boldsymbol{x}_k)$ form a bounded sequence. Then every limit point of the sequence*

$$(\boldsymbol{\lambda}_0, \ldots, \boldsymbol{\lambda}_k) \tag{2.76}$$

generated by the cutting plane method gives the optimal solution to (2.46).

Proof. See Bertsekas (1999). □

The update in (2.75) can be efficiently minimized using linear programming. The downside with the cutting plane method is that we need to solve a growing linear program in each iteration. For a large number of constraints, this might be intractable.

2.5 Decomposition methods

Suppose that a function f lends itself into a natural decomposition of two functions as

$$f(\boldsymbol{x}) = f_1(\boldsymbol{x}) + f_2(\boldsymbol{x}). \tag{2.77}$$

The two optimization approaches presented in this section, *dual decomposition* and *alternating direction method of multipliers*, use Lagrangian duality to split the optimization task into two subproblems; one optimizing f_1 and one optimizing f_2.

2.5.1 Dual decomposition

Given the function in (2.77), consider

$$\begin{aligned} \underset{\boldsymbol{x},\boldsymbol{y}\in\mathcal{X}^n}{\text{minimize}} \quad & f_1(\boldsymbol{x}) + f_2(\boldsymbol{y}) \\ \text{subject to} \quad & \boldsymbol{x} = \boldsymbol{y}. \end{aligned} \tag{2.78}$$

Note that we have introduced a new set of variables $\boldsymbol{y}$ constrained to be equal to $\boldsymbol{x}$. The problem in (2.78) is a special case of (2.39). When (2.78) is optimized using dual ascent, it is often referred to as *dual decomposition*. The iterative algorithm can be expressed as

$$\boldsymbol{x}_{k+1} = \underset{\boldsymbol{x}\in\mathcal{X}^n}{\operatorname{argmin}}\, L(\boldsymbol{\lambda}, \boldsymbol{x}, \boldsymbol{y}_k), \tag{2.79}$$

$$\boldsymbol{y}_{k+1} = \underset{\boldsymbol{y}\in\mathcal{X}^n}{\operatorname{argmin}}\, L(\boldsymbol{\lambda}, \boldsymbol{x}_k, \boldsymbol{y}), \tag{2.80}$$

$$\boldsymbol{\lambda}_{k+1} = \boldsymbol{\lambda}_k + \tau_k(\boldsymbol{x}_{k+1} - \boldsymbol{y}_{k+1}). \tag{2.81}$$

The optimization in (2.79) and (2.80) are independent of each other and can be optimized separately, this is the strength of dual decomposition. Even better is that we are not restricted to only two subproblems. The procedure can be repeated, resulting in a problem decomposed into any number of subproblems.

2.5.2 Alternating direction method of multipliers

A downside with the augmented Lagrangian is that the dual problem (2.63) is not separable. In this section we present the *Alternating Direction Method*

of Multipliers (ADMM) which is similar to performing dual ascent on the augmented Lagrangian, while ignoring the issue of separability. We restrict our attention to linear constraints and consider

$$\begin{aligned} \underset{\boldsymbol{x},\boldsymbol{y}\in\mathcal{X}^n}{\text{minimize}} \quad & f_1(\boldsymbol{x}) + f_2(\boldsymbol{y}) \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{y} = \boldsymbol{z}. \end{aligned} \tag{2.82}$$

The ADMM algorithm performs the following updates

$$\boldsymbol{x}_{k+1} = \underset{\boldsymbol{x}\in\mathcal{X}^n}{\operatorname{argmin}}\, L_\rho(\boldsymbol{\lambda}_k, \boldsymbol{x}, \boldsymbol{y}_k), \tag{2.83}$$

$$\boldsymbol{y}_{k+1} = \underset{\boldsymbol{x}\in\mathcal{X}^n}{\operatorname{argmin}}\, L_\rho(\boldsymbol{\lambda}_k, \boldsymbol{x}_{k+1}, \boldsymbol{y}), \tag{2.84}$$

$$\boldsymbol{\lambda}_{k+1} = \boldsymbol{\lambda}_k + \rho_k \left(\boldsymbol{A}\boldsymbol{x}_{k+1} + \boldsymbol{B}\boldsymbol{y}_{k+1} - \boldsymbol{z}\right). \tag{2.85}$$

Proposition 2.15. *Assume that f_1 and f_2 are convex and that each sublevel set to f_1 and f_2 is closed and bounded. Furthermore, assume that L_0 in* (2.62) *has a saddle point. That is, there exists $(\boldsymbol{x}^\star, \boldsymbol{y}^\star, \boldsymbol{\lambda}^\star)$ for which*

$$L_0(\boldsymbol{x}^\star, \boldsymbol{y}^\star, \boldsymbol{\lambda}) \leq L_0(\boldsymbol{x}^\star, \boldsymbol{y}^\star, \boldsymbol{\lambda}^\star) \leq L_0(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\lambda}^\star) \tag{2.86}$$

holds for all $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\lambda}$. Then

$$f_1(\boldsymbol{x}_k) + f_2(\boldsymbol{y}_k) \to p^\star \text{ as } k \to \infty, \tag{2.87}$$

where $p^\star$ is the optimal solution to (2.85).

Proof. See Boyd, Parikh, et al. (2011). □

The convergence results hold for fixed ρ_k. Letting ρ_k increase with k can improve convergence speed and the convergence results for the algorithm hold, as long as ρ_k eventually converges.

It is worth noting that the underlying mechanic behind the convergence proof of (2.15) is not based on dual ascent, see Eckstein (2012). This is further emphasized by the fact that we need to restrict ourselves to linear constraints. Furthermore, ADMM does not generalize to three subproblems such as

$$f(\boldsymbol{x}) = f_1(\boldsymbol{x}) + f_2(\boldsymbol{x}) + f_3(\boldsymbol{x}). \tag{2.88}$$

This decomposition may result in a non-convergent algorithm, see Chen et al. (2014).

Remark 2.16. *Many methods discussed in this chapter will converge to the global optimum under certain conditions. In practice, all of these conditions may not be fulfilled. For instance, the models used for computer vision are seldom convex. However, applying methods on non-convex functions may still make sense. For methods based on duality, the duality gap still holds and after the algorithm has terminated we know how far away we are from the global optimum even if the function is non-convex.*

2.6 Boolean optimization

In this section we consider the family of functions which can be expressed as

$$f : \mathbf{B}^n \to \mathbf{R}, \tag{2.89}$$

where $\mathbf{B} = \{0, 1\}$. A function on this form is known as a *pseudo-boolean function* and can uniquely be expressed as a multi-linear polynomial,

$$\begin{aligned} f(\boldsymbol{x}) = a_0 + \sum_{i=1}^{n} a_i x_i + \sum_{1 \le i < j \le n} a_{ij} x_i x_j \\ + \sum_{1 \le i < j < \ell \le n} a_{ijl} x_i x_j x_l + \cdots . \end{aligned} \tag{2.90}$$

The *order* of the pseudo-boolean function is the degree of the polynomial.

Minimization of f is known to be NP-hard so we will have to rely on approximative algorithms or consider subsets of the family. A subset of functions which is easier to optimize are the *submodular functions*.

Definition 2.17. *Let $\vee$ be element-wise maximum and $\wedge$ element-wise minimum. A function f is* submodular *if*

$$f(\boldsymbol{x} \vee \boldsymbol{y}) + f(\boldsymbol{x} \wedge \boldsymbol{y}) \le f(\boldsymbol{x}) + f(\boldsymbol{y}), \quad \textit{for all } \boldsymbol{x}, \boldsymbol{y} \in \mathbf{B}^n. \tag{2.91}$$

An equivalent and sometimes more convenient characterization is

$$\frac{\partial^2 f}{\partial x_i \partial x_j}(\boldsymbol{x}) \le 0, \textit{ for all } 1 \le i < j \le n \textit{ and } \boldsymbol{x} \in \mathbf{B}^n, \tag{2.92}$$

where $\frac{\partial f}{\partial x}$ is defined as the symbolic derivative of (2.90)*, see Strandmark (2012).*

We will briefly outline the current state of the art for minimizing submodular functions in the field of computer vision. Typically the number of variables is very large so general methods such as the ellipsoid method, see Fujishige (2005), will be intractable for real-world applications.

Order 2**.** Ivunescu (1965) showed that the problem of minimizing a second-order submodular function can be reformulated into a max-flow problem, which can be efficiently optimized. In the next section we will describe this reformulation.

Order 3**.** Billionnet and Minoux (1985) showed how to reformulate a third-order submodular function into a second-order submodular function by introducing additional variables. This reformulation is known as a *reduction*.

Order ≥ 4**.** Any submodular function can be described as

$$f(\boldsymbol{x}) = \sum_{a \in \mathcal{F}} f_a(\boldsymbol{x}), \tag{2.93}$$

where each $f_a(\boldsymbol{x})$ is submodular. In Kolmogorov (2012) an efficient algorithm is given if the order of each f_a is low. This is the typical form of submodular functions of interest for computer vision.

Remark 2.18. *The results of Ivunescu (1965) and Billionnet and Minoux (1985) were popularized in computer vision community by Boykov, Veksler, et al. (2001) and Kolmogorov and Zabih (2004).*

2.6.1 Submodular functions of order 2

Let G be a graph with vertices, V, connected by directed edges, E, where each edge has an associated weight, $w_{ij} \geq 0$, connecting vertex i to vertex j. There are two special vertices; the source s and the sink t. This graph construction is referred to as an *st-graph*. A cut in the graph is a partition of the vertices into two sets S and T such that $S \cup T = V$, $S \cap T = \emptyset$ and $s \in S$ and $t \in T$. The value of the cut is the sum of all edges going from S to T. Finding minimum-cuts is a well studied area with very efficient

algorithms. For computer vision problems there are special implementations with attractive execution times see Boykov and Kolmogorov (2004); Goldberg et al. (2011).

Let $\boldsymbol{x}$ be an indicator variable where

$$x_i = \begin{cases} 0 & \text{if } i \in S, \\ 1 & \text{if } i \in T. \end{cases} \tag{2.94}$$

Then any minimum-cut problems can be expressed as

$$\min_{\boldsymbol{x} \in \mathbf{B}^n} \sum_{i=1}^{n} w_{it}(1 - x_i) + \sum_{i=1}^{n} w_{si} x_i + \sum_{1 \leq i < j \leq n} w_{ij}(1 - x_i)x_j, \tag{2.95}$$

which is equivalent to,

$$\begin{aligned} \min_{\boldsymbol{x} \in \mathbf{B}^n} & \sum_{i=1}^{n} \left(w_{si} - w_{it}\right) x_i + \sum_{i=1}^{n} w_{it} \\ & + \sum_{1 \leq i < j \leq n} w_{ij} x_j + \sum_{1 \leq i < j \leq n} (-w_{ij}) x_i x_j. \end{aligned} \tag{2.96}$$

Proposition 2.19. *Any second-order submodular function on the form* (2.90) *with a specific constant* a_0 *can be transformed to the form in* (2.96).

Proof. Using the definition of a submodular function we know that each $a_{ij} \leq 0$. For the binary term set

$$w_{ij} = -a_{ij}, \tag{2.97}$$

it follows that $w_{ij} \geq 0$. For any unary term set

$$w_i = \sum_{j=1}^{n} -a_{ij} + a_i, \tag{2.98}$$

and then set

$$w_{si} = \begin{cases} w_i & \text{if } w_i \geq 0, \\ 0 & \text{if } w_i < 0, \end{cases} \tag{2.99}$$

$$w_{it} = \begin{cases} 0 & \text{if } w_i \geq 0, \\ w_i & \text{if } w_i < 0. \end{cases} \tag{2.100}$$

By construction $w_{si} - w_{it} \geq 0$. Finally we have equality if the constant a_0 fulfills

$$a_0 = \sum_{i=1}^{n} w_{it}. \tag{2.101}$$

□

Corollary 2.20. *Any second-order submodular function on the form* (2.90) *can be minimized by performing minimum cut on a st-graph.*

Proof. As the proposition shows (2.90) and (2.96) are equivalent up to known constant. The constant will not influence the minimizer to (2.96). □

In the computer vision community using Corollary (2.20) is usually referred to as performing *graph-cuts*.

2.6.2 General functions

We will now turn our attention on how to minimize non-submodular functions via a relaxation known as *roof duality*. The idea behind roof duality is to find a submodular lower bound relaxation, g, to the original function f. The idea was originally given in Hammer et al. (1984) for second-order functions. It was later extended to higher-order functions in Kolmogorov (2012). Kahl and Strandmark (2012) were the first to show how to efficiently find the submodular relaxation for higher-order functions. Using roof duality for higher-order functions will be referred to as performing *generalized roof duality*.

Remark 2.21. *In Hammer et al. (1984) maximization of functions were considered and g was a upper bound on the function value, hence the name* roof *dual.*

The list of properties we wish our submodular relaxation to have is

1. $g : \mathbf{B}^{2n} \to \mathbf{R}$ is submodular.

2. $g(\boldsymbol{x}, \overline{\boldsymbol{x}}) = f(\boldsymbol{x}) \quad$ for all $\boldsymbol{x} \in \mathbf{B}^n$

3. $g(\boldsymbol{x}, \boldsymbol{y}) = g(\overline{\boldsymbol{y}}, \overline{\boldsymbol{x}}) \quad$ for all $(\boldsymbol{x}, \boldsymbol{y}) \in \mathbf{B}^{2n}$ (symmetry),

where $\overline{\boldsymbol{x}} = \mathbf{1} - \boldsymbol{x}$.

Note that the relaxation g has twice as many variables as f. But since g is submodular it is easy to optimize. Let

$$(\boldsymbol{x}^\star, \boldsymbol{y}^\star) = \operatorname*{argmin}_{(\boldsymbol{x},\boldsymbol{y})\in\mathbf{B}^{2n}} g(\boldsymbol{x}, \boldsymbol{y}). \tag{2.102}$$

It follows by construction that

$$\min_{(\boldsymbol{x},\boldsymbol{y})\in\mathbf{B}^{2n}} g(\boldsymbol{x}, \boldsymbol{y}) \leq \min_{\boldsymbol{x}\in\mathbf{B}^n} g(\boldsymbol{x}, \overline{\boldsymbol{x}}) = \min_{\boldsymbol{x}\in\mathbf{B}^n} f(\boldsymbol{x}). \tag{2.103}$$

The minimum of g works as a lower bound for the minimum of the original function f. By construction there is some freedom in how to choose our lower bounding function g. A natural choice is to choose the lower bound giving us the highest value:

$$\begin{aligned} \underset{g,\ell}{\text{maximize}} \quad & \ell \\ \text{subject to} \quad & g(\boldsymbol{x}, \boldsymbol{y}) \geq \ell, \quad \text{for all}(\boldsymbol{x}, \boldsymbol{y}) \in \mathbf{B}^{2n}, \\ & g \text{ satisfies } (1) - (3). \end{aligned} \tag{2.104}$$

Proposition 2.22. *Restricting g to be symmetric (property 3) has no effect on the maximum of the lower bound.*

Proof. See Kahl and Strandmark (2012). □

For the second-order case (2.104) has a closed form solution. Every symmetric second-order g can be written on the form

$$\begin{aligned} g(\boldsymbol{x}, \boldsymbol{y}) = & \frac{1}{2} \sum_i b_i (x_i + \bar{y}_i) + \sum_i b_{ii} x_i \bar{y}_i \\ & + \frac{1}{2} \sum_{i<j} \big(b_{ij} (x_i x_j + \bar{y}_i \bar{y}_j) + c_{ij} (x_i \bar{y}_j + \bar{y}_i x_j) \big). \end{aligned} \tag{2.105}$$

Proposition 2.23. *An optimal submodular relaxation g, of a second-order pseudo-boolean function, in the sense of* (2.104)*, is given by* (2.105) *with the coefficients*

$$b_i = a_i \qquad \textit{for } 1 \leq i \leq n, \tag{2.106}$$
$$b_{ii} = 0 \qquad \textit{for } 1 \leq i \leq n, \tag{2.107}$$
$$b_{ij} = \min(a_{ij}, 0) \qquad \textit{for } 1 \leq i < j \leq n, \tag{2.108}$$
$$c_{ij} = \max(a_{ij}, 0) \qquad \textit{for } 1 \leq i < j \leq n. \tag{2.109}$$

Proof. See Kahl and Strandmark (2012). □

This result is equivalent to the construction in Boros and Hammer (2002) which is used in Rother et al. (2007). Before we consider higher-order functions we need some of the most useful results of the relaxation.

Definition 2.24. *For any $\boldsymbol{x} \in \mathbf{B}^n$ and $(\boldsymbol{x}^\star, \boldsymbol{y}^\star)$ we define the* overwrite *operator $\mathbf{B}^n \times \mathbf{B}^{2n} \to \mathbf{B}^n$ as*

$$\boldsymbol{x} \leftarrow (\boldsymbol{x}^\star, \boldsymbol{y}^\star) = \boldsymbol{u} \tag{2.110}$$

where

$$u_i = \begin{cases} x_i^\star & \textit{if } x_i^\star \neq y_i^\star, \\ x_i & \textit{otherwise}, \end{cases} \qquad \textit{for } i = 1, \ldots, n. \tag{2.111}$$

Proposition 2.25 (Autarky)**.** *Let g be a function satisfying* $(1) - (3)$ *and*

$$(\boldsymbol{x}^\star, \boldsymbol{y}^\star) \in \operatorname{argmin} g. \tag{2.112}$$

Then $f\left(\boldsymbol{x} \leftarrow (\boldsymbol{x}^\star, \boldsymbol{y}^\star)\right) \leq f(\boldsymbol{x})$ for all $\boldsymbol{x}$.

Proof. See Kolmogorov (2012); Hammer et al. (1984). □

The result gives us a way to iteratively minimize f using proposed relaxations g.

Corollary 2.26 (Persistency)**.** *Let g be a function satisfying (1)-(3) and*

$$(\boldsymbol{x}^\star, \boldsymbol{y}^\star) \in \operatorname{argmin} g. \tag{2.113}$$

If $\boldsymbol{x} \in \operatorname{argmin}(f)$, then $\boldsymbol{x} \leftarrow (\boldsymbol{x}^\star, \boldsymbol{y}^\star) \in \operatorname{argmin}(f)$.

Proof. See Kahl and Strandmark (2012). □

This result gives us *partial optimality*. All variables replaced by the overwrite function are equal to some solution $\boldsymbol{x}^\star \in \operatorname{argmin}(f)$.

Definition 2.27. *Every variable replaced by a overwrite function is said to be* labeled. *All other variables as said to be* unlabeled.

Remark 2.28. *A minimizer of g that labels all variables gives us the optimal solution.*

In Kahl and Strandmark (2012) relaxations for higher-order functions are found by simplifying (2.104) and iteratively reducing the number of variables using persistency. In each of these iterations a large linear program needs to be solved to find g. This might be more expensive than to minimize the relaxed function g. To combat this Kahl and Strandmark (2012) also introduce a number of heuristic ways to find g. The lower bound function given might not be optimal but the persistency and autarky results still hold making these heuristics useful in some applications.

2.7 Multi-label optimization

In this section we consider the family of function which can be expressed as

$$f : \boldsymbol{L}^n \to \mathbf{R}, \tag{2.114}$$

where $\boldsymbol{L} = \{0, \ldots, k\}$. Each possible value of $\boldsymbol{L}$ will be referred to as a *label* and each possible solution $\boldsymbol{x}$ will be referred to as a *assignment*. Generally the functions we are interested in can be expressed as

$$f(\boldsymbol{x}) = \sum_{a \in \mathcal{F}} f_a(\boldsymbol{x}_a), \tag{2.115}$$

where $\boldsymbol{x}_a$ is the restriction of $\boldsymbol{x}$ to the *factor* a, representing a subset of $\boldsymbol{x}$, $\mathcal{F}$ is a set of factors, and

$$f_a : \boldsymbol{L}^{|a|} \to \mathbf{R}. \tag{2.116}$$

The largest number of variables covered by any factor is known as the *order*, or arity, of the function. For each variable x_i, we will introduce a vector of indicator variables, $\boldsymbol{x}_i = \big[\boldsymbol{x}_i(0) \dots \boldsymbol{x}_i(k)\big]$, with the property

$$\boldsymbol{x}_i(j) = \begin{cases} 1 & \text{if } x_i = j, \\ 0 & \text{otherwise.} \end{cases} \tag{2.117}$$

We will use boldface $\boldsymbol{x}_i$ to denote indicator variables and regular x_i to denote the original variables. The indicator variables are a useful tool when analyzing different methods used to minimize (2.115). More on this later in the chapter.

Remark 2.29. *For boolean variables x_i is an indicator variable.*

We will start to focus on function consisting of symmetric factors of order at most two. They can be expressed as

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} U_i(x_i) + \sum_{i=1}^{n} \sum_{j=1}^{n} B_{ij}(x_i, x_j), \tag{2.118}$$

where $U_i : \boldsymbol{L} \to \mathbf{R}$ and $B_{ij} : \boldsymbol{L} \times \boldsymbol{L} \to \mathbf{R}$ are the *unary* and *binary* terms respectively. We will only consider symmetric binary terms, that is

$$B_{ij}(x_i, x_j) = B_{ji}(x_j, x_i) \quad \text{for all } i \text{ and } j. \tag{2.119}$$

It is possible to represent the function in (2.115) as a graph where each variable, and indirectly each unary term, is represented by a vertex. Furthermore each binary term B_{ij}, is represented as an edge connecting the two vertices associated with the variables x_i and x_j. Figure 2.7 gives an example construction. This representation of (2.118) will be used throughout this chapter.

2.7.1 Dynamic programming

We will now focus on how to find the minimizer of (2.118) if the function can be represented as a tree graph with a distinct root. For this subset of functions we are able to find the global minimizer to (2.118) using dynamic programming.

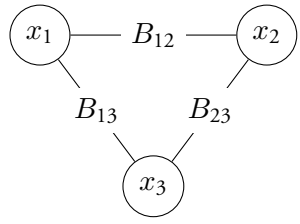


Figure 2.7: A graph representation of the function $f(\boldsymbol{x}) = U_1(x_1) + U_2(x_2) + U_3(x_3) + B_{12}(x_1, x_2) + B_{23}(x_1, x_2) + B_{13}(x_1, x_3)$.

Let $d(i, j)$ be the number of edges on the shortest path from vertex i and j in the graph representation. Number the variable such that n is the root vertex of the tree and for each vertex i define a parent $p(i)$, the children $\boldsymbol{c}(i)$ and all descendants $\boldsymbol{d}(i)$ as

$$\begin{aligned} \boldsymbol{c}(i) &= \{j : B_{ij} \neq 0 \text{ and } d(i,n) - d(j,n) = 1\}, \\ \boldsymbol{d}(i) &= \{j : B_{ij} \neq 0 \text{ and } d(i,n) - d(j,n) > 0\}, \\ p(i) &= \{j : B_{ij} \neq 0 \text{ and } d(i,n) - d(j,n) = -1\}. \end{aligned} \tag{2.120}$$

The functions we consider in this section can now be expressed as

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} \left(U_i(x_i) + \sum_{j \in \boldsymbol{c}(i)} B_{ij}(x_i, x_j) \right). \tag{2.121}$$

The minimization of $f(\boldsymbol{x})$ can be split into recursive subproblems as,

$$D_k(y) = \min_{\{\boldsymbol{x} \,:\, x_k = y\}} \left(\sum_{i \in \{\boldsymbol{d}(k), k\}} \left(U_i(x_i) + \sum_{j \in \boldsymbol{c}(i)} B_{ij}(x_i, x_j) \right) \right). \tag{2.122}$$

Note that

$$\min_{y \in \boldsymbol{L}} D_n(y) = \min_{\boldsymbol{x} \in \boldsymbol{L}^n} f(\boldsymbol{x}). \tag{2.123}$$

Examples of all these definitions are given in Figure 2.8. The following proposition gives us an efficient algorithm for minimizing $D_k(y)$.

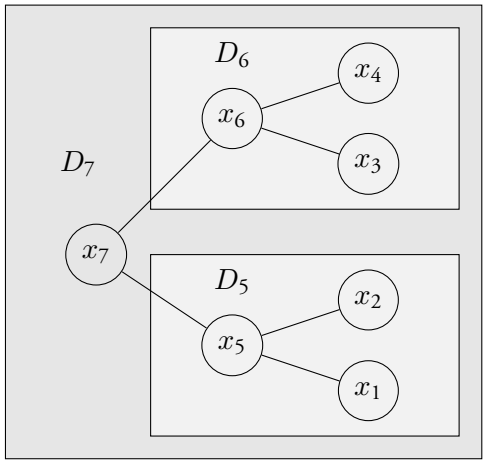


Figure 2.8: Example graph representation of $f(\boldsymbol{x})$ with the subproblem D_7, D_6, and D_5 in (2.30) marked. Each vertex corresponds to a variable and each edge corresponds to a binary term. x_7 is the root. $\boldsymbol{c}(7) = \{5, 6\}$, $p(7) = \emptyset$ and $\boldsymbol{d}(7) = \{1, 2, 3, 4, 5, 6\}$.

Proposition 2.30. *For any $k \in \{1, \ldots, n\}$, the function $D_k(y)$ defined in* (2.122) *can be computed as*

$$D_k(y) = \begin{cases} U_k(y) & \boldsymbol{c}(k) = \emptyset \\ U_k(y) + \min\limits_{\{\boldsymbol{x} \,:\, x_k = y\}} \sum\limits_{i \in \boldsymbol{c}(k)} \big(B_{ki}(y, x_i) + D_i(x_i) \big) & \boldsymbol{c}(k) \neq \emptyset, \end{cases}$$

with the notation given in (2.120).

Proof.

$$
\begin{aligned}
D_k(y) &= \min_{\{\boldsymbol{x}\,:\,x_k=y\}} \left(\sum_{i\in\{\boldsymbol{d}(k),k\}} \left(U_i(x_i) + \sum_{j\in\boldsymbol{c}(i)} B_{ij}(x_i,x_j) \right) \right). \\
&= U_k(y) + \min_{\{\boldsymbol{x}\,:\,x_k=y\}} \left(\sum_{i\in\boldsymbol{c}(k)} B_{ki}(y,x_i) \right. \\
&\left. + \sum_{i\in\boldsymbol{d}(k)} \left(U_i(x_i) + \sum_{j\in\boldsymbol{c}(i)} B_{ij}(x_i,x_j) \right) \right) \\
&= U_k(y) + \min_{\{\boldsymbol{x}\,:\,x_k=y\}} \sum_{i\in\boldsymbol{c}(k)} \left(B_{ki}(y,x_i) + D_i(x_i) \right) \qquad (2.124)
\end{aligned}
$$

□

Using Proposition (2.30) we can calculate the optimal value $f(\boldsymbol{x}^\star)$ in an efficient manner. What we most of the time are aiming for however is to recover $\boldsymbol{x}^\star = \operatorname{argmin}_{\boldsymbol{x}} f(\boldsymbol{x})$. The next proposition helps us with that.

Proposition 2.31. *Given a function of the form in* (2.118)*. The optimal solution*

$$
\boldsymbol{x}^\star = \operatorname*{argmin}_{\boldsymbol{x}\in\boldsymbol{L}^n} f(\boldsymbol{x}), \qquad (2.125)
$$

can be computed as

$$
x_k^\star = \begin{cases} \operatorname*{argmin}_{y\in\boldsymbol{L}} D_k(y) & p(k) = \emptyset \\ \operatorname*{argmin}_{y\in\boldsymbol{L}} D_k(y) + B_{p(k)k}\left(y, x^\star_{p(k)}\right) & p(k) \neq \emptyset, \end{cases} \qquad (2.126)
$$

where $D_k(y)$ is defined in (2.122) *and the $p(k)$ is defined in* (2.120)*.*

Remark 2.32. *If each $x_k^\star$ is calculated breadth-first starting at the root, $x^\star_{p(k)}$ is always known.*

Proof. If $p(k) = \emptyset$ the result follows from the definition. For any other k consider the set

$$
\mathcal{C} = \big\{ \{\boldsymbol{d}(n), n\} \backslash \{\boldsymbol{d}(k), k\} \big\}. \qquad (2.127)
$$

It follows that

$$x_k^\star = \operatorname*{argmin}_{y \in \boldsymbol{L}} \min_{\{\boldsymbol{x} \,:\, x_k = y\}} f(\boldsymbol{x}) \tag{2.128}$$

$$= \operatorname*{argmin}_{y \in \boldsymbol{L}} \min_{\{\boldsymbol{x} \,:\, x_k = y\}} \sum_{i \in \{\boldsymbol{d}(n), n\}} \left(U_i(x_i) + \sum_{j \in \boldsymbol{c}(i)} B_{ij}(x_i, x_j) \right) \tag{2.129}$$

$$= \operatorname*{argmin}_{y \in \boldsymbol{L}} \min_{\{\boldsymbol{x} \,:\, x_k = y\}} \left(\sum_{i \in \{\boldsymbol{d}(k), k\}} \Big(U_i(x_i) + \sum_{j \in \boldsymbol{c}(i)} B_{ij}(x_i, x_j) \Big) \right. \tag{2.130}$$

$$\left. + \underbrace{\sum_{i \in \mathcal{C}} \Big(U_i(x_i) + \sum_{j \in \boldsymbol{c}(i)} B_{ij}(x_i, x_j) \Big)}_{\text{independent of } y} \right)$$

$$= \operatorname*{argmin}_{y \in \boldsymbol{L}} \Big(D_k(y) + B_{p(k)k}\Big(y, x^\star_{p(k)}\Big) \Big). \tag{2.131}$$

□

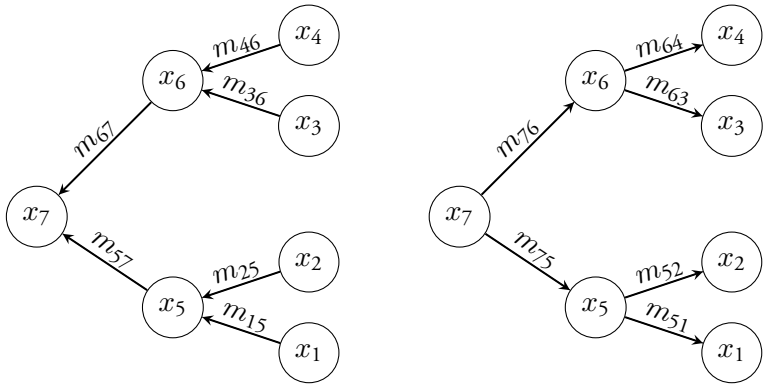
We can now summarize this section.

Minimize (2.121) using dynamic programming.

1. Use Proposition 2.30 to compute D_k for every variable k.
2. Use Proposition 2.31 to recover the optimal solution $\boldsymbol{x}^\star$.

2.7.2 Belief propagation

In the statistical literature minimizing functions of form (2.118) is usually done via belief propagation, see e.g. Weiss and Freeman (2001). In this section we will show the connection between belief propagation and the dynamic programming approach discussed in the previous section. We will once again only consider functions which can be represented by a tree graph. A message from variable i to variable j is defined as



(a) Inward pass. (b) Outward pass.

Figure 2.9: The order in which the messages are passed for the inward and outward pass. The message are calculated sequentially in breadth-first-order. For the inward pass m_{46}, m_{36}, and m_{25} are calculated before m_{67} and m_{57}. Similarly for the outward pass m_{67} and m_{57} are calculated before m_{46}, m_{36}, m_{25}, and m_{15}.

$$m_{ij}(x_j) = \min_{x_i} \left(U_i(x_i) + B_{ij}(x_i, x_j) + \sum_{\ell \in \{\mathcal{N}(i) \setminus j\}} m_{\ell i}(x_i) \right), \tag{2.132}$$

where $\mathcal{N}(i) = \{\boldsymbol{c}(i), p(i)\}$ and each $m_{ij}(x_j)$ is initialized to be zero. Calculating $m_{ij}(x_i)$ for all x_i will be referred to as *passing the message vector* from i to j. The messages are passed first from the leaf variables towards the root. This is known as the *inward pass*. After that the messages are passed from the root towards each leaf. This is known as the *outward pass*. These two passes are illustrated in Figure 2.9.

Proposition 2.33. *Consider D_k defined in* (2.30)*. For the inward pass*

$$D_k(y) = U_k(y) + \sum_{j \in \boldsymbol{c}(k)} m_{jk}(y), \tag{2.133}$$

where the messages m is defined in (2.132)*.*

Proof. Proof by induction. For the inward pass we have

$$\{\mathcal{N}(i) \backslash p(i)\} = \boldsymbol{c}(i). \tag{2.134}$$

If $\boldsymbol{c}(k) = \emptyset$ then it holds by definition of D_k. Now assume it hold for all variables in $\boldsymbol{d}(k)$. By definition it then holds for all variables in $\boldsymbol{c}(k)$. From Proposition 2.30 and using symmetry of the binary term we have

$$\begin{aligned} D_k(y) &= U_k(y) + \min_{\{\boldsymbol{x} \,:\, x_k = y\}} \sum_{i \in \boldsymbol{c}(k)} \Big(B_{ik}(x_i, y) + D_i(x_i) \Big) && (2.135) \\ &= U_k(y) + \\ &\min_{\{\boldsymbol{x} \,:\, x_k = y\}} \sum_{i \in \boldsymbol{c}(k)} \Big(B_{ik}(x_i, y) + U_i(x_i) + \sum_{j \in \boldsymbol{c}(i)} m_{ji}(y) \Big) && (2.136) \\ &= U_k(y) + \sum_{j \in \boldsymbol{c}(k)} m_{jk}(y). && (2.137) \end{aligned}$$

□

Definition 2.34. *The* min-marginals *for a variable k corresponds to*

$$\min_{\{\boldsymbol{x} \,:\, x_k = s\}} f(\boldsymbol{x}), \tag{2.138}$$

for any $s \in \boldsymbol{L}$.

Corollary 2.35. *Let n be the root of the tree, then*

$$\min_{\{\boldsymbol{x} \,:\, x_n = y\}} f(\boldsymbol{x}) = U_n(y) + \sum_{j \in \mathcal{N}(n)} m_{jn}(y) \tag{2.139}$$

Theorem 2.36. *After both the inward and outward messages are passed, the min-marginals are given by*

$$\min_{\{\boldsymbol{x} \,:\, x_i = y\}} f(\boldsymbol{x}) = U_i(y) + \sum_{j \in \mathcal{N}(i)} m_{ji}(y), \tag{2.140}$$

where the messages m is defined in (2.132).

Proof. We begin to note that from Proposition 2.33

$$m_{ip}(y) = \min_{\{\boldsymbol{x}\,:\,x_p=y\}} \Big(U_i(x_i) + B_{ip}(x_i, y) + \sum_{\ell \in \boldsymbol{c}(i)} m_{\ell i}(x_i) \Big), \quad (2.141)$$

$$= \min_{\{\boldsymbol{x}\,:\,x_p=y\}} \Big(D_i(x_i) + B_{ip}(x_i, y) \Big). \quad (2.142)$$

By corollary 2.35 it holds for the root. Now assume that it holds for variable $p = p(i)$. From Proposition 2.33 we can rewrite it as follows

$$U_i(y) + \sum_{\ell \in \mathcal{N}(i)} m_{\ell i}(y) \quad (2.143)$$

$$= D_i(y) + m_{pi}(y) \quad (2.144)$$

$$= D_i(y) + \min_{\{\boldsymbol{x}\,:\,x_i=y\}} \Big(U_p(x_p) + B_{pi}(x_p, y) + \sum_{\ell \in \mathcal{N}(p)} m_{\ell p}(x_p) - m_{ip}(x_p) \Big). \quad (2.145)$$

$$= D_i(y) + \min_{\{\boldsymbol{x}\,:\,x_i=y\}} \Bigg(U_p(x_p) + B_{pi}(x_p, y) + \sum_{\ell \in \mathcal{N}(p)} m_{\ell p}(x_p) - \min_z \Big(D_i(z) + B_{ip}(z, x_p) \Big) \Bigg). \quad (2.146)$$

Reordering and using symmetry of B_{ip} we get

$$\min_{\{\boldsymbol{x}\,:\,x_i=y\}} \Bigg(U_p(x_p) + \sum_{\ell \in \mathcal{N}(p)} m_{\ell p}(x_p) + D_i(y) + B_{ip}(y, x_p) - \min_z \big(D_i(z) + B_{ip}(z, x_p) \big) \Bigg). \quad (2.147)$$

Recall that $D_i(y)$ handles the subset of the function rooted at variable i. We can add terms to extend this to include all variables in f. Furthermore

$$\min_{\boldsymbol{x}_p} \big(U_p(x_p) + \sum_{\ell \in \mathcal{N}(p)} m_{\ell p}(x_p) \big) = \min_{\boldsymbol{x}} f(\boldsymbol{x}), \quad (2.148)$$

by construction. This allows us to rewrite (2.147) into,

$$\min_{\boldsymbol{x}} \big(f(\boldsymbol{x}) + f(\boldsymbol{x})_{|x_i=y} - f(\boldsymbol{x})\big) = \min_{\{\boldsymbol{x}\,:\,x_i=y\}} f(\boldsymbol{x}), \tag{2.149}$$

where $f(\boldsymbol{x})_{|x_i=y}$ is the restriction of f to $x_i = y$. □

We can now summarize this section.

Minimize (2.121) using belief propagation.

1. Starting at the leaf variables send messages inwards towards the root variable using (2.132), until all inward message has been sent.
2. Starting at the root variable send message outwards towards the leaf variables using (2.132), until all outward messages has been sent.
3. Use Theorem 2.36 to recover the optimal solution $\boldsymbol{x}^\star$.

Belief propagation is exact on the subset of problems to which we have restricted our attention. In practice it has however often been used on general problems where the connectivity cannot described as a tree, see e.g. Tappen and Freeman (2003). When this method is applied it usually goes under the name loopy belief propagation (LBP). LBP is not guaranteed to yield an optimal solution, nor give any lower bound or even converge.

Reparametrization

For optimization schemes based on *message passing*, like belief propagation, it is possible to save memory by storing the message implicitly by reparametrization of the problem, see Wainwright et al. (2004); Rother et al. (2007).

In this section we will denote the function in (2.118) as $f_{\boldsymbol{w}}$, where $\boldsymbol{w}$ consist of all functions $U_i(x_i)$ and $B_{ij}(x_i, x_j)$ defining f in (2.118). If

$$f_{\boldsymbol{w}}(\boldsymbol{x}) = f_{\boldsymbol{w}'}(\boldsymbol{x}) \text{ for all } \boldsymbol{x}, \tag{2.150}$$

then $\boldsymbol{w}'$ is a *reparametrization* of $\boldsymbol{w}$.

The goal of this section is to reparametrize the function such that,

$$U_i'(x_i) = U_i(x_i) + \sum_{j \in \mathcal{N}(i)} m_{ji}(x_i), \qquad \text{for all } x_i \in \boldsymbol{L}. \tag{2.151}$$

That is the unary terms correspond to the min-marginals. By performing this reparametrization it is possible to implement message based optimization without explicitly storing the message during optimization. Suppose that we have calculated m_{ij}, the messages from vertex i to vertex j. Then instead of storing it, we can reparametrize the function as follows; for each $x_i \in \boldsymbol{L}$ set

$$\begin{aligned} U_i'(x_i) &= U_i(x_i) + m_{ij}(x_i), \\ B_{ij}'(x_i, x_j) &= B_{ij}(x_i, x_j) - m_{ij}(x_i) \quad \text{for all } x_j \in \boldsymbol{L}. \end{aligned} \tag{2.152}$$

This is repeated for each message going sent to i. The final reparametrization becomes

$$\begin{aligned} U_i'(x_i) &= U_i(x_i) + \sum_{\ell \in \mathcal{N}(i)} m_{\ell i}(x_i), \\ B_{ij}'(x_i, x_j) &= B_{ij}(x_i, x_j) - m_{ij}(x_i) \qquad \text{for all } x_j \in \boldsymbol{L}. \end{aligned} \tag{2.153}$$

Proposition 2.37. *After the reparametrization given in* (2.153) *the message in* (2.132) *is given by*

$$m_{ij}(x_j) = \min_{x_i} \left(U_i'(x_i) + B_{ij}'(x_i, x_j) \right). \tag{2.154}$$

Proof.

$$\min_{x_i} \left(U_i'(x_i) + B_{ij}'(x_i, x_j) \right) \tag{2.155}$$

$$= \min_{x_i} \left(U_i(x_i) + \sum_{\ell \in \mathcal{N}(i)} m_{\ell i}(x_i) + B_{ij}(x_i, x_j) \quad - m_{ij}(x_j) \right) \tag{2.156}$$

$$= \min_{x_i} \left(U_i(x_i) + B_{ij}(x_i, x_j) + \sum_{\ell \in \{\mathcal{N}(i) \setminus j\}} m_{\ell i}(x_i) \right) \tag{2.157}$$

$$= m_{ij}(x_j). \tag{2.158}$$

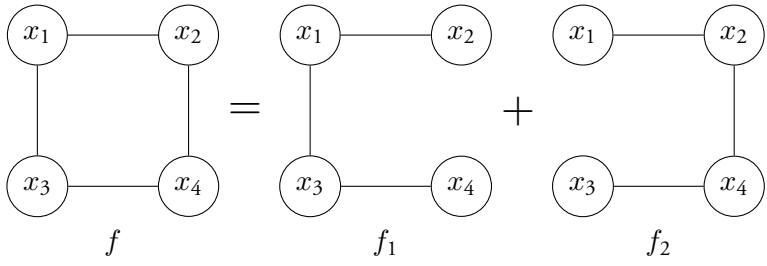


Figure 2.10: Example decomposition of f into f_1 and f_2. All functions are represented as graphs, where there the unary and binary terms are represented by vertices and edges respectively. It is straightforward to construct unary and binary terms in f_1 and f_2 such that $f(\boldsymbol{x}) = f_1(\boldsymbol{x}) + f_2(\boldsymbol{x})$ for all $\boldsymbol{x}$.

□

This proposition shows that (2.132) can be optimized using only the current parameterization, the messages are not needed to be stored.

Efficient message passing

If each message in (2.132) is calculated using brute-force, then $\mathcal{O}\big(|L|^2\big)$ evaluations are performed. For problems with many labels this might be intractable. If we restrict our binary functions to

$$B_{ij}(x_i, x_j) = \min\big(c|x_i - x_j|, \tau\big) \quad \text{or} \tag{2.159}$$

$$B_{ij}(x_i, x_j) = \min\big(c(x_i - x_j)^2, \tau\big), \tag{2.160}$$

where $\tau, c \in \mathbf{R}$; then Felzenszwalb and Huttenlocher (2006) showed how the messages can be found using $\mathcal{O}(|L|)$ evaluations via a distance transform.

2.7.3 Decomposition methods

In this section we will use Lagrangian duality to derive two efficient algorithms able to minimize arbitrary functions on the form given in (2.118). The resulting algorithms can be seen as a combination of the efficient solvers

for tree structure problems, and the decomposition methods discussed in earlier sections.

In this section we will assume that the function has decomposed into tree structured subfunctions, $\mathcal{S} = \{1, \ldots, m\}$, such that

$$f(\boldsymbol{x}) = f^1(\boldsymbol{x}) + \cdots f^m(\boldsymbol{x}), \qquad \text{for all } \boldsymbol{x} \in \boldsymbol{L}^n. \tag{2.161}$$

Figure 2.10 shows an example of such a decomposition. Throughout this section will use binary indicator variables. Each variable $x_i \in \boldsymbol{L}$ will be represented by the binary indicator variable $\boldsymbol{x}_i$. Furthermore, a new set indicator variables $\boldsymbol{x}_{ij}$ is introduced, encoding the assignments of variables x_i and x_j.

In this section *each* subproblem s, will be given their own set of binary indicator variables denoted as $\boldsymbol{x}_i^s$. The indicator variables for the original function f will be denoted by $\boldsymbol{y}$. Furthermore, $\boldsymbol{y}_s$ will be used to denote the restriction of $\boldsymbol{y}$ to subproblem s. For any subfunction $f^s : \boldsymbol{L}^n \to \mathbf{R}$ we define

$$f^s(\boldsymbol{x}^s) = f^s(\tilde{\boldsymbol{x}}), \tag{2.162}$$

where $\tilde{\boldsymbol{x}} \in \boldsymbol{L}^n$ are converted from the indicator variables $\boldsymbol{x}^s$ as

$$\tilde{x}_i = k \quad \text{if } \boldsymbol{x}_i^s(k) = 1. \tag{2.163}$$

The variables $\boldsymbol{x}_{ij}^s$ are not part of the original problem and we will enforce them to coincide with our original variables by restricting our attention to the set

$$\mathcal{X} = \left\{ \boldsymbol{x} : \begin{array}{ll} \sum\limits_{a \in \boldsymbol{L}} \boldsymbol{x}_i(a) = 1 & \text{for all } i \;\; : U_i \;\; \neq 0 \\ \sum\limits_{b \in \boldsymbol{L}} \boldsymbol{x}_{ij}(a,b) = \boldsymbol{x}_i(a) & \text{for all } ij : B_{ij} \neq 0,\, b \in \boldsymbol{L} \\ \boldsymbol{x}_i(a) \in \mathbf{B} & \text{for all } i \;\; : U_i \;\; \neq 0,\, a \in \boldsymbol{L} \\ \boldsymbol{x}_{ij}(a,b) \in \mathbf{B} & \text{for all } ij : B_{ij} \neq 0,\, a, b \in \boldsymbol{L} \end{array} \right\}. \tag{2.164}$$

Furthermore, let $\mathcal{X}^s$ be the same set of constraints for each set of variables $\boldsymbol{x}^s$. The minimization of (2.118) can now be expressed as the following

constrained problem

$$\begin{aligned} \min_{\{\boldsymbol{x}^s \in \mathcal{X}^s\}, \boldsymbol{y} \in \mathcal{X}} \quad & \sum_{s \in \mathcal{S}} f^s(\boldsymbol{x}^s) \\ \text{subject to} \quad & \boldsymbol{x}^s = \boldsymbol{y}_s \qquad \text{for all } s \in \mathcal{S}. \end{aligned} \tag{2.165}$$

The *dual function* is given by

$$d(\boldsymbol{\lambda}) = \min_{\{\boldsymbol{x}^s \in \mathcal{X}^s\}, \boldsymbol{y} \in \mathcal{X}} \sum_{s \in \mathcal{S}} f^s(\boldsymbol{x}^s) + \sum_{s \in \mathcal{S}} \left(\boldsymbol{\lambda}^s\right)^{\mathrm{T}} \left(\boldsymbol{x}^s - \boldsymbol{y}_s\right), \tag{2.166}$$

constrained to the domain

$$\boldsymbol{\Lambda} = \left\{ \boldsymbol{\lambda} : \begin{array}{ll} \sum_{s \in \mathcal{S}} \boldsymbol{\lambda}_i^s(a) = 0 & \text{for all } i \;\; : U_i \neq 0, a \in \boldsymbol{L} \\ \sum_{s \in \mathcal{S}} \boldsymbol{\lambda}_{ij}^s(a,b) = 0 & \text{for all } ij : B_{ij} \neq 0, a, b \in \boldsymbol{L} \end{array} \right\}. \tag{2.167}$$

Without these constraints $d(\boldsymbol{\lambda}) = -\infty$ for some $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$. The *dual problem* is then given by

$$\underset{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}}{\text{maximize}}\, d(\boldsymbol{\lambda}). \tag{2.168}$$

The dual function (2.166) is also called the *Lagrangian relaxation* of the primal problem (2.165). This is because the inequality constraint of the primal problem (2.165) is relaxed by the dual variables, $\boldsymbol{\lambda}$, in (2.166).

Remark 2.38. *We are working with indicator variables. For a problem with $|L|$ labels each variable $\boldsymbol{x}_i$ in* (2.165) *produces $|L|$ equality constraints. We could also have chosen to work with the variables directly; however, this would lead to fewer constraints. Later in this chapter we will discuss how the number of constraints can influence the optimization of the dual problem.*

There are two reasons for introducing the dual function; first it gives us a lower bound and secondly the dual function is easier to optimize, since it is concave.

Remark 2.39. *Any optimization algorithm for the dual problem* (2.168) *can be implemented via reparametrization, just like in Section 2.7.2. In Kolmogorov (2006) maximization of $d(\boldsymbol{\lambda})$ is interpreted as maximizing the dual function, with respect to a reparametrization.*

In order to simplify the presentation we will restrict our attention to decompositions where each subproblem is given the same weight. This allows us to set

$$U_i^s(a) = \frac{U_i(a)}{\mathcal{S}(i)} \text{ for all } a \in \boldsymbol{L}, \tag{2.169}$$

$$B_{ij}^s(a,b) = \frac{B_{ij}(a,b)}{\mathcal{S}(\{i,j\})} \text{ for all } a,b \in \boldsymbol{L}, \tag{2.170}$$

where $\mathcal{S}(I)$ is the number of subproblems containing the variables in the set I.

The strength of the relaxation

It might come as a surprise the we introduced redundant binary constraints in (2.164). For any pair of variables (i,j) forcing

$$\begin{cases} \boldsymbol{x}_i^s(a) = \boldsymbol{x}_i^t(a) \\ \boldsymbol{x}_j^s(b) = \boldsymbol{x}_j^t(b) \end{cases} \quad \text{for all } s,t \in \mathcal{S} \text{ and } a,b \in \boldsymbol{L}, \tag{2.171}$$

will implicitly enforce

$$\boldsymbol{x}_{ij}^s(a,b) = \boldsymbol{x}_{ij}^t(a,b). \tag{2.172}$$

We will motivative this in two different ways. One using a linear programming relaxation, where the indicator variables are relaxed to $[0,1]$, and one using the dual problem where the equality constraints are relaxed by the dual variables.

We start by considering the dual problem. Let $\boldsymbol{\lambda}$ be all dual variables associated with the constraint set $\mathcal{X}$ given in (2.164). Given one of the constraints c, used to define $\mathcal{X}$; let $\boldsymbol{\lambda}_c \subset \boldsymbol{\lambda}$ denote the dual variables used by c. Furthermore, for subproblem s, let $\boldsymbol{x}_c^s \subset \boldsymbol{x}^s$ and $\boldsymbol{y}_{c \cap s} \subset \boldsymbol{y}_s$ denote the restriction of $\boldsymbol{x}^s$ and $\boldsymbol{y}_s$, to the variables used by c. Given some constraint set $\mathcal{C}$, the dual function in (2.166) can now be expressed as

$$d_{\mathcal{C}}(\boldsymbol{\lambda}) = \sum_{s \in \mathcal{S}} f^s(\boldsymbol{x}^s) + \sum_{c \in \mathcal{C}} \sum_{s \in \mathcal{S}} \left(\boldsymbol{\lambda}_c^s\right)^{\mathrm{T}} \left(\boldsymbol{x}_c^s - \boldsymbol{y}_{c \cap s}\right), \tag{2.173}$$

and the dual problem in (2.168) can now be expressed as

$$\max_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}_{\mathcal{C}}} d_{\mathcal{C}}(\boldsymbol{\lambda}), \tag{2.174}$$

where

$$\boldsymbol{\Lambda}_{\mathcal{C}} = \left\{ \boldsymbol{\lambda}_c^s : \sum_{s \in \mathcal{S}} \boldsymbol{\lambda}_c(a) = 0 \text{ for all } a \in \boldsymbol{L}, s \in \mathcal{S} \text{ and } c \in \mathcal{C} \right\}. \tag{2.175}$$

As each constraint has associated dual variables, removing the constraints c can be thought of as restricting $\boldsymbol{\lambda}_c = \mathbf{0}$. This observation give us our next result.

Proposition 2.40. *Given two sets of constraints*

$$\mathcal{C}_1 \subset \mathcal{C}_2, \tag{2.176}$$

the following holds

$$\max_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}_{\mathcal{C}_1}} d_{\mathcal{C}_1}(\boldsymbol{\lambda}) \leq \max_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}_{\mathcal{C}_2}} d_{\mathcal{C}_2}(\boldsymbol{\lambda}). \tag{2.177}$$

Proof. Introduce $C = \mathcal{C}_1 \cap \mathcal{C}_2$ and $D = \mathcal{C}_2 \backslash \mathcal{C}_1$ and define

$$\boldsymbol{\Lambda}_{\mathcal{C}_2'} = \begin{Bmatrix} \boldsymbol{\lambda}_c : \sum_{s \in \mathcal{S}} \boldsymbol{\lambda}_c^s(a) = 0 & \text{for all } a \in \boldsymbol{L} \text{ and } c \in C, \\ \boldsymbol{\lambda}_d : \boldsymbol{\lambda}_d^s(a) = \mathbf{0} & \text{for all } a \in \boldsymbol{L}, s \in \mathcal{S}, \text{ and } d \in D \end{Bmatrix}.$$

$\boldsymbol{\Lambda}_{\mathcal{C}_2'}$ and $\boldsymbol{\Lambda}_{\mathcal{C}_2}$ contains the same variables but $\boldsymbol{\Lambda}_{\mathcal{C}_2'}$ is more restricted. It directly follows that

$$\max_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}_{\mathcal{C}_1}} d_{\mathcal{C}_1}(\boldsymbol{\lambda}) = \max_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}_{\mathcal{C}_2'}} d_{\mathcal{C}_2'}(\boldsymbol{\lambda}) \leq \max_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}_{\mathcal{C}_2}} d_{\mathcal{C}_2}(\boldsymbol{\lambda}). \tag{2.178}$$

□

This shows that adding more constraints can potentially increase our lower bound. Taking Proposition 2.40 to the extreme, by including every constraint would give the best dual problem. This is of course not tractable even for moderately sized problem.

Definition 2.41. *A relaxation with the constraint set $\mathcal{C}_1$ is said to be* stronger *than a relaxation with constraint set $\mathcal{C}_2$ if*

$$\mathcal{C}_2 \subsetneq \mathcal{C}_1. \tag{2.179}$$

We will now consider the linear programming relaxation. In Werner (2010) the minimization of f is formulated as a relaxed linear program, sometimes called the *Schlesinger LP relaxation*. Each relaxation is specified by two sets, the factors $\mathcal{F}$, and coupling scheme J. For each $a \in \mathcal{F}$ and for each possible labeling $\boldsymbol{x}_a$ an indicator variable $\tau_a(\boldsymbol{x}_a) \in \{0, 1\}$ is introduced; the integrality constraint is then relaxed to $\tau_a(\boldsymbol{x}_a) \in [0, 1]$. The coupling set, J, contains pairs (a, b)

$$a \in \mathcal{F}, \quad b \subset a, \tag{2.180}$$

which means that $(\mathcal{F}, J)$ is a directed acyclic graph. The simplest choice is

$$J = \Big\{(a, \{i\}) \ : a \in \mathcal{F}, i \in a\Big\}. \tag{2.181}$$

In Kolmogorov and Schoenemann (2012) this choice is called a *relaxation with singleton separators*. For each edge $(a, b) \in J$ we add a *consistency* constraint between a and b. The resulting relaxation is given by

$$\text{minimize} \quad \sum_{a \in \mathcal{F}} \sum_{\boldsymbol{x}_a} f_a(\boldsymbol{x}_a)\tau_a(\boldsymbol{x}_a) \tag{2.182a}$$

$$\text{subject to} \quad \sum_{\boldsymbol{x}_a} \tau_a(\boldsymbol{x}_a) = 1 \qquad \text{for all } a \in \mathcal{F}, \tag{2.182b}$$

$$\sum_{\boldsymbol{x}_a \,:\, (\boldsymbol{x}_a)_{a \cap b} = \boldsymbol{x}_b} \tau_a(\boldsymbol{x}_a) = \tau_b(\boldsymbol{x}_b) \quad \text{for all } (a, b) \in J, \tag{2.182c}$$

$$\tau_a(\boldsymbol{x}_a) \geq 0 \qquad \text{for all } a \in \mathcal{F}. \tag{2.182d}$$

Constraints which were redundant in the original problem may no longer be redundant in the relaxed problem. Adding more linear constraints may yield a stronger relaxation.

Now suppose that we have chosen $(\mathcal{F}, J)$ to correspond to the constraint set $\mathcal{X}$ in (2.164), then Kolmogorov (2006) shows that the linear programming relaxation (2.182) is a dual problem to (2.168) and that strong duality holds.

Optimizing (2.168)

So far in this section we have decomposed the function and talked about the potential quality of the solutions. But we have yet to cover the most important part, how to optimize (2.168). In this section we will cover two competing methods both able to optimize (2.168), *dual ascent* and *average ascent*. In the computer vision community maximizing (2.168) using dual ascent is usually referred to as *dual decomposition* and maximizing it using average ascent is referred to as *TRW-s*. In order to avoid confusion we will use the established nomenclature and use TRW-s and dual decomposition.

TRW-s

Wainwright et al. (2005) introduced tree-reweighted message passing. In this initial formulation the algorithm is not guaranteed to converge. Modifications in Kolmogorov (2006) led to a convergent algorithm called tree-reweighted sequential message passing (TRW-s). The algorithm belongs to a class of optimization routines knowns as tree-consistency bound optimization introduced by Meltzer et al. (2009). These optimization algorithms are usually expressed in terms of reparameterization of the original problem. In this section we will link this description to the arguably more straightforward interpretation of updating the dual variables $\boldsymbol{\lambda}$ and optimizing (2.168). First we need to introduce two min-marginals functions as

$$b_i(a) = \min_{\{\boldsymbol{x} \,:\, x_i = a\}} f(\boldsymbol{x}), \tag{2.183}$$

$$b_{ij}(a, b) = \min_{\{\boldsymbol{x} \,:\, x_i = a, x_j = b\}} f(\boldsymbol{x}). \tag{2.184}$$

Identically b_i^s and b_{ij}^s denote the min-marginals for subproblem s. In each iteration for any unary variable i, define the *average* as

$$\overline{b_i}(x_i) = \frac{1}{S(\{i\})} \sum_{s \in \mathcal{S}} b_i^s(x_i) \tag{2.185}$$

And similarly for the binary term connecting i and j, define the average as

$$\overline{b_{ij}}(x_i, x_j) = \frac{1}{S(\{i, j\})} \sum_{s \in \mathcal{S}} b_i^s(x_i) + b_{ij}^s(x_i, x_j) + b_j^s(x_j) \tag{2.186}$$

In the algorithm the binary and unary terms are successively reparameterized such that

$$\overline{b_i}(x_i) = U_p^s(x_i), \tag{2.187}$$

$$\overline{b_{ij}}(x_i, x_j) = U_i^s(x_i) + B_{ij}^s(x_i, x_j) + U_j^s(x_j). \tag{2.188}$$

In the binary update $U_i(x_i)$ is always fixed, only $B_{ij}(x_i, x_j)$ and $U_j(x_j)$ are updated. If each subproblem is optimized using message-passing and reparametrization; then, the min-marginals are simply the current reparameterized binary and unary weights.

In order to present an efficient version of TRW-s we need to introduce a special ordering of the variables. The subproblems $s \in \mathcal{S}$ are said to be *monotonic* if there exist an ordering function o, such that if

$$u_1^s, \ldots, u_k^s, \tag{2.189}$$

are consecutive variables in each subproblem then

$$\big(o(u_1^s), \ldots, o(u_k^s)\big), \tag{2.190}$$

is a monotonic sequence. We can now describe TRW-s in a compact way.

Minimize (2.118) using TRW-s implemented via reparametrization.

1. For each unary term i in increasing order $o(i)$ do:
   (a) Average the unary dual variables using (2.192).
   (b) For every binary term (s, t) with $o(t) > o(s)$ do:
      i. Average the binary dual variables using (2.193).
      ii. Send the message m_{st}.
2. Reverse order $o(u) = n + 1 - o(u)$.
3. Check for convergence otherwise go back to step 1.

Kolmogorov (2006) shows that by introducing the monotonic ordering the messages sent in each iteration are actually correct. That is, they would

have been the same even if we had reoptimized each subproblem after each averaging operation.

Next we will show the connection between averaging and an update scheme on the dual variables. This will show that the averaging operations actually corresponds to an ascent method, which we will call *average ascent*. Instead of viewing (2.185) as a reparametrization step we can view it as an update of the dual variables as

$$b_i^s(x_i) + \lambda_p^s(x_i) = \overline{b_i}(x_i). \tag{2.191}$$

The averaging step for variable i can thus be interpreted as the dual update

$$\lambda_i^s(x_i) = \overline{b_i}(x_i) - b_i^s(x_i). \tag{2.192}$$

Similarly the dual variables associated with i and j can be updated as

$$\lambda_{ij}^s(x_i, x_j) = \overline{b_{ij}}(x_i, x_j) - b_{ij}^s(x_i, x_j) - b_i^s(x_i) - b_q^t(x_j). \tag{2.193}$$

Each averaging step reparametrizes the problem such that the current min-marginals for each subproblem coincide. Summing over all subproblems it directly follows that

Proposition 2.42. *After averaging in* (2.192) *or* (2.193) *the dual variables*

$$\boldsymbol{\lambda} \in \boldsymbol{\Lambda}. \tag{2.194}$$

where $\boldsymbol{\Lambda}$ *is defined in* (2.167).

The next theorem show that this update scheme actually is an ascent algorithm.

Theorem 2.43. *Let* $\boldsymbol{\lambda}_k$ *be the dual variables after* k *iterations of TRW-s then either* $\boldsymbol{\lambda}_k$ *has converged or*

$$d(\boldsymbol{\lambda}_k) \geq d(\boldsymbol{\lambda}_{k-1}). \tag{2.195}$$

where d is defined in (2.166).

Proof. See Kolmogorov (2006). □

The next theorem show that TRW-s works well with a common, and relatively easy class of optimization problems.

Theorem 2.44. *Given a second-order submodular pseudo-boolean function TRW-s converges to the optimal solution.*

Proof. See Kolmogorov and Wainwright (2012). □

The fact that TRW-s uses average ascent is one of TRW-s biggest advantages. After averaging, the unary and binary terms defining each subproblems are all equivalent. This allows us to implement TRW-s in an efficient manner, by only storing one version of each unary and binary term for every subproblem The gain is two-fold; firstly, this allows us to save a lot of memory, and secondly the implementation will be faster, due to less memory overhead.

Dual decomposition

Dual decomposition was first explored by Komodakis et al. (2007); Komodakis et al. (2011) for solving general pseudo-boolean functions and in Strandmark, Kahl, and Schoenemann (2011) for its parallelization ability. Given the dual function in (2.166) we can directly from Proposition (2.11) find a supergradient and optimize the dual function using dual ascent.

Minimize (2.118) using dual decomposition.

1. Initialize with $\boldsymbol{\lambda} = \mathbf{0}$.
2. For all $s \in \mathcal{S}$ compute $(\boldsymbol{x}^S)^\star = \operatorname{argmin}_{\boldsymbol{x}} f_{\boldsymbol{\lambda}}(\boldsymbol{x})$.
3. Project the supergradients onto $\boldsymbol{\Lambda}$

$$s\boldsymbol{\lambda}_i^{s'} = \boldsymbol{\lambda}_i^s + \tau\left(\boldsymbol{x}_p^{s\star} - \frac{\sum_{s\in\mathcal{S}} \boldsymbol{x}^{s\star}}{|\mathcal{S}(i)|}\right) \tag{2.196}$$

$$\boldsymbol{\lambda}_{ij}^{s'} = \boldsymbol{\lambda}_{ij}^s + \tau\left(\boldsymbol{x}_{ij}^{T\star} - \frac{\sum_{s\in\mathcal{S}} \boldsymbol{x}_{ij}^{s\star}}{|\mathcal{S}(\{i,j\})|}\right) \tag{2.197}$$

   where τ is some step length.

4. Check for convergence, if the algorithm has not converged restart at step 2.

Comparing TRW-s to Dual decomposition

In this section we will cover some pros and cons with TRW-s and dual decomposition.

Theorem 2.45. *The solution given by dual decomposition on a set of subproblem $\mathcal{S}$, is at least as good as the solution given TRW-s.*

Proof. See Komodakis et al. (2011). □

Furthermore, Komodakis et al. gives an example where TRW-s get stuck in a local optima, while dual decomposition finds the global optimal solution.

Min-marginals. TRW-s requires that the min-marginals are computed, dual decomposition only requires the global minimizer. If the subproblems can be optimized using belief propagation the min-marginals are given directly. However, it does not easily generalize to different decompositions. For instance if we would like to split the subproblems into submodular parts, then calculating the min-marginals is very costly, see Kohli and Torr (2006).

Speed and memory. Dual decomposition needs to store and update each subproblem separately, whereas TRW-s needs only to store one problem instance representing every subproblem. For decompositions with many subproblem this could lead to huge memory savings and also potential speedups thanks to less memory overhead. This has been shown experimentally in Kappes et al. (2013) where TRW-s is faster than dual decomposition.

Step length. The convergence of dual decomposition stems from the fact that we chose the step length τ as a decreasing function taking smaller and smaller steps. For TRW-s we do not choose the step length and it may not decrease, this can be interpreted as TRW-s taking slightly more greedy steps.

2.7.4 General functions

We will briefly review some methods for minimizing general functions of the form given in (2.115). Both TRW-s and Dual decomposition can be generalized to solver higher-order functions.

Generalized TRW-s

Recall the linear programming relaxation given in (2.182). Given any coupling scheme, J defined in (2.181) the set $(\mathcal{F}, J)$ defines a acyclic graph. This is exploited by Kolmogorov and Schoenemann (2012) to generalize TRW-s to higher-order functions. The generalized version works with any coupling scheme J and each $(a, b) \in J$ get its own message m_{ab}. A generalization of monotonic chains is also given which makes an efficient implementation possible.

Dual decomposition

The already presented algorithm for second-order functions can be generalizes to higher-order functions, see Komodakis et al. (2011). In their formulation the functions is decomposed into "subgraphs", and consistency is enforced between the different subgraphs. A example construction would be to choose each factor, $a \in \mathcal{F}$, as subgraph and enforce consistency between all subgraphs, with overlapping variables, using dual variables.

2.8 Improving existing solutions

It is quite common to be stranded with a set of possibly incomplete solutions to a problem. It might be a solution from a previous frame in a video sequence, or a incomplete solution given by roof duality. In this section a few methods are presented which improve upon already given solutions.

2.8.1 Fusion moves

Fusion moves is an iterative method used to optimize a large family of functions. Given a set of proposed solutions, or *proposals*, $\mathcal{S}$, fusion moves iteratively combines them into one final solution. The number proposed solutions is finite, so the resulting optimization problem is discrete. We are however not restricted to functions with discrete domain.

Fusion moves were introduce in Boykov, Veksler, et al. (2001) for a limited set of functions and was called α-expansion. This result was generalized in Lempitsky et al. (2010) and given the name fusion moves.

The two mentioned approaches are restricted in that they only fuse two solutions at the same time. In the section we outline how this can be ex-

tended to fusing arbitrary number of solutions simultaneously. *Simultaneous fusion* was first explored in Veksler (2009).

Fusing two solutions

We will restrict our attention to function on the form

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} U_i(x_i) + \sum_{1 \leq i < j \leq n} B_{ij}(x_i, x_j), \tag{2.198}$$

where $U_i : \mathbf{R} \to \mathbf{R}$ and $B_{ij} : \mathbf{R} \times \mathbf{R} \to \mathbf{R}$. Introduce a set of indicator variables $\boldsymbol{z} = \{z_1, \ldots, z_n\}$ and suppose we have two proposed solutions $\boldsymbol{a}$ and $\boldsymbol{b}$. For notational brevity let $\overline{z}_i = (1 - z_i)$. We can now introduce a function which for each variable i will choose either the solution from a or b as

$$\begin{aligned} h(\boldsymbol{z}) = &\sum_{i=1}^{n} z_i U_i(a) + \overline{z}_i U_i(b) \\ &+ \sum_{1 \leq i < j \leq n} z_i z_j B(a_i, a_j) + \overline{z}_i z_j B(b_i, a_j) \\ &+ \sum_{1 \leq i < j \leq n} \overline{z}_i z_j B(b_i, a_j) + \overline{z}_i \overline{z}_j B(b_i, b_j). \end{aligned} \tag{2.199}$$

It directly follows that

$$h(\mathbf{0}) = f(\boldsymbol{b}), \tag{2.200}$$

$$h(\mathbf{1}) = f(\boldsymbol{a}). \tag{2.201}$$

The *fusion move* is performed by solving

$$\boldsymbol{z}^\star = \operatorname*{argmin}_{\boldsymbol{z} \in \mathbf{B}^n} h(\boldsymbol{z}). \tag{2.202}$$

Solving (2.202) via roof duality will be useful even if we cannot label every variable. Let g be the submodular relaxation used for roof duality and let $(\boldsymbol{x}^\star, \boldsymbol{y}^\star) \in \operatorname{argmin} g$. Now we can use the overwrite function to define

$$\mathbf{1} \leftarrow (\boldsymbol{x}^\star, \boldsymbol{y}^\star) = \boldsymbol{z}. \tag{2.203}$$

From Proposition (2.25) it follows that

$$h(\boldsymbol{z}) \leq h(\mathbf{1}) = f(\boldsymbol{a}). \tag{2.204}$$

We can construct our solution, $\boldsymbol{y}$, to the fusion problem as

$$y_i = \begin{cases} a_i & \text{if } z_i = 1 \\ b_i & \text{if } z_i = 0. \end{cases} \tag{2.205}$$

By constructing we have

$$f(\boldsymbol{y}) = h(\boldsymbol{z}) \leq h(\mathbf{1}) = f(\boldsymbol{a}), \tag{2.206}$$

After performing a fusion move we have a new solution $\boldsymbol{y}$ which is at least as good as the initial solution $\boldsymbol{a}$. Constructing $\boldsymbol{y}$ from $\boldsymbol{a}$ and $\boldsymbol{b}$ in (2.203) will be referred to as performing *binary fusion* of $\boldsymbol{a}$ and $\boldsymbol{b}$.

Remark 2.46. *If function h in* (2.199) *is a submodular we do not need to use roof duality. The function can be efficiently optimized using the method described in Section 2.6.1.*

Each fusion move is guaranteed to generate a solution at least as good as the previous one. This gives us an optimization procedure. Generate a large set of proposals $\mathcal{S}$ and then fuse them one by one. This will be referred to as performing *iterative binary fusion*.

Fusing n solutions

The iterative binary fusion has some problems, the order in which we fuse the proposal may influence the end result and we have no guarantee that we have fused our proposals in an optimal way.

Suppose we have a function on the form (2.198) and m proposed solutions

$$\mathcal{S} = \left\{\boldsymbol{x}^1, \ldots, \boldsymbol{x}^m\right\}. \tag{2.207}$$

Introduce a set of indicator variables $\boldsymbol{z} = \{\boldsymbol{z}_1 \cdots \boldsymbol{z}_n\}$ where $\boldsymbol{z}_i : \boldsymbol{L} \to \mathbf{B}$ and

$$\boldsymbol{z}_i(\ell) = \begin{cases} 1 & \text{if } z_i = \ell, \\ 0 & \text{if } z_i \neq \ell. \end{cases} \tag{2.208}$$

We can now generalize (2.199) by introducing

$$\begin{aligned} s(\boldsymbol{z}) = &\sum_{i=1}^{n}\sum_{a=1}^{m}\boldsymbol{z}_i(a)U_i(x_i^a) \\ &\sum_{1\leq i<j\leq n}\sum_{a=1}^{m}\sum_{b=1}^{m}\boldsymbol{z}_i(a)\boldsymbol{z}_j(b)B_{ij}(x_i^a, y_i^b). \end{aligned} \tag{2.209}$$

By construction we have

$$s(\boldsymbol{i}) = f(\boldsymbol{x}^i). \tag{2.210}$$

The optimization problem we needs to solve is

$$\boldsymbol{z}^\star = \underset{\boldsymbol{z}\in\boldsymbol{L}^n}{\operatorname{argmin}}\, s(\boldsymbol{z}) \tag{2.211}$$

The function is of the same form as (2.115) in Section 2.7. Section 2.7 describes methods which can be used to optimize (2.211). Given any solution we extract a fused solution $\boldsymbol{y}$ as

$$y_i = x_i^{z_i^\star}, \quad \text{for all } i = 1, \ldots, n. \tag{2.212}$$

Optimizing (2.211) for all proposals in $\mathcal{S}$ and constructing $\boldsymbol{y}$ using (2.212) will be referred to as performing *simultaneous fusion*. If we are able to find the global optimal to (2.211) it directly follows that

$$f(\boldsymbol{y}) \leq f(\boldsymbol{x}^i), \quad \text{for all } i = 1, \ldots, m. \tag{2.213}$$

The advantages with simultaneous fusion is clear; given any number of proposed solutions, it will find a solution to f, at least as good as iterative binary fusion. The downside is that solving (2.211) is substantially harder than solving (2.202).

2.8.2 Completing solutions

After minimizing a function using roof duality we may end up with some unlabeled variables, that is an incomplete solution. We will now briefly cover two methods which can be used to generate a complete solution.

Probing

For all variables with persistency we know the optimal labeling, see Corollary 2.26. These variables can be contracted and we end up with a function $\tilde{f}$ with potentially less variables than f. Given a variable z in $\tilde{f}$ construct two new functions

$$\begin{aligned} \tilde{f}_0 &= \tilde{f}_{|z=0}, \\ \tilde{f}_1 &= \tilde{f}_{|z=1}. \end{aligned} \tag{2.214}$$

Minimizing $\tilde{f}_0$ and $\tilde{f}_1$ using roof duality may give us additional persistencies. Now assume that the persistences for $\tilde{f}_0$ and $\tilde{f}_1$ agree on some variables $\boldsymbol{x}$. Since they agree for both labelings of z, they are persistent in the original problem f and all variables in $\boldsymbol{x}$ can be contracted. The trick is that by fixing z we may end up different set of persistencies than just running roof duality on f.

Iteratively performing the steps above for one or more variables is known as *probing*, a method introduced by Boros, Hammer, and Tavares (2006). If the number of variables fixed in each iteration is increasing then probing will converge to the optimal solution. However, the worst case is brute-force testing all combinations of all unlabeled variables; this is intractable even for moderate sized sets of unlabeled variables.

Improve

Another way to generate a complete solution is to use binary fusion and fuse the incomplete solution with any complete solution. This was proposed by Rother et al. (2007) and can be seen as a precursor to the more general fusion moves framework. If the complete solution is randomly generated this is known as running *improve* on an incomplete roof duality solution.

Chapter 3

Partial enumeration

This chapter deals with optimization of high-order multi-label functions. The function variables are assumed to be placed on a grid, and all higher-order interactions are assumed to be contained inside small *patches* on this grid. This grid structure is used in many computer vision applications.

Exhaustive search is a naive and general optimization approach, where the cost of all possible variable assignments are enumerated. In this chapter, a general optimization method, *partial enumeration*, is introduced. The method is based on enumeration of small patches and it reduces complex high-order formulations to *pairwise weighted constraint satisfaction problems*, which can be efficiently optimized using standard methods like TRW-s.

To showcase the strength of partial enumeration, it is benchmarked against different state-of-the-art algorithms on a number of computer vision tasks: a novel curvature segmentation approach, binary deconvolution and higher-order stereo regularization.

3.1 Algorithm

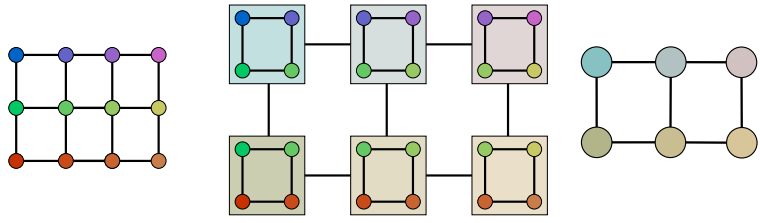
This chapter considers minimization of functions,

$$f : \boldsymbol{L}^k \rightarrow \mathbf{R}, \tag{3.1}$$

given on the form

$$f(\boldsymbol{x}) = \sum_{a \in \mathcal{F}} f_a(\boldsymbol{x}_a), \tag{3.2}$$

where $\boldsymbol{x}_a$ is the restriction of $\boldsymbol{x}$ to the *factor* a, representing a subset of $\boldsymbol{x}$, $\boldsymbol{L} = \{1, \ldots, k\}$, $\mathcal{F}$ is a set of factors, and $f_a : \boldsymbol{L}^{|a|} \rightarrow \mathbf{R}$.



(a) f (b) Patches (c) f_{pe}

Figure 3.1: Example of a partial enumeration for a function which can be represented by a grid. Each variable is shown as a vertex and the binary terms are shown as edges, connecting the two variables they are a function of. In (a) the graph representation for f is shown. In (b) patches are formed by grouping pixels in a sliding window fashion. Note that a variable may be part of more than one patch. In this representation the new binary terms enforce consistency between patches. In (c) the patches are enumerated resulting in f_{pe}. In this representation the higher-order interactions in f are transformed into the unary terms of f_{pe}.

Select a new set of factors denoted $\mathcal{V}$, such that

$$a \subseteq b \quad \text{for some } b \in \mathcal{V}, \text{ given any } a \in \mathcal{F}. \tag{3.3}$$

Each factor in $b \in \mathcal{V}$ will be referred to as a *patch*. By construction each factor $a \in \mathcal{F}$ is fully contained inside a patch $b \in \mathcal{V}$. Two patches $a, b \in \mathcal{V}$, are said to be *overlapping* if $a \cap b \neq \varnothing$.

We are now ready to reformulate f; take any set of patches $\mathcal{V}$, for each patch, $a \in \mathcal{V}$, enumerate the cost of every possible assignment of $\boldsymbol{x}_a$. This turns a set of variables $\boldsymbol{x}_a \in \boldsymbol{L}^{|a|}$, into one *patch variable* with higher label space:

$$X_a \in \left\{0, \ldots, |\boldsymbol{L}|^{|a|}\right\} = \boldsymbol{L}_a. \tag{3.4}$$

By doing this we have *partially enumerated* the function f. For any patch $a \in \mathcal{V}$, the patch assignment costs can now be encoded into a unary term by a function

$$U_a : \boldsymbol{L}_a \to \mathbf{R}. \tag{3.5}$$

Denote the space of all patch variables as

$$\mathcal{X} = \bigtimes_{a \in \mathcal{V}} \boldsymbol{L}_a. \tag{3.6}$$

Definition 3.1. *The assignment of two variables*

$$X_a \in \boldsymbol{L}_a \text{ and } X_b \in \boldsymbol{L}_b, \tag{3.7}$$

is said to be consistent *if their corresponding representations*

$$\boldsymbol{x}_a \in \boldsymbol{L}^{|a|} \text{ and } \boldsymbol{x}_b \in \boldsymbol{L}^{|b|}, \tag{3.8}$$

agree for all overlapping variables. This will be denoted as $X_a \sim X_b$. The set of all variables $\boldsymbol{X} \in \mathcal{X}$ is said to be consistent if all overlapping patches are consistent.

The patch variables can be forced to be consistent using a binary term. We can now reformulate (3.2) as

$$f_{\text{pe}} : \mathcal{X} \to \mathbf{R}, \tag{3.9}$$

where

$$f_{\text{pe}}(\boldsymbol{X}) = \sum_{a \in \mathcal{V}} U_a(X_a) + \sum_{(a,b) \in \mathcal{E}} B_{ab}(X_a, X_b), \tag{3.10}$$

with the binary term defined as

$$B_{ab}(X_a, X_b) = \begin{cases} 0 & \text{if } X_a \sim X_b \\ \infty & \text{otherwise .} \end{cases} \tag{3.11}$$

The minimization of f_{pe} is a *weighted constraint satisfaction problem*, formulated as the minimization of a second-order multi-label function. By construction f_{pe} can be minimized with standard approaches discussed in Section 2.7. An example of this reformulation is given in Figure 3.1.

It remains to specify how to choose the set of binary terms $\mathcal{E}$. One possibility would be to select all pairs overlapping patches. However, in some cases we may be able to choose a smaller set.

Definition 3.2. *We call a set of edges* $\mathcal{E}$ valid *if it for every variable,* $x_i \in \boldsymbol{L}$*, and every pair of patches* $a, b \in \mathcal{V}$ *with* $x_i \in \boldsymbol{x}_a$ *and* $x_i \in \boldsymbol{x}_b$*, there exists a sequence of patches*

$$\mathcal{S} = (s_0, s_1, \dots s_n), \tag{3.12}$$

in $\mathcal{E}$*, with* $s_0 = a$*,* $s_n = b$*, and* $x_i \in \boldsymbol{x}_{s_i} \cap \boldsymbol{x}_{s_{i+1}}$.

In other words the consistency is enforced via a sequence of constraints linking patch a and patch b.

Proposition 3.3. *For a valid set of edges* $\mathcal{E}$*, the labeling* $\boldsymbol{X}$ *is consistent if and only if*

$$\sum_{(a,b)\in\mathcal{E}} B_{ab}(X_a, X_b) = 0, \tag{3.13}$$

for all $(a, b) \in \mathcal{E}$.

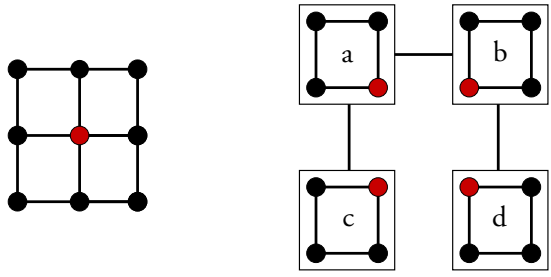
Proof. One direction is trivial: if $\boldsymbol{X}$ is consistent then each binary term $B_{ab}(X_a, X_b) = 0$. For the other direction it suffices to show that every variable is consistent. Consider any variable x_i. For every $a, b \in \mathcal{V}$ for which $x_i \in \boldsymbol{x}_a$ and $x_i \in \boldsymbol{x}_b$ there exists a sequence of patches S, by construction, connecting the two such that $B_{s_i s_{i+1}}(X_{s_i}, X_{s_{i+1}}) = 0$ for all $s_i \in \mathcal{S}$. From this it follows that each x_i must be consistent over all patches in $\mathcal{V}$. □

This proposition tells us that we do not need to include every possible overlapping $(a, b) \in \mathcal{E}$, it suffices that they are implicitly included. An example where we have removed some constraints is given in Figure 3.2.

3.1.1 Efficient message passing

Throughout this chapter we will optimize f_{pe} using the message passing based method TRW-s. Since the number of labels can be large for higher-order factors it is essential to compute the messages fast. The messages sent during optimization has the form

$$m_{ab}(X_b) = \min_{X_a}(B_{ab}(X_a, X_b) + h(X_a)), \tag{3.14}$$

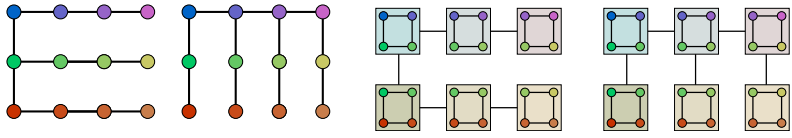


(a) A 3×3 grid (b) Four 2×2 patches

Figure 3.2: Example of how to use Proposition 3.3. in (a) a 3×3 grid has been split into four 2×2 patches named a, b, c, and d. The intersection $a \cap d = x$ is highlighted. In (b) the sequence (c, a, b, d) connects all patches enforcing consistency for the variable x over all patches. Therefore there is no need to add edges enforcing consistency directly between for instance c and d.

where h is some function of the patch label X_a. Due to the special structure of the binary term (3.11), we can find the optimal message very efficiently. Given any pair variables X_a and X_b, order the labels into disjoint group pairs such that $X_a \sim X_b$ for all labels in each group pair. The message values $m_{ab}(X_b)$ for all the X_b in the same group can now be found by searching for the smallest value of $h(X_a)$ in its paired group. The label order depends on the set $\mathcal{E}$; however, it does not change during optimization and can therefore be precomputed at startup. The bottleneck is therefore searching the groups for the minimal value, which can be done in linear time.

This process does not require all the possible patch assignments to be considered. For larger patches many assignments might be unwanted, they can simply be set to be inconsistent and never explicitly stored. This observation is imperative for applications using larger patches, as the label space grows rapidly as a function of the patch size.



(a) Decomposition of f (b) Decomposition of f_{pe}

Figure 3.3: An example decomposition of f and f_{pe} in Figure 3.1. (a) After the decomposition of f, each subfunction lacks some binary terms of f (missing edges). (b) For the decomposition of f_{pe} however, all binary terms of f are still present. (The removed binary terms enforces consistency.) The subfunctions in (b) contains more structure of f compared to the subfunctions in (a).

3.1.2 Strength of relaxations

In the previous section the original function f was reformulated into an equivalent form f_{pe}. For higher-order functions the reformulation allows for efficient optimization using TRW-s. In this section it will be shown that even for second-order multi-label functions, which TRW-s can optimize directly, the reformulation is advantageous. Intuitively each part of the decomposition contains more structure for f_{pe} compared to f, see Figure 3.3. By comparing the decomposition of f and f_{pe} we will show that the resulting relaxation used by f_{pe} is stronger than the relaxation of f.

Consider f_{pe} in (3.10), the Schlesinger LP relaxation is given by

$$\text{minimize} \quad \sum_{a\in\mathcal{F}} \sum_{\boldsymbol{x}_a} f_a(\boldsymbol{x}_a)\tau_a(\boldsymbol{x}_a) \tag{3.15a}$$

$$\text{subject to} \quad \sum_{\boldsymbol{x}_a} \tau_a(\boldsymbol{x}_a) = 1 \qquad \text{for all } a \in \mathcal{F}, \tag{3.15b}$$

$$\sum_{\boldsymbol{x}_a : \boldsymbol{x}_a \sim \boldsymbol{x}_b} \tau_a(\boldsymbol{x}_a) = \tau_b(\boldsymbol{x}_b) \qquad \text{for all } (a,b) \in J, \tag{3.15c}$$

$$\tau_a(\boldsymbol{x}_a) \geq 0 \qquad \text{for all } a \in \mathcal{F}, \tag{3.15d}$$

where $\boldsymbol{x}_a \sim \boldsymbol{x}_b$ means that labelings $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$ are consistent on the overlap area and $\tau_a(\boldsymbol{x}_a) \in [0,1]$ is the relaxed indicator variable. Adding more edges to the coupling scheme J gives more constraints and leads to

the same or stronger relaxation. The simplest choice is to set

$$J = \{(a, \{i\}) : a \in \mathcal{F}, i \in a\}. \tag{3.16}$$

This coupling scheme is known as relaxation with singleton separators. TRW-s uses the coupling scheme

$$J = \{(a, b) \in \mathcal{E}\}, \tag{3.17}$$

see Kolmogorov (2006). In this section we will show that f_{pe} uses a larger coupling scheme than f, when they are both optimized using TRW-s.

Theorem 3.4. *Given a second-order multi-label function f,* (3.2)*, and its partial enumeration reformulation f_{pe},* (3.10)*. Then the Schlesinger LP relaxation for f_{pe}, with the coupling scheme used by TRW-s,* (3.17)*, is equivalent to the Schlesinger LP relaxation for f, with*

$$\begin{aligned} \mathcal{F} &= \{a : a \subseteq \hat{a} \textit{ for some } \hat{a} \in \mathcal{V}\}, \\ J &= \{(a, b) : a, b \in \mathcal{F}, b \subset a\}. \end{aligned} \tag{3.18}$$

Proof. First, let us write down the Schlesinger LP for (3.10). It uses variables $\hat{\tau}_a(\boldsymbol{x}_a)$ for $a \in \mathcal{F}$ where $\boldsymbol{x}_a \in \mathcal{L}_a$ and variables $\hat{\tau}_{ab}(\boldsymbol{x}_a, \boldsymbol{x}_b)$ for $(a, b) \in \mathcal{E}$ where $(\boldsymbol{x}_a, \boldsymbol{x}_b) \in \mathcal{L}_a \times \mathcal{L}_b$. The variables $\hat{\tau}_{ab}(\boldsymbol{x}_a, \boldsymbol{x}_b)$ and $\hat{\tau}_{ba}(\boldsymbol{x}_b, \boldsymbol{x}_a)$ are treated as the same variable. If $\boldsymbol{x}_a \nsim \boldsymbol{x}_b$ then $\hat{\tau}_{ab}(\boldsymbol{x}_a, \boldsymbol{x}_b)$ will be zero at the optimum, since the associated cost is infinite. Thus, we can eliminate such variables from the formulation. The LP can now be expressed as

$$\begin{aligned}
\text{minimize} \quad & \sum_{a \in \mathcal{V}} \sum_{\boldsymbol{x}_a} f_a(\boldsymbol{x}_a) \hat{\tau}_a(\boldsymbol{x}_a) && && \text{(3.19a)} \\
\text{subject to} \quad & \sum_{\boldsymbol{x}_a} \hat{\tau}_a(\boldsymbol{x}_a) = 1 && \text{for all } a \in \mathcal{V}, && \text{(3.19b)} \\
& \sum_{\boldsymbol{x}_a : \boldsymbol{x}_a \sim \boldsymbol{x}_b} \hat{\tau}_{ab}(\boldsymbol{x}_a, \boldsymbol{x}_b) = \hat{\tau}_b(\boldsymbol{x}_b) && \text{for all } (a, b) \in \mathcal{E}, && \text{(3.19c)} \\
& \hat{\tau}_a(\boldsymbol{x}_a) \geq 0 && \text{for all } a \in \mathcal{F}, && \text{(3.19d)} \\
& \hat{\tau}_{ab}(\boldsymbol{x}_a, \boldsymbol{x}_b) \geq 0 && \text{for all } (a, b) \in \mathcal{E} && \text{(3.19e)} \\
& && \text{such that } \boldsymbol{x}_a \sim \boldsymbol{x}_b.
\end{aligned}$$

Our goal is to show that this LP is equivalent to the LP (3.15) with the graph $(\mathcal{F}, J)$ defined in (3.18). Let Ω and $\hat{\Omega}$ be the feasible sets of the LP (3.15) and LP (3.19) respectively. We will prove the claim by showing that there exist cost-preserving mappings $\hat{\Omega} \to \Omega$ and $\Omega \to \hat{\Omega}$.

Mapping $\hat{\Omega} \to \Omega$. Given a vector $\hat{\tau} \in \hat{\Omega}$, we define vector τ as follows: consider $\gamma \in \mathcal{F}$. By the definition of $\mathcal{F}$, there exists $a \in \mathcal{V}$ with $\gamma \subseteq a$. We set

$$\tau_\gamma(\boldsymbol{x}_\gamma) = \sum_{\boldsymbol{x}_a : \boldsymbol{x}_a \sim \boldsymbol{x}_\gamma} \hat{\tau}_a(\boldsymbol{x}_a) \tag{3.20}$$

Let us show that this definition does not depend on the choice of a. Suppose there are two patches $a, b \in \mathcal{V}$ with $\gamma \subseteq a \cap b$. We consider three cases:

Case 1: $\gamma = a \cap b$, $(a, b) \in \mathcal{E}$. Using condition (3.19c) for pairs (a, b) and (b, a), we obtain the desired result:

$$\begin{aligned}
\sum_{\boldsymbol{x}_a : \boldsymbol{x}_a \sim \boldsymbol{x}_\gamma} \hat{\tau}_a(\boldsymbol{x}_a) &= \sum_{\boldsymbol{x}_a : \boldsymbol{x}_a \sim \boldsymbol{x}_\gamma} \; \sum_{\boldsymbol{x}_b : \boldsymbol{x}_b \sim \boldsymbol{x}_a} \hat{\tau}_{ab}(\boldsymbol{x}_a, \boldsymbol{x}_b) \\
&= \sum_{\boldsymbol{x}_a : \boldsymbol{x}_a \sim \boldsymbol{x}_\gamma} \; \sum_{\boldsymbol{x}_b : \boldsymbol{x}_b \sim \boldsymbol{x}_\gamma} \hat{\tau}_{ab}(\boldsymbol{x}_a, \boldsymbol{x}_b) \\
= \sum_{\boldsymbol{x}_b : \boldsymbol{x}_b \sim \boldsymbol{x}_\gamma} \; \sum_{\boldsymbol{x}_a : \boldsymbol{x}_a \sim \boldsymbol{x}_b} \hat{\tau}_{ab}(\boldsymbol{x}_a, \boldsymbol{x}_b) &= \sum_{\boldsymbol{x}_b : \boldsymbol{x}_b \sim \boldsymbol{x}_\gamma} \hat{\tau}_b(\boldsymbol{x}_b).
\end{aligned}$$

Case 2: $\gamma \subset a \cap b$, $(a, b) \in \mathcal{E}$. Denote $\gamma' = a \cap b$. Using the result that we just proved we obtain

$$\begin{aligned}
\sum_{\boldsymbol{x}_a : \boldsymbol{x}_a \sim \boldsymbol{x}_\gamma} \hat{\tau}_a(\boldsymbol{x}_a) &= \sum_{\boldsymbol{x}_{\gamma'} : \boldsymbol{x}_{\gamma'} \sim \boldsymbol{x}_\gamma} \; \sum_{\boldsymbol{x}_a : \boldsymbol{x}_a \sim \boldsymbol{x}_{\gamma'}} \hat{\tau}_a(\boldsymbol{x}_a) \\
&= \sum_{\boldsymbol{x}_{\gamma'} : \boldsymbol{x}_{\gamma'} \sim \boldsymbol{x}_\gamma} \; \sum_{\boldsymbol{x}_b : \boldsymbol{x}_b \sim \boldsymbol{x}_{\gamma'}} \hat{\tau}_b(\boldsymbol{x}_b) = \sum_{\boldsymbol{x}_b : \boldsymbol{x}_b \sim \boldsymbol{x}_\gamma} \hat{\tau}_b(\boldsymbol{x}_b).
\end{aligned}$$

Case 3: $(a, b) \notin \mathcal{E}$. Let $(a_0, \dots, a_k)$ be the sequence specified in proposition 3.3 for patches a and b. As proved above, for each i, it holds that

$$\sum_{\boldsymbol{x}_{a_i} : \boldsymbol{x}_{a_i} \sim \boldsymbol{x}_\gamma} \hat{\tau}_{a_i}(\boldsymbol{x}_{a_i}) = \sum_{\boldsymbol{x}_{a_{i+1}} : \boldsymbol{x}_{a_{i+1}} \sim \boldsymbol{x}_\gamma} \hat{\tau}_{a_{i+1}}(\boldsymbol{x}_{a_{i+1}}).$$

Using an induction argument, we obtain the desired result. We have now proved that definition of $\tau_\gamma(\boldsymbol{x}_\gamma)$ does not depend on the choice of a. Showing that obtained vector τ satisfies (3.15c) for each $(\alpha, \beta) \in J$ is straightforward: in the definition (3.20) for $\tau_a(\boldsymbol{x}_a)$ and $\tau_\beta(\boldsymbol{x}_\beta)$ we need to select the same cluster $\hat{a} \in \mathcal{V}$ with $b \subset a \subseteq \hat{a}$, then (3.15c) easily follows. Condition (3.19b) implies (3.15b) for all $a \in \mathcal{V}$; combining this with (3.15c) gives condition (3.15b) for all $a \in \mathcal{F}$. We proved that $\tau \in \Omega$.

Mapping $\Omega \to \hat{\Omega}$. Consider vector $\tau \in \Omega$. We define

$$\hat{\tau}_a(\boldsymbol{x}_a) = \tau_a(\boldsymbol{x}_a), \tag{3.21}$$

for patches $a \in \mathcal{V}$ and labelings $\boldsymbol{x}_a$. For each edge $(a, b) \in \mathcal{E}$ and labelings $\boldsymbol{x}_a \sim \boldsymbol{x}_b$ we define

$$\hat{\tau}_{ab}(\boldsymbol{x}_a, \boldsymbol{x}_b) = \begin{cases} \frac{\tau_a(\boldsymbol{x}_a)\tau_b(\boldsymbol{x}_b)}{\tau_\gamma(\boldsymbol{x}_\gamma)} & \tau_\gamma(\boldsymbol{x}_\gamma) \neq 0 \\ 0 & \tau_\gamma(\boldsymbol{x}_\gamma) = 0, \end{cases} \tag{3.22}$$

where $\gamma = a \cap b$. Let us show that (3.19c) holds for a pair $(a, b) \in \mathcal{E}$ and a fixed labeling $\boldsymbol{x}_a$. Denote $\gamma = a \cap b$, and let $\boldsymbol{x}_\gamma$ be the restriction of $\boldsymbol{x}_a$ to γ. If $\tau_\gamma(\boldsymbol{x}_\gamma) = 0$ then from (3.15c),(3.15d) we have $\tau_a(\boldsymbol{x}_a) = 0$, and so both sides of (3.19c) are zeros. Otherwise we can write

$$\begin{aligned} \sum_{\boldsymbol{x}_b : \boldsymbol{x}_b \sim \boldsymbol{x}_a} \hat{\tau}_{ab}(\boldsymbol{x}_a, \boldsymbol{x}_b) &= \frac{\tau_a(\boldsymbol{x}_a)}{\tau_\gamma(\boldsymbol{x}_\gamma)} \sum_{\boldsymbol{x}_b : \boldsymbol{x}_b \sim \boldsymbol{x}_a} \tau_b(\boldsymbol{x}_b) \\ = \frac{\tau_a(\boldsymbol{x}_a)}{\tau_\gamma(\boldsymbol{x}_\gamma)} \sum_{\boldsymbol{x}_b : \boldsymbol{x}_b \sim \boldsymbol{x}_\gamma} \tau_b(\boldsymbol{x}_b) &= \frac{\tau_a(\boldsymbol{x}_a)}{\tau_\gamma(\boldsymbol{x}_\gamma)} \cdot \tau_\gamma(\boldsymbol{x}_\gamma) = \hat{\tau}_a(\boldsymbol{x}_a) \end{aligned} \tag{3.23}$$

where we used condition (3.15c). We proved that $\hat{\tau} \in \hat{\Omega}$.

We finished the construction of mappings $\hat{\Omega} \to \Omega$ and $\Omega \to \hat{\Omega}$. In both cases we have $\hat{\tau}_a(\boldsymbol{x}_a) = \tau_a(\boldsymbol{x}_a)$ for $a \in \mathcal{V}$, and therefore the mappings are cost-preserving. □

Remark 3.5. *The factors ($\mathcal{F}$) and coupling scheme (J) in* (3.18) *satisfies:*

1. *If $a \in \mathcal{F}$ then any non-empty subset of a is also in $\mathcal{F}$.*
2. *If $a, b \in \mathcal{F}$ and $b \subseteq a$ then $(a, b) \in J$.*

TRW-s solves the dual of the Schlesinger LP (Kolmogorov (2006)). From this it follows that a larger factor set ($\mathcal{F}$) and a larger coupling scheme (J) correspond to a stronger relaxation. A stronger relaxation can give better, but never worse, optimization results as discussed in Section 2.7.3.

Corollary 3.6. *Given a second-order multi-label function f, let f_{pe} be the partial enumeration reformulation as given in* (3.10) *using patches of size at least 2×2. Then TRW-s uses a stronger relaxation when optimizing f_{pe} than when optimizing f.*

Proof. A second-order multi-label function can be expressed as

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} U_i(x_i) + \sum_{(i,j)\in E} B_{ij}(x_i, x_j). \tag{3.24}$$

For the problem of minimizing f, TRW-s uses:

$$\begin{aligned} \mathcal{F} &= \big\{i \cup j : (i,j) \in E\big\} \cup \{1, \ldots, n\}, \\ J &= \big\{(i,j) \in E\big\}. \end{aligned} \tag{3.25}$$

Now consider f_{pe} and the patches $\mathcal{V}$ used in the reformulation. By construction, given any $a \in E$, there exists a patch $v \in \mathcal{V}$ such that $a \subsetneq v$. The corresponding coupling scheme and factors used by TRW-s when optimizing f_{pe} are given in (3.18). The factor set and coupling scheme in (3.18) are clearly larger than the ones in (3.25). □

The following corollary follows from the same argumentation.

Corollary 3.7. *Given a second-order multi-label function f, let f_n be the partial enumeration reformulation as given in* (3.10) *with patches of size $n \times n$. Then TRW-s uses a stronger relaxation when optimizing f_{n+1} than when optimizing f_n.*

The corollary tells us that increasing the patch size also increases the strength of the relaxation. For very difficult problems, using patches larger than the factors in (3.2) may be worthwhile.

3.1.3 Higher-order functions and related work

Partial enumeration can also be used to optimize functions with higher-order terms. In this section we will relate partial enumeration to two other methods able to do the same: generalized TRW-s (GTRW-s), introduced in Kolmogorov and Schoenemann (2012), and dual decomposition, introduced in Komodakis et al. (2011).

GTRW-s is formulated to work with any type of relaxation, including the relaxation in (3.18). For a general type of relaxation, however, GTRW-s is very complex to implement. The currently available implementation, see Schoenemann (2013), only enforces consistency between single variables and pairs of variables. This is a weaker relaxation than the corresponding relaxation used by partial enumeration with 2×2 patches.

Partial enumeration is similar to the dual decomposition construction of Komodakis et al. In their formulation the function is decomposed into, possibly overlapping, patches (in their paper the patches are called "subgraphs" and are more general.) The patches are then enforced to be consistent, just like for the partial enumeration formulation. There is, however, an important difference: For partial enumeration, consistency is strongly enforced using the binary terms in (3.11). This allows each subproblem, which is solved exactly, to contain many more variables than just the variables from one patch, see Figure 3.3. For dual decomposition, consistency is only loosely enforced via dual variables. Then each subproblem, which is solved exactly, contains only the variables from *one* patch. Furthermore, the following contributions of this chapter can be carried over to the dual decomposition formulation of Komodakis et al. (2011):

- *Limited label space of each patch.* It is not required to consider every possible assignment of a patch. For larger patches this is needed to make the algorithm tractable.

- *Limited number of consistency terms.* The consistency between patches does not need to be enforced directly for all overlapping patches; they need only to be enforced indirectly, cf. Definition 3.2.

- *Stronger relaxation.* Even for functions which dual decomposition can minimize directly, without forming patches, decomposition into overlapping patches makes the relaxation stronger.

This chapter approaches the minimization of f differently than dual decomposition and GTRW-s. The function f is first reformulated into an equivalent form, f_{pe}. The function f_{pe} can then be minimized by any of the readily available and highly optimized methods able to minimize a second-order function with large label space.

3.2 Curvature regularization

This section will introduce a novel approach to segmentation using curvature regularization. The curvature costs will be directly encoded into the patches, making these segmentation problems a perfect benchmark for partial enumeration. The segmentation will be given by the minimizer of

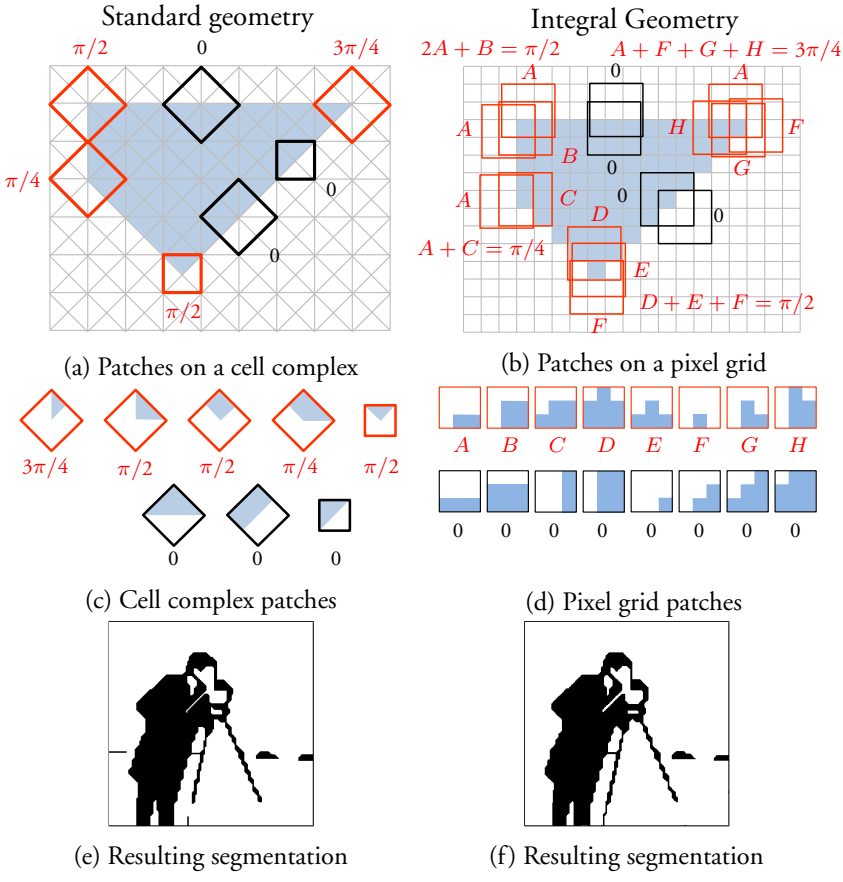
$$f(S) = \int_{\text{int}(S)} d(x)\, dx + w \int_{\partial S} |\kappa(s)| ds, \tag{3.26}$$

where $\kappa(s)$ is curvature, w is a weighting parameter and $d(x)$ is a data term.

A common approach for evaluating curvature is to work with the edges in a mesh of complex patches as used by Schoenemann et al. (2012); El-Zehiry and Grady (2010); Strandmark and Kahl (2011). In Figure 3.4 the proposed approach based on integral geometry is compared to a complex-patch technique.

In Schoenemann et al. (2012) curvature is reduced to unary terms using auxiliary variables. Their integer linear programming approach is formulated over a large number of binary variables defined on fine geometric primitives (vertices, faces, edges, pairs of edges, etc), which are tied by constraints. In contrast, for the proposed approach, unary representation of curvature uses larger-scale geometric primitives (overlapping patches) tied by consistency constraints. The number of corresponding variables is significantly smaller, but the label space is larger.

Another approach to curvature regularization is to enforce the regularization cost using high-order factors, both El-Zehiry and Grady (2010) and Strandmark and Kahl (2011) follow this route. Despite technical differences in the underlying formulations and optimization algorithms, the proposed patch-based approach for complexes and the approaches of Schoenemann et al. (2012); Strandmark and Kahl (2011) use geometrically equivalent models for approximating curvature. That is, all of these models would



(a) Patches on a cell complex

(b) Patches on a pixel grid

(c) Cell complex patches

(d) Pixel grid patches

(e) Resulting segmentation

(f) Resulting segmentation

Figure 3.4: Evaluating the curvature of a image segment on a complex using complex geometry and on a grid using integral geometry. At high resolution, any segment S can be approximated by a polygon. Thus, to minimize functions like $\int_{\partial S} |\kappa(s)| ds$ we need to evaluate all corners. Overlapping patches are created for each vertex on a complex (a) and for each pixel on a grid (b). A patch on a complex consists of all cells adjacent to a vertex and a grid patch is a square window centered at a pixel. Each corner on a complex (a) can be directly evaluated from a labeling of a single patch using standard geometry. However, each corner on a grid (b) should be evaluated using integral geometry by summing over multiple patches covering the corner. Black patches occur at straight boundaries and should have zero curvature. Red patches correspond to curved boundaries. The weights $\{A, \ldots, H\}$ for all assignments in (d) can be precomputed. The accuracy of integral geometry approach to curvature on a grid is comparable to the standard basic geometry used on complexes, see (c) and (d).

produce the same solution if there were exact global optimization algorithms for them. The optimization algorithms for these models do however vary in quality, memory, and run-time efficiency.

In practice, grid patches are easier to implement than complex patches because of the grids' regularity and symmetry. Grid patches were also recently used for curvature evaluation in Shekhovtsov et al. (2012). Unlike the integral geometry in Figure 3.4(c), their method computes a minimum response over a number of affine filters encoding some learned "soft" patterns. The response to each filter combines the deviation from the pattern and the cost of the pattern. The mathematical justification of this approach to curvature estimation is not fully explained and several presented plots indicate its limited accuracy. As stated in Shekhovtsov et al. (2012), "the plots do also reveal the fact that we consistently overestimate the true curvature cost."

3.2.1 Patch-cost assignments

We will now explain in detail how curvature cost can be approximated for different patch sizes. For small patches the curvature estimations are very coarse and we are unable to penalize small curvature costs. As the patch size grows the precision increases. Throughout this chapter we only concern ourselves with the absolute curvature of polygons. For a polygon, the integral of the absolute curvature corresponds to the sum of the exterior angles in the polygon, see Bruckstein et al. (2001). Hence, for each patch size we need only to determine the exterior angles of the polygonal segmentation boundary.

Patches of size 2×2

Patches of size 2×2 are the smallest ones we consider, these are able to approximate curvature with $\pi/2$ precision. Each patch has 4 pixel boundaries that intersect in the middle of the patch. To compute the curvature contribution of a patch we need to determine which of the 4 pixel boundaries also belong to the segmentation boundary. If two neighboring pixels are sharing a boundary and have different assignments, then their common boundary belongs to the segmentation boundary. Each patch can take 16 values corresponding to assignments of the individual pixels. It is straightforward to assign each of these assignments its curvature cost.

Figure 3.5 shows all the possible labelings for this case. The unary terms are added to the cost of each different labeling. Note that each pixel may be in as many as four different patches. The patch costs need to be reweighted to compensate for this.

Patches of size 3×3

Patches of size 3×3 are able to approximate the curvature cost with $\pi/4$ precision. Since patches are overlapping, changes in boundary direction will be visible in the assignments of more than one patch. We need to make sure that the total contribution of the patch assignments equals the curvature of the segmentation boundary.

To determine the assignments in the case of 3×3 patches, we generate windows of size 5×5. These windows contain the binary assignments that would result from a segmentation boundary which transitions between two directions at the center of the window, see Figures 3.6. By looking in these windows it is possible to determine all patch assignments that are present in the vicinity of such a transition and constrain their sum to be the correct curvature penalty.

Consider for example the window in Figure 3.7. If we let the labels of the patch be l_a, where $a \in \{0, \ldots, 511\}$ encodes the assignments of the individual pixels, then the patch in Figure 3.7 gives us the linear constraint

$$l_{38} + l_{311} + l_{447} + l_{452} + l_{486} + l_{503} + l_{504} + l_{508} + l_{510} = \frac{3\pi}{4}. \quad (3.27)$$

In a similar way each window/boundary transition gives us a linear equality constraint. In addition, it is required that $l_a \geq 0$ for all $a \in \{0, \ldots, 511\}$ and that $l_a = \infty$ for the labels that do not occur in any of the windows. This gives us a system of linear equalities and inequalities. To find a solution we randomly select a linear objective function (with positive entries) and solve the resulting linear program. Since the system is underdetermined the individual label costs can vary depending on the random objective function. However, the linear equalities ensure that the resulting curvature estimate obtained for each transition between boundary directions is correct when combining patch assignments.

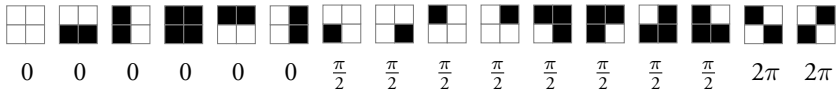


Figure 3.5: All 16 patch assignments used to encode curvature cost for 2×2 patches. Note that there are only four unique patches up to rotation inversion and reflection.

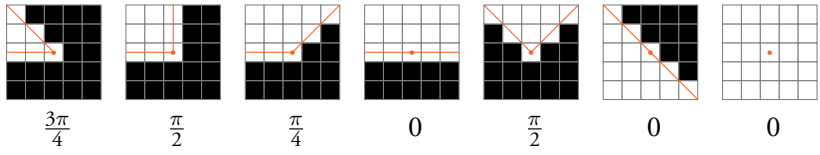


Figure 3.6: Seven of the 5×5 windows used for computing curvature cost for 3×3 patches. The rest are obtained through rotations, reflections, and inversions of the pixel assignments. The line shows the interpretation of the segmentation boundary and its corresponding curvature is given below.

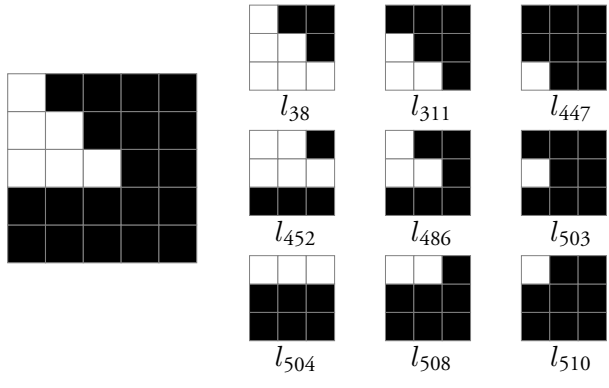


Figure 3.7: First window of Figure 3.6 and its patch assignments.

Patch size	Precision	$\lvert \boldsymbol{L} \rvert$
2×2	$\pi/2$	16
3×3	$\pi/4$	122
5×5	$\pi/8$	2422

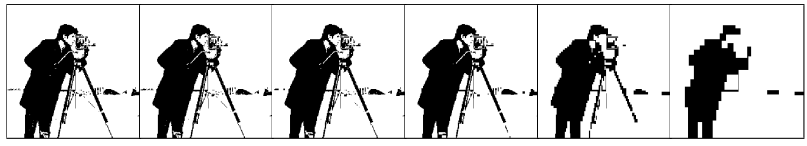
Table 3.1: The three different patch sized use to approximate curvature. Larger patches yields better precision, at the cost of increased label space.

Patches of size 5×5

Patches of size 5×5 are able to approximate curvature cost with $\pi/8$ precision. It is straightforward in direct analogy to the 3×3 case to create even larger training patches to calculate the cost for every possible 5×5 patch. With these larger patches comes additional problems. For the 5×5 patches, the number of possible enumerations is large, $2^{25} = 33554432$. If we restrict our attention to boundaries with penalties in $\{0, \pi/8, \ldots, 2\pi\}$, many of these enumerations can be discarded. We end up with 2422 different labelings for each 5×5 patch. Any other labeling is given infinite cost and is never explicitly stored. Similarly, for the 3×3 patches we only need to store 122 labelings. This approach significantly simplifies the optimization of f_{pe} by reducing the label space for each patch. The results are summarized in Table 3.1.

3.2.2 $\pi/2$ precision curvature

As observed by El-Zehiry and Grady (2010) the 2×2 curvature interaction can be represented by a third-order pseudo-boolean function. In the 8-connected setting, pairs of edges can be used to approximate the angles of the polygon and thus the curvature. The resulting function can be reduced into a second-order function and minimized by a number of different methods. However, it may still be useful to form patches of size 2×2, even though they are larger than necessary, since this reformulation results in a stronger relaxation. In Figure 3.8, the partial enumeration reformulation is compared to the third-order approach of El-Zehiry and Grady. In the figure the reformulation is also compared to generalized roof duality (GRD). GRD is able to directly optimize the third-order function. The function however is very difficult for GRD to minimize and it fails for strong regularization.

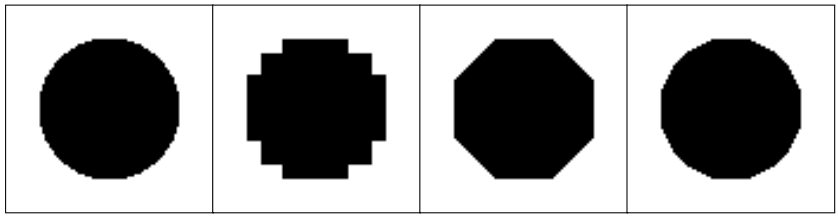


w	$\frac{10^{-4}}{4}$	$\frac{10^{-3}}{4}$	$\frac{10^{-2}}{4}$	$\frac{10^{-1}}{4}$	$\frac{10^{0}}{4}$	$\frac{10^{1}}{4}$

w	Partial Enumeration			El-Zehiry and Grady	
	Obj.	Lower bound	Time (s)	Obj.	Lower bound
$10^{-4}/4$	**4677**	**4677**	0.934	**4677**	**4677**
$10^{-3}/4$	**4680**	**4680**	0.961	**4680**	**4680**
$10^{-2}/4$	**4707**	**4707**	2.43	4709	4705
$10^{-1}/4$	**4910**	**4910**	3.98	5441	4501
$10^{0}/4$	**5833**	**5833**	14.3	16090	−16039
$10^{1}/4$	**7605**	**7605**	28.8	15940	−19990

w	GRD		GRD-heuristic	
	unlabeled	Time (s)	unlabeled	Times(s)
$10^{-4}/4$	**0**%	10737	**0**%	7.08
$10^{-3}/4$	**0**%	9287	**0**%	10.7
$10^{-2}/4$	0.2%	10731	0.2%	7.32
$10^{-1}/4$	6.7%	12552	6.8%	6.96
$10^{0}/4$	100%	12337	100%	10.9s
$10^{1}/4$	100%	7027	100%	22.2

Figure 3.8: Global optimal results for 2×2 curvature with different regularization weight w are shown in the top row. The function in El-Zehiry and Grady (2010) is optimized using TRW-s. Generalized roof duality (GRD) fails to recover a complete labeling for strong regularization. Partial enumeration optimized via TRW-s managed to recover the global optimal solution for all regularization strengths.



(a) Data term (b) 2×2 (c) 3×3 (d) 5×5

Figure 3.9: Segmentation results on a 81×81 pixel image with very high regularization weight, w. 2×2 patches clearly favor horizontal and vertical boundaries. 3×3 patches favor directions that are multiples of $\pi/4$. 5×5 patches favor directions that are multiples of $\pi/8$.

3.2.3 $\pi/4$ and $\pi/8$ precision curvature

For patches of size 2×2 it is only possible to encourage horizontal and vertical boundaries. To make the model more accurate, and include directions that are multiples of $\pi/4$ radians, larger patches are needed, see Figure 3.4. This section presents experiments using 3×3 and 5×5 patches, which are able to encourage angles of $\pi/4$ and $\pi/8$.

In Figure 3.9 a simple example is given. The data term is constructed as a discretized circle, giving all pixels within some radius a very low cost and all other pixels a very high cost. Since there is no noise in the image, truncating the data term would correspond to the optimal solution given arbitrary fine precision. The regularization penalty, w, used in these experiments is very high, exposing the preferred angles for each patch size.

In Figure 3.11 an experiment on a real image is given. In this experiment the original problem is optimized using GTRW-s, and the reformulated problem is optimized using a number of different solvers; TRW-s, MPLP and LBP. TRW-s and LBP is also used with and without the efficient message-passing method discussed in Section (3.1.1). Of all methods tried, TRW-s with efficient message passing is a clear winner both in terms of quality of the solution and execution time. This point is further highlighted in Figure 3.10 where the convergence plots for 2×2 patches are given, for all the different optimizers.

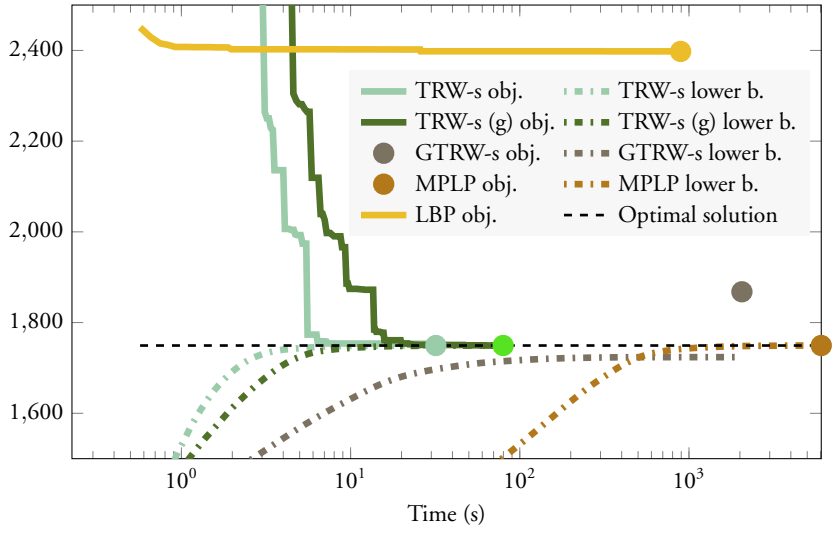
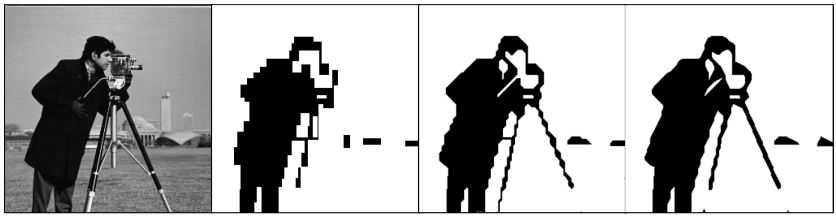


Figure 3.10: Logarithmic time plot for the objective value and lower bound over time for the 2×2 patches experiment in Figure 3.11.



(a) Image (b) 2×2 (c) 3×3 (d) 5×5

PE	Optimizer	Obj.	Lower b.	Time (s)	Obj.	Lower b.	Time (s)
Yes	TRW-s	**1749.4**	**1749.4**	**21**	**1505.7**	**1505.7**	**355**
Yes	TRW-s (g)	**1749.4**	**1749.4**	1580	**1505.7**	**1505.7**	41503
Yes	MPLP	**1749.4**	**1749.4**	6584	‡	‡	‡
Yes	LBP	2397.7		1565	∗		3148
No	GTRW-s	1867.9	1723.8	2189	99840	1312.6	10785
		2×2			3×3		

PE	Optimizer	Obj.	Lower b.	Time (s)
Yes	TRW-s	**1417.0**	**1416.6**	**8829**
Yes	TRW-s (g)	‡	‡	‡
Yes	MPLP	‡	‡	‡
Yes	LBP	∗		157532
No	GTRW-s	‡	‡	‡
		5×5		

Figure 3.11: Segmentation of the cameraman, 256×256 pixels, using 2×2, 3×3 and 5×5 patches with $w = 1$. The image I is scaled to $[0, 1]$ and the data term is $d(\boldsymbol{x}) = I(\boldsymbol{x}) - 0.5$. TRW-s and LBP uses efficient message passing whereas TRW-s (g) uses general message passing. GTRW-s encodes the higher-order costs directly, all other methods solves the same partial enumeration reformulation. Thus, GTRW-s solves a weaker relaxation which is confirmed by the results. All algorithms have an upper bound of 10,000 iterations. In addition, for LBP, TRW-s and GTRW-s is stopped when the lower bound stops increasing. MPLP is also stopped if the duality gap is less than 10^{-4}.
(‡) Creating the problem instance not feasible due to excessive memory usage. (∗) Inconsistent labeling.

3.3 Binary deconvolution

In binary deconvolution the goal is to, given a noise image y, reconstruct the original binary image x. The original image is assumed to be blurred by a known convolution matrix H as

$$y = Hx. \tag{3.28}$$

The binary assumption makes it possible to reformulate the inverse problem into a second-order pseudo-boolean function. The resulting function f is non-submodular if H is non-symmetric and it is usually hard to minimize. Each second-order term is possible to cover by a 2×2 patch, but this enumeration is not strong enough. To get a stronger relaxation, patches of 3×3 are formed.

Figure 3.12 gives an example result, comparing partial enumeration to a number of different methods. Partial enumeration is the only method able to recover the optimal solution.

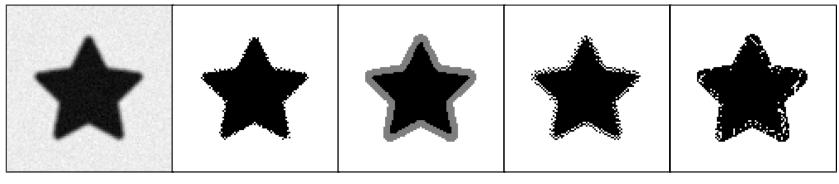


Image PE RD TRW-s MPLP

	Obj.	Lower bound	Unlabeled	Times (s)
Partial enumeration	**7.1553**	**7.1553**	**0**	15.45
Roof duality	241.761	-57.542	909	2.59
TRW-s	18.6821	-57.542	**0**	**1.04**
MPLP	184.2140	-57.542	**0**	777.91

Figure 3.12: Binary deconvolution on a 100×100 pixel image. Each algorithm has a maximum of 1,000 iterations. The resulting images are given in the top row. The unlabeled pixels for roof duality are shown in gray. For roof duality the objective value is given by setting each unlabeled pixel to be background. The partial enumeration reformulation was optimized using TRW-s with efficient message passing.

3.4 Dense stereo regularization

In this section, functions in Woodford et al. (2009) are optimized. The regularization part of the function penalizes the second-order derivatives of the disparity map, either using a truncated ℓ_1- or ℓ_2-penalty. The second-order derivatives are estimated from three consecutive disparity values, vertically or horizontally, resulting in third-order interactions. The resulting objective function is a third-order pseudo-boolean function. The function is optimized using fusion moves. Each fusion move can be reduced to a second-order function by introducing auxiliary variables. For these experiments the problem is decomposed into 3×3 patches which contain the third-order interaction. In Woodford et al. (2009) the resulting relaxation is then solved using roof duality. Table 3.2 shows the results for the CONES dataset from Scharstein and Szeliski (2003).

	Obj. value	Lower bound	Time (s)
Partial Enumeration	$\mathbf{1.4558 \cdot 10^{10}}$	$\mathbf{1.4558 \cdot 10^{10}}$	315.3342
Roof duality	$1.4637 \cdot 10^{10}$	$1.4518 \cdot 10^{10}$	**1.9216**
PE/RD	0.9958	1.0019	180.6859

(a) ℓ_1 regularization, RD left on average 24% of the variables unlabeled.

	Obj. value	Lower bound	Time (s)
Partial Enumeration	$\mathbf{1.3594 \cdot 10^{10}}$	$\mathbf{1.3594 \cdot 10^{10}}$	428.2557
Roof duality	$1.5165 \cdot 10^{10}$	$1.0484 \cdot 10^{10}$	**4.6479**
PE/RD	0.9092	1.1652	111.8597

(b) ℓ_2 regularization, RD left on average 64% of the variables unlabeled.

Table 3.2: Averaged stereo results on CONES sequence when fusing SegPln proposals in Woodford et al. (2009). To ensure that each subproblem is identical for the two approaches, the solution from roof duality is used for partial enumeration in each fusion move. The partial enumeration reformulation was optimized using TRW-s with efficient message passing.

3.5 Conclusions

In this chapter, it has been shown how to reformulate a large class of functions into an equivalent form. The reformulation is theoretically justified by showing that it is beneficial when the optimization is performed using TRW-s. Experimentally, the benefits are also shown for the following applications: a novel curvature segmentation framework, binary deconvolution and dense stereo estimation.

Chapter 4

Parametric max-flow

In this chapter, we focus on solving the *parametric max-flow problem*.

Problem 4.1 (Parametric max-flow)**.** *Let* $\boldsymbol{\gamma} \in \mathbf{R}^d$*. Find* all *minimizers to*

$$f_{\boldsymbol{\gamma}}(\boldsymbol{x}) = \sum_{i=1}^{n} U_i(x_i) + \sum_{i=1}^{n} \sum_{j>i}^{n} B_{ij}(x_i, x_j) + \sum_{i=1}^{n} \sum_{j=1}^{d} \gamma_j c_{ij} x_i \quad (4.1)$$

where $\boldsymbol{x} \in \mathbf{B}^n$ *and* $f_{\mathbf{0}}$ *is submodular, given any choice of* $\boldsymbol{\gamma}$*. Each solution defines a region in* $\mathbf{R}^d$ *and the set of all these regions is called the* solution diagram.

The parametric max-flow problem with $d = 1$ is known as the *one-parameter max-flow problem*. This special case has found its uses in computer vision. In Kolmogorov, Boykov, et al. (2007), ratios of form

$$r(\boldsymbol{x}) = \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} \quad \text{with } q(\boldsymbol{x}) > 0, \quad (4.2)$$

are minimized using a one-parameter max-flow formulation. In Carreira and Sminchisescu (2010), one-parameter max-flow is used to generate sets of segmentations as part of their object recognition pipeline.

The parametric max-flow problem with $d \geq 2$ is not well studied in computer vision, even though forms of the commonly used Chan-Vese function belong to this family of problems.

Problem 4.2 (Chan-Vese, Chan and Vese (2001))**.** *Given an image*

$$I : [1, a] \times [1, b] \to [0, 1] \quad (4.3)$$

with $ab = n$ pixels, approximate the foreground and the background using only two values, μ and λ, by solving

$$\begin{aligned}\underset{\boldsymbol{x}\in\mathbf{B}^n}{\operatorname{argmin}} \min_{\mu,\lambda\in\mathbf{R}} &\sum_{i=1}^{n}(1-x_i)\big(I(i)-\mu\big)^2 + x_i\big(I(i)-\lambda\big)\big)^2 \\ &+ \sum_{i=1}^{n}\sum_{j>i}^{n} B_{ij}(x_i,x_j),\end{aligned} \tag{4.4}$$

where $I(i)$ is the intensity of pixel i.

In Strandmark, Kahl, and Overgaard (2009) Problem 4.2 is solved via branch and bound, where the sub solver is an efficient one-parameter max-flow solver. We can reformulate Problem (4.2) into the following problem.

Problem 4.3 (Complete Chan-Vese)**.** *Find* all *solutions optimal for any choice of μ and λ in Problem* (4.2).

Proposition 4.4. *Solving Problem 4.3 corresponds to solving a two-parameter max-flow problem, and a subset of its solutions is the optimal solution to Problem* (4.2).

Proof. The unary term of (4.4) can be reformulated as as

$$\sum_{i=1}^{n} 2\big(\lambda^2-\mu^2\big)x_i + 2I(i)(\mu-\lambda)x_i + \mu^2 - 1 - 2I(i)\mu. \tag{4.5}$$

Here $\mu^2 - 1 - 2I(i)\mu$ is independent of $\boldsymbol{x}$ and can be removed since it will not influence the solution. Introduce new variables as

$$\begin{aligned}\gamma_1 &= \lambda^2 - \mu^2, \\ \gamma_2 &= \mu - \lambda.\end{aligned} \tag{4.6}$$

The variable change allows us to define

$$c_{i1} = 2, \tag{4.7}$$

$$c_{i2} = 2I(i). \tag{4.8}$$

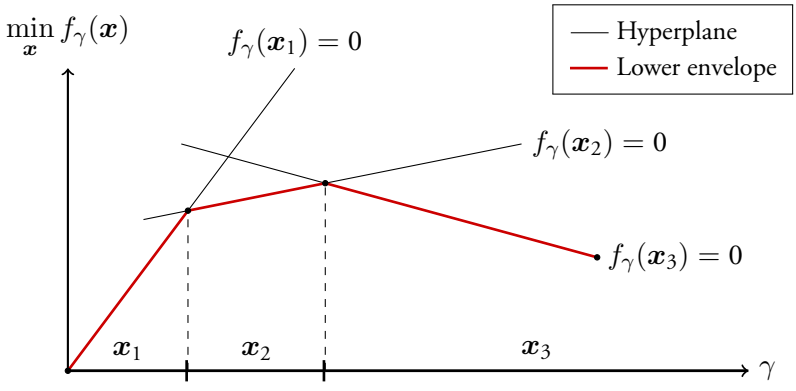


Figure 4.1: An simple example showing the hyperplanes and solution diagram for $\gamma \in \mathbf{R}$. The three variable assignments $\boldsymbol{x}_1, \boldsymbol{x}_2$ and $\boldsymbol{x}_3$ all define a hyperplane in $\mathbf{R}^2$ (a line.) The solution diagram is simply the partition of $\mathbf{R}$ shown along the γ-axis.

Finally this allows us to rewrite (4.4) into

$$f_{\boldsymbol{\gamma}}(\boldsymbol{x}) = \sum_{i=1}^{n} \sum_{j>i}^{n} B_{ij}(x_i, x_j) + \sum_{i=1}^{n} \left(\gamma_1 c_{i1} + \gamma_2 c_{i2}\right). \tag{4.9}$$

This function is of the form given in Problem 4.1 showing the first claim.

Secondly, we have found all solutions given any parameter choice, in particular this includes the optimal choice in the sense of Problem (4.2). □

4.1 Algorithms

4.1.1 Exact solution

In this section we will motivate and describe the algorithm of Fernández-Baca and Srinivasan (1991) which can solve the parametric max-flow problem for any dimension of the parameter vector $\boldsymbol{\gamma}$. Solving the parametric max-flow problem can also be expressed as finding the function

$$h(\boldsymbol{\gamma}) = \min_{\boldsymbol{x} \in \mathbf{B}^n} f_{\boldsymbol{\gamma}}(\boldsymbol{x}), \tag{4.10}$$

where $\gamma \in \mathbf{R}^d$. Here h is piecewise-linear and concave function and the graph to h defines a concave polytope in $\mathbf{R}^{d+1}$. Any solution

$$\boldsymbol{y} = \underset{\boldsymbol{x} \in \mathbf{B}^n}{\operatorname{argmin}} f_{\boldsymbol{\gamma}_y}(\boldsymbol{x}), \tag{4.11}$$

defines a hyperplanes in $\mathbf{R}^{d+1}$ as

$$h(\boldsymbol{\gamma}_y) + ((\boldsymbol{\gamma} - \boldsymbol{\gamma}_y) \cdot \boldsymbol{c})^{\mathrm{T}} \boldsymbol{y} = 0, \tag{4.12}$$

where $\boldsymbol{c}$ is a vector of all weights in (4.1) and $\cdot$ denote elementwise multiplication. Now suppose we have evaluated $h(\gamma)$ for k different values of γ giving us k hyperplanes. We can define the lower envelope of all these hyperplanes as

$$\begin{aligned} C^k(\boldsymbol{\gamma}) = \min\big\{ & h(\boldsymbol{\gamma}_0) + ((\boldsymbol{\gamma} - \boldsymbol{\gamma}_0) \cdot \boldsymbol{c})^{\mathrm{T}} \boldsymbol{x}_0), \ldots, \\ & h(\boldsymbol{\gamma}_{k-1}) + ((\boldsymbol{\gamma} - \boldsymbol{\gamma}_{k-1}) \cdot \boldsymbol{c})^{\mathrm{T}} \boldsymbol{x}_{k-1} \big\}. \end{aligned} \tag{4.13}$$

Now if we were to evaluate every possible $\boldsymbol{\gamma} \in \mathbf{R}^d$, we would end up with $C^k(\boldsymbol{\gamma}) = h(\boldsymbol{\gamma})$. This is not possible to do and fortunately not needed. The idea behind the algorithm is to successively build the polytope defining the graph to h. For each point $\boldsymbol{\gamma} \in \mathbf{R}^d$, for which we solve (4.11) we do the following.

1. Intersect the hyperplane in (4.12) with the polytope.
2. Keep track on any added corners points in the polytope produced by the intersection.

In Fernández-Baca and Srinivasan (1991) it is shown that we only need to consider the corners points produced by the intersection. Once we have evaluated all corners points we end up with $C^k(\boldsymbol{\gamma}) = h(\boldsymbol{\gamma})$. This summed up and formalized in Algorithm 1.

Remark 4.5. *Dual ascent corresponds to solving*

$$\underset{\boldsymbol{\gamma} \in \mathbf{R}^d}{\textit{maximize}}\ h(\boldsymbol{\gamma}), \tag{4.14}$$

where $\boldsymbol{\gamma}$ are the dual variables. The parametric max-flow problem on the other hand finds the optimal solutions given any $\boldsymbol{\gamma}$. From this it follows that the optimal dual ascent solution can be found in the solution diagram to the parametric max-flow problem.

Algorithm 1 Solving the parametric max-flow problem.

Let $u(\boldsymbol{\gamma})$ be the current approximation of $h(\boldsymbol{\gamma})$ in (4.10).
u is stored as the polytope defining the graph of u.
Let $\boldsymbol{c}$ be a list of unvisited points where $\boldsymbol{u} \in \mathbf{R}^d$ for all $\boldsymbol{u} \in \boldsymbol{c}$.

Restrict the search space to $\boldsymbol{\gamma}_{\min} \times \boldsymbol{\gamma}_{\max}$.
Add all points in the search space to $\boldsymbol{c}$.
Initialize u such that $u(\boldsymbol{\gamma}) = \infty$ for each point in the search space.
while c is non-empty **do**
  $\boldsymbol{\gamma} \leftarrow \boldsymbol{u} \in \boldsymbol{c}$.
  $\boldsymbol{c} \leftarrow \boldsymbol{c} - \boldsymbol{u}$.
  $\boldsymbol{y} \leftarrow \min_{\boldsymbol{x} \in \mathbf{B}^n} f_{\boldsymbol{\gamma}}(\boldsymbol{x})$.
  Update the approximation $u(\boldsymbol{\gamma})$ by intersecting the hyperplane in (4.12) with graph of u.
  Add all new intersection points to $\boldsymbol{c}$.
  Remove any point in $\boldsymbol{c}$ above hyperplane.
return u.

Proposition 4.6. *Let $T(n)$ be the times it takes to solve*

$$\operatorname*{argmin}_{\boldsymbol{x} \in \mathbf{B}^n} f_{\gamma}(\boldsymbol{x}), \tag{4.15}$$

and let b be the number of solutions in the solution diagram. The computational complexity of Algorithm 1 is

$$\mathcal{O}\left(bT(n) + b^2\right). \tag{4.16}$$

Proof. Using appropriate data structures it is shown in Fernández-Baca and Srinivasan (1991). □

4.1.2 Approximate solution

Suppose we are only interested in finding an approximate solution diagram. Depending on application different approximations might be useful. If we are only looking to get k solutions, Algorithm 1 can be stopped once k solutions have been found. However, this will give us no information on how well the current solution polytope approximate the graph to h. This is something the algorithm described in this section remedies.

Let $\ell(\gamma)$ and $u(\gamma)$ be any lower and upper bound functions to $h(\gamma)$ for all $\gamma \in \mathbf{R}^d$. We would like to stop algorithm once

$$g(\boldsymbol{\gamma}) = u(\boldsymbol{\gamma}) - \ell(\boldsymbol{\gamma}) \leq a \quad \text{for all } \boldsymbol{\gamma} \in \mathbf{R}^d, \tag{4.17}$$

for some constant a. In each iteration k of Algorithm 1 the approximate lower envelope of (4.13) define an upper bound to h as

$$u(\boldsymbol{\gamma}) = \min_{\boldsymbol{\gamma} \in \mathbf{R}^d} C^k(\boldsymbol{\gamma}). \tag{4.18}$$

Furthermore, the upper boundary of the convex hull of each visited points defines a lower bound function $\ell(\gamma)$, see Figure 4.2.

Proposition 4.7. *Consider Algorithm 1 and the gap g, defined in* (4.17). *Let $\mathbf{v}$ be all points in $\mathbf{R}^d$ which have been visited by the algorithm and $\mathbf{c}$ be all corner points in the current solution diagram. Then*

$$\operatorname*{argmax}_{\boldsymbol{\gamma} \in \mathbf{R}^d} g(\boldsymbol{\gamma}) \in \{\boldsymbol{c} \setminus \mathbf{v}\}. \tag{4.19}$$

Proof. First note that both graphs of $u(\boldsymbol{\gamma})$ and $\ell(\boldsymbol{\gamma})$ are built up by facets which each has a unique normal direction. Let $f_u(\boldsymbol{\gamma})$ and $f_\ell(\boldsymbol{\gamma})$ denote the facets associated with $u(\boldsymbol{\gamma})$ and $\ell(\boldsymbol{\gamma})$ respectively for $\boldsymbol{\gamma}$. Now divide $\mathbf{R}^d$ into regions such that $r \subset \mathbf{R}^d$ and

$$f_u(\boldsymbol{\gamma}_1) = f_u(\boldsymbol{\gamma}_2) \qquad \text{for all } \boldsymbol{\gamma}_1, \boldsymbol{\gamma}_2 \in r, \tag{4.20}$$

$$f_\ell(\boldsymbol{\gamma}_1) = f_\ell(\boldsymbol{\gamma}_2) \qquad \text{for all } \boldsymbol{\gamma}_1, \boldsymbol{\gamma}_2 \in r. \tag{4.21}$$

Consider any region r. We have two different cases:

1. If the normal direction f_u is equal to the normal direction of f_v. Then $g(\gamma)$ is equal for all $\gamma \in r$ and it suffices to check the corner points of the region.

2. If the normal directions of f_u and f_v differ then the gap is largest at the boundary and it suffices to check corner points of the region.

The corner points of all these regions are built up by the set $\boldsymbol{c}$. However, by construction $g(\boldsymbol{\gamma}) = 0$ for all $\boldsymbol{\gamma} \in \mathbf{v}$. It follows that we only need to check the points in $\{\boldsymbol{c} \setminus \mathbf{v}\}$. □

Using Proposition 4.7 it is straightforward to construct an approximate algorithm, by adding convergence criteria to a Algorithm 1, resulting in Algorithm 2.

Algorithm 2 Approximate solution to the parametric max-flow problem.

Let $\ell(\boldsymbol{\gamma})$ be the upper boundary of the convex hull of all visited points.
Let $u(\boldsymbol{\gamma})$ be the current approximation of $h(\boldsymbol{\gamma})$ in (4.10).
u is stored as the polytope defining the graph of u.
Let $\boldsymbol{c}$ be a list of unvisited points where $\boldsymbol{u} \in \mathbf{R}^d$ for all $\boldsymbol{u} \in \boldsymbol{c}$.

Restrict the search space to $\boldsymbol{\gamma}_{\min} \times \boldsymbol{\gamma}_{\max}$.
Add all points in the search space to $\boldsymbol{c}$.
Initialize u such that $u(\boldsymbol{\gamma}) = \infty$ for each corner in the search space.
Initialize l such that $l(\boldsymbol{\gamma}) = -\infty$ for each corner in the search space.
while c is non-empty **do**
  $\boldsymbol{\gamma} \leftarrow \boldsymbol{u} \in \boldsymbol{c}$.
  $\boldsymbol{c} \leftarrow \boldsymbol{c} - \boldsymbol{u}$.
  $\boldsymbol{y} \leftarrow \min_{\boldsymbol{x} \in \mathbf{B}^n} f_{\boldsymbol{\gamma}}(\boldsymbol{x})$.
  Update the lower bound $\ell(\boldsymbol{\gamma})$ using $\boldsymbol{y}$.
  Update the approximation $u(\boldsymbol{\gamma})$ by intersecting the hyperplane in (4.12) with graph of u.
  Add all new intersection points to $\boldsymbol{c}$.
  Remove any point in $\boldsymbol{c}$ above hyperplane.
  Check convergence, $u(\boldsymbol{\gamma}) - \ell(\boldsymbol{\gamma}) < a$ for all $\boldsymbol{\gamma} \in \boldsymbol{c}$.
return u.

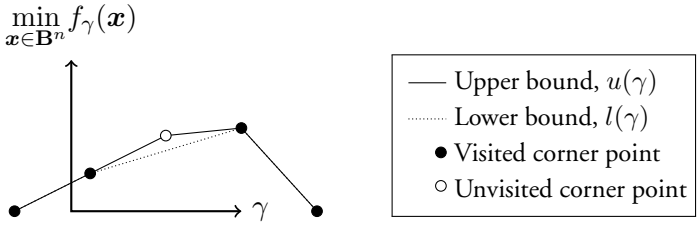


Figure 4.2: The idea behind the approximate algorithm. Suppose we have visited four corner points, and there is one corner point yet to consider. The solid line $u(\gamma)$ correspond to the current estimate of $C^k(\gamma)$ in (4.13). The dashed line $l(\gamma)$ correspond to the upper boundary of the convex hull of all visited points. The graph of h in (4.10) is somewhere in between the two lines.

4.2 Theoretical results

4.2.1 One-parameter

For notational simplicity define a one-parameter instance of (4.1) as

$$f_\lambda(\boldsymbol{x}) = \left(\sum_{i=1}^{n} a_i + b_i \lambda \right) x_i + \sum_{i=1}^{n} \sum_{j>i}^{n} B_{ij}(x_i, x_j). \tag{4.22}$$

An important question to answer is: how many solutions are there?

Proposition 4.8. *Given a one-parameter max-flow problem given in* (4.22)*. If each b_i has the same sign, then there can be at most n solutions*

Proof. Suppose each b_i is positive then:

- if $\lambda \to \infty$ then $\underset{\boldsymbol{x} \in \mathbf{B}^n}{\operatorname{argmin}} f_\lambda(\boldsymbol{x}) = \mathbf{0}$,
- if $\lambda \to -\infty$ then $\underset{\boldsymbol{x} \in \mathbf{B}^n}{\operatorname{argmin}} f_\lambda(\boldsymbol{x}) = \mathbf{1}$.

For some λ the optimal labeling for x_i changes from 0 to 1. Further decreasing λ will never make $x_i = 0$ less costly. Hence each variable changes label at most one time. The maximum number of solutions is reached if each variable changes labeling at a different λ.

The case when each b_i is negative follow by an analogous derivation. □

When the sign of b_i is unrestricted it is much harder to constrain the maximum number of solutions. There are very rough upper bound results.

Proposition 4.9. *Given a one-parameter max-flow problem on form* (4.22) *with n variables. Then there can be at most $\mathcal{O}\left(2^{\sqrt{n}}\right)$ solutions.*

Proof. See Carstensen (1983). □

The maximum number of assignments is 2^n so $\mathcal{O}\left(2^{\sqrt{n}}\right)$ is a signification reduction. Still for computer vision applications n may be very large and $2^{\sqrt{n}}$ is in most cases intractable.

4.2.2 Two-parameters

For notational simplicity define a two-parameter instance of (4.1) as

$$f_{\{\lambda,\mu\}}(\boldsymbol{x}) = \sum_{i=1}^{n} (a_i + c_i\lambda + d_i\mu)\, x_i + \sum_{i=1}^{n}\sum_{j>i}^{n} B_{ij}(x_i, x_j). \quad (4.23)$$

One might hope that restricting $c_i, d_i \geq 0$ will lower the bound on the number of solutions similar to the one-parameter case. Unfortunately this is not the case.

Proposition 4.10. *Given a two-parameter max-flow problem of the form given in* (4.23)*. Restricting the signs of all c_i and d_i to be equal results in a problem with at least as many solutions as some one-parameter max-flow problem given on form* (4.22)*.*

Proof. We will prove this by taking any one-parameter problem and transforming it to a two-parameter problem with the given restrictions. We begin by considering the case when c_i and d_i are all non-negative. Now given any one-parameter problem we can construct a two-parameter problem fulfilling the restrictions as

$$\begin{cases} c_i &= b_i \\ d_i &= 0 \end{cases} \text{ if } \quad b_i \geq 0, \quad (4.24)$$

and

$$\begin{cases} c_i &= 0 \\ d_i &= -b_i \end{cases} \text{ if } \quad b_i < 0. \quad (4.25)$$

Now the line $\lambda = -\mu$ in the solution diagram to (4.23) corresponds to

$$f_\lambda(\boldsymbol{x}) = \sum_{i=1^n} (a_i + c_i\lambda_i - d_i\lambda_i)x_i + \sum_{i=1}^{n}\sum_{j>i}^{n} B_{ij}(x_i, x_j) \quad (4.26)$$

$$= \sum_{i=1^n} (a_i + b_i\lambda)x_i + \sum_{i=1}^{n}\sum_{j>i}^{n} B_{ij}(x_i, x_j). \quad (4.27)$$

This is a function of form (4.22) with no restrictions on the coefficients. The case of non-positive coefficients can be shown using the same technique. □

4.3 Experiments

An essential part of Algorithm 1 is repeated intersection of hyperplanes. If these intersections are calculated with float-number precision, numerical issues are bound to occur. To overcome this issue, the intersections are calculated using exact arithmetic via the CGAL library, see Hachenberger and Kettner (2000). This is made possible by restricting all coefficients in (4.1) to be integers. The resulting coefficients for each facet in the solution polytope can then be represented by rational numbers. Due to limitations in CGAL, the algorithm has only been implemented for $\gamma \in \mathbf{R}^2$.

A random problem. All random problem instances are constructed on a $n \times n$ grid as

$$\begin{aligned} f_n(\boldsymbol{x}) = &\sum_{i=1}^{n^2} (a_i + c_i\lambda + d_i\mu)\, x_i \\ &+ \sum_{i=1}^{n^2} \sum_{j>i}^{n^2} B_{ij}(x_i(1-x_j) + (1-x_i)x_j). \end{aligned} \tag{4.28}$$

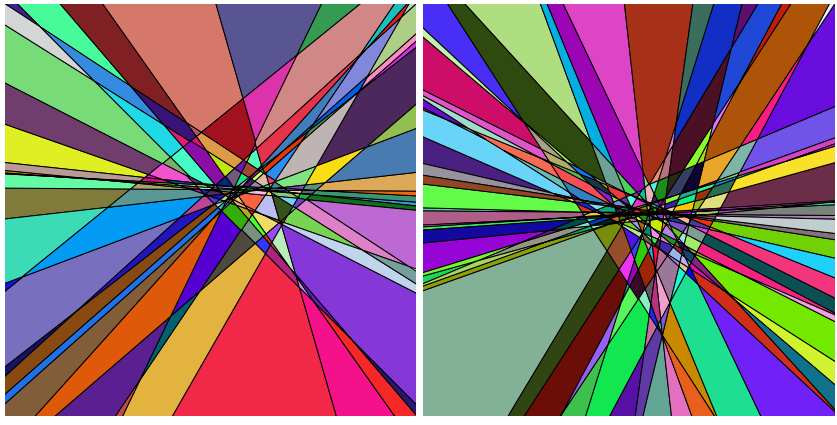
The coefficients $a_i, b_i, B_{ij} \in \mathbf{Z}$ are all chosen randomly. Furthermore, the connectivity is restricted by imposing

$$|B_{ij} \neq 0| \leq 4 \quad \text{for all } (i,j) \in [1, n^2] \times [1, n^2]. \tag{4.29}$$

Solution diagram. In Figure 4.3 the solution diagram for two randomly generated problems is given.

One parameter. By choosing $d_i = 0$ in (4.28) for all i, the problem becomes a standard one-parameter problem with arbitrary signs on the parameter coefficients. Results on random problem instances are given in Figure 4.4.

Two parameters. In this experiment random functions, f_n, of increasing size are solved, in order to study how well the algorithm scales with size. The results are shown in Figure 4.5, indicating that the algorithm does scale very poorly with increased problem size.



(a) f_5, 25 variables and 306 solutions. (b) f_6, 36 variables and 638 solutions.

Figure 4.3: Solution diagrams for two random problem instances of Problem 4.28 with hidden axes. Each region in the solution diagram is given a random color.

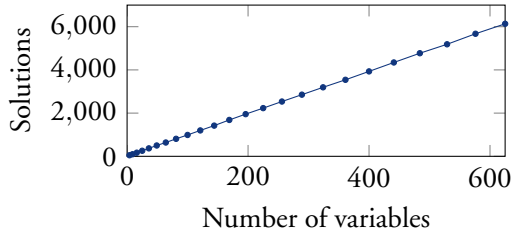


Figure 4.4: Number of solutions averaged over 100 random one-parameter ($d_i = 0$) instances of (4.28).

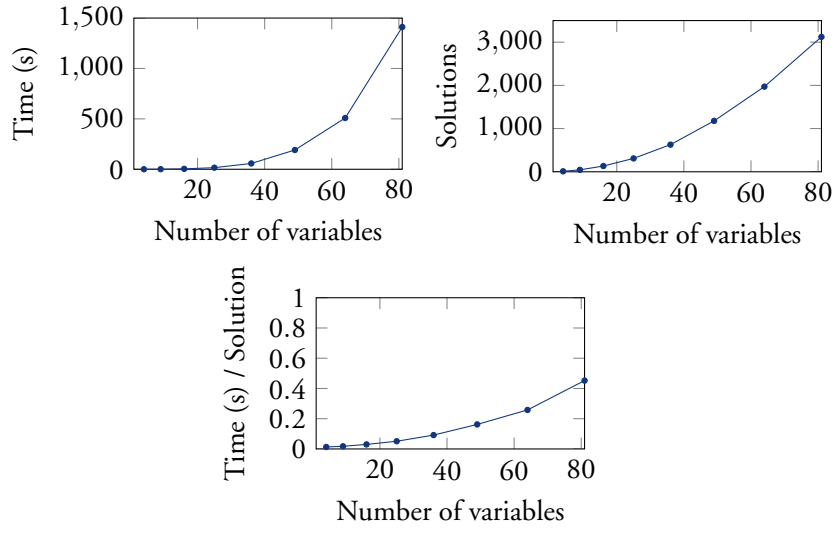


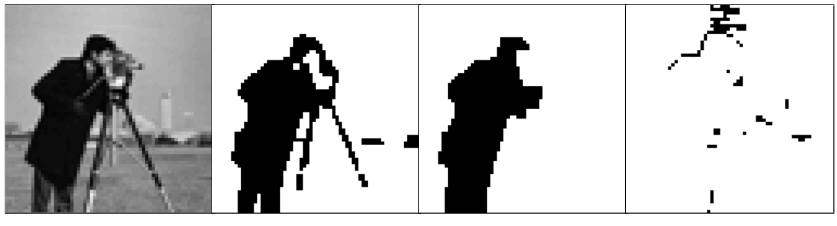
Figure 4.5: Execution time and number of solutions averaged over 100 random two-parameter instances of (4.28).

Precision. In order to emphasize the need for exact arithmetic, a version of the algorithm using floating-point precision is implemented. This version is denoted by DOUBLE and the version using exact arithmetic is denoted as EXACT. Both versions are compared in Table (4.1). As expected, the floating-point precision algorithm fails to find a large number of solutions even for the small problems considered in the experiments.

Chan-Vese. In the last experiment the complete Chan-Vese Problem is solved for a small 64×64 pixel image, see Figure 4.6. Even for the relatively small image, the number of possible parameter choices is huge.

	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9
EXACT	11	45	133	309	628	1178	1971	3122
DOUBLE	11	44	128	289	589	1101	1855	2925
Difference	0	1	5	20	39	77	116	197

Table 4.1: The number of solutions for Problem (4.28). Comparing intersection performed using floating-point precision (DOUBLE) and exact arithmetic (EXACT). As the problem size grows, the number of solutions which DOUBLE fails to find increases.



(a) Image (b) 1 (c) 3000 (d) 12552

Solution	Objective value	λ	μ
1	6035	0.5804	0.1054
3000	7348	0.5690	0.1018
12552	22042	0.7507	0.4578

Figure 4.6: The complete Chan-Vese problem solved for a 64×64 pixel image, execution time 6.1 hours. The total number parameter choices leading to different segmentation results are 12552. The top row shows the image along with the best, the 3000th and the seldom seen worst solution.

4.4 Conclusions

As indicated by the experiments, the algorithm does not scale well enough with the size of the problem to be usable in real-world vision applications. The largest obstacle is in the core of Algorithm 1, where hyperplanes are intersected over and over again. By restricting ourselves to integer weights, we only need to use exact arithmetic of rational numbers. However, the effort required to store and perform computations with these rational numbers grows with the number of intersections, as indicated in Figure 4.5. The natural remedy to this is to resort to floating-point numbers. However, as seen in Table 4.1, even for small problems, the numerical issues lead to many solutions being lost. This highlights the most important points of this chapter:

- If every solution of the parametric max-flow problem is wanted, exact arithmetic is needed.
- For most real-world applications, exact arithmetic is too slow.

Comparing Figure 4.4 to Figure 4.5, we can see that the one-parameter max-flow problem seems to grow linearly with the number of variables. The two-parameter max-flow problem on the other hand, seems to be growing much faster, resulting in a large number of solutions even for smaller problems. This is highlighted in Figure 4.6 where the complete Chan-Vese problem is solved. The solver is almost unusably slow, even for the small 64×64 pixel image used in the example.

If, as in Carreira and Sminchisescu (2010), parametric max-flow only is used to generate a few hypotheses, exact solutions are not needed. Instead, it would be advisable to simply use some approximate algorithm as discussed in Section 4.1.2.

For the one-parameter max-flow problem, specialized solvers exist, see Gallo et al. (1989).

Chapter 5

Curves

This chapter describes a general method for finding thin, elongated structures in images and volumes, regularized with respect to high-order differential properties. These structures can be modeled as curves. In its most simple form, the method reduces itself to performing shortest-path calculation on a graph using Dijkstras algorithm. When using shortest-path on a ordinary graph, where the graph represents the image, the only property of the curve which we are able to penalize is its length. In real-world applications, it is often desirable to penalize more complex properties, such as curvature. Mumford (1994) showed that Euler's elastica – the line integral of the squared curvature – conforms better to intuitive completion of boundary curves than length.

In many applications, the curve should closely follow the shape of an underlying surface which is locally planar. In such situations, it may be advantageous to penalize torsion. Figure 5.1 shows a drawing of a heart. For coronary arteries, it is not ideal to penalize length or curvature, since the coronary arteries are neither short nor do they have low curvature. Here, penalizing torsion would be an ideal prior.

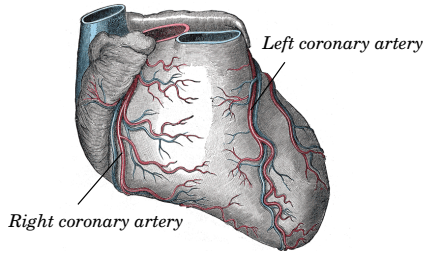


Figure 5.1: A drawing of a heart. The thinner coronary vessels are long with high curvature, but have low torsion as they lie approximately in a plane.

The flexibility of the framework presented in this chapter is highlighted by two extensions. The first one is the geodesic shortest-path problem, which is the problem of finding the shortest path between two points on a surface — the geodesic distance. Gulshan et al. (2010) use the geodesic distance as a regularization prior, Schwarz et al. (2012) use the geodesic distance as a distance measure and Bronstein et al. (2006) use the geodesic distance to match surfaces.

The second extension concerns subsidiary constraints on the minimizing curve, encoded using *resource constraints*. This enforces global constraints on the shortest path. In Irnich and Desaulniers (2005), this formulation is employed for vehicle routing and crew scheduling. The global constraint can be used to limit the maximum length of the curve or to limit the integrated curvature.

5.0.1 Related work

A classical approach for computing the global minimum of an active contour model is based on shortest path, see Cohen and Kimmel (1997); Fischler et al. (1981). This is a flexible and frequently used tool for extracting 1D structures in both 2D and 3D images, especially in medical image analysis, see Lesage et al. (2009). Most of these methods use only weighted length regularization, even though the idea of applying shortest path to higher-order functions involving curvature is not new. One early example of minimizing a curvature-based objective functions can be found in Amini et al. (1990). However, in order to obtain reasonable execution times, only a small band around the initial contour is considered, making the approach resemble a local refinement technique. In Schoenemann et al. (2011), the elastic ratio function is proposed for edge-based 2D image segmentation. The ratio is able to penalize the curvature of the segmentation boundary and finds the global optima on a discrete mesh. Still, on a CPU, the execution times are up to several hours for medium-sized images. One advantage of their approach is that it is possible to implement it in a parallel manner on a GPU, which drastically reduces execution times.

An extension of the live-wire framework with curvature priors is presented in Wang (2005). The algorithm is applied to image squares of up to 80×80 pixels with 8-connectivity. In Péchaud et al. (2009), the standard 2D shortest path is lifted to 4D by incorporating 12 discrete ori-

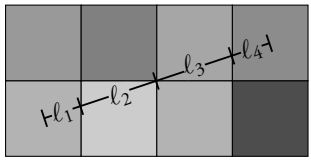
entation angles and radii in the estimation, and hence implicitly penalizing curvature. Many nice examples of the benefits with curvature are given in both Péchaud et al. (2009) and Wang (2005), but no execution times are reported. In contrast, the proposed method is used with significantly larger resolutions of both the image and orientations with tractable execution times. In Krueger et al. (2013), a heuristic method is developed for minimizing a pseudo-elastica for 2D segmentation. The computational complexity of their algorithm is attractive, but at the price of approximate solutions. The proposed algorithm has comparable execution times and the same asymptotic complexity $\mathcal{O}(d^2 n \log nd)$ where n is the number of pixels and d depends on the neighborhood size.

Line graphs have been used before to add "turning costs" to shortest-path problems Caldwell (1961); Winter (2002). This chapter continues this work by showing how to optimize curvature, going beyond relatively small graphs, refining the solution with local optimization, and showing applications to medical problems. In particular, Winter (2002) mentioned increased performance as an open question, an issue that will study in detail in this chapter.

Finding the geodesic shortest path is well studied for analytic surfaces, see Pressley (2010). Several works consider geodesic shortest path on polyhedral surfaces, see Kapoor (1999); Sharir and Schorr (1986). This chapter covers surfaces implicitly given by a depth map. For these surfaces, the distance is usually approximated by the euclidean distance, see e.g. Gulshan et al. (2010); Schwarz et al. (2012). In this chapter, a closed form solution for the distance is given. Furthermore, it is shown that using the correct distance can greatly improve the results at a low computational cost.

The resource constrained shortest-path problem has been optimized using Lagrangian relaxations previously in the literature, see of instance Beasley and Christofides (1989); Mehlhorn and Ziegelmann (2000). In this chapter, it shown how this easily generalizes to function penalizing higher-order properties, such as curvature. There is also previous work on constraining local properties of a curve. In Reeds et al. (1990); Vendittelli et al. (1999) Dubins paths are used for path-planing in robotics as they approximate the mechanical limitations of robots, yielding physically feasible trajectories. The framework proposed in this chapter allows for straightforward implementation of constraints of this type.

In MacCormick and Fitzgibbon (2013) local optimization of a curva-



$$\int_0^\ell D(\gamma(s))ds = \sum_{i=1}^4 \ell_i D(i)$$

Figure 5.2: The data term interpolated using nearest neighborhood for a two-dimensional problem.

ture dependent function is optimized using LBFGS. The derivatives are calculated by hand and the authors noted that it was "tedious to implement and debug." In this chapter, local optimization is also performed using LBFGS, but the derivatives are determined using automatic differentiation. Using automatic differentiation is simpler to implement and can be computationally more efficient than symbolic differentiation, see Bendtsen and Stauning (1996).

5.1 Problem formulation

We are interested in finding a curve γ, in two or three dimensions, which minimizes a data-dependent objective function give on the form

$$\begin{aligned} \underset{\gamma, L}{\text{minimize}} \int_0^L \Big(& f(\gamma(s)) + g(\gamma'(s)) \\ & + h(\gamma''(s)) + \ell(\gamma'''(s)) \Big) ds \\ \text{subject to} \quad & \gamma(0) \in E_{\text{start}}, \gamma(L) \in E_{\text{end}}, \end{aligned} \tag{5.1}$$

where $\gamma : [0, L] \to \mathbf{R}^{2 \text{ or } 3}$ is parametrized by arc length. The functions f, g, h, and ℓ are all restricted to be non-negative. The curve has to start in the set E_{start} and end in the set E_{end}. The main focus of the chapter is

$$f(\gamma(s)) = D(\gamma(s)), \tag{5.2}$$
$$g(\gamma'(s)) = \rho, \tag{5.3}$$
$$h(\gamma''(s)) = \sigma\kappa(s)^2, \tag{5.4}$$
$$\ell(\gamma'''(s)) = \nu\tau(s)^2, \tag{5.5}$$

where D is a data term, κ is the curvature of γ, and τ is the torsion of γ. The scalars ρ, σ and ν are weighting factors for length, curvature and torsion, respectively.

Another problem, expressible as (5.1), is finding geodesics on surfaces defined by some depth map D. For this problem $f = h = \ell = 0$ and

$$\int_0^L g(\gamma'(s))ds, \tag{5.6}$$

is the *geodesic distance* on the surface defined by D.

Furthermore, if we would like to constrain f at some point s to be at most m, we can enforce this replacing it with

$$\tilde{f}(\gamma(s)) = \begin{cases} \infty & f(\gamma(s)) > m \\ f(\gamma(s)) & \text{otherwise.} \end{cases} \tag{5.7}$$

Any other term in (5.1) can be constrained in a similar manner. In Section 5.3 it is shown how the integral in (5.1) also can be constrained.

In this chapter, (5.1) is solved by considering piecewise linear curves γ. In order to approximate k-th derivative of γ we use $k + 1$ points. In the remainder of this section we show how to approximate all costs discussed as a function of either $2, 3$, or 4 points.

5.1.1 Data-dependent term (two points)

The integral of $D(\gamma(s))$ is approximated by nearest neighbor interpolation as

$$\int_0^\ell D(\gamma(s))ds \approx \sum_{i=1}^{n} \ell_i D(i), \tag{5.8}$$

where ℓ_i is the curve segment length for which voxel i is closest to $\gamma(s)$, see Figure 5.2 for an example.

5.1.2 Curvature (three points)

Every line segment has zero curvature, so we will need at least three points to approximate curvature. Let (x_i, y_i, z_i) for $i = \{1, 2, 3\}$ be three ordered

points. The quadratic B-spline associated with these points is

$$\boldsymbol{r}(t) = \frac{1}{2} \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} 1 & -2 & 1 \\ -2 & 2 & 0 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} t^2 \\ t \\ 1 \end{bmatrix}. \tag{5.9}$$

The curvature is defined as

$$\kappa(t) = \frac{\|\boldsymbol{r}'(t) \times \boldsymbol{r}''(t)\|}{\|\boldsymbol{r}'(t)\|^3}, \tag{5.10}$$

see Pressley (2010). It is possible to analytically compute the squared curvature as well as the length element $ds = \|\boldsymbol{r}'(t)\|dt$ of the spline using symbolic mathematics software, see Section 5.A.

There is no closed solution to $\int_0^1 \kappa(t)^2 ds$, but it can be approximated with numerical integration. In this chapter, the integral is approximated using the trapezoidal rule.

5.1.3 Torsion (four points)

Every curve contained in a plane has zero torsion and any given triplet of points always lies in a plane. This means means we need at least four points, (x_i, y_i, z_i), $i = \{1, \ldots, 4\}$, to approximate torsion. Torsion uses derivatives of the third-order, which require cubic B-splines. The cubic B-spline associated with these points is

$$\boldsymbol{r}(t) = \frac{1}{6} \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \\ x_4 & y_4 & z_4 \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}. \tag{5.11}$$

The torsion is given by

$$\tau(t) = \det \begin{bmatrix} \boldsymbol{r}'(t) & \boldsymbol{r}''(t) & \boldsymbol{r}'''(t) \end{bmatrix} / \|\boldsymbol{r}'(t) \times \boldsymbol{r}''(t)\|^2, \tag{5.12}$$

see Pressley (2010). Similarly to curvature, the torsion can be calculated analytically, see Section 5.B.

Figure 5.3 shows an example curve (a helix), for which the exact curvature and torsion are well known: they are both constant. Figure 5.4 shows the error when approximating the curvature and torsion using three and four points, respectively.

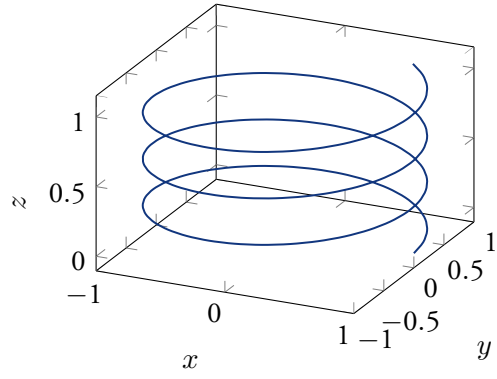


Figure 5.3: Example curve $(x, y, z) = (\cos 6\pi t, \sin 6\pi t, t)$.

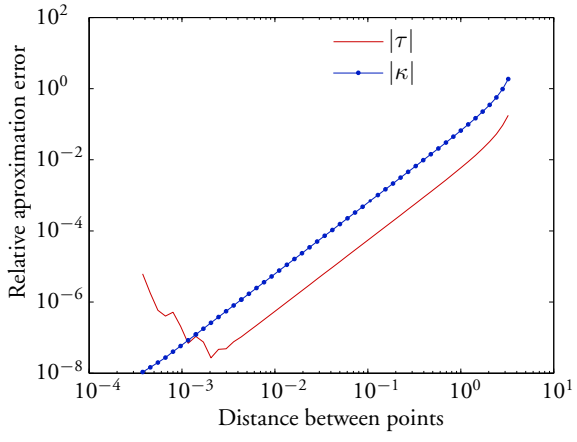


Figure 5.4: Error when computing the curvature (κ) and torsion (τ) using the middle point of a spline fit to three or four points on the curve in Figure 5.3.

5.1.4 Distance on a surface (two points)

We restrict our attention to the case where surfaces are defined implicitly by the depth in a number of grid points, as defined by a depth map D. The curve $\gamma(s)$ on the surface D is defined as

$$\gamma(s) = \big(x(s), y(s), D(x(s), y(s))\big), \tag{5.13}$$

where $r(s) = (x(s), y(s))$ is a curve in $\mathbf{R}^2$. This approach was used in Gulshan et al. (2010), where geodesic distance was defined by the euclidean distance between between points on γ. This corresponds to interpolating D using *linear interpolation*. This gives rise two related issues:

1. How do we define distance between two points whose depth is not given in the depth map.

2. What kind of surface are we implicitly defining using this interpolation.

Consider any patch of four points. Four points does not in general lie on a surface, so simply using euclidean distance leads to contradictions. The underlying surface may be different depending on which pair of points are compared. The implicitly defined surface is *inconsistent*.

Nearest-neighborhood interpolation gives consistent estimation of the surface. Yet at each point the curve changes nearest neighbor, the surface may be discontinuous.

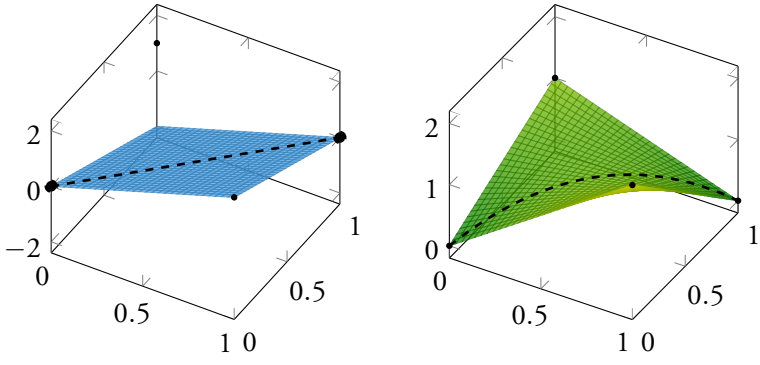
Bilinear interpolation is the simplest interpolation scheme giving a continuous surfaces. Approximating the surface depth, d, by bilinear interpolation yields an expression which can be simplified as

$$d(x, y) = d_{11}xy + d_{10}x + d_{01}y + d_{00} \tag{5.14}$$

where $(x, y) \in [0, 1] \times [0, 1]$. The coefficients are given by

$$d_{11} = D(0, 0) + D(1, 1) - D(1, 0) - D(0, 1), \tag{5.15}$$
$$d_{10} = D(1, 0) - D(0, 0), \tag{5.16}$$
$$d_{01} = D(0, 1) - D(0, 0), \tag{5.17}$$

where $D(i, j)$ are the depths defined by discrete depth map in the four corners of the patch. Figure 5.5 shows the difference between the linear and the bilinear interpolation for a patch.



(a) Linear interpolation, $d(x, y) = 2(x - y)$, line length $\sqrt{2} \approx 1.4142$.

(b) Bilinear interpolation, $d(x, y) = -3xy + 2x + y$, line length ≈ 2.1572.

Figure 5.5: The difference between linear and bilinear interpolation, when measuring the length of curve lying on a surface. The depth map is only defined in the four corner points shown in both images. For linear interpolation we show a possible plane which is consistent with three of the four points.

A curve from (x_0, y_0) to (x_1, y_1) can be parametrized as

$$\begin{aligned} r(t) &= (x(t), y(t)) && (5.18)\\ &= (t(x_1 - x_0) + x_0, t(y_1 - y_0) + y_0), && (5.19)\end{aligned}$$

with $t \in [0, 1]$. It turns out that even though we are using bilinear interpolation we still get a closed form solution for the arc length:

$$\begin{aligned} &\int_0^1 \sqrt{x'(t)^2 + y'(t)^2 + d'(t)^2}dt = \\ &\begin{cases} L & \text{if } d_{11} = 0 \text{ or } \Delta x = 0 \text{ or } \Delta y = 0, \\ B & \text{otherwise,} \end{cases} \end{aligned} \tag{5.20}$$

where

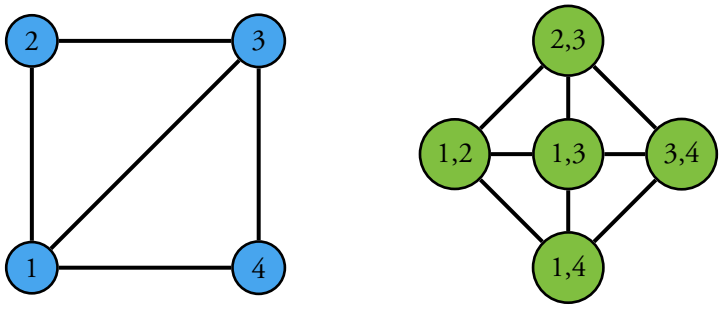
$$\begin{aligned} L &= \sqrt{\Delta x^2 + \Delta y^2 + (\Delta x d_{10} + \Delta y d_{01})^2}, \\ B &= \frac{|a|}{2}\Big(c \log \left|\frac{b + 1 + \sqrt{g}}{b + \sqrt{f}}\right| + (b + 1)\sqrt{g} - b\sqrt{f}\Big), \end{aligned} \tag{5.21}$$

$$\begin{aligned} a &= 2d_{11}\Delta x \Delta y, & f &= b^2 + c, \\ b &= \frac{y_0 \Delta x + x_0 \Delta y}{2\Delta x \Delta y} + \frac{d_{01}\Delta y + d_{10}\Delta x}{a}, & g &= 1 + 2b + f, \\ c &= \frac{\Delta x^2 + \Delta y^2}{a^2}, & \Delta x &= x_1 - x_0, \\ & & \Delta y &= y_1 - y_0. \end{aligned} \tag{5.22}$$

Note that if $\Delta x, \Delta y$ or d_{11} equals 0, then the bilinear interpolation along $r(t)$ is linear. If the curve passes through several patches the line is cut into pieces and the length is measured separately in each patch.

5.2 Shortest paths in line graphs

In this section we will show how optimize (5.1) by restricting γ to lie in a predefined mesh represented by a graph, G. We will assume that the reader is familiar with basic graph theory, see e.g Papadimitriou and Steiglitz



(a) Graph (G)

(b) $L(G)$ of the graph in (a)

Figure 5.6: Example construction of a line graph.

(1998). Given a data volume, D, each voxel is represented by a vertex in G. Edges connecting the vertices in G represent possible line-segments for γ, see Figure 5.7. Given a start and end sets we can find then find shortest path very efficiently using Dijkstra's algorithm, see Dijkstra (1959).

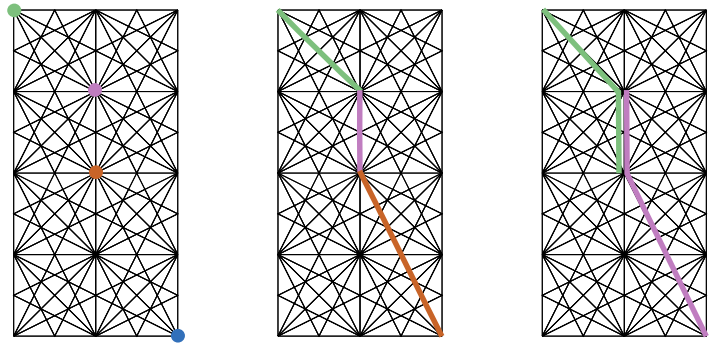
Definition 5.1. *We refer to the number edges pointing to a vertex as the* connectivity *of that vertex. If all vertices have the same connectivity we will refer to that as the* connectivity *of the problem instance.*

For example, adding all edges for a 2D problem with distance less than or equal to 2.5 gives a connectivity of 16. This construction connects two vertices and makes it possible to penalize functions of at most two points. In order to penalize higher-order regularization like curvature and torsion another graph construction is needed.

Definition 5.2. *For a graph G, the* line graph $L(G)$ *is given by a graph where each edge in G is a vertex in $L(G)$ and each pair of edges connected by a vertex in G is a edge in $L(G)$. See Figure 5.6 for an example construction.*

Functions on form (5.1) **with** $h(\gamma) = \ell(\gamma) = 0$

The objective functions is a function of *two points* and the original graph suffices, see Figure 5.7(a).



(a) Using vertices (b) Using edges (c) Using edge pairs

Figure 5.7: The same path through a graph, edges shown in black, represented in three different ways. The graph vertices, shown in color, correspond to (a) vertices, (b) edges, and (c) edge pairs in the graph.

Functions on form (5.1) **with** $\ell(\gamma) = 0$

Function $h(\gamma)$ requires at least three points to be calculated. Form $L(G)$ of the original graph. The start set of $L(G)$ consists of all vertices in $L(G)$ whose corresponding edge in G connects to a vertex in E_{start}. In the same way the end set of $L(G)$ is constructed. By construction each edge in $L(G)$ connects three vertices in the original graph G, see Figure 5.7(b).

Any function on form (5.1)

Function $\ell(\gamma)$ requires at least four points to be calculated. We repeat the graph construction for the previous case and form $L(L(G))$. Now each edge connects four vertices in the original graph G, see Figure 5.7(c).

Going beyond (5.1)**, penalizing functions of** k **points**

It is possible to penalize any function of k points by nesting the line graph construction $k-2$ times. The main issue with penalizing even higher-order functions is that the size of the nested line graph grows very rapidly. Suppose we would like to penalize a function of k points and thereby for instance approximating the $k-1$ derivative of γ. Given a problem with a total of n points and a connectivity of c. The corresponding line graph would have $\mathcal{O}(nc^{k-2})$ vertices and $\mathcal{O}(nc^{k-1})$ edges.

5.3 The resource constrained shortest path

It is possible to extend (5.1) by introducing hard constraints such as maximum length, curvature or torsion of the curve. This problem is known as *resource constrained shortest path*. In this section we solve this problem using a Lagrangian relaxation, this was first done by Beasley and Christofides (1989).

Consider the primal problem

$$\begin{aligned}
\underset{\gamma, L}{\text{minimize}} \quad & \int_0^L \Big(f(\gamma(s)) + g(\gamma'(s)) \\
& \qquad + h(\gamma''(s)) + \ell(\gamma'''(s)) \Big) ds \\
\text{subject to} \quad & \gamma(0) \in E_{\text{start}}, \gamma(L) \in E_{\text{end}} \\
& \int_0^L g\left(\gamma'(s)\right) ds \leq u_1 \\
& \int_0^L h\left(\gamma''(s)\right) ds \leq u_2 \\
& \int_0^L \ell\left(\gamma'''(s)\right) ds \leq u_3.
\end{aligned} \tag{5.23}$$

Define the dual function as

$$\begin{aligned}
& d(\lambda_1, \lambda_2, \lambda_3) = \\
\underset{\gamma, L}{\text{minimize}} & \int_0^L \Big(f(\gamma(s)) + g(\gamma'(s)) \\
& + h(\gamma''(s)) + \ell(\gamma'''(s)) \Big) ds \\
& + \lambda_1 \left(\int_0^L g(\gamma'(s)) ds - u_1 \right) \\
& + \lambda_2 \left(\int_0^L h(\gamma''(s)) ds - u_2 \right) \\
& + \lambda_3 \left(\int_0^L \ell(\gamma'''(s)) ds - u_3 \right) \\
\text{subject to } & \gamma(0) \in E_{\text{start}}, \gamma(L) \in E_{\text{end}},
\end{aligned} \tag{5.24}$$

and the dual problem as

$$\begin{aligned} &\underset{\lambda_1,\lambda_2,\lambda_3}{\text{maximize}} \quad d(\lambda_1,\lambda_2,\lambda_3) \\ &\text{subject to } \lambda_i \geq 0 \quad \text{for all } i \in \{1,2,3\}. \end{aligned} \tag{5.25}$$

For a feasible solution to the dual problem it follows that,

$$\begin{aligned} d(\lambda_1,\lambda_2,\lambda_3) \leq \int_0^L \Big(& f(\gamma(s)) + g(\gamma') \\ & + h(\gamma''(s)) + \ell(\gamma'''(s)) \Big) ds. \end{aligned} \tag{5.26}$$

This result is known as *weak duality*. Using this result we can instead of minimizing the primal problem, maximize the dual problem.

The dual problem is maximized using the cutting-plane method. Note that d can efficiently be evaluated, by shortest path computations, by simply modifying the functions f, g, and h in (5.1).

5.4 Local optimization

So far we only discussed finding an optimal solution on a discrete mesh. Ideally we would like a larger search space. This can be achieved by first finding the optimal solution on the discrete mesh and then performing local optimization, for example using gradient descent, on the discrete solution. The major concern with most local optimization methods is that we need to find the derivatives of the function. Approximating them via finite differences can lead to issues with numerical stability. Deriving them analytically by hand can be very cumbersome and is prone to human error. A better approach is to derive the derivatives automatically via dual numbers.

Dual numbers are an extension to the real numbers much like the imaginary numbers. Given a real number a adjoin a new element ε and define a dual numbers as $z = a + \varepsilon b$, with the additional property that $\varepsilon^2 = 0$.

Now consider a function f and its Taylor expansion around a,

$$f(a+\varepsilon) = \sum_{k=0}^{\infty} \frac{\varepsilon^k f^{(k)}(a)}{k!} \qquad = f(a) + \varepsilon f'(a). \tag{5.27}$$

Using the dual number we can recover the derivative of by simply reading the coefficients in front of ε.

It is possible to efficiently implement Automatic Differentiation (AD) using dual numbers in a programming language supporting templates, see Bendtsen and Stauning (1996), such as C++. The compiler is able to generate and optimize multiple overloads of the same function; one for scalar floating point evaluation and another for dual numbers, which gives the derivative.

In this chapter, automatic differentiation is used to perform LBFGS using the software by Strandmark (2013). This allows us to locally optimize the curve without calculating any derivatives symbolically or numerically. Since the problem is relatively low-dimensional, local optimization is typically much faster than finding the discrete solution using Dijkstra.

5.5 Implementation

Dijkstra's algorithm for computing the shortest path is well known. An important property is the fact that it does not require the graph to be stored. Two things are required:

1. The number of vertices along with a start and end sets.

2. An oracle that given a vertex returns its neighbors and edge weights.

Using an oracle means that the weight does not have to be precomputed for every edge in the graph. As Section 5.1 shows, computing the edge weights can become quite involved.

Creating the line graphs explicitly in memory is not needed; they can be derived on the fly from the original graph. This observation is crucial for higher-order regularization as the number of edges in nested line graphs explodes. Furthermore, assuming that each vertex have the same type of neighborhood system, then the graph itself does not have to be stored. This is assumption is true for all the experiments performed in this chapter.

The same implementation of Dijkstra's algorithm is used for all experiments in this chapter. It uses the C++ standard library for all data structures. The limiting factor memory-wise is the priority-queue; each entry requires 12 bytes and consists of a `int` index and `double` weight.

A cache is used in order to avoid performing the same computations twice. For example, the curvature cost between two edges is the same when both edges are translated by the same amount. Thus, the integrated curvature needs only be computed once for each configuration of edge pairs. The cache has to be fast, since when working with torsion there are millions of different configurations of two edge pairs. For general connectivity the implementation uses an `std::map` with the coordinates (mean subtracted) as keys. For a fixed connectivity all costs are precomputed and stored in a `std::vector` for more efficient lookup.

5.5.1 The A*-algorithm

A common improvement to Dijkstra's algorithm is A*, which changes the vertex visitation order. It requires a function l, which for each vertex returns a lower bound of the distance to the end set. Any such l yields a correct algorithm. If $l \equiv 0$, then A* becomes the standard Dijkstra algorithm.

In some cases, finding a good lower bound is easy. For example, when computing the path with shortest length, the distance "as the crow flies", ignoring D, can be used as a lower bound, ignoring any obstacles in the graph. For higher-order regularization, however, computing a useful lower bound is more difficult (it has to be done really fast to make a difference). One option is to set $h(\gamma) = \ell(\gamma) = 0$ and solve (5.1) for all vertices (this is fast), and use the result as a lower bound. This lower bound will be evaluated in experimental section of this chapter.

5.5.2 Parallelization

Computing the shortest path between two vertices efficiently in parallel is not a trivial task. Therefore Dijkstra's is sequentially run and the neighbors of each vertex are sequentially computed. The oracle can compute the edge weights for all neighbors in parallel on a multi-core CPU. Since these computations are a large portion of the total computational cost (even with the above-mentioned cache), some parallelization is obtained.

5.6 Experiments

This section gives many examples showcasing the strength of higher-order regularization; both on synthetic and real data.

5.6.1 Synthetic Experiments

River segmentation

This experiment highlights the difference between length and curvature regularization. The river image in Figure 5.8 is manually sampled at different locations. The samples are used to construct the data term for every pixel as the shortest (Euclidean) distance in L*a*b* space to any of the color samples of the river delta. E_{start} is the upper boundary and E_{end} is the lower boundary of the image.

Figure 5.8 presents the results. With curvature regularization it is possible to find a long path which does not turn much. No amount of length regularization is able to find this path in Figure 5.8. Local optimization is able to improve the shortest-path solution and is most effective when curvature regularization is used. In Figure 5.9 the improvements of local optimization are highlighted, note that the solution given after local optimization is much smoother, resulting in much lower curvature cost.

The execution times are heavily dependent on the regularization as shown in Figure 5.10. For length regularization, the execution time increases with the regularization strength. In the extreme case, $\rho = \infty$, all lines up to the Euclidean shortest path need to be considered before the algorithm terminates. The execution times for curvature penalty behave differently. For low regularization, a strong data term together with the A* lower bound will leave most of the graph unvisited when the shortest path is found. For $\sigma = \infty$, only the straight lines are considered, which also leaves most of the graph unvisited. The longest execution time occurs with medium strong regularization, as most of the graph needs to be visited. The execution times are compared in Figure 5.11 for Dijkstra and A*, and it is evident that large speed-ups are obtained with A*.

Torsion

In Figure 5.12 an experiment indicating the difference between curvature and torsion regularization is shown. High torsion regularization forces the curve to stay within a plane, but within the plane, the curve may have high length and curvature. The graph used for torsion regularization had about 280 billion edges. The technique of not storing the line graph explicitly is of course imperative for such a graph.

Geodesic shortest path

The last synthetic experiment covers geodesic shortest paths — the shortest path between two points on a surface. Define a surface as the function graph of

$$f(x, y) = 10 \left|\cos(x) \sin(y)\right|. \tag{5.28}$$

Sampling f on a grid results in a discrete version of the function graph. This discrete function graph will be used as a surface in this experiment. The set E_{start} is chosen as the upper left corner of the grid and E_{end} is chosen as the lower right corner of the grid. The geodesic shortest-path problem is solved using both linear and bilinear interpolation on the discretized function graph in Figure 5.13. This gives two curves for which the lengths are determined by numerical integration of the analytical function graph of f. For low resolutions, bilinear interpolation gives much better results. As the sample rate goes up, the linear and bilinear interpolations give basically identical solutions.

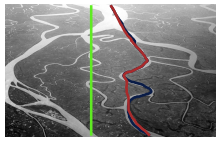
In Figure 5.14, the shortest-path problem is solved on the surface defined by the function graph of

$$g(x, y) = 10\sqrt{x^2 + y^2}. \tag{5.29}$$

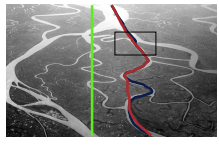
The shortest-path solution is compared to a refined solution obtained after local optimization. This experiment highlights the benefits of local optimization which yields a much smoother curve, which fits the model significantly better.

Image

Data term

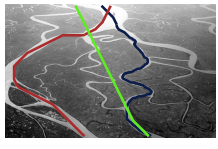


Only length reg.

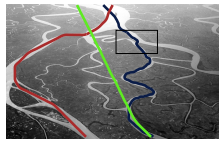


After local optimization

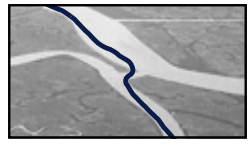
Close-up



Only curvature reg.



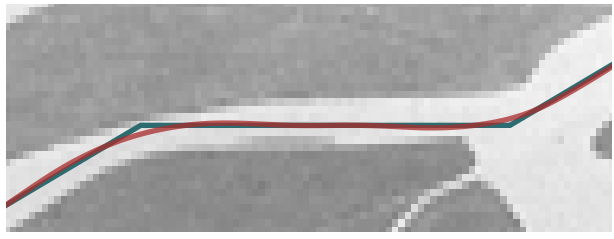
After local optimization



Close-up

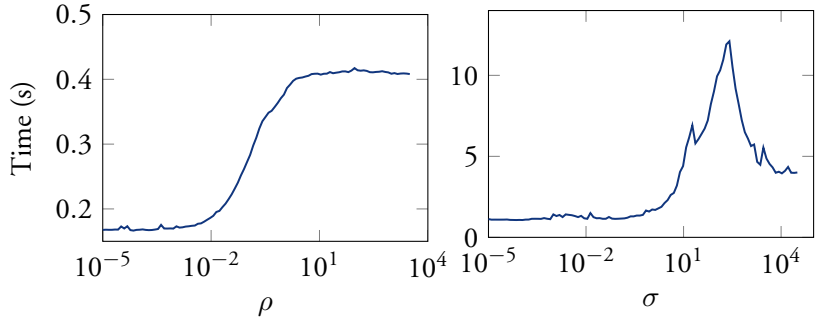
	■	■	■	■	■	■
ρ	0.25	1	1000	0	0	0
σ	0	0	0	0.25	50	500
Time (s)						
A*	0.37	0.42	0.47	0.82	6.55	8.00
Local optimization	0.12	0.03	0.01	18.16	9.01	14.4
Local optimization decreased objective value by						
	1.5%	1.1%	0%	14%	48%	13%

Figure 5.8: Segmentation highlighting the difference between length and curvature regularization. The start set E_{start} is the upper boundary of the image and the end set E_{end} is the lower boundary. The images on the second and third row use only length and curvature regularization respectively. No amount of length regularization is able to find the long and smooth river path. The underlying graph has 220,443 vertices (373×591) and 3,509,756 edges (16-connectivity).



	Data term	Curvature term
Shortest path	68.71	65.66
+Local optimization	64.21	5.61

Figure 5.9: The effect of refining the shortest-path solution using local optimization. The image shows a close-up of the shortest-path problem solved in Figure 5.8, with regularization settings: $\rho = 0$ and $\sigma = 50$.

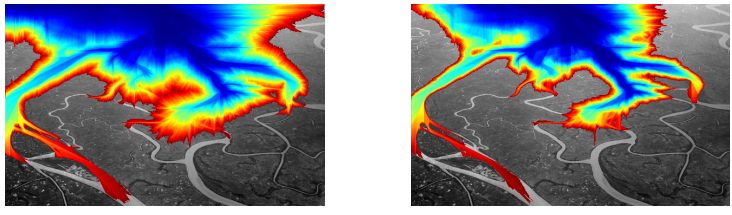


(a) Only length regularization

(b) Only curvature regularization

Figure 5.10: Execution time as a function of regularization strength for the example in Figure 5.8. For purely length regularization the execution time is increasing with the regularization strength. For purely curvature regularization the longest execution times occur for medium strong regularization.

σ	Dijsktra		A*	
	Vertices visited	Time (s)	Vertices visited	Time (s)
0.25	886,966	3.45	196,553	0.86
50	1,328,045	9.70	885,415	6.67
500	916,061	9.14	740,683	7.54



(a) Dijkstra, $\sigma = 50$ (b) A*, $\sigma = 50$

Figure 5.11: The number of vertices visited using Dijkstra's and A* for the curvature experiments in Figure 5.8. The images show the order in which the vertices were visited for medium curvature with and without A*, from blue (early) to red (late). Naturally, the heuristic works best for low curvature regularization.

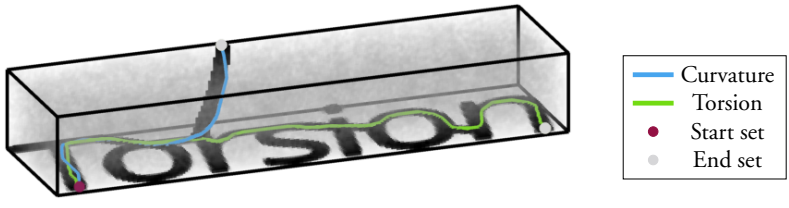
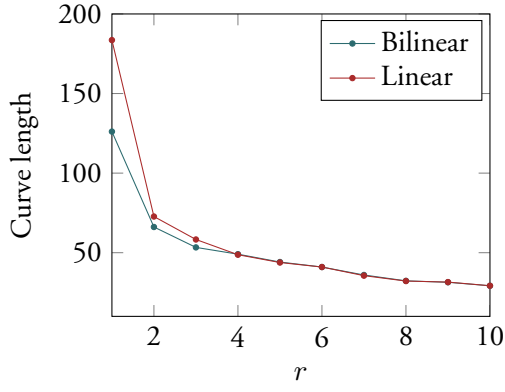
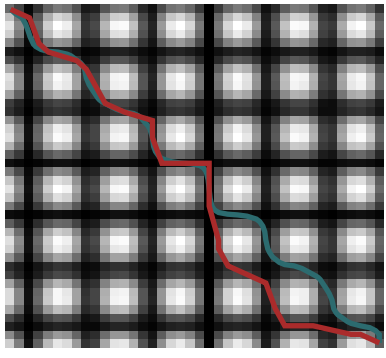


Figure 5.12: Synthetic 3D experiment for torsion with volume-rendered data term. Darker regions correspond to lower cost. The segmentation is performed with either very high curvature or very high torsion regularization. The underlying graph has 91,000 vertices ($35 \times 130 \times 20$), 13,286,000 edges (146-connectivity) and 1,939,756,000 edge pairs. Run times are: 0.1670 seconds for curvature and 6.8991 seconds for torsion. More difficult problems will take substantially longer time to solve.

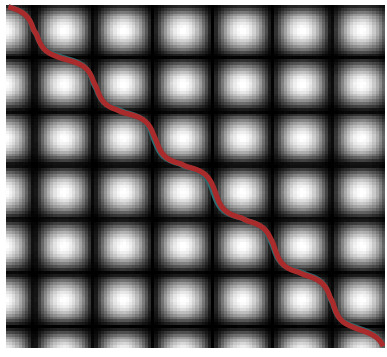


(a) f is sampled on a grid with resolution $20r \times 20r$.



	Length	Time (s)
Linear	72.7285	0.3530
Bilinear	66.1618	0.2183

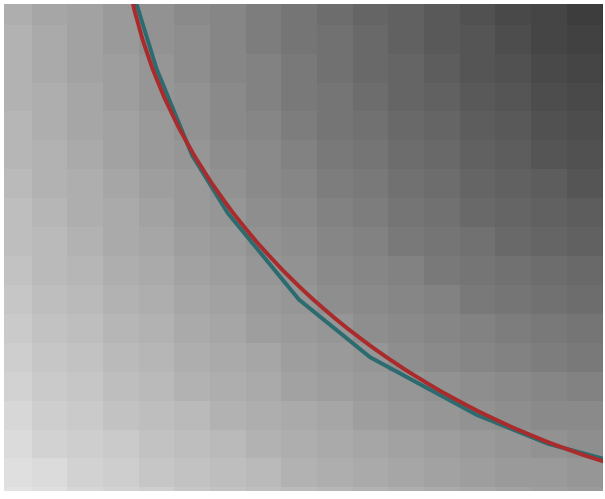
(b) f is sampled on a 40×40 grid ($r = 2$). The underlying graph has 1,600 vertices (40×40) and 76,800 edges (48-connectivity).



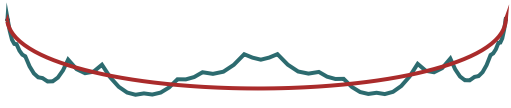
	Length	Time (s)
Linear	40.9942	0.9393
Bilinear	40.9931	1.0214

(c) f is sampled on a 120×120 grid ($r = 6$). The underlying graph has 14,400 vertices (120×120) and 691,200 edges (48-connectivity).

Figure 5.13: The geodesic shortest path from the top left corner to the bottom right corner, on a discretized function graph of $f(x, y) = 10|\sin x \cos y|$. For lower sample rates, r, the curves determined via bilinear interpolation are significantly better than the curves determined via linear interpolation.



(a) Close-up of a corner of the resulting curves.



(b) The curves on the surface (the surface is not shown).

	Length	Time (s)
Shortest path	76.6101	0.0514
+Local optimization	66.4362	0.2454
Half circle	66.6332	

Figure 5.14: The geodesic shortest path on the function graph of $10\sqrt{x^2+y^2}$. The start and end sets are chosen as two opposing points with distance 21.2132 from origin. The underlying graph has 25,000 vertices (50×50) and 120,000 edges (48-connectivity). Local optimization is performed on the shortest-path solution. The surface is convex, hence the local optimizer shines and produces a solution with a much shorter length.

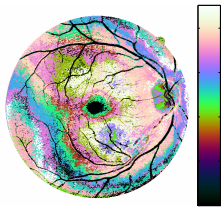
5.6.2 2D Retinal images

Automated segmentation of vessels in retinal images is an important problem in medical image analysis, which, for example, can be used when screening for diabetic retinopathy. This section investigates whether curvature regularization is useful for this task on the publicly available dataset given in Staal et al. (2004). Figure 5.15 shows the experiment. Shortest paths are iteratively computed from a user-provided start point to any point on the image boundary. A curve could start anywhere along the previously found curves, but not end close to a previous end point.

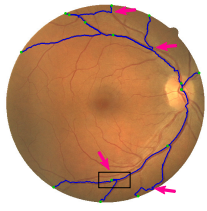
Experiments are performed with different amounts of length regularization in Figure 5.15. No or low length regularization resulted in noisy, wiggling paths. This problem expectedly disappeared for medium regularization, but sharp turns were still present in the solution (sharp turns usually change direction in the vessel tree). When using too high regularization, Figure 5.15(e), the solution almost ignores the data term and prefers paths along the image boundary. In contrast, curvature regularization is able to more correctly capture the vessel tree.

5.6.3 Multi-view reconstruction of space curves

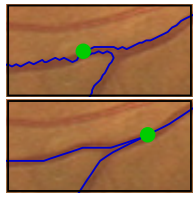
Curvature has previously been used to reconstruct space curves from multiple calibrated views of a static scene, see Kahl and August (2003), but only using local optimization. Given start and end points, the curves can now be optimally reconstructed. Figure 5.16 shows an experiment on the same data as Kahl and August (2003). The tree is reconstructed iteratively in the same way as in Figure 5.15 and, as expected, length regularization introduces similar artifacts. Integrating the image along the projected 3D edge gives the edge cost for a single view. The data term used for an edge is simply the maximum of the cost over all views.



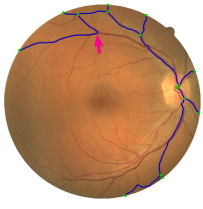
(a) Data term



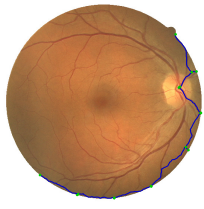
(b) $\rho = \sigma = 0$.
Mean time: 0.41s



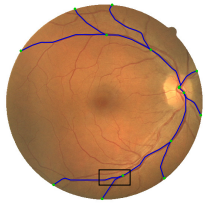
(c) Details of (b) and (f), respectively



(d) $\rho = 0.1, \sigma = 0$.
Mean time: 0.56s

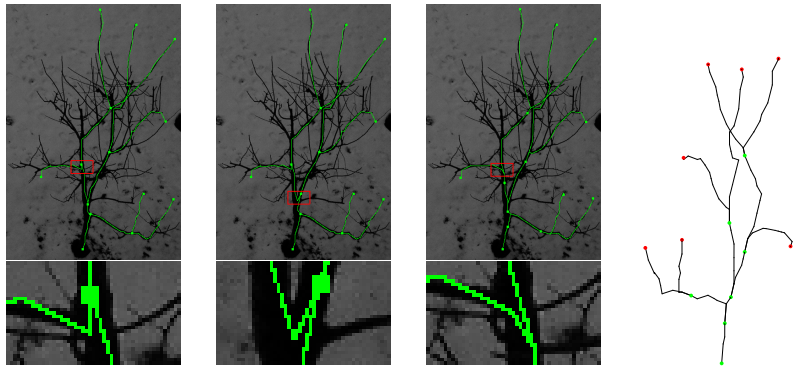


(e) $\rho = 1, \sigma = 0$.
Mean time: 0.66s



(f) $\rho = 0, \sigma = 100$.
Mean time: 10.1s

Figure 5.15: Segmenting a vessel tree with 8 branches in a 2D retinal image from the Staal et al. (2004) data set. The green dots show the computed best starting points for the new branches. The underlying graph has 329,960 vertices (584×565) and 10,496,766 edges (32-connectivity). The arrows point at sharp turns where the segmented vessel changes direction in the vessel tree and at particularly noisy parts. All length regularizations (b-d) have various issues and curvature regularization (e) finds a reasonable tree.



$\rho = 10, \sigma = 0$ $\rho = 100, \sigma = 0$ $\rho = 0, \sigma = 10$ The tree

Figure 5.16: Reconstructing a tree in 3D from 4 different views, only one view is shown. (a)-(b) Using only length regularization gives similar artifacts (sharp turns) as in Figure 5.15. (c) Replacing the length regularization with curvature regularization improves the resulting reconstruction. The underlying graph has 125,000 vertices and 24,899,888 edges (218-connectivity). Run times are 4.8 minutes for length and 13.8 hours for curvature. The data comes from Kahl and August (2003).

5.6.4 3D coronary artery centerline extraction

Finding the centerlines of coronary arteries is of high clinical importance, see Schaap et al. (2009). This task is however very time consuming when performed manually. This section uses the public data set "Rotterdam Coronary Artery Algorithm Evaluation Framework" from Schaap et al. (2009). The data set consisting of 32 CT angiography scans. Frequency analysis of the CT volumes in Schaap et al. (2009) show that the resolution is artificially high in the dataset, see Friman et al. (2008). In the experiments the volumes are downsampled to half the original size in each dimension without any loss of information.

Most of the methods which have submitted to the Rotterdam challenge use some sort of shortest path formulation in some part of their algorithm. Reported results vary between 70–98% overlap, with different amounts of human interaction and model complexity.

For the experiments the data term proposed by Metz et al. (2008) is used. The data term is based on Frangis' *vesselness* measure, see Frangi et al. (1998). The vesselness measure is constructed by analyzing the eigenvalues of Hessian matrix in each voxel of the volume. As is usual for these measures the volume is first smoothed with a Gaussian kernel, with standard deviation σ, allowing the Hessian calculation and Gaussian smoothing to be combined into one filter. The eigenvalues of the Hessian at voxel p are sorted as,

$$|\lambda_1(p)| \leq |\lambda_2(p)| \leq |\lambda_3(p)|. \tag{5.30}$$

The vesselness is then defined as

$$\mathcal{V}(p) = \begin{cases} 0 & \lambda_2(p) > 0 \text{ or } \lambda_3(p) > 0 \\ (1 - \exp(A)) \exp(B) \left(1 - \exp(C)\right) & \text{otherwise,} \end{cases} \tag{5.31}$$

where

$$A = -\frac{1}{2\alpha^2} \left(\frac{\lambda_2(p)}{\lambda_3(p)} \right)^2 \tag{5.32}$$

$$B = -\frac{1}{2\beta^2} \frac{\lambda_1(p)^2}{|\lambda_2(p)\lambda_3(p)|} \tag{5.33}$$

$$C = -\frac{1}{2c^2} \left(\lambda_1(p)^2 + \lambda_2(p)^2 + \lambda_3(p)^2 \right). \tag{5.34}$$

The motivations for the different functions are as follows.

A: distinguishes between plate-like and line-like structures.

B: accounts for deviations from a blob-like structure.

C: captures that the norm of the eigenvalues should be small in the background.

We are now ready to define the data term used by Metz et al. First the voxel volume undergoes a soft thresholding

$$T(p) = \frac{1}{2}\Big(\operatorname{erf}\left(b\left(I(p) - a_1\right) + 1\right)\Big)\Big(1 - \frac{1}{2}\left(\operatorname{erf}\left(b\left(I(p) - a_2\right)\right) + 1\right)\Big). \tag{5.35}$$

The voxel intensity, $I(p)$, is measured in Hounsfield and the error function is defined as

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x \exp(-t^2) dt. \tag{5.36}$$

The parameters are tuned to avoid bronchi and calcium intensities. The data term for voxel p is defined as

$$D(p) = \frac{1}{\mathcal{V}(p)T(p) + \varepsilon} \tag{5.37}$$

All parameters are chosen as in Metz et al. (2008),

$$\begin{aligned} &\alpha = \frac{1}{2}, \qquad \beta = \frac{1}{2}, \qquad c = 230, \quad \sigma = 0.92\text{mm}, \\ &a_1 = -24, \quad a_2 = 576, \quad b = 0.1, \quad \varepsilon = 10^{-4}. \end{aligned} \tag{5.38}$$

The start and end set needs to be manually chosen and is extracted from the ground truth data. In all experiments the connectivity is 96. Each coronary artery volume has a resolution of about $200 \times 200 \times 150$ voxels.

Two different regularization schemes are compared:

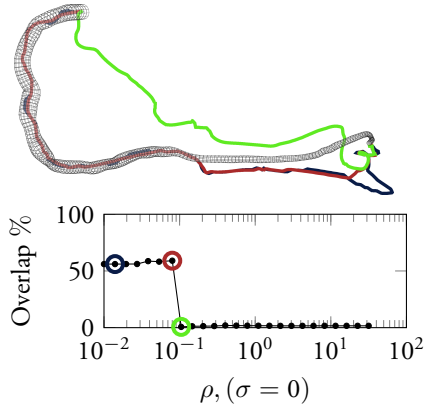
- LENGTH: vary ρ and keep $\sigma = 0$,
- CURVATURE: vary σ and keep $\rho = 0$.

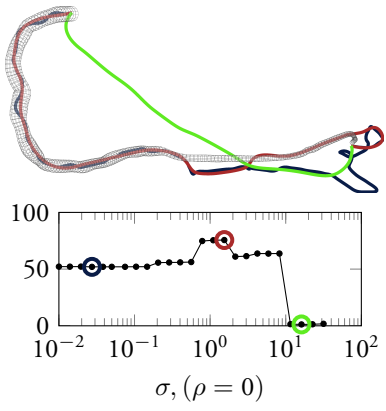
	Overlap %	Regularization		Wins	Time (s)
	Mean ±std	ρ	σ		Mean ±std
LENGTH					
Shortest path	91.90 ±15.10	0.08	0.00	12	15.62 ±8.04
+Local opt.	91.94 ±15.04	0.08	0.00	11	16.59 ±8.16
CURVATURE					
Shortest path	87.51 ±22.09	0.00	0.05	10	81.91 ±10.62
+Local opt.	87.80 ±22.14	0.00	0.05	11	89.35 ±11.76

Table 5.1: Quantitative results for coronary artery centerline extraction on the training set of Schaap et al. (2009). The execution times include the construction of the data the data term.

	Data term $\int_0^L D(\gamma(s))ds$	Length $\int_0^L 1ds$	Curvature $\int_0^L \kappa(s)^2 ds$
	Mean ±std	Mean ±std	Mean ±std
Ground truth	8.05 ±10.38	130.08 ±47.67	2.59 ±1.46
LENGTH			
Shortest path	3.29 ±2.91	131.96 ±48.96	18.55 ±9.92
+Local opt.	2.87 ±2.33	128.57 ±47.84	359.58 ±422.61
CURVATURE			
Shortest path	3.23 ±2.78	140.90 ±61.96	9.50 ±5.51
+Local opt.	2.89 ±2.47	137.89 ±60.72	3.76 ±3.54

Table 5.2: Curve characteristics for all coronary arteries in the training set of Schaap et al. (2009).





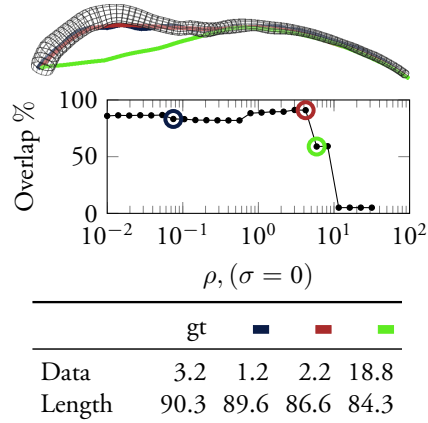
	gt			
Data	47.8	7.8	9.1	12.6
Length	220.1	271.2	238.7	184.7
Curvature	2.9	15.8	8.8	7.8

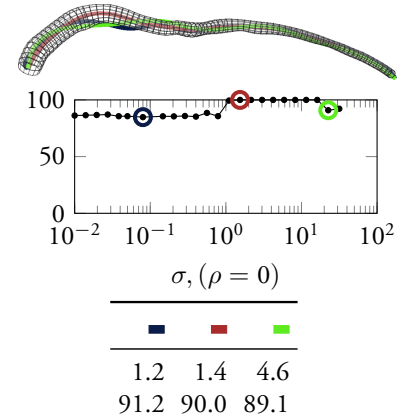
(a) Varying length regularization

8.1	10.8	42.9
302.2	243.3	138.2
16.6	2.8	0.4

(b) Varying curvature regularization

Figure 5.17: Biggest advantage for LENGTH, the curve characteristics are defined as in Table 5.2. The ground truth (gt) is shown as a black wireframe.





	gt			
Data	3.2	1.2	2.2	18.8
Length	90.3	89.6	86.6	84.3
Curvature	0.9	11.7	1.5	0.2

(a) Varying length regularization

1.2	1.4	4.6
91.2	90.0	89.1
1.2	0.4	0.3

(b) Varying curvature regularization

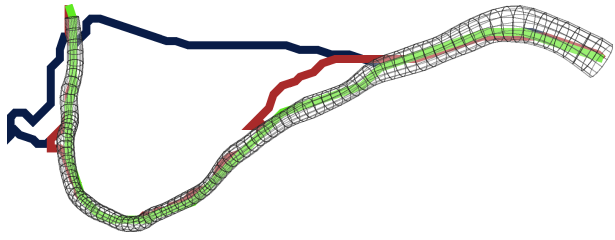
Figure 5.18: Biggest advantage for CURVATURE, the curve characteristics are defined as in Table 5.2. The ground truth (gt) is shown as a black wireframe.

Grid search gives the optimal values, $\rho = 0.08$ and $\sigma = 0.05$ for the two approaches. In Table 5.1 the result on the training set of Schaap et al. (2009) is given. The execution times are attractive compared to other methods reported in Schaap et al. (2009). The error is measured as overlap, which is defined as the percentage of the extracted curve which lies inside a ground truth artery given by both a centerline and radius.

Performing local optimization improves the results slightly and is fast compared to the shortest path execution time. LENGTH and CURVATURE are best at about the same number of arteries whereas LENGTH have slightly higher mean overlap. The resulting curves' data term, length and curvature is compared to the ground truth in Table 5.2. The curves recovered with CURVATURE have curve characteristics closer resembling to the ground truth compared to LENGTH.

To further investigate the advantages and disadvantages of the different regularizations, the two arteries which performed best for LENGTH and CURVATURE are compared. In Figure 5.17 the coronary artery which yielded the best result for LENGTH is shown. High curvature regularization is able to find the best possible solution, but still, there is no regularization which gives a result close to the ground truth due to a weak data term. In Figure 5.18 the coronary artery which yielded the best result for CURVATURE is shown. The centerline in this artery is very smooth which curvature models very well.

For the last experiment suppose that there is some natural limitations on the curve like maximum length or maximum curvature. The resulting problem is a resource constrained shortest-path problem. In Figure 5.19 an example is given showing the added benefits of constraining the total length and curvature .



	Ground truth	■	■	■
Overlap	100%	35.24%	78.43%	99.23%
Length limit (u_1)		∞	120	120
Curvature limit (u_2)		∞	∞	2
Data	11.78	5.17	6.16	9.69
Length	111.55	123.83	118.49	111.17
Curvature	1.37	15.08	12.34	1.81
Time (s)		18.10	288.75	657.00

Figure 5.19: An example where global constraints improves the result for coronary artery segmentation. Using only length regularization, $\rho = 0.08$, the resulting blue curve has low overlap with the ground truth. For the red curve more knowledge is added, constraining the length of the curve to be less than or equal to 120. The cutting-plane algorithm terminated with duality gap of $8.8 \cdot 10^{-11}$. For the green curve even more knowledge is added; the curve is constrained to have integrated squared curvature of less than or equal to 2. The algorithm terminated with duality gap of $1.9 \cdot 10^{-9}$. The curve characteristics are defined as in Table 5.2 and the ground truth artery is shown as a black wireframe.

5.7 Conclusions

This chapter has demonstrated the possibility of incorporating higher-order regularization, such as curvature and torsion, into shortest-path problems. The underlying framework has been applied to several different applications, in several different settings, showing the general applicability of the method.

The fact that the discretized problem converges to the underlying continuous problem follows from the fact that quadratic and cubic splines approximate second- and third-order derivatives well. The convergence is also demonstrated in practice in Figure 5.4. Further, by using local optimization, the initial discrete solution obtained from shortest path is improved. The benefits of using local optimization should be clear from Table 5.1 and Figure 5.14.

With the growing popularity of depth-map-generating cameras, the need to measure distances on surfaces is increasing. Previous approaches implicitly assume that the surface can be represented by a polyhedron which is not true for an arbitrary depth map. In Section 5.6.1, it is shown that this approximation breaks down for low resolution images. The section also shows how to efficiently use bilinear interpolation instead, improving the accuracy of the measured distance.

Figure 5.19 shows that constraining the maximum length and integrated curvature can improve the results. For some applications, this can be very useful, for instance if it is known that a vessel cannot be longer than x or a that robot will be worn out after a certain amount of turning.

Even though, for all applications in this chapter, the execution time is polynomial in the data size, using torsion regularization cannot be considered practical. Using torsion regularization is only possible for small problems. However, for many problems, using curvature regularization is not that computationally expensive. The usefulness of curvature regularization in medical imaging problems is shown, both in 2D (Figure 5.15) and in 3D (Figure 5.18).

The quality of the solution is dependent on the connectivity of the mesh, the higher connectivity the better. Previous works have used 8-connectivity, as in Wang (2005), and 16-connectivity, as in Krueger et al. (2013). However, the experiments in this chapter indicate that 16-connectivity is the bare minimum. Here, 32-connectivity or more has been used for all experiments.

Appendix

5.A Calculating curvature

Curvature uses second-order derivatives, which requires a quadratic B-splines to be approximated. Let (x_i, y_i, z_i) for $i = 1, 2, 3$ be three ordered points. The quadratic B-spline associated with these points is

$$\boldsymbol{r}(t) = \frac{1}{2} \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} 1 & -2 & 1 \\ -2 & 2 & 0 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} t^2 \\ t \\ 1 \end{bmatrix}. \tag{5.39}$$

The curvature is defined as $\kappa(t) = \frac{\|\boldsymbol{r}'(t) \times \boldsymbol{r}''(t)\|}{\|\boldsymbol{r}'(t)\|^3}$.

$$\kappa(t)^2 = \frac{\begin{array}{r}(c\,\Delta y - b\,\Delta z)^2 + (c\,\Delta x - a\,\Delta z)^2 \\ + (b\,\Delta x - a\,\Delta y)^2\end{array}}{\Big((a\,t - \Delta x)^2 + (b\,t - \Delta y)^2 + (c\,t - \Delta z)^2\Big)^3}, \tag{5.40}$$

where

$$\begin{aligned} a &= x_1 - 2x_2 + x_3, & \Delta x &= x_1 - x_2, \\ b &= y_1 - 2y_2 + y_3, & \Delta y &= y_1 - y_2, \\ c &= z_1 - 2z_2 + z_3, & \Delta z &= z_1 - z_2. \end{aligned} \tag{5.41}$$

With the same notation, the length element $ds = \|\boldsymbol{r}'(t)\|dt$ is equal to

$$ds = \sqrt{(a\,t - \Delta x)^2 + (b\,t - \Delta y)^2 + (c\,t - \Delta z)^2}. \tag{5.42}$$

5.B Calculating torsion

Torsion uses third-order derivatives, which requires a cubic B-splines to be approximated. Let (x_i, y_i, z_i) for $i = 1, 2, 3, 4$ be four ordered points. The cubic B-splines associated with these points is

$$\boldsymbol{r}(t) = \frac{1}{6} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \\ x_4 & y_4 & z_4 \end{bmatrix}. \tag{5.43}$$

The torsion

$$\tau(t) = \det \begin{bmatrix} \boldsymbol{r}'(t) & \boldsymbol{r}''(t) & \boldsymbol{r}'''(t) \end{bmatrix} / \|\boldsymbol{r}'(t) \times \boldsymbol{r}''(t)\|^2, \tag{5.44}$$

of the B-spline can also be expressed in closed form; let

$$\begin{aligned} N = {} & y_3\Big(x_2z_4 - x_2z_1 + x_4(z_1 - z_2)\Big) \\ & - y_4\Big(x_2z_3 - x_2z_1 + x_1(z_2 - z_3)\Big) \\ & - x_3\Big(y_1(z_2 - z_4) + y_4(z_1 - z_2)\Big) \\ & - y_2\Big(x_1(z_3 - z_4) - x_3(z_1 - z_4) + x_4(z_1 - z_3)\Big) \\ & + y_1(x_2z_3 - x_2z_4) + x_1y_3(z_2 - z_4) + x_4y_1(z_2 - z_3) \end{aligned} \tag{5.45}$$

and

$$D = (s_5s_3 - s_2s_6)^2 + (s_4s_3 - s_1s_6)^2 + (s_2s_4 - s_1s_5)^2, \tag{5.46}$$

where

$$
\begin{aligned}
s_1 &= \frac{1}{2}(z_1 - 3z_2 + 3z_3 - z_4)t^2 \\
&\quad + (2z_2 - z_1 - z_3)t + \frac{z_1}{2} - \frac{z_3}{2}, \\
s_2 &= \frac{1}{2}(y_1 - 3y_2 + 3y_3 - y_4)t^2 \\
&\quad + (2y_2 - y_1 - y_3)t + \frac{y_1}{2} - \frac{y_3}{2}, \\
s_3 &= \frac{1}{2}(x_1 - 3x_2 + 3x_3 - x_4)t^2 \\
&\quad + (2x_2 - x_1 - x_3)t + \frac{x_1}{2} - \frac{x_3}{2}, \\
s_4 &= z_1 - 2z_2 + z_3 - t(z_1 - 3z_2 + 3z_3 - z_4), \\
s_5 &= y_1 - 2y_2 + y_3 - t(y_1 - 3y_2 + 3y_3 - y_4), \\
s_6 &= x_1 - 2x_2 + x_3 - t(x_1 - 3x_2 + 3x_3 - x_4).
\end{aligned}
\tag{5.47}
$$

Then $\tau = N/D$ and $ds = \sqrt{s_1^2 + s_2^2 + s_3^2}$.

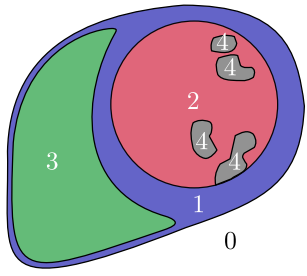
Chapter 6

Multi-region segmentation

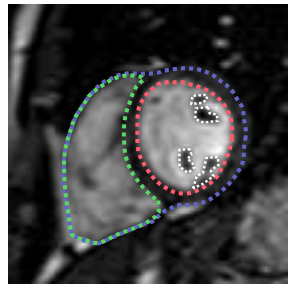
The field of medical imaging is full of challenging segmentation tasks. The aim of this chapter is to segment multiple regions simultaneously using a model which encompasses both the underlying appearance and the shape of the different regions, as well as their geometric relationships. This is often overlooked in present methods. For example, many successful cardiac segmentation approaches concentrate on segmenting the left ventricle (LV), as this part is the most interesting for diagnostic purposes. Still, quantifiable information about the cardiac function can be gained from segmenting the right ventricle (RV) as well. The proposed framework allows for the construction of a model of the *entire* heart, where the final result is improved compared to segmenting the parts independently.

The main contribution of this chapter is a multi-region segmentation framework with good optimizability. The framework builds on the multi-region scheme presented by Delong and Boykov (2009). In their paper, it is shown that geometric relationships, for instance when one object is included in another, can be modeled as a second-order pseudo-boolean function. The key property which makes efficient optimization possible is that the resulting objective function is submodular. Not all geometric relationships, however, are submodular. Delong and Boykov used roof duality for these more difficult problems. However, roof duality is too memory-intensive for large three-dimensional problems. In this chapter, the resulting non-submodular function is instead minimized using dual ascent.

Another contribution is the evaluation of the optimization framework for medical segmentation problems. The cardiac segmentation model is applied to publicly available data and the optimization framework is compared to roof duality in terms of memory and speed.



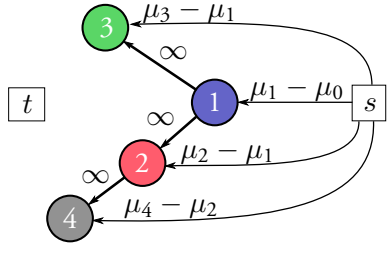
(a) Four-region model



(b) MR view

r	$\boldsymbol{x}_p$
0	$(0,0,0,0)$
1	$(1,0,0,0)$
2	$(1,1,0,0)$
3	$(1,0,1,0)$
4	$(1,1,0,1)$

(c) Representation



(d) Graph

Figure 6.1: (a) A constructed short-axis view showing how the heart is modeled. Region 0 is the background, region 1 contains myocardium *and* the left and right ventricular cavities. Region 2 is the left ventricular cavity, and region 3 the right ventricular cavity. Region 4 is the papillary muscles of the left ventricle. (b) An example of a slice from a short-axis image acquired with MRI where all four regions have been manually delineated. (c) The Boolean representation of the four regions reflect their geometric relationships as given in (a). (d) 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.

6.1 Multi-region framework

It is often easer to get a general concept after given an example. To ease into the general model consider an example construction given in Figure 6.1. The inclusion constraints are enforced by adding terms of infinite cost.

Let $\mathcal{R}$ be the set of region labels excluding the background and let $\mathcal{P}$ be the set of voxel indices. Each voxel p should be assigned a region label $r \in \mathcal{R} \cup \{0\}$ where 0 is the background region. Introduce $\boldsymbol{x} \in \mathbf{B}^n$, where $n = |\mathcal{R}||\mathcal{P}|$, $\mathbf{B} = \{0, 1\}$, and $\boldsymbol{x}$ is indexed as x_p^r, with $r \in \mathcal{R}$ and $p \in \mathcal{P}$. Furthermore, let $\boldsymbol{x}^r$ represents all Boolean variables associated with region r and $\boldsymbol{x}_p$ represents all Boolean variables associated with voxel p. Each voxel in the image is represented by $|\mathcal{R}|$ Boolean variables, which will make it possible to directly encode geometric relationships between regions, like inclusion and exclusion.

Figure 6.1(c) shows the correspondence between r and $\boldsymbol{x}_p$ for the cardiac model. The inclusion of region 2 and 3 inside region 1 is encoded in the Boolean representation by settings the first boolean variable to one. Similarly, region 4 is contained in both region 1 and region 2 and consequently, the first two Boolean variables are set to one.

The objective function to be minimized can be expressed as

$$f(\boldsymbol{x}) = \underbrace{D(\boldsymbol{x})}_{\text{data}} + \underbrace{R(\boldsymbol{x})}_{\text{regularization}} + \underbrace{G(\boldsymbol{x})}_{\text{geometric}}, \tag{6.1}$$

whose three components are, in order, the data terms, the regularization terms and the geometric interaction terms. For every voxel p, the data terms introduce a cost for each labeling of $\boldsymbol{x}_p$:

$$D(\boldsymbol{x}) = \sum_{p \in \mathcal{P}} \sum_{r \in \mathcal{R}} D_p^r \left(x_p^r\right). \tag{6.2}$$

The regularization terms use a connectivity $\mathcal{N}$ to favor smooth and correctly located boundaries:

$$R(\boldsymbol{x}) = \sum_{p \in \mathcal{P}} \sum_{q \in \mathcal{N}(p)} \sum_{r \in \mathcal{R}} R_{pq}^r \left(x_p^r, x_q^r\right). \tag{6.3}$$

The geometric interaction terms associate a cost with labeling voxel p in region i with different labelings for voxel q in region j. These terms are

used either to attract or repel different regions to each other:

$$G(\boldsymbol{x}) = \sum_{p\in\mathcal{P}} \sum_{q\in\mathcal{N}(p)} \sum_{\substack{(i,j)\in\mathcal{R}\\ i\neq j}} G_{pq}^{ij}\left(x_p^i, x_q^j\right). \tag{6.4}$$

6.1.1 Data term

The data terms are constructed from the probability of each voxel belonging to any of the regions. We define

$$\mu_r(p) = -\log\left(\mathrm{P}\left(x_p^r = 1\right)\right), \tag{6.5}$$

for voxel p and region r, where $P\left(x_p^r = 1\right)$ is the probability of voxel p belonging to region r. Region i is *parent* to region j if region j is forced to be contained inside i *directly*. By directly we mean that if region j is forced to be contained inside region i via another region k, we only consider k as a parent to region j. Regions not forced to be contained inside any specific regions is defined to have the background, $r = 0$, as parent.

As an example, consider the cardiac model in Figure 6.1. Region 4 has just one parent — region 2. Now consider any region r and let $\mathcal{G}_r$ denote the set of all parents to r, then we construct the data term as

$$D_p^r\left(x_p^r\right) = x_p^r\left(\mu_r\left(p\right) - \sum_{g\in\mathcal{G}_r}\mu_g\left(p\right)\right), \tag{6.6}$$

for all $p \in \mathcal{P}$ and $r \in \mathcal{R}$. Examples of these constructions are given in Figures 6.1 and 6.7. The reason this construction works is most easily explained through an example.

Example 6.1. *Consider the cardiac model in Figure 6.1. According to* (6.6) *we end up with:*

$$\begin{aligned}\sum_{r=1}^{4} D_p^r\left(x_p^r\right) = \quad & x_p^4\left(\mu_4\left(p\right) - \mu_2\left(p\right)\right) + x_p^3\left(\mu_3\left(p\right) - \mu_1\left(p\right)\right) \\ + \quad & x_p^2\left(\mu_2\left(p\right) - \mu_1\left(p\right)\right) + x_p^1\left(\mu_1\left(p\right) - \mu_0\left(p\right)\right).\end{aligned} \tag{6.7}$$

Now consider a voxel assigned to region 4 *from the model. We know that* $x_p^1 = x_p^2 = x_p^4 = 1$ *and* $x_p^3 = 0$ *. It follows that*

$$\begin{aligned} D_p^1(1) + D_p^2(1) + D_p^3(0) + D_p^4(1) = \\ 1(\mu_4(p) - \mu_2(p)) + 0(\mu_3(p) - \mu_1(p)) \\ +1(\mu_2(p) - \mu_1(p)) + 1(\mu_1(p) - \mu_0(p)) = \\ \mu_4(p) - \mu_0(p). \end{aligned} \tag{6.8}$$

The reason this construction works is that Boolean variables with parents are linked to their parents by the geometric interaction term. The final cost for assigning a voxel to a region is added up like a telescopic sum resulting in $\mu_r - \mu_0$ for each region r.

6.1.2 Regularization term

The regularization weights are chosen differently for each region, in a method related to the discussion in Grady and Jolly (2008). For each region i the regularization term is chosen as

$$R_{pq}^r\left(x_p^r, x_q^r\right) = w_{pq}\left(\frac{x_p^r x_q^r}{1 + \beta\left(\mathrm{P}\left(x_p^r = r\right) - \mathrm{P}\left(x_q^r = r\right)\right)^2}\right), \tag{6.9}$$

where β can be used to tune the regularization. The neighborhood $\mathcal{N}$ for the regularization is in the experiments chosen as 18-connectivity. The multipliers, w_{pq}, give different weights to different types of edges. One common choice is $w_{pq} = 1/\operatorname{dist}(p,q)$; however, we instead use the arguably more correct way described in Boykov and Kolmogorov (2003) based on solid angles. The fact that MRI has anisotropic resolution is very important to take into consideration both when calculating the distance between voxels and when using the method from Boykov and Kolmogorov (2003).

6.1.3 Geometric interaction term

Some regions should be contained inside other regions, while other regions should be forced apart. This is controlled by the geometric interaction terms.

Submodular interaction terms. Suppose region j should be *contained* inside region i. This is accomplished by settings

$$G_{pp}^{ij}(0,1) = \infty \quad \text{for all } p \in \mathcal{P}. \tag{6.10}$$

This term is clearly submodular. It is also possible to enforce a margin between two regions by setting

$$G_{pq}^{ij}(0,1) = \infty \quad \text{for all } p \in \mathcal{P}, \tag{6.11}$$

where q is taken in some neighborhood $\mathcal{N}(p)$ of p. As an example, let $\mathcal{N}(p)$ be the 8-connected neighborhood of p. Now region j will not only be forced to be inside region i, it will be forced to be slightly smaller than region i.

Non-submodular interaction terms. Similarly if region i should be *excluded* from region j set

$$G_{p,p}^{ij}(1,1) = \infty, \quad \text{for all } p \in \mathcal{P}. \tag{6.12}$$

This term is non-submodular. In some special cases the Boolean variables can be transformed, allowing for a submodular construction with exclusion constraints, see Delong and Boykov (2009) for details. However, this is not possible for either model discussed in this chapter.

6.2 Problem formulation

The standard approach for minimizing non-submodular functions of this type is to use roof duality, see Delong and Boykov (2009). This chapter will show that it is possible to optimize the functions using dual ascent resulting in a fast and memory-efficient method. For any bounded function the exclusion term for two binary variables x and y can be replaced by an equality or inequality constraint as

$$xy = 0 \qquad \text{or} \qquad x + y - 1 \leq 0. \tag{6.13}$$

Let $f'(\boldsymbol{x})$ be the objective function without the non-submodular exclusion constraints. By this construction f' will be easy to minimize.

Let $\boldsymbol{g}(\boldsymbol{x}) \leq \mathbf{0}$ encode the exclusion constraints. Adding these constraints gives us the new problem

$$\begin{aligned} \underset{\boldsymbol{x}\in\mathbf{B}^n}{\text{minimize}} \quad & f'(\boldsymbol{x}) \\ \text{subject to} \quad & \boldsymbol{g}(\boldsymbol{x}) \leq \mathbf{0}. \end{aligned} \tag{6.14}$$

This is the *primal* problem. We have now separated the easy part from the difficult non-submodular constraints.

The primal problem can in principle be solved as an integer programming problem. However, this is not a tractable approach due to the large number of variables. Instead, we look at the *dual* problem:

$$\begin{aligned} \underset{\boldsymbol{\lambda}\in\mathbf{R}^c}{\text{maximize}} \quad & d(\boldsymbol{\lambda}) \\ \text{subject to} \quad & \boldsymbol{\lambda} \geq \mathbf{0}, \end{aligned} \tag{6.15}$$

where the *dual function* is defined as

$$d(\boldsymbol{\lambda}) = \min_{\boldsymbol{x}\in\mathbf{B}^n} \left(f'(\boldsymbol{x}) + \boldsymbol{\lambda}^{\mathrm{T}} \boldsymbol{g}(\boldsymbol{x}) \right), \tag{6.16}$$

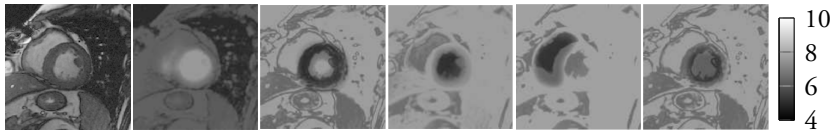
and c is the number of exclusion constraints.

In this chapter, we optimize the dual problem (6.15) using the dual ascent algorithm with the aggressive step length discussed in Section 2.4.1. Dual ascent evaluates $d(\boldsymbol{\lambda})$ iteratively, efficiency of this evaluation is of utmost importance. By rewriting the exclusion term into an inequality constraint, evaluating the $d(\boldsymbol{\lambda})$ corresponds minimization of a submodular function. The fact that we can perform efficient optimization stems from this fact. Furthermore, the structure of each dual function is very similar and a lot calculations can be reused in each evaluation as described in Kohli and Torr (2007).

6.3 Cardiac segmentation

The heart below the atrioventricular plane is modeled by four different regions as shown in Figures 6.1(a-b). The joint model describes both the geometry of the different regions and their appearances in the MR images. In the cardiac model, region 1 contains both region 2 and region 3. This is modeled by the use of geometric interaction terms as

$$G_{pp}^{12}(0,1) = \infty \quad \text{and} \quad G_{pp}^{13}(0,1) = \infty \quad \text{for all } p \in \mathcal{P}. \tag{6.17}$$



(a) Slice (b) μ_0 (c) μ_1 (d) μ_2 (e) μ_3 (f) μ_4

Figure 6.2: Example of μ_r for the slice shown in (a). Recall that $\mu_r(p) = -\log\left(\mathrm{P}\left(x_p^r = 1\right)\right)$. A lower intensity corresponds to higher probability.

Furthermore, the left ventricular papillary muscle must be inside the left ventricle. This is modeled as

$$G_{pp}^{24}(0,1) = \infty \quad \text{for all } p \in \mathcal{P}, \tag{6.18}$$

see Figure 6.1(d). We also want to exclude region 2 from 3; that is, add terms of the form $G_{pp}^{23}(1,1) = \infty$. These terms, however, become non-submodular. We want to handle the non-submodular terms using Lagrangian duality and setup the primal optimization problem as

$$\begin{aligned} \min_{\boldsymbol{x} \in \mathbf{B}^n} \quad & f'(\boldsymbol{x}) \\ \text{subject to} \quad & \boldsymbol{x}^2 + \boldsymbol{x}^3 - \mathbf{1} \leq \mathbf{0}, \end{aligned}$$

where $f'(\boldsymbol{x})$ the objective function without the non-submodular terms. The data terms construction given in Figures 6.1(c-d) results in:

$$\begin{aligned} D_p^1(1) &= \mu_1(p) - \mu_0(p), \quad D_p^2(1) = \mu_2(p) - \mu_1(p), \\ D_p^3(1) &= \mu_3(p) - \mu_1(p), \quad D_p^4(1) = \mu_4(p) - \mu_2(p), \end{aligned} \tag{6.19}$$

and $D_p^r(0) = 0$, for all $r \in \mathcal{R}$ and $p \in \mathcal{P}$.

For the heart model the spatial probability is split into four categories: left ventricle, right ventricle, myocardium, and background. Similarly the intensity is split into three categories: blood, muscle, and background. The probability for each region is then calculated with the assumption that the spatial and intensity distributions are independent. An example of the final μ_r's can be found in Figure 6.2. The spatial distribution is estimated by first resizing each image in the training data to the same size using bilinear interpolation. Then a binary mask is constructed for each

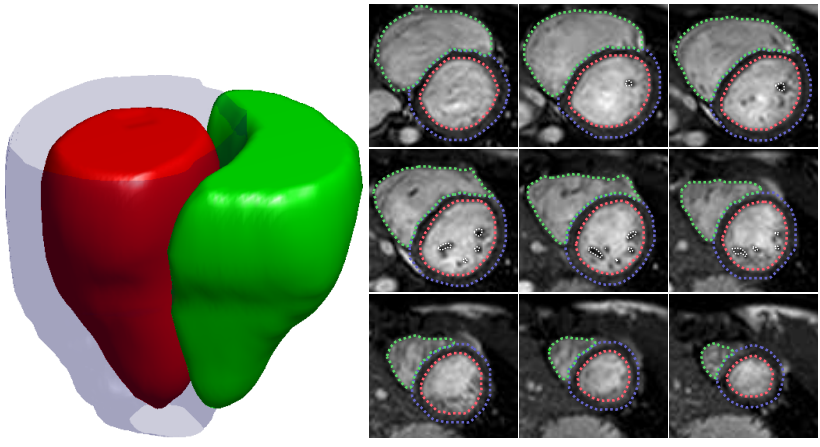
category. The masks are enlarged and smoothed and then they are all added together constructing the final probability mask. The intensity distribution for each region is estimated by collecting all intensities from the examples in the training data. The histogram of intensities is then smoothed and a distribution is constructed. For both the location and intensity probability a lowest probability is set, in order to capture occurrences unseen in the training data. The user selects which slices to be segmented and selects a center point of the right and left ventricle in *one* slice. The two center points are used to roughly align the hearts in order to get good spatial statistics. The algorithm can handle slices lacking any of the regions. Badly captured MRIs are identified by looking at the distribution of the intensities. If there are multiple peaks in the histogram close to each other for the lower intensities, the image is assumed to be too bright and the intensity distribution is shifted to fit an average histogram.

In all ground truth data considered only the left ventricular epicardium is delineated. The model is not restricted to this — it segments the full myocardium. In order to compare the results with the ground truth all myocardium which is not part of the left ventricular epicardium must be removed. To do this, the thickness of the septum is approximated as the shortest distance between the left and right ventricles in the resulting segmentation. Then the outlying myocardium is removed based on this thickness approximation. Another assumption on the segmentation is that the left ventricle and the myocardium are convex. The resulting segmentation is taken as the convex hull in each slice.

Due to the regularization, the segmentation sometimes misses the most apical slice. From the user input it is known which slices the left and right ventricles are contained in and it would be wasteful to throw this information away. The user input is utilized by naturally extending the segmentation into the apical slice. This is done by taking the segmentation from another slice, shrinking it slightly and inserting at the bottom.

6.3.1 Experiments

The segmentation is only performed on the slices of the heart which are fully below the atrioventricular plane. The quality of the segmentation is measured by the *dice metric* given by $2\,|A \cap B| \,/\, (|A| + |B|)$, where A is the ground truth segmentation and B is the computed segmentation. The



(a) 3D rendering (b) 10 slices

Figure 6.3: Example segmentation from LUND. The color scheme is the same as in Figure 6.1.

	End systole			End diastole		
	LV endo.	LV epi.	RV	LV endo.	LV epi.	RV
Multi	0.87 ±0.05	0.88 ±0.05	0.80 ±0.11	0.96 ±0.02	0.93 ±0.03	0.91 ±0.07
Single	0.47 ±0.25	0.86 ±0.04	0.42 ±0.14	0.62 ±0.12	0.90 ±0.03	0.57 ±0.14

Table 6.1: Results measured in the dice metric for LUND reported as mean ± one standard deviation. On the first row the full multi-region model is used. On the second row each region is segmented separately. Note that the multi-region model has a huge influence on the segmentation results.

algorithm is evaluated on two datasets: LUND and SUNNYBROOK. Each dataset is trained and evaluated separately.

LUND consists of cine short-axis steady-state free-precession MR images of 62 healthy normal hearts captured on a Philips Interera CV 1.5T with a five-channel cardiac synergy coil. Each heart has the left and right ventricular endocardium and the left ventricular epicardium manually delineated by an expert. The dataset is split into two equally sized parts, one used for training and one used for evaluation. Results are given in the first row of Table 6.1 and an example segmentation is given in Figure 6.3. Three clinical parameters are also evaluated: the left ventricular mass has an error of 15.6 ± 11.5 g, the left and right ventricular ejection fraction errors are 5.6 ± 2.9 % and 7.1 ± 5.2 %, respectively.

The method is also compared to a simplified version where each region is segmented separately, see the second row of Table 6.1. Without the complete multi-region model, the localization of the ventricles becomes very difficult and the blood pools are often overestimated. A few typical examples where the multi-region model improves the segmentation are given in Figures 6.4 and 6.5.

SUNNYBROOK consists of 30 patients with different heart diseases and is split up into two equally sized parts, one for training and one for evaluation. The dataset was used in a segmentation challenge, see Radau et al. (2009). SUNNYBROOK lacks ground truth for the right ventricles, this was manually constructed by a non-expert. Therefore, this ground truth is only used for training and not for evaluation. The results, given by the evaluation code used in the challenge, are given in Table 6.3 along with results from competing methods. The evaluation code in the challenge calculates the dice metric per slice and averages over all slices.

The small number of training data of SUNNYBROOK gives the proposed method a disadvantage as there are only 15 hearts spanning over three different diseases and one group of normals. Image-driven methods do not suffer from the small training set as they do not need to be trained. The limited number of training examples impedes the model since there are too few examples of variation in shape for each disease and the normals. The intensity model is less affected by this but would still benefit from a larger training set. Note that all diseases and normals are covered by one model.

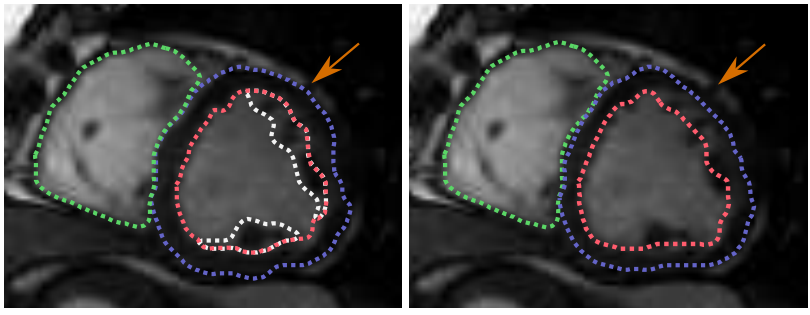
The model is also optimized using roof duality (RD). If RD is unable to label all variables, the methods "probing" or "improve" are used to obtain a

complete labeling, this is denoted as *RD-P* and *RD-I* respectively. For RD-I and dual ascent, the same termination criterion is used: either the relative duality gap must be smaller than 10^{-4} or a maximum of 25 iterations must be reached. RD-P is terminated either if all variables are labeled or after a maximum of 12 hours execution time. If some variables still are unlabeled after 12 hours they are set to 0. Dual ascent is faster and uses less memory than both RD-I and RD-P. The final results for all the optimization methods in terms of quality of segmentation are virtually identical, see Table 6.2. In particular, all methods achieve small duality gaps. Only RD-P encounters some problems on the SUNNYBROOK dataset, where the larger duality gap is a result of 3 hearts that are not completely labeled after 12 hours of probing. The progress of the duality gap and dice over time for the different optimization methods is depicted in Figure 6.6.

To summarize, dual ascent finds a globally optimal solution for 52% of the hearts. For the other hearts, from the very small relative duality gap, it is certain that dual ascent finds a solution close to the global optimum. For 4 out of a total of 46 hearts, probing takes more than 12 hours. This highlights the problem with probing — there is no real guarantee that the computations will be done within a reasonable time; on some problem instances, probing is unable to return a complete solution, even after weeks of execution time.

It is possible to extend the cardiac model to also include papillary muscles in the right ventricle; only one more variable is needed per voxel. Initial experiments gave worse results for both the right ventricle segmentation and the myocardium segmentation with the added region. The new region had a tendency to overflow into the septum since this would give region 3 a rounder shape, resulting in lower regularization cost. Therefore, the model used in this chapter only has four regions.

The LUND dataset is manually delineated using both short- and long-axis images. For a number of hearts, the most basal slice of the short-axis images containing the left ventricular cavity also cuts through to the atrium. For these slices it is hard or even impossible to manually delineate the left ventricle solely based on information from the short-axis images. When the ground truth was produced, long-axis images were used to properly segment them. It would be desirable for the algorithm to incorporate information from long-axis images as well so that the algorithm could handle these few slices.

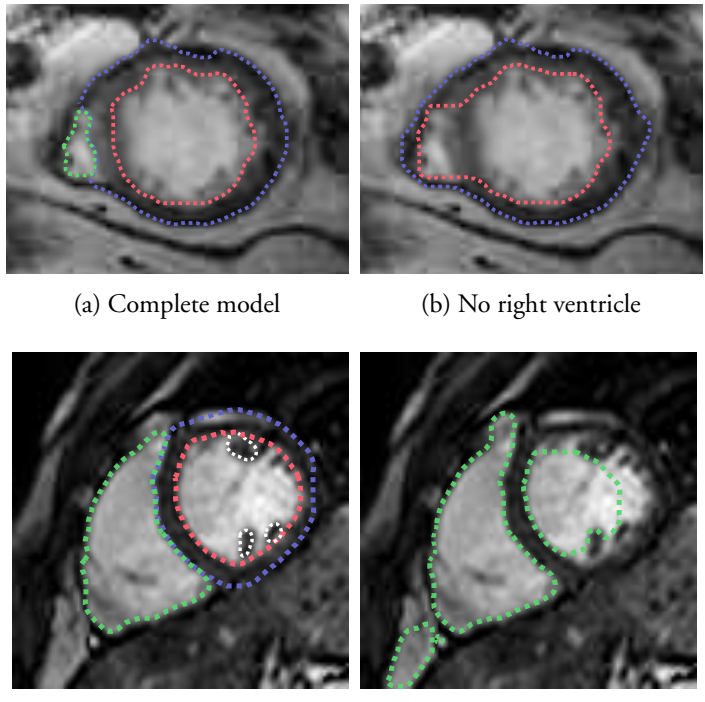


(a) Complete model (b) No papillary muscles

Figure 6.4: Example showing that the complete model including the papillary muscles can improve the segmentation of the left ventricle and the myocardium.

Method	Memory (MB)		Time (s)	
	SUNNYBROOK	LUND	SUNNYBROOK	LUND
Dual ascent	2727 ±680	2103 ±788	46 ±27	30 ±27
RD-I	5038 ±985	3913 ±1407	135 ±165	80 ±113
RD-P	5041 ±1014	3949 ±1402	6109 ±12451	1934 ±7984
Method	Relative duality gap		Dice (average)	
	SUNNYBROOK	LUND	SUNNYBROOK	LUND
Dual ascent	0.00054 ±0.0013	0.00054 ±0.0021	0.888 ±0.0484	0.892 ±0.0815
RD-I	0.00016 ±0.00034	0.00049 ±0.0021	0.888 ±0.0485	0.892 ±0.0816
RD-P	0.0011 ±0.0030	0.00056 ±0.0021	0.888 ±0.0484	0.892 ±0.0825

Table 6.2: Memory consumption of different optimization methods in megabyte and their relative duality gap. The resolution of the data in SUNNYBROOK is on average $146 \times 146 \times 10 \times 2$ voxels and for LUND on average $126 \times 126 \times 10 \times 2$ voxels. The third dimension, the number of slices, varied from heart to heart. Comparing each problem instance, instead of the total mean, both versions of RD used ≈ 1.9 times more memory than the dual ascent algorithm. The dice measure is calculated as an average over each region where ground truth is available in the datasets.



(a) Complete model (b) No right ventricle

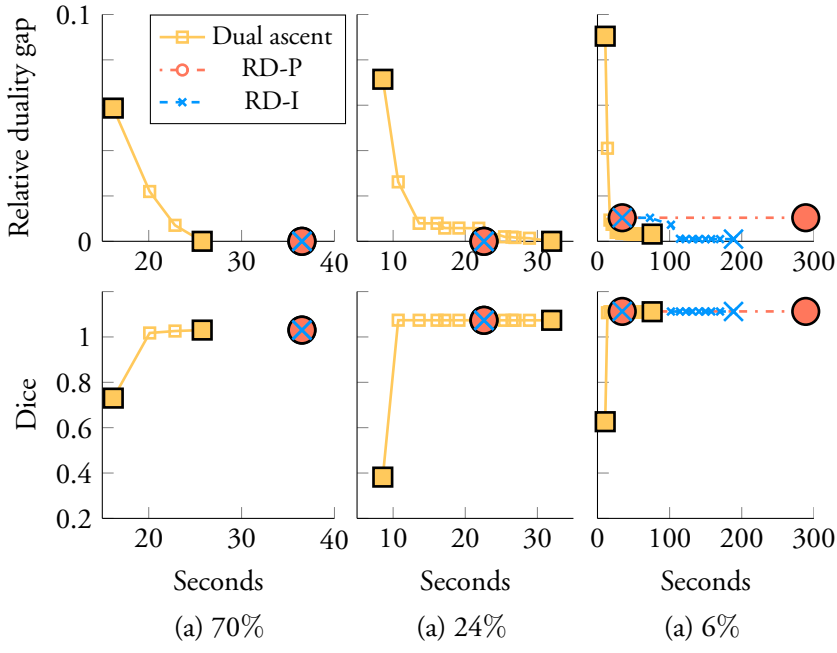
(c) Complete model (d) Only right ventricle

Figure 6.5: Examples of how modeling multiple regions improves the segmentation of the ventricular epicardium and endocardium. The color scheme is the same as in Figure 6.1.

Method	Dice		LV Mass (g)	LV ejection fraction (%)
	LV endo.	LV epi.		
This ch.	0.86 ±0.05	0.92 ±0.02	27.1 ±28.3	12.5 ±8.7
A	0.86 ±0.04	0.93 ±0.01	23 ±?	14 ±?
B	0.89 ±0.03	0.94 ±0.02	21.6 ±14.6	8.08 ±5.06
C	0.89 ±0.03	0.93 ±0.01	28.7 ±18.7	7.02 ±4.78
D	?	0.93 ±?	†	?
E	0.81 ±?	0.91 ±?	?	?
F	0.89 ±0.04	0.92 ±0.02	†	†
G	0.89 ±0.04	0.94 ±0.01	?	?
H	0.88 ±0.04	0.93 ±0.02	31.8 ±17.7	8.35 ±5.78

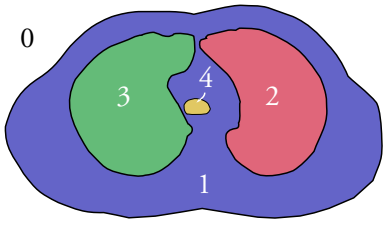
A: Marák et al. (2009)
B: Lu et al. (2009)
C: Wijnhout et al. (2009)
D: Casta et al. (2009)
E: O'Brien et al. (2009)
F: Constantinides et al. (2009)
G: Huang et al. (2009)
H: Jolly (2009)

Table 6.3: Results for SUNNYBROOK. "?" means that the result is not reported in the corresponding paper. "†" means that the result is not directly comparable. Mass and ejection fraction is reported as the difference between manual and automatic value.

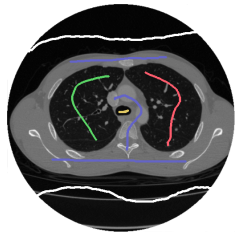


(a) 70% (a) 24% (a) 6%

Figure 6.6: The typical progress for all optimization methods compared in this chapter. For all methods the results in the dice metric are virtually identical. The results can be divided into three different categories: In (a) all iterations of dual ascent are done before the initial RD calculations are completed. This happens for 70% of the hearts. In (b) RD manages to label all variables before dual ascent terminated, which occurs for 6% of the hearts. In (c) RD is unable to label all variables and RD-P converge very slowly, which happened for 24% of the hearts. The top row shows the relative duality gap and the bottom row the dice metric, as a function of time for the three methods. The first and last iterations are highlighted and for RD-P the number of iterations is limited to 25.



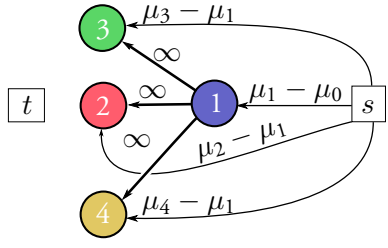
(a) Four-region model



(b) Seeds provided by user

r	$\boldsymbol{x}_p$
0	$(0,0,0,0)$
1	$(1,0,0,0)$
2	$(1,1,0,0)$
3	$(1,0,1,0)$
4	$(1,0,0,1)$

(c) Representation



(d) Graph

Figure 6.7: (a) A diagram showing the model used for lung segmentation. Region 0 is the background, region 1 is the body, regions 2 and 3 are the right and left lungs, respectively, and region 4 is the throat. (b) The seeds in one slice used for the segmentation. In a clinical setting, these are provided by a physician. (c) The Boolean representation of the four regions. (d) Graph construction for one voxel, showing the geometrical relationships.

6.4 Lung segmentation

The second application of this chapter is segmentation of lungs in a full-body X-ray CT scan. The model is shown in Figure 6.7 and uses four regions: the body, region 1, the two lungs, region 2 and 3, and the heart together with the throat, region 4. Regions 2, 3, and 4 are all forced to be contained inside region 1 by adding the terms,

$$G_p^{12}(0,1) = \infty, \quad G_p^{13}(0,1) = \infty, \text{ and } G_p^{14}(0,1) = \infty \text{ for all } p \in \mathcal{P}. \tag{6.20}$$

The largest difference to the cardiac model is that more than one separation is needed to be enforced:

$$\begin{aligned} \underset{\boldsymbol{x}\in\mathbf{B}^n}{\text{minimize}} \quad & f'(\boldsymbol{x}) \\ \text{subject to} \quad & \boldsymbol{x}^2 + \boldsymbol{x}^3 + \boldsymbol{x}^4 - \mathbf{1} \leq \mathbf{0}. \end{aligned} \tag{6.21}$$

Alternatively, the three-variable constraint could equivalently be replaced with three constraints of the same type as in the previous section

$$\begin{aligned} \boldsymbol{x}^2 + \boldsymbol{x}^3 - \mathbf{1} \leq \mathbf{0}, \\ \boldsymbol{x}^2 + \boldsymbol{x}^4 - \mathbf{1} \leq \mathbf{0}, \\ \boldsymbol{x}^3 + \boldsymbol{x}^4 - \mathbf{1} \leq \mathbf{0}. \end{aligned} \tag{6.22}$$

The data terms construction can be seen in Figure 6.7(d). We get:

$$\begin{aligned} D_p^1(1) = \mu_1(p) - \mu_0(p), \quad D_p^2(1) = \mu_2(p) - \mu_1(p), \\ D_p^3(1) = \mu_3(p) - \mu_1(p), \quad D_p^4(1) = \mu_4(p) - \mu_1(p), \end{aligned} \tag{6.23}$$

and $D_p^r(0) = 0$, for all $r \in \mathcal{R}$ and $p \in \mathcal{P}$.

The user gives ground truth seeds only in one slice of the data as shown in Figure 6.7(b). The background is removed by thresholding on an intensity level between the seeds given from the background and the body. The seeds are then used to build intensity histograms for the five regions which are then used to estimate the intensity distribution. No kind of spatial statistics is estimated, the probability is approximated by a fading gradient from the left and right side of each slice. See Figure 6.10 for example data terms.

6.4.1 Experiments

The method is tested on a full-body X-ray CT dataset with seeds as shown in Figure 6.7. A sample result from a few slices can be seen in Figure 6.8. The execution time for roof duality is 39 seconds and for dual ascent 29 seconds. Both methods give exactly the same solution.

One thing not taken into account is the fact that the structure of each function is highly repetitive. For instance, all geometric interaction terms are equal and they need not be stored explicitly. A specialized-purpose solver reducing the memory usage is introduced in Rykfors (2012); it only supports submodular constraints and certain fixed neighborhoods.

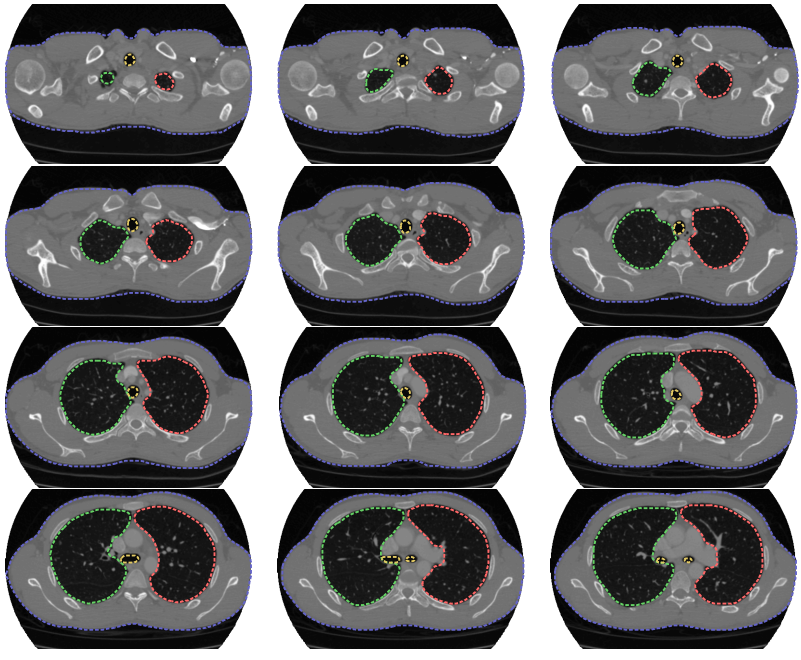
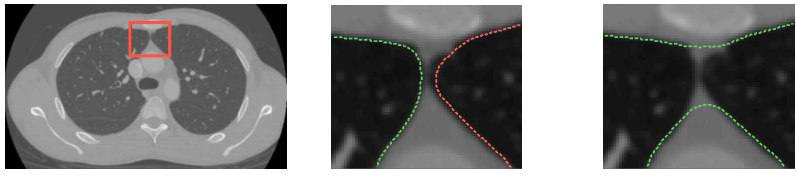
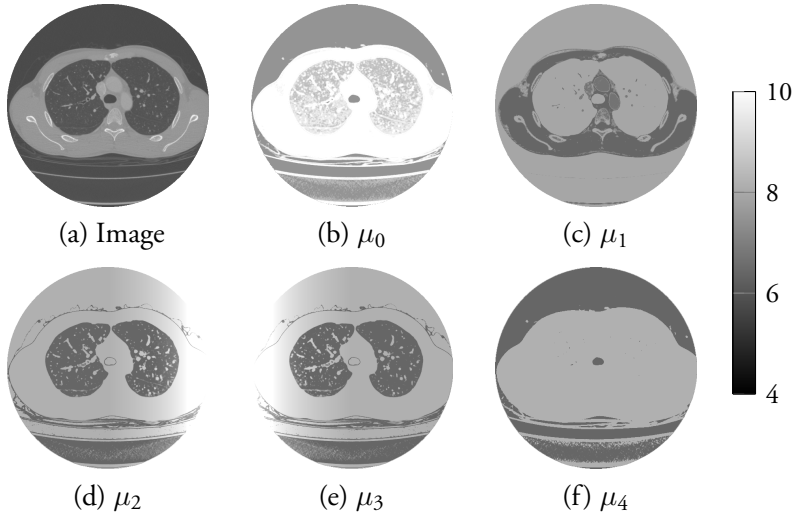


Figure 6.8: Sample results from the segmentation, the same color coding as in Figure 6.7 is used.



(a) Region of interest (b) Forced separation (c) No separation

Figure 6.9: In (a) the region of interest is marked with a rectangle. In (b) the model has constraints that force the two regions to be separate. In (c) the exclusion constraint is removed. This results in a segmentation where one region wrongly overflows into the other. Note that the image data for the correct segmentation is very weak and hence it is necessary to encode this prior information into the model.



(a) Image (b) μ_0 (c) μ_1

(d) μ_2 (e) μ_3 (f) μ_4

Figure 6.10: An example slice from the dataset with the data term for all 5 regions shown.

6.5 Conclusions

In this chapter, it has been demonstrated, through experimental results, that a multi-region model achieves significantly better results than segmenting the regions separately. See Figures 6.4, 6.5, and 6.9. Enforcing geometric constraints and, more generally, incorporating prior information into the model results in qualitative improvements. The qualitative improvements are not always captured by quantitative measures such as the dice metric.

The optimization method based on dual ascent, proposed in this chapter, outperforms roof duality, both in terms of speed and memory consumption.

Application of the multi-region framework on cardiac segmentation, using a publicly available data base, achieves results on par with dedicated left ventricle methods. There are fine-tunings which can be made to improve the segmentation performance, such as a better data term and better parameter choices. However, the obtained results are still encouraging.

Finally, the model can easily be modified to fit other medical imaging tasks; by adding and removing regions, as demonstrated by the small differences between the heart and the lung models.

Chapter 7

Dense stereo

This chapter deals with the classical dense stereo problem: given two or more input images, estimate the depth or disparity of every pixel in one of the images. To accomplish this, pixels in the different images are matched using some matching criterion. Most matching criteria suffers from ambiguity which is a result of e.g. textureless patches or occlusions. This turns dense stereo estimation into an ill-posed problem and therefore regularization has to be incorporated.

The most popular regularizers are the first-order ones, see Boykov, Veksler, et al. (2001); Felzenszwalb and Huttenlocher (2006); Kolmogorov and Zabih (2002). These typically penalize assignments where neighboring pixels have different disparity or depth. Their popularity is in large part due to the fact that they often result in submodular formulations when applying move-making algorithms such as α-expansion (Boykov, Veksler, et al. (2001)) or fusion moves (Lempitsky et al. (2010)). On the downside, this type of regularization favors fronto-parallel planes, since surfaces with non-zero disparity derivatives or depth derivatives are penalized.

To address this problem, Birchfield and Tomasi (1999) use 3D labels corresponding to arbitrary 3D planes. The interaction cost between coplanar points is zero and therefore this approach is suitable for piecewise planar scenes. Li and Zucker (2010) use pairwise interactions between 3D labels (encoding disparity and disparity derivatives) to penalize non-smooth surfaces. In contrast, Woodford et al. (2009) use regular disparity labels to encode second derivative priors. However, the optimization problem is difficult due to the introduction of non-submodular third-order terms used to approximate the second derivatives.

In this chapter, we will encode second derivative priors using labels similar to those used by Li and Zucker (2010). The resulting problem is optimized using iterative fusion of proposals. It will be shown, both

theoretically and experimentally, that the second-order fusion problems are easier to optimize than their third-order counterpart. The formulation also allows us to fuse more than two proposals simultaneously. Lastly, it is shown how a large family of proposals can be generated using local optimization.

7.1 A second-order regularization prior

In this chapter we will let the depth or disparity for each pixel be encoded by tangent planes. The intersection between the viewing ray and each tangent plane will be the estimated depth or disparity. By interpreting the planes as tangents of the viewed 3D surface, we can encourage smooth solutions by penalizing neighboring points which deviate from neighboring tangents. The standard approach to dense stereo is to simply encode each pixel with either depth or disparity. Hence, it might seem counterintuitive to make the label space larger by introducing tangent planes. But in this chapter, it will be shown that the resulting optimization problem is actually easier to solve when using the higher-order label space formulation.

Throughout this chapter we will let $\mathcal{I}$ be the reference image, for which we are to perform dense stereo estimation. The two pixels $\boldsymbol{p} \in \mathcal{I}$ and $\boldsymbol{q} \in \mathcal{I}$ are always assumed to be neighbors in the image.

7.1.1 Tangent planes and spaces

Assume a pinhole camera model where a camera has been calibrated and normalized to be of form $[\boldsymbol{I}\ \mathbf{0}]$. The projection of a scene point, $\mathbf{P} = (X, Y, Z)$, into a pixel, $\boldsymbol{p} = (x, y) \in \mathcal{I}$, is then simply given by

$$x = X/Z, \tag{7.1}$$

$$y = Y/Z. \tag{7.2}$$

Given a depth function z and a disparity function d, the point in *scene space* corresponding to the pixel $\boldsymbol{p}$ is given by

$$\mathbf{P} = \big(\boldsymbol{p}, 1\big) z(\boldsymbol{p}), \tag{7.3}$$

the corresponding point in *depth space* is given by

$$\mathbf{P}_z = \big(\boldsymbol{p}, z(\boldsymbol{p})\big), \tag{7.4}$$

and the corresponding point in *disparity space* is given by

$$\mathbf{P}_d = \big(\boldsymbol{p}, d(\boldsymbol{p})\big). \tag{7.5}$$

Now consider any plane in scene space. Any such plane can be expressed as all points $\boldsymbol{x} \in \mathbf{R}^3$ fulfilling

$$\boldsymbol{n}^{\mathrm{T}}\boldsymbol{x} + a = 0, \tag{7.6}$$

where $\boldsymbol{n} \in \mathbf{R}^3$ and $a \in \mathbf{R}$. This plane can in terms of a depth function z be expressed as

$$\boldsymbol{n}^{\mathrm{T}}\big((\boldsymbol{p}, 1)z(\boldsymbol{p})\big) + a = 0. \tag{7.7}$$

This is equivalent to

$$z(\boldsymbol{p}) = -\frac{a}{\boldsymbol{n}^{\mathrm{T}}(\boldsymbol{p}, 1)}. \tag{7.8}$$

Since the disparity, d, is inversely proportional to the depth, it immediately follows that:

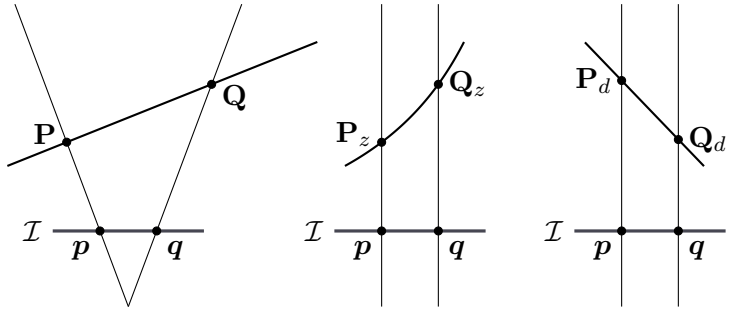
$$d(\boldsymbol{p}) = -b\frac{\boldsymbol{n}^{\mathrm{T}}(\boldsymbol{p}, 1)}{a}, \tag{7.9}$$

where b is the baseline of the stereo setup. In Figure 7.1, a plane in 2D (a line) is shown in scene, depth, and disparity space.

From (7.8) and (7.9) an important fact follows: The image of a plane viewed in depth space is not necessarily a plane itself. Hence, it does not make sense to penalize deviations from tangent planes in depth space. In contrast, the image of a plane viewed in disparity space is a plane itself. This allows us to penalize deviations from tangent planes directly in disparity space. For depth space, we have to work with tangent planes in scene space, making the derivations slightly more complicated.

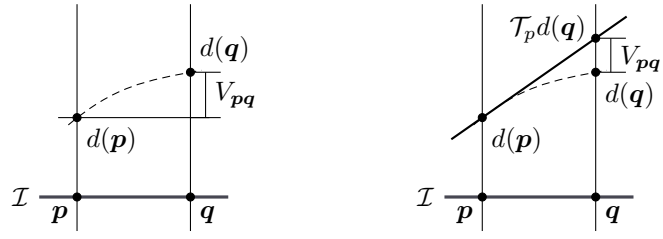
7.1.2 Rectified cameras and disparity

We will start by assuming that the cameras in the stereo setup have been rectified, since this allows us to work in disparity space. For multiple views, this places some restrictions on the camera positions which are usually not



(a) Scene space $(\boldsymbol{p}, 1)z(\boldsymbol{p})$ (b) Depth space $(\boldsymbol{p}, z(\boldsymbol{p}))$ (c) Disparity space $(\boldsymbol{p}, d(\boldsymbol{p}))$

Figure 7.1: The same plane viewed in scene, depth, and disparity space with the viewing rays for the two pixels $\boldsymbol{p}$ and $\boldsymbol{q}$ and their corresponding space points marked out. Note that a plane in scene space is also a plane in disparity space.



(a) Standard stereo regularization (b) Proposed stereo regularization

Figure 7.2: Geometric interpretation of the regularization term for rectified cameras. The image plane, $\mathcal{I}$, and the viewing rays of pixel $\boldsymbol{p}$ and $\boldsymbol{q}$ are marked. The dashed line indicates a possible disparity function d. In (a) a standard first-order stereo regularizer is shown. The difference between the disparity estimates, $V_{\boldsymbol{pq}} = |d(\boldsymbol{p}) - d(\boldsymbol{q})|$, is penalized. In (b) the proposed stereo regularizer is shown. The distance between the tangent planes, $V_{\boldsymbol{pq}} = |\mathcal{T}_{\boldsymbol{p}} d(\boldsymbol{q}) - d(\boldsymbol{q})|$, is penalized.

fulfilled in general image collections, see Seitz (2001) for further details. We will therefore relax the restrictions on the camera positions in Section 7.1.3 where we work with regular cameras.

To each pixel $\boldsymbol{p} \in \mathcal{I}$, we assign a tangent plane which locally approximates the disparity. We can think of these tangents as samples of the disparity function and its derivatives. By the function $\mathcal{T}_{\boldsymbol{p}} d : \mathcal{I} \to \mathbf{R}$, we mean the tangent at the point $\boldsymbol{p}$, seen as a function of the entire image, that is

$$\mathcal{T}_{\boldsymbol{p}} d(\boldsymbol{x}) = d(\boldsymbol{p}) + \boldsymbol{\nabla d}(\boldsymbol{p})^T (\boldsymbol{x} - \boldsymbol{p}), \tag{7.10}$$

where $d(\boldsymbol{p})$ and $\boldsymbol{\nabla d}(\boldsymbol{p})$ are the assigned disparity and disparity gradient (with respect to the image coordinate system) at pixel $\boldsymbol{p}$. We define the interaction between neighboring pixels as

$$V_{\boldsymbol{pq}} = |\mathcal{T}_{\boldsymbol{p}} d(\boldsymbol{q}) - d(\boldsymbol{q})|, \tag{7.11}$$

see Figure 7.2. Intuitively, if the surface is smooth, then the tangent plane should be a good approximation of the surface. Therefore $V_{\boldsymbol{pq}}$ is expected to be small for smooth surfaces. Using Taylor expansion we get

$$d(\boldsymbol{q}) \approx d(\boldsymbol{p}) + \boldsymbol{\nabla d}(\boldsymbol{p})^T (\boldsymbol{q} - \boldsymbol{p}) + \frac{1}{2} (\boldsymbol{q} - \boldsymbol{p})^T \boldsymbol{H}(\boldsymbol{p}) (\boldsymbol{q} - \boldsymbol{p}), \tag{7.12}$$

where $\boldsymbol{H}(p)$ is the Hessian of d at p. It follows that

$$V_{\boldsymbol{pq}} \approx \left| \frac{1}{2} (\boldsymbol{q} - \boldsymbol{p})^T \boldsymbol{H}(\boldsymbol{p}) (\boldsymbol{q} - \boldsymbol{p}) \right|. \tag{7.13}$$

The regularization, $V_{\boldsymbol{pq}}$, measures the second derivative of the disparity function, d, at $\boldsymbol{p}$ in direction $\boldsymbol{q} - \boldsymbol{p}$.

7.1.3 Regular cameras and depth

In many real-world situations, rectified cameras may not be available. In such cases we work with depth rather than disparity. Once again deviations from neighboring tangent planes are penalized, see Figure 7.3. This time, however, we are considering tangent planes in scene space. By the function $\mathcal{T}_{\boldsymbol{p}} z : \mathcal{I} \to \mathbf{R}$, we mean the tangent at the point $\boldsymbol{p} \in \mathcal{I}$, seen as a function of the entire image, encoded by the depth $z(\boldsymbol{p})$ and the depth

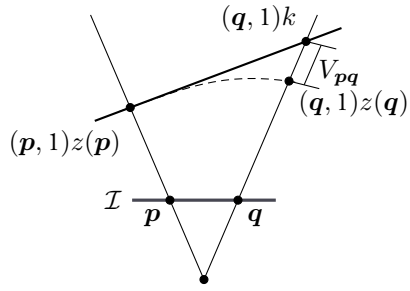


Figure 7.3: Geometric interpretation of the regularization term for regular cameras. The regularization term, $V_{\boldsymbol{pq}}$, measures the deviation from the neighboring tangent along the viewing ray.

gradient $\boldsymbol{\nabla z}(\boldsymbol{p})$. Assume a pinhole camera model where a camera has been calibrated and normalized to be of form $[\boldsymbol{I}\ \boldsymbol{0}]$. The point

$$(\boldsymbol{q},1)k = (\boldsymbol{q},1)\mathcal{T}_{\boldsymbol{p}}z(\boldsymbol{q}) \tag{7.14}$$

is the intersection between the tangent plane of pixel $\boldsymbol{p}$ and the viewing ray of pixel $\boldsymbol{q}$. The line between $(\boldsymbol{p},1)z(\boldsymbol{p})$ and $(\boldsymbol{q},1)k$ is contained in the tangent plane $\mathcal{T}_{\boldsymbol{p}}z(\boldsymbol{q})$ and can therefore be found by linearizing the curve

$$(\boldsymbol{p}+t\mathbf{v},1)z(\boldsymbol{p}+t\mathbf{v}). \tag{7.15}$$

Here $\mathbf{v}$ is a vector chosen such that

$$\boldsymbol{q} = \boldsymbol{p} + s\mathbf{v}, \tag{7.16}$$

where s is the distance between the two pixels. Linearization gives

$$\boldsymbol{l}(t) = (\boldsymbol{p},1)z(\boldsymbol{p}) + t\left((\boldsymbol{p},1)z'_{\mathbf{v}}(p) + (\mathbf{v},0)z(\boldsymbol{p})\right). \tag{7.17}$$

At the intersection point $(\boldsymbol{q},1)k$, we have

$$\boldsymbol{l}(t) = k(\boldsymbol{q},1) = k\left((\boldsymbol{p},1) + s(\mathbf{v},0)\right). \tag{7.18}$$

Identification of the coefficients yields

$$k = z(\boldsymbol{p}) + z'_{\mathbf{v}}(\boldsymbol{p})t, \tag{7.19}$$

$$s = \frac{z(\boldsymbol{p})t}{z(\boldsymbol{p}) + z'_{\mathbf{v}}(\boldsymbol{p})t}, \tag{7.20}$$

$$t = \frac{z(\boldsymbol{p})s}{z(\boldsymbol{p}) - sz'_{\mathbf{v}}(\boldsymbol{p})}, \tag{7.21}$$

$$k = \frac{z(\boldsymbol{p})^2}{z(\boldsymbol{p}) - sz'_{\mathbf{v}}(\boldsymbol{p})}. \tag{7.22}$$

The regularization can thus be expressed as

$$V_{\boldsymbol{pq}} = |\mathcal{T}_{\boldsymbol{p}} z(\boldsymbol{q}) - z(\boldsymbol{q})| \, \|(\boldsymbol{q}, 1)\| \tag{7.23}$$

$$= |k - z(\boldsymbol{q})| \, \|(\boldsymbol{q}, 1)\| \tag{7.24}$$

$$= \left| \frac{z(\boldsymbol{p})^2}{z(\boldsymbol{p}) - sz'_{\mathbf{v}}(\boldsymbol{p})} - z(\boldsymbol{q}) \right| \|(\boldsymbol{q}, 1)\| . \tag{7.25}$$

Remark 7.1. *The scalar $\|(\boldsymbol{q}, 1)\|$ can be incorporated into a regularization constant for the regularization term.*

7.2 General-order regularization priors

In the previous sections we used tangent planes to create the regularization. It is possible to use higher-order local models to encode more complex regularizations. Let $\mathcal{A}_{\boldsymbol{p}} f$ be a Taylor approximation of order n, then the interaction

$$V_{\boldsymbol{pq}} = |\mathcal{A}_{\boldsymbol{p}} f(\boldsymbol{q}) - f(\boldsymbol{q})|, \tag{7.26}$$

would be a $n+1$ order regularization penalty. At the same time it is possible to add a penalty for derivatives of order at most n, using only data terms. For example, if we to each pixel assigns a quadratic function instead of a tangent, then the interaction penalizes third-order derivatives. In this case first- and second-order derivatives can be encoded into the data term. Table 7.1 shows properties for some different choices of labels spaces.

Labels	Regularize	Data term	Submodular proposals
Depth or disparity	f'	f	Constant f
Tangent planes	f''	f and f'	Constant f'
Second-order approximations	f'''	f, f', and f''	Constant f''
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Table 7.1: Characterization of possible regularization terms given different label spaces and their submodular proposals for fusion moves. The depth or disparity function is denoted by f and its first, second, and third derivatives in the direction of a neighboring pixel is denoted by f', f'' and f'''. Note that higher-order label spaces also allows more information to be included in the data term.

Using the methods shown later in Proposal 7.4 on page 168, it is easy to see that if the proposed solutions fulfill

$$\mathcal{A}_{\boldsymbol{p}} f(\boldsymbol{q}) = f(\boldsymbol{q}), \tag{7.27}$$

then the fusion move will be submodular. Hence the n'th order surfaces give submodular interactions.

Example 7.2. *The regularization interaction for the zero-order Taylor expansion corresponds to*

$$V_{\boldsymbol{pq}} = |f(\boldsymbol{p}) - f(\boldsymbol{q})|, \tag{7.28}$$

which is the standard first-order stereo regularization from Boykov, Veksler, et al. (2001), see Figure 7.2. Fusion moves using this regularization term corresponds to α-expansion.

7.3 Problem formulation

The dense stereo estimation problem is formulated as finding the minimizer of the objective function

$$E(f) = \underbrace{\sum_{\boldsymbol{p}\in\mathcal{I}} D_{\boldsymbol{p}}(f)}_{\text{data terms}} + \underbrace{\sum_{\boldsymbol{p}\in\mathcal{I}} \sum_{\boldsymbol{q}\in\mathcal{N}(\boldsymbol{p})} R_{\boldsymbol{pq}}(f)}_{\text{regularization terms}}, \tag{7.29}$$

where $\mathcal{N}(\boldsymbol{p})$ is a predefined neighborhood to $\boldsymbol{p}$. For rectified cameras $f = d$ denotes the disparity function, and for regular cameras $f = z$ denotes the depth function. In the experimental section two different objective functions will be used. One is based on *normalized cross correlation* and the other is based on *pixel-wise photoconsistency*. The second function is introduced in order to facilitate comparison to the work of Woodford et al. (2009). Figure 7.4 shows the difference between the data terms used by the two objective functions.

Normalized cross correlation

Normalized cross correlation (NCC) compares a patch around a pixel to a patch around the potential matching pixel. Let p_l and p_r denote the two patches to be matched and $(\boldsymbol{x}_1^l, \ldots, \boldsymbol{x}_n^l)$ and $(\boldsymbol{x}_1^r, \ldots, \boldsymbol{x}_n^r)$ denote the pixels in each patch. Furthermore, let μ_i denote the mean and σ_i the standard deviation for the pixel intensities for patch i. The normalized cross correlation is given by

$$\text{NCC}(p_l, p_r) = \frac{1}{n} \sum_{i=1}^{n} \frac{\Big(I_l\big(\boldsymbol{x}_i^l\big) - \mu_l\Big)\Big(I_r\big(\boldsymbol{x}_i^r\big) - \mu_r\Big)}{\sigma_l \sigma_r}. \tag{7.30}$$

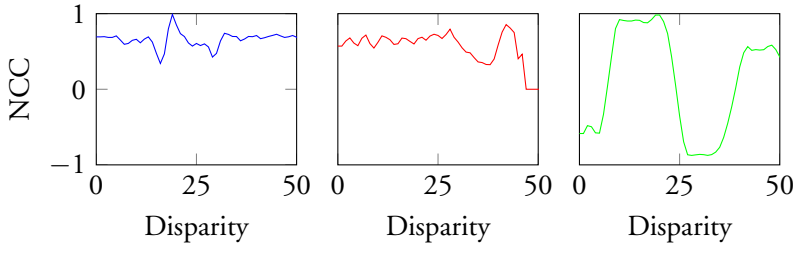
The value ranges from -1 to 1 where 1 is a perfect match. In this chapter, NCC is computed for 3×3 patches.

An extra cost to assignments of planes which are roughly parallel to the viewing rays is also added. The reason for doing this is that we are unlikely to be able to see many pixels from such planes (and if we do, the data term that we have computed using fronto-planar patches is probably not accurate). The data term is given the extra term:

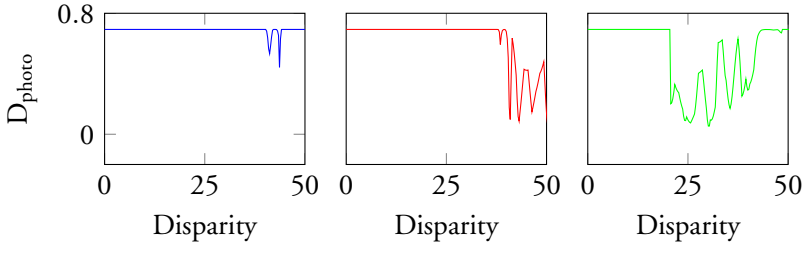
$$(1 - \boldsymbol{n}_{\boldsymbol{p}}^T \mathbf{v}_{\boldsymbol{p}})^{2k}, \tag{7.31}$$

(a) Left image

(b) Right image



(c) Normalized cross correlation, 3×3 pixel patches



(d) Pixel-wise photoconsistency, $\sigma_d = 30$

Figure 7.4: Comparing normalized cross correlation (NCC) to pixel-wise photoconsistency (D_{photo}) for a pair of stereo images. In the top row the two images are shown along with three color coded points. The two following rows show the resulting normalized cross correlation and pixel-wise photoconsistency for the three points.

where $\boldsymbol{n_p}$ is the normal of the plane assigned to $\boldsymbol{p}$ and $\mathbf{v_p}$ is the direction of the viewing ray in $\boldsymbol{p}$ (in the 3D space the viewing ray direction will be $\boldsymbol{p}/\|\boldsymbol{p}\|$ and in disparity space $(0, 0, 1)$). The constant k is selected large enough so that the penalty effects tangents with high tilt ($k = 10$ for all experiments).

The regularization term is simply modeled as

$$R_{\boldsymbol{pq}}(f) = \min\left(V_{\boldsymbol{pq}}(f), \tau\right), \tag{7.32}$$

where $\tau \in \mathbf{R}$ is a threshold introduced in order to preserve discontinuities. Now we can define the objective function as

$$\begin{aligned} E_{\text{ncc}}(f) = -\mu \sum_{\boldsymbol{p}\in\mathcal{I}} \text{NCC}(f) + (1 - \boldsymbol{n}_{\boldsymbol{p}}^T \mathbf{v}_{\boldsymbol{p}})^{2k} \\ + \sum_{\boldsymbol{p}\in\mathcal{I}} \sum_{\boldsymbol{q}\in\mathcal{N}(\boldsymbol{p})} \min\left(V_{\boldsymbol{pq}}(f), \tau\right), \end{aligned} \tag{7.33}$$

where the parameter μ controls the trade-off between the data term and the regularization.

Pixel-wise photoconsistency

This section introduces an objective function based on Woodford et al. (2009). The data terms are identical while the regularization terms are modeled to be as close as possible to regularization of Woodford et al.

Each data term is based on pixel-wise photoconsistency between two pixels, $\boldsymbol{x}_l$ and $\boldsymbol{x}_r$, which is defined as

$$\text{D}_{\text{photo}}(\boldsymbol{x}_l, \boldsymbol{x}_r) = \log(2) - \log\left(1 + \exp\left(-\frac{\|I(\boldsymbol{x}_l) - I(\boldsymbol{x}_r)\|^2}{\sigma_d}\right)\right), \tag{7.34}$$

where σ_d is a model parameter. Pixel-wise photoconsistency is cheaper to compute than normalized cross correlation as it only takes pairs of pixels into consideration, but it might be less robust to noise.

The regularization terms used by Woodford et al. is defined as

$$W(\mathcal{N})\tau \min\left(\frac{\left|S\big(\mathcal{N}\big)\right|}{\tau}, 1\right)^{\gamma}, \tag{7.35}$$

where neighborhood, $\mathcal{N}$, is a collection of, horizontal and vertical, triples $(\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{r})$ of neighboring points and $S\big(\mathcal{N}\big)$ is an approximation of second derivative defined as

$$S(\mathcal{N}) = f(\boldsymbol{p}) - 2f(\boldsymbol{q}) + f(\boldsymbol{r}). \tag{7.36}$$

$W(\mathcal{N})$ is a weight depending on a segmentation of the image. If $\mathcal{N}$ contains pixels from several segments $W(\mathcal{N})$ takes a low value, if all pixels are from the same segment it takes a high value.

To achieve a similar regularization with the proposed model the neighborhood $\mathcal{N}$ is changed to normal 4-connectivity. Now $\mathcal{N}$ consist of collection of pairs $(\boldsymbol{p}, \boldsymbol{q})$ and the second derivative is penalized using,

$$S\big((\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{r})\big) = V_{\boldsymbol{pq}} + V_{\boldsymbol{qp}}. \tag{7.37}$$

The resulting regularization term can be expressed as

$$R_{\boldsymbol{pq}}(f) = W\left(\mathcal{N}\right)\tau \min\left(\frac{V_{\boldsymbol{pq}}(f)}{\tau}, 1\right)^{\gamma}, \tag{7.38}$$

where $\gamma \in \{1, 2\}$. For all experiments in this chapter the parameters are chosen as as in Woodford et al. (2009).

The objective function can now be defined as

$$E_{\text{photo}}(f) = \sum_{\boldsymbol{p}\in\mathcal{I}} \mathrm{D}_{\text{photo}}(f) + \sum_{\boldsymbol{p}\in\mathcal{I}} \sum_{\boldsymbol{q}\in\mathcal{N}(\boldsymbol{p})} W\left(\mathcal{N}\right)\tau \min\left(\frac{V_{\boldsymbol{pq}}(f)}{\tau}, 1\right)^{\gamma}. \tag{7.39}$$

where $\gamma \in \{1, 2\}$ and $\gamma = i$ gives ℓ_i regularization.

7.4 Optimization

In this section it will be shown how to minimize (7.29) by fusing solution proposals. It will also be shown that binary fusion moves are often submodular and how to go beyond binary fusion and fuse multiple proposals simultaneously. The discussion on how to generate proposals will be postponed to the next section.

7.4.1 Binary fusion

Given a current function f, and a proposal function n, the fusion move algorithm allows pixels to change their labels from the tangents of f to the tangents of n. For the objective function in (7.29) the binary fusion function can be expressed as

$$g(\boldsymbol{x}) = \sum_{\boldsymbol{p}\in\mathcal{I}} g_{\boldsymbol{p}}(x_{\boldsymbol{p}}) + \sum_{\boldsymbol{p}\in\mathcal{I}} \sum_{\boldsymbol{q}\in\mathcal{N}(\boldsymbol{p})} g_{\boldsymbol{pq}}(x_{\boldsymbol{p}}, x_{\boldsymbol{q}}), \tag{7.40}$$

where

$$\begin{aligned} g_{\boldsymbol{p}}(x_{\boldsymbol{p}}) &= \overline{x}_{\boldsymbol{p}} D_{\boldsymbol{p}}(f) + x_{\boldsymbol{p}} D_{\boldsymbol{p}}(n), \\ g_{\boldsymbol{pq}}(x_{\boldsymbol{p}}, x_{\boldsymbol{q}}) &= \overline{x}_{\boldsymbol{p}}\overline{x}_{\boldsymbol{q}} R_{\boldsymbol{pq}}(f, f) + x_{\boldsymbol{p}}\overline{x}_{\boldsymbol{q}} R_{\boldsymbol{pq}}(n, f) \\ &\quad + x_{\boldsymbol{p}}\overline{x}_{\boldsymbol{q}} R_{\boldsymbol{pq}}(f, n) + x_{\boldsymbol{p}} a_{\boldsymbol{q}} R_{\boldsymbol{pq}}(n, n), \end{aligned} \tag{7.41}$$

where $\overline{x} = 1 - x$. A fusion move is performed by solving

$$\underset{\boldsymbol{x}\in\mathbf{B}^{|\mathcal{I}|}}{\operatorname{argmin}}\, g(\boldsymbol{x}). \tag{7.42}$$

Lemma 7.3. *The function in* (7.40) *is submodular if and only if*

$$R_{\boldsymbol{pq}}(n, n) + R_{\boldsymbol{pq}}(f, f) \leq R_{\boldsymbol{pq}}(f, n) + R_{\boldsymbol{pq}}(n, f). \tag{7.43}$$

Proof. By definition the first order term $g_{\boldsymbol{p}}$, will not influence the submodularity. For submodularity it suffices to check that

$$g_{\boldsymbol{pq}}(1, 1) + g_{\boldsymbol{pq}}(0, 0) \leq g_{\boldsymbol{pq}}(0, 1) + g_{\boldsymbol{pq}}(1, 0), \tag{7.44}$$

holds for all (p, q), which by construction holds as long as (7.43) holds. On the other hand if (7.43) does not hold, then it directly follows that

$$g_{\boldsymbol{pq}}(1, 1) + g_{\boldsymbol{pq}}(0, 0) > g_{\boldsymbol{pq}}(0, 1) + g_{\boldsymbol{pq}}(1, 0), \tag{7.45}$$

showing that (7.40) is non-submodular. □

Using this lemma it will be shown that some proposals are easy to fuse. We will begin by restricting the attention to a certain class of regularization terms, namely

$$R_{\boldsymbol{pq}}(f, n) = c_{\boldsymbol{pq}} V_{\boldsymbol{pq}}(f, n) = c_{\boldsymbol{pq}} \left| \mathcal{T}_{\boldsymbol{p}} f(\boldsymbol{q}) - f(\boldsymbol{q}) \right|, \tag{7.46}$$

where $c_{\boldsymbol{pq}} \geq 0$ is some constant.

Plane proposals

Proposition 7.4. *Assume that the proposal function n is a plane. Then the binary fusion function in* (7.40) *is submodular if we restrict the regularization term to be of the form in* (7.46).

Proof. Since n is a tangent plane we have

$$\mathcal{T}_{\boldsymbol{p}} n(\boldsymbol{q}) = n(\boldsymbol{q}), \tag{7.47}$$

And therefore $R_{\boldsymbol{pq}}(n, n) = 0$. Furthermore,

$$\begin{aligned} R_{\boldsymbol{pq}}(f, f) &= c_{\boldsymbol{pq}} \left| \mathcal{T}_{\boldsymbol{p}} f(\boldsymbol{q}) - f(\boldsymbol{q}) \right| && (7.48)\\ &= c_{\boldsymbol{pq}} \left| \mathcal{T}_{\boldsymbol{p}} f(\boldsymbol{q}) - n(\boldsymbol{q}) + \mathcal{T}_{\boldsymbol{p}} n(\boldsymbol{q}) - f(\boldsymbol{q}) \right| && (7.49)\\ &\leq c_{\boldsymbol{pq}} \left| \mathcal{T}_{\boldsymbol{p}} f(\boldsymbol{q}) - n(\boldsymbol{q}) \right| + c_{\boldsymbol{pq}} \left| \mathcal{T}_{\boldsymbol{p}} n(\boldsymbol{q}) - f(\boldsymbol{q}) \right| && (7.50)\\ &= R_{\boldsymbol{pq}}(f, n) + R_{\boldsymbol{pq}}(n, f). && (7.51) \end{aligned}$$

From this it directly follows that

$$R_{\boldsymbol{pq}}(f, f) + R_{\boldsymbol{pq}}(n, n) \leq R_{\boldsymbol{pq}}(n, f) + R_{\boldsymbol{pq}}(f, n), \tag{7.52}$$

and via Lemma 7.3 the result follows. □

General proposals

Next we derive more general sufficient conditions for submodularity of the fusion move.

Proposition 7.5. *Assume that the current solution f and the proposed solution n are both convex (or alternatively both concave) between $\boldsymbol{p}$ and $\boldsymbol{q}$. Then the binary fusion function in* (7.40) *is submodular if we restrict the regularization term to be of the form in* (7.46).

Proof. To get the result first note that if both f and n are convex then they are both bounded from below by their tangent planes:

$$\begin{aligned} f(\boldsymbol{q}) &\geq \mathcal{T}_{\boldsymbol{p}} f(\boldsymbol{q}) && (7.53)\\ n(\boldsymbol{q}) &\geq \mathcal{T}_{\boldsymbol{p}} n(\boldsymbol{q}). && (7.54) \end{aligned}$$

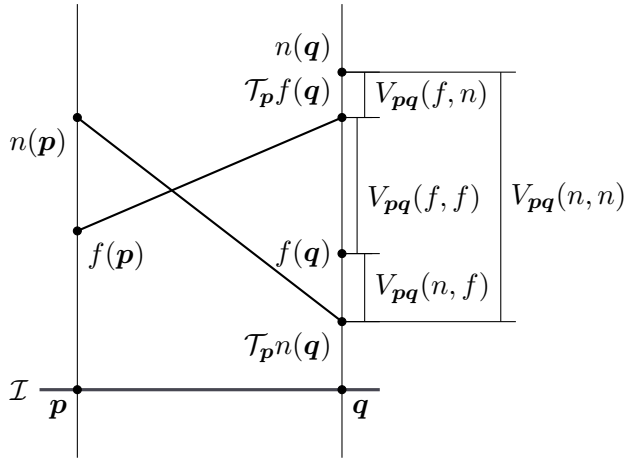


Figure 7.5: An example of a possible non-submodular term when fusing the solution f and the proposed solution n. Using the regularization term in (7.60) with $c_{\boldsymbol{pq}} > 0$, it follows that $R_{\boldsymbol{pq}}(n,n) + R_{\boldsymbol{pq}}(f,f) > R_{\boldsymbol{pq}}(f,n) + R_{\boldsymbol{pq}}(n,f)$.

Using this it follows that

$$R_{\boldsymbol{pq}}(f,f) + R_{\boldsymbol{pq}}(n,n) = \tag{7.55}$$

$$c_{\boldsymbol{pq}}\Big(f(\boldsymbol{q}) - \mathcal{T}_{\boldsymbol{p}} f(\boldsymbol{q}) + n(\boldsymbol{q}) - \mathcal{T}_{\boldsymbol{p}} n(\boldsymbol{q})\Big) \leq \tag{7.56}$$

$$c_{\boldsymbol{pq}}|n(\boldsymbol{q}) - \mathcal{T}_{\boldsymbol{p}} f(\boldsymbol{q})| + c_{\boldsymbol{pq}}|f(\boldsymbol{q}) - \mathcal{T}_{\boldsymbol{p}} n(\boldsymbol{q})| = \tag{7.57}$$

$$R_{\boldsymbol{pq}}(f,n) + R_{\boldsymbol{pq}}(n,f). \tag{7.58}$$

From this it follows that

$$R_{\boldsymbol{pq}}(f,f) + R_{\boldsymbol{pq}}(n,n) \leq R_{\boldsymbol{pq}}(n,f) + R_{\boldsymbol{pq}}(f,n), \tag{7.59}$$

and via Lemma 7.3 the result follows.

It is easy to see that the same statement is true if both n and f are concave. In this case the inequalities in (7.53) and (7.54) are switched which means that the signs of (7.56) are switched. □

Figure 7.5 shows an example where the fusion function have a non-submodular term. In this case both the functions f and n have a relatively large second-order derivative. Choosing for example f at $\boldsymbol{p}$ and n at $\boldsymbol{q}$ (or vise versa) gives a function with small second-order derivative.

Truncating the regularization

To make the regularization discontinuity preserving it is desirable to introduce threshold, $\tau > 0$, on the regularization. Extend (7.46) to

$$R_{\boldsymbol{pq}}(f, n) = c_{\boldsymbol{pq}} \min(V_{\boldsymbol{pq}}(f, n), \tau), \tag{7.60}$$

where $c_{\boldsymbol{pq}} \geq 0$. The result from Proposition 7.4 carries over to the truncated term.

Proposition 7.6. *Assume that the proposal function n is a plane. Then the binary fusion function in* (7.40) *is submodular if we restrict the regularization term to be of the form in* (7.60).

Proof. Using the same argument as in Proposition 7.4 it follows that $V_{\boldsymbol{pq}}(n, n) = 0 \implies R_{\boldsymbol{pq}}(n, n) = 0$, and

$$\begin{aligned} R_{\boldsymbol{pq}}(f, f) &= c_{\boldsymbol{pq}} \min(V_{\boldsymbol{pq}}(f, f), \tau) && (7.61)\\ &\leq c_{\boldsymbol{pq}} \min\left(V_{\boldsymbol{pq}}(f, n) + c_{\boldsymbol{pq}} V_{\boldsymbol{pq}}(n, f), \tau\right) && (7.62)\\ &\leq c_{\boldsymbol{pq}} \min\left(V_{\boldsymbol{pq}}(f, n), \tau\right) + c_{\boldsymbol{pq}} \min\left(V_{\boldsymbol{pq}}(n, f), \tau\right) && (7.63)\\ &= R_{\boldsymbol{pq}}(f, n) + R_{\boldsymbol{pq}}(n, f). && (7.64) \end{aligned}$$

From this we get

$$R_{\boldsymbol{pq}}(f, f) + R_{\boldsymbol{pq}}(n, n) \leq R_{\boldsymbol{pq}}(f, n) + R_{\boldsymbol{pq}}(n, f), \tag{7.65}$$

and via Lemma 7.3 the result follows. □

Proposition 7.5 does not carry over for (7.60). The result can fail for surfaces with large second derivative because of the added threshold.

Remark 7.7. *For ℓ_1 regularization, the assumptions made on the objective function in* (7.60) *holds for the objective function E_{ncc} in* (7.33) *and the objective function E_{photo} in* (7.39).

7.4.2 Simultaneous fusion

Binary fusion moves are limited in that they can only fuse two proposals at a time. Therefore, the end result may depend on the order that the proposals are fused. In this section we will go beyond binary fusion moves and fuse hundreds of proposals simultaneously.

Assume that pixel $\boldsymbol{p}$ is assigned the tangent of function $f_i : \mathcal{I} \rightarrow \mathbf{R}$ at $\boldsymbol{p}$ and pixel $\boldsymbol{q}$ is assigned the tangent of $f_j : \mathcal{I} \rightarrow \mathbf{R}$ at $\boldsymbol{q}$. In this section we will restrict our attention to the subset of the objective functions in (7.29), whose regularization term can be expressed as

$$R_{\boldsymbol{pq}}(f_i, f_j) = c_{\boldsymbol{pq}} \min \Big(|\mathcal{T}_{\boldsymbol{p}} f_i(\boldsymbol{q}) - f_j(\boldsymbol{q})|^{\gamma}, \tau \Big), \tag{7.66}$$

where $\gamma \in \{1, 2\}$ and $\gamma = i$ gives ℓ_i regularization.

Remark 7.8. *The assumptions made on the objective function in* (7.66) *holds for the objective function* E_{ncc} *in* (7.33) *and the objective function* E_{photo} *in* (7.39).

Given a set of proposals, $S = \{f_1, \ldots, f_m\}$, fusing all proposals simultaneously corresponds to solving the multi-label problem

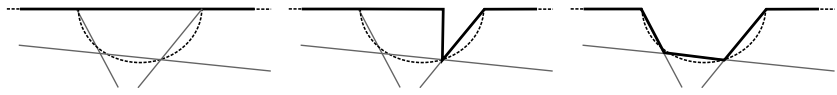
$$\underset{\boldsymbol{x} \in \boldsymbol{L}^{|\mathcal{I}|}}{\operatorname{argmin}} \sum_{\boldsymbol{p} \in \mathcal{I}} g_{\boldsymbol{p}}(x_{\boldsymbol{p}}) + \sum_{\boldsymbol{p} \in \mathcal{I}} \sum_{p \in \mathcal{N}(p)} g_{\boldsymbol{pq}}(x_{\boldsymbol{p}}, x_{\boldsymbol{q}}), \tag{7.67}$$

where $L = \{1, \ldots, m\}$ and

$$\begin{aligned} g_{\boldsymbol{p}}(x_{\boldsymbol{p}}) &= \sum_{i=1}^{m} \boldsymbol{x}_{\boldsymbol{p}}(i) D_{\boldsymbol{p}}(f_i), \\ g_{\boldsymbol{pq}}(x_{\boldsymbol{p}}, x_{\boldsymbol{q}}) &= \sum_{i=1}^{m} \sum_{j=1}^{m} \boldsymbol{x}_{\boldsymbol{p}}(i) \boldsymbol{x}_{\boldsymbol{q}}(j) R_{\boldsymbol{pq}}(f_i, f_i), \end{aligned} \tag{7.68}$$

where $\boldsymbol{x}_{\boldsymbol{p}}$ is the indicator variable for $x_{\boldsymbol{p}}$.

Why? One natural question to ask is: is it not sufficient to consider binary fusion? One reason for simultaneous fusion is that we are able to avoid local optima, as illustrated in Figure 7.6. Another advantage is that we can



(a) Start solution (b) One plane (c) Three planes

Figure 7.6: Motivational example for simultaneously fusion. a) Suppose a real surface is represented by the dashed curve, the current best solution is given by the thick black line and three proposals are shown as the three gray lines. b) A possible solution after binary fusion with one of the proposals, note that this solution incurs a large regularization cost. Because of the cost this proposal might never be successfully fused. c) If we fuse all three planes simultaneously we can jump directly to a better solution.

get guarantees on the solution. That is, given n proposals we know how far away from the optimal fusion of the n proposals the solution is. If we perform binary fusion iteratively we can only get guarantees on each fusion step but we do not know if the order of the fusion has led us into a local optima.

Efficient optimization

In this section it will be shown how (7.67) can be efficiently optimized using message-passing based methods, such as TRW-s. Each message is a vector of the same dimension as the number of possible labels. Let $m_{\boldsymbol{pq}}$ denote the message $\boldsymbol{p}$ sends to $\boldsymbol{q}$. The messages can be expressed as

$$m_{\boldsymbol{pq}}(x_{\boldsymbol{q}}) = \min_{x_{\boldsymbol{p}} \in \boldsymbol{L}} \big(g_{\boldsymbol{pq}}(x_{\boldsymbol{p}}, x_{\boldsymbol{q}}) + h(x_{\boldsymbol{p}}) \big), \tag{7.69}$$

where

$$h(x_{\boldsymbol{p}}) = D_{\boldsymbol{p}}(x_{\boldsymbol{p}}) + \sum_{\boldsymbol{s} \in \{\mathcal{N}(\boldsymbol{p}) \setminus \boldsymbol{q}\}} m_{\boldsymbol{sp}}(x_{\boldsymbol{p}}). \tag{7.70}$$

Straightforward computation of messages using the formula (7.69) takes $\mathcal{O}\big(n^2\big)$ evaluations, where n is the number of labels. This section will show how to use a generalized distance transform to compute the messages in $\mathcal{O}(n)$. In principle the labels belong to a three-dimensional space and computing distance transforms in this space may seem difficult. However, due to the fact that the interaction is defined using distances along the

viewing ray we can apply regular one-dimensional distance transforms with little modifications. The only extra step required is sorting the labels according to their disparity or depth as this can vary between points.

Efficient distance transform

In Felzenszwalb and Huttenlocher (2006) a distance transform is introduced that can handle standard stereo regularization. It is possible to modify their approach to work with the second-order stereo regularization proposed in this chapter.

We will start by simplifying (7.66) into

$$R_{\boldsymbol{pq}}(f_i, f_j) = c_{\boldsymbol{pq}} |\mathcal{T}_{\boldsymbol{p}} f_i(\boldsymbol{q}) - f_j(\boldsymbol{q})|^{\gamma}, \tag{7.71}$$

the truncation will be handled later on in this section. With slight abuse of notation, we will express the message updates as a function of the proposal as

$$m_{\boldsymbol{pq}}(f_j) = \min_{i \in \{0,\dots,n\}} R_{\boldsymbol{pq}}(f_i, f_j) + h(f_i), \tag{7.72}$$

where

$$h(f_i) = D_{\boldsymbol{p}}(f_i) + \sum_{\boldsymbol{s} \in \{\mathcal{N}(\boldsymbol{p}) \setminus \boldsymbol{q}\}} m_{\boldsymbol{sp}}(f_i) \tag{7.73}$$

Suppose that we would like to fuse the proposals, $\mathcal{S} = \{f_0, \dots, f_n\}$, for a point pair $(\boldsymbol{p}, \boldsymbol{q})$ introduce,

$$\boldsymbol{Q}_{\boldsymbol{q}} = \{f_0(\boldsymbol{q}), \dots, f_n(\boldsymbol{q})\} = \{q_0, \dots, q_n\}, \tag{7.74}$$

$$\boldsymbol{Q}'_{\boldsymbol{pq}} = \{\mathcal{T}_{\boldsymbol{p}} f_0(\boldsymbol{q}), \dots, \mathcal{T}_{\boldsymbol{p}} f_n(\boldsymbol{q})\} = \{q'_0, \dots, q'_n\}. \tag{7.75}$$

Since we are only considering points along the viewing ray, each point in the two sets are represented by their distance from the camera center.

ℓ_1 **regularization.** To compute the message $m_{\boldsymbol{pq}}$, in case of ℓ_1 regularization, we need to evaluate the function

$$\ell(f_i) = \min_{i \in \{0,\dots,n\}} c_{\boldsymbol{pq}} |q'_i - q| + h(f_i) \tag{7.76}$$

for all $f_i \in \mathcal{S}$. Each point can be used to represent a cone rooted at

$$\left(q'_i, h(f_i)\right), \tag{7.77}$$

with slope

$$c_{\boldsymbol{pq}}|q'_i - q|. \tag{7.78}$$

If we construct the lower envelope of all these cones then we can solve (7.76) in constant time, resulting in a linear time message update.

The order at which the points in $\boldsymbol{Q}'_{\boldsymbol{pq}}$ appear along the viewing ray depends on the tilt of the tangent functions. Therefore the same set of tangents at p will have different orderings for different neighboring pair (p, q). To be able to compute the distance transform and evaluate $\ell(f)$ in linear time we thus need to maintain an ordering of the labels for each neighborhood. Since the proposals and neighborhoods are fixed for each fusion problem, the sorting can be done once at start up and maintained during the optimization.

The distance transform sequentially constructs the lower envelope of the cones by first sorting $\boldsymbol{Q}'_{\boldsymbol{pq}}$ and then going through each point $q' \in \boldsymbol{Q}'_{\boldsymbol{pq}}$. In what follows, let i be the index of point q'_i and let j be the last point added to the lower envelope. For each point q'_i drawn from $\boldsymbol{Q}'_{\boldsymbol{pq}}$ one of the following three cases may occur

1. $h(f_i)$ is above the current lower envelope at q'_i.
2. $h(f_i) + c_{\boldsymbol{pq}}|q'_i - q'_j|$ is below the current lower envelope at q'_i.
3. Neither 1 nor 2.

For case 1) we can just skip point i as it will never be part of the lower envelope. For case 2) the previously added interval to the lower envelope is removed and we repeat the comparison for interval $j - 1$. In case 3) the lower envelope intersects the cone associated with point i. For ℓ_1 regularization two cones intersect when

$$q = \frac{h(f_i) - h(f_j) + c_{\boldsymbol{pq}}|q'_i + q'_j|}{2c_{\boldsymbol{pq}}}. \tag{7.79}$$

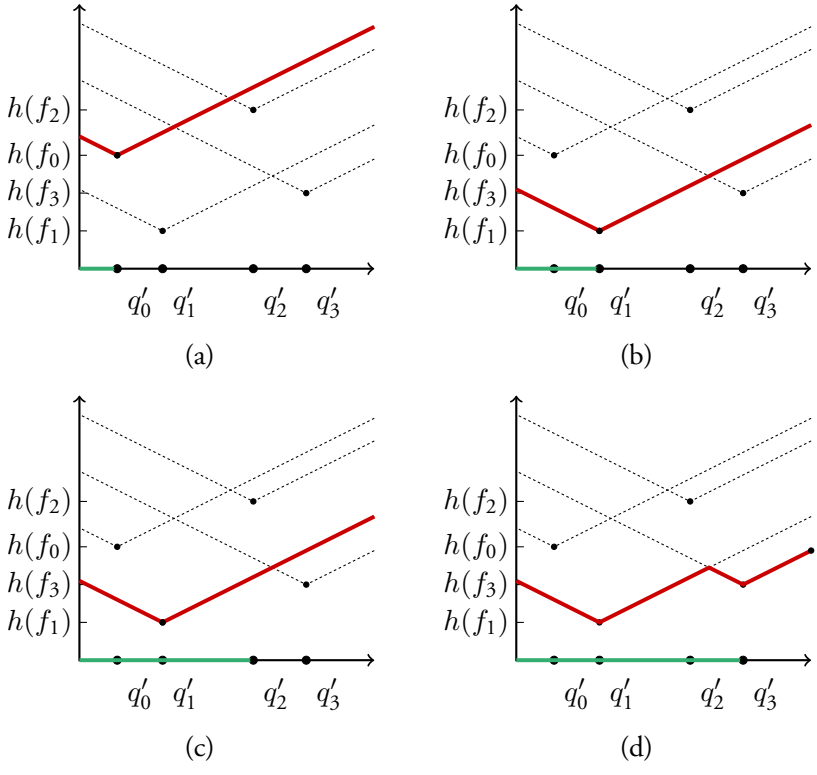


Figure 7.7: Example construction of lower envelope for ℓ_1 regularization assuming the points are ordered in ascending order. The dashed lines shows $c_{\boldsymbol{pq}}|q'_i - q| + h(f_i)$. The current estimated lower envelope is shown by a thick red line and the progress is indicated by the green line. (a) The algorithms is initialized. (b) $c_{\boldsymbol{pq}}|q'_1 - q'_0| + h(f_1) < h(f_0)$. The cone belonging to q'_0 will never be part of the lower envelope. (c) $c_{\boldsymbol{pq}}|q'_2 - q'_1| + h(f_1) < h(f_2)$. The cone belonging to q'_2 will never be part of the lower envelope. Finally in (d) the current estimate of the lower bound is intersecting with q'_3's cone. The lower envelope is divided into two intervals. One belonging to q'_1 and one to q'_3.

The intersection is calculated and the new interval is added to the lower envelope. An example lower envelope is constructed in Figure 7.7.

After the lower envelope is constructed the message cost for $m_{\boldsymbol{pq}}(f_j)$ is just the height of the lower envelope at position q_j. When using standard stereo regularization the sets $\boldsymbol{Q}$ and $\boldsymbol{Q}'$ are usually the same for all pixels. When using regularization proposed in this chapter, they most likely differ.

A final thing to consider is when we add truncation to the regularization term resulting in

$$R_{\boldsymbol{pq}}(f_i, f_j) = c_{\boldsymbol{pq}} \min\left(|\mathcal{T}_{\boldsymbol{p}} f_i(\boldsymbol{q}) - f_j(\boldsymbol{q})|, \tau\right), \tag{7.80}$$

where $\tau \in \mathbf{R}$ is some truncation level. The highest cost any message can have is

$$\rho = \min_{f_i \in \mathcal{S}} h(f_i) + c_{\boldsymbol{pq}}\tau. \tag{7.81}$$

Adding the truncation is now easy. First ignore the truncation, and calculate the message cost as if there is no truncation, call these message $m'_{\boldsymbol{pq}}$. The message cost with the truncation can now be calculated as

$$m_{\boldsymbol{pq}}(f_i) = \min(m'_{\boldsymbol{pq}}(f_i), \rho) \quad \text{for all } f_i \in \mathcal{S}. \tag{7.82}$$

Finding ρ and updating all messages requires $\mathcal{O}(n)$ calculations. Pseudocode for the algorithm is given in Algorithm 3.

ℓ_2 regularization. The only modification needed is related to the construction of the lower envelope, which is described in Felzenszwalb and Huttenlocher (2006). The rest is carried over from the ℓ_1 regularization algorithm. For ℓ_2 regularization the lower envelope consists of parabolas of the form $c_{\boldsymbol{pq}}(q'_i - q)^2 + h(f_i)$. The lower envelope is sequentially constructed by considering intersection of the parabolas, which for parabolas (i, j) is given by

$$q = \frac{\left(h(f_i) + c_{\boldsymbol{pq}}(q'_i)^2\right) - \left(h(f_j) + c_{\boldsymbol{pq}}(q'_j)^2\right)}{2c_{\boldsymbol{pq}}\left(q'_i - q'_j\right)}. \tag{7.83}$$

Let j be the last added point to the lower envelope and let z_j be the first point in the interval where it is part of the lower envelope. For each

point i drawn from $\boldsymbol{Q}'_{\boldsymbol{pq}}$ and as long as $q'_i \neq q'_j$, consider the intersection of parabolas i and j, denote their intersection q. One of the following two cases occur

1. $q > z_j$. The intersection is added to the lower envelope.
2. $q \leq z_j$. Repeat intersection with parabola $j - 1$.

When $q'_i = q'_j$, the parabolas do not intersect and the parabola with smallest $h(f_i)$ is kept in the lower envelope. This case is not covered in Felzenszwalb and Huttenlocher (2006) and correspond to several proposals intersecting the q-ray at the same point. This occurs frequently in practice.

Complexity of the message updates for ℓ_1 and ℓ_2 regularization. Let n be the number of proposals. The sorting of points is done only once and outside of the message update. For the lower envelope, the maximum number of intervals is $n + 1$. Each interval is visited at most 3 times. Once when it is added, once when it is intersected and once when it is removed. All other parts of the algorithm are at most single nested loops of size n. Hence the complexity for each message update is $\mathcal{O}(n)$.

Algorithm 3 Message update for ℓ_1 regularization.

function DISTANCE(x, y)
  return $c_{\boldsymbol{pq}}|x - y|$

function UPDATE_INTERVALS(interval, start, intersection)
  v[interval] $\leftarrow$ start ▷ Label belonging to this interval.
  z[interval] $\leftarrow$ intersection ▷ Lower limit of this interval.
  z[interval + 1] $\leftarrow \infty$ ▷ Upper limit of this interval.

Precondition: $\boldsymbol{Q}$ and $\boldsymbol{Q}'$ are sorted.
function MESSAGE_UPDATE($\boldsymbol{Q}, \boldsymbol{Q}', c_{\boldsymbol{pq}}, \tau$)
  UPDATE_INTERVALS$(0, 0, -\infty)$
  `maxVal` $\leftarrow \infty$
  for $i \leftarrow 1$ **to** $n - 1$ **do**
    `maxVal` $\leftarrow \min($`maxVal`$, h(f_i))$
    $j \leftarrow 0$ ▷ Number of intervals in the lower envelope.
    for $l = i$ **to** 0 **do**
      if DISTANCE $\left(q'_i, q'_j\right) + h_{v[j]} \leq h(f_i)$ **then** ▷ Case 1
        break
      else if DISTANCE $\left(q'_i, q'_j\right) + h(f_i) < h_{v[j]}$ **then** ▷ Case 2
        if j = 0 **then**
          UPDATE_INTERVALS$(0, i, -\infty)$
        else
          $j \leftarrow j - 1$
      else ▷ Case 3
        $j \leftarrow j + 1$
        intersection $\leftarrow \frac{(h(f_i) - h(f_j) + c_{\boldsymbol{pq}}(q'_i + q'_j)}{2c_{\boldsymbol{pq}}}$
        UPDATE_INTERVALS $(j, i,$ intersection$)$
        break
  $j \leftarrow 0$
  `maxVal` $\leftarrow$ `maxVal` $+ c_{\boldsymbol{pq}}\tau$. ▷ Largest message cost.
  for $i \leftarrow 1$ **to** $n - 1$ **do**
    while $z[j + 1] < q_i$ **do**
      $j \leftarrow j + 1$
    $\boldsymbol{m}[i] \leftarrow \min\left(\text{DISTANCE}\left(q_i, q'_j\right) + h_{v[j]}, \texttt{maxVal}\right)$
  return $\boldsymbol{m}$

Algorithm 4 Message update for ℓ_2 regularization.

function DISTANCE(x, y)
  return $c_{\boldsymbol{pq}}(x-y)^2$

function UPDATE_INTERVALS(interval, start, intersection)
  v[interval] $\leftarrow$ start ▷ Label belonging to this interval.
  z[interval] $\leftarrow$ intersection ▷ Lower limit of this interval.
  z[interval $+ 1$] $\leftarrow \infty$ ▷ Upper limit of this interval.

Precondition: $\boldsymbol{Q}$ and $\boldsymbol{Q}'$ are sorted.

function MESSAGE_UPDATE($\boldsymbol{Q}, \boldsymbol{Q}', c_{\boldsymbol{pq}}, \tau$)
  UPDATE_INTERVALS$(0, 0, \infty)$
  `maxVal` $\leftarrow \infty$
  for $i \leftarrow 1$ **to** $n-1$ **do**
    `maxVal` $\leftarrow \min($`maxVal`$, h(f_i))$
    $j \leftarrow 0$ ▷ Number of intervals in the lower envelope.
    for $l = i$ **to** 0 **do**
      if $q'_j = q'_i$ **then**
        if $h(f_j) > h(f_i)$ **then**
          if $j = 0$ **then**
            UPDATE_INTERVALS$(0, i, \infty)$
          else
            $j \leftarrow j - 1$
        else
          break ▷ Parabola always above the lower envelope.
      else
        $j \leftarrow j + 1$
        intersection $\leftarrow \frac{\left(h(f_i)+c_{\boldsymbol{pq}}q_i'^2\right)-\left(h(f_j)+c_{\boldsymbol{pq}}q_j'^2\right)}{2c_{\boldsymbol{pq}}(q'_i-q'_j)}$
        UPDATE_INTERVALS$(j, i,$ intersection$)$
        break
  $j \leftarrow 0$
  `maxVal` $\leftarrow$ `maxVal` $+ c_{\boldsymbol{pq}}\tau$. ▷ Largest message cost.
  for $i \leftarrow 1$ **to** $n-1$ **do**
    while $z[j+1] < q_i$ **do**
      $j \leftarrow j + 1$
    $\boldsymbol{m}[i] \leftarrow \min\left(\text{DISTANCE}\left(q_i, q_j\right) + h_{v[j]}, \texttt{maxVal}\right)$
  return $\boldsymbol{m}$

7.5 Proposal generation

Both binary and simultaneous fusion are dependent on good proposals to yield good results. In this section we will give a number of different methods to generate proposals.

A useful and simple proposal is to choose f to correspond to tangent planes with constant depth or disparity. In Woodford et al. (2009) these proposals are given the abbreviation SameUni. For regular stereo regularization with one-dimensional labels these proposals corresponds to the constant proposals used by α-expansion in Boykov, Veksler, et al. (2001).

Another popular method is to generate tangent planes. Given a small set of points in a region estimate the best 3D points by considering only the data term. For most data terms this is rather inexpensive. From these points it possible to generate tangent planes using RANSAC. In Woodford et al. (2009) the regions are determined by over segmenting the image using 14 different settings and fitting a plane inside each region, resulting in 14 proposals called SegPln. It straight forward to generate higher-order surface using this same method.

In Woodford et al. (2009) proposals are generated by smoothing the current best solution, these proposals are called Smooth. For the proposed model its possible to emulate this by averaging the normal directions among the tangent planes. The next section will introduce a new of set proposals.

7.5.1 Local refinement

A usual approach for improving discrete solutions is the perform some sort of local optimization. In essence this is what the Smooth proposals are trying to accomplish. In this section we will generate proposals using alternating direction method of multipliers (ADMM) which can be initialized from any solution. Any proposals generated this way can be seen as a local refinement of the current solution. Note that we still use these proposal as any other proposal and fuse them. In this way we can guarantee that our objective function is not increasing, even though the ADMM optimization have no guarantees for non-convex functions.

We will begin to restrict our attention to the subset of the objective functions in (7.29), whose regularization terms can be expressed as

$$R_{\boldsymbol{pq}}(z) = c_{\boldsymbol{pq}} \min\Big(|\mathcal{T}_{\boldsymbol{p}} z(\boldsymbol{q}) - z(\boldsymbol{q})|^{\gamma}, \tau\Big), \tag{7.84}$$

where $\tau \in \mathbf{R}$ is some threshold level, $c_{\boldsymbol{pq}} \in \mathbf{R}$ some weight, and $\boldsymbol{\gamma} \in \mathbf{R}_+$. Furthermore, we assume that $R_{\boldsymbol{pq}}$ is piecewise differentiable.

In this section we will only cover regular cameras. This allows us to introduce

$$W_{\boldsymbol{pq}}(z) = \mathcal{T}_{\boldsymbol{p}} z(\boldsymbol{q}) - z(\boldsymbol{q}) \ = \frac{z(\boldsymbol{p})^2}{z(\boldsymbol{p}) - s z'_{\mathbf{v}}(\boldsymbol{p})} - z(\boldsymbol{q}). \tag{7.85}$$

The objective function in (7.29) can now be expressed as

$$\sum_{\boldsymbol{p} \in \mathcal{I}} D_{\boldsymbol{p}}(z) + \sum_{\boldsymbol{p} \in \mathcal{I}} \sum_{\boldsymbol{q} \in \mathcal{N}(\boldsymbol{p})} R_{\boldsymbol{pq}}\left(W_{\boldsymbol{pq}}(z)\right). \tag{7.86}$$

where the regularization terms are given by

$$R_{\boldsymbol{pq}}(W_{\boldsymbol{pq}}(z)) = c_{\boldsymbol{pq}} \min(|W_{\boldsymbol{pq}}|^{\gamma}, \tau). \tag{7.87}$$

Furthermore, we will also assume that data term, $D_{\boldsymbol{p}}$, can be densely sampled such that its minima can be found by simply searching all the sample points.

Remark 7.9. *The assumptions made on the objective function in* (7.86) *holds for the objective function* E_{ncc} *in* (7.33) *and the objective function* E_{photo} *in* (7.39).

Optimizing (7.86) is typically very challenging since the data term is often non-differentiable with lots of local minima. In addition, the regularization term is a sum of non-convex functions. To handle these problems the terms are decoupled by two new sets of variables; $\boldsymbol{x} \in \mathbf{R}^m$ and $\boldsymbol{y} \in \mathbf{R}^{|\mathcal{I}|}$, where m is the number of binary terms. These variables are constrained to be

$$x_{\boldsymbol{pq}} = W_{\boldsymbol{pq}}(z) \qquad \text{for all } \boldsymbol{p} \in \mathcal{I}, \boldsymbol{q} \in \mathcal{N}(\boldsymbol{p}), \tag{7.88}$$

$$y_{\boldsymbol{p}} = z(\boldsymbol{p}) \qquad \text{for all } \boldsymbol{p} \in \mathcal{I}. \tag{7.89}$$

The augmented Lagrangian is now

$$\begin{aligned} L(\boldsymbol{x}, \boldsymbol{y}, z, \boldsymbol{\lambda}) = &\sum_{\boldsymbol{p}\in\mathcal{I}} \sum_{\boldsymbol{q}\in\mathcal{N}(\boldsymbol{p})} R_{\boldsymbol{pq}}(x_{\boldsymbol{pq}}) \\ &+ \sum_{\boldsymbol{p}\in\mathcal{I}} \sum_{\boldsymbol{q}\in\mathcal{N}(\boldsymbol{p})} \lambda_{\boldsymbol{pq}}(x_{\boldsymbol{pq}} - W_{\boldsymbol{pq}}(z)) \\ &+ \sigma \sum_{\boldsymbol{p}\in\mathcal{I}} \sum_{\boldsymbol{q}\in\mathcal{N}(\boldsymbol{p})} (x_{\boldsymbol{pq}} - W_{\boldsymbol{pq}}(z))^2 \\ &+ \sum_{\boldsymbol{p}\in\mathcal{I}} \left(\lambda_{\boldsymbol{p}}(y_{\boldsymbol{p}} - z(\boldsymbol{p})) + \sigma(y_{\boldsymbol{p}} - z(\boldsymbol{p}))^2\right) \\ &+ \sum_{\boldsymbol{p}\in\mathcal{I}} D_{\boldsymbol{p}}(y_{\boldsymbol{p}}). \end{aligned} \tag{7.90}$$

When applying ADMM we get three subproblems:

$$\underset{x_{\boldsymbol{pq}}}{\text{minimize}}\ R_{\boldsymbol{pq}}(x_{\boldsymbol{pq}}) + \lambda_{\boldsymbol{pq}}(x_{\boldsymbol{pq}} - W_{\boldsymbol{pq}}(z)) + \sigma(x_{\boldsymbol{pq}} - W_{\boldsymbol{pq}}(z))^2, \tag{7.91}$$

$$\begin{aligned} \underset{z}{\text{minimize}}\quad \sum_{\boldsymbol{p}\in\mathcal{I}} \Bigg(&\sum_{\boldsymbol{q}\in\mathcal{N}(\boldsymbol{p})} \lambda_{\boldsymbol{pq}}(x_{\boldsymbol{pq}} - W_{\boldsymbol{pq}}(z)) \\ &+ \sigma(x_{\boldsymbol{pq}} - W_{\boldsymbol{pq}}(z))^2 \\ &+ \left(\lambda_{\boldsymbol{p}}(y_{\boldsymbol{p}} - z(\boldsymbol{p})) + \sigma(y_{\boldsymbol{p}} - z(\boldsymbol{p}))^2\right) \Bigg), \end{aligned} \tag{7.92}$$

and

$$\underset{y_{\boldsymbol{p}}}{\text{minimize}}\quad \lambda_{\boldsymbol{p}}(y_{\boldsymbol{p}} - z(\boldsymbol{p})) + \sigma(y_{\boldsymbol{p}} - z(\boldsymbol{p}))^2 + D_{\boldsymbol{p}}(y_{\boldsymbol{p}}). \tag{7.93}$$

In addition we obtain the dual update rules

$$\lambda_{\boldsymbol{pq}}^{k+1} = \lambda_{\boldsymbol{pq}}^{k} + \sigma(x_{\boldsymbol{pq}} - W_{\boldsymbol{pq}}(z)), \tag{7.94}$$

$$\lambda_{\boldsymbol{p}}^{k+1} = \lambda_{\boldsymbol{p}}^{k} + \sigma(y_{\boldsymbol{p}} - z(\boldsymbol{q})), \tag{7.95}$$

see Boyd, Parikh, et al. (2011) for details. This decoupling of terms has the following positive effects: The terms $R_{\boldsymbol{pq}}(x_{\boldsymbol{pq}})$ and $D_{\boldsymbol{p}}(y_{\boldsymbol{p}})$, that are

non-smooth and difficult to approximate locally, end up in two different subproblems both of which are separable and where optimization can be carried out over individual pixels separately, greatly reducing the search space. The coupling between pixels appears in problem (7.92) where the involved functions are smooth and can be optimized locally using standard descent methods. In the following subsections, solution strategies for each individual problem is outlined.

Problem (7.91)

To solve (7.91) first note that the optimum must be in either a stationary point or in a transition between differentiable segments of the function $R_{\boldsymbol{pq}}$. Since we will be using $R_{\boldsymbol{pq}}(x_{\boldsymbol{pq}}) = \min(|x_{\boldsymbol{pq}}|, \tau)$ in the experiments we illustrate the process using this choice. There are four cases:

1. $|x_{\boldsymbol{pq}}| > \tau$. Taking derivatives of (7.91) gives

$$\lambda_{\boldsymbol{pq}} + 2\sigma(x_{\boldsymbol{pq}} - W_{\boldsymbol{pq}}(z)) = 0. \tag{7.96}$$

   Solving for $x_{\boldsymbol{pq}}$ gives the stationary point. Note that the solution may violate $|x_{\boldsymbol{pq}}| > \tau$. In this case the solution is false and there is no stationary point in the interval. However, since we compare the value of all "candidate" minimizers we do not have to test for this. We are guaranteed that one of the candidates is the global minimizer of (7.91).

2. $-\tau < x_{\boldsymbol{pq}} < 0$. In this case the stationary point is given by

$$-1 + \lambda_{\boldsymbol{pq}} + 2\sigma(x_{\boldsymbol{pq}} - W_{\boldsymbol{pq}}(z)) = 0. \tag{7.97}$$

3. $0 < x_{\boldsymbol{pq}} < \tau$. Here the stationary point is given by

$$1 + \lambda_{\boldsymbol{pq}} + 2\sigma(x_{\boldsymbol{pq}} - W_{\boldsymbol{pq}}(z)) = 0. \tag{7.98}$$

4. In addition we need to test the two transition points $x_{\boldsymbol{pq}} = \pm\tau$ and $x_{\boldsymbol{pq}} = 0$.

Problem (7.92)

The objective function in (7.92) is similar to non-linear least squares problem. We will apply a Levenberg-Marquardt approach to solve it. We linearize the residual

$$x_{\boldsymbol{pq}} - \left(\frac{z(\boldsymbol{p})^2}{z(\boldsymbol{p}) - s z'_{\mathbf{v}}(\boldsymbol{p})} - z(\boldsymbol{q}) \right). \tag{7.99}$$

Note that $y_{\boldsymbol{p}} - z(\boldsymbol{q})$ is already linear in terms of the unknowns ($z(\boldsymbol{p})$,$z(\boldsymbol{q})$ and $z'_{\mathbf{v}}(\boldsymbol{p})$), and does therefore not require modification. The approximation that we get from the linearization is in most cases very accurate. This can be heuristically explained by looking at the equivalent expression

$$x_{\boldsymbol{pq}} - \left(z(\boldsymbol{p}) + s z'_{\mathbf{v}}(\boldsymbol{p}) + \frac{(s z'_{\mathbf{v}}(\boldsymbol{p}))^2}{z(\boldsymbol{p}) - s z'_{\mathbf{v}}(\boldsymbol{p})} - z(\boldsymbol{q}) \right). \tag{7.100}$$

Assuming that s is small (recall that this is the distance between pixels) the nonlinear term is likely to be neglectable for reasonable values of the derivative $z'_{\mathbf{v}}(\boldsymbol{p})$.

Problem (7.93)

Since the function $D_{\boldsymbol{p}}$ is one-dimensional and sampled densely it is easy to optimize it by simply searching the sample values. To solve (7.93) we simply recompute the samples of $D_{\boldsymbol{p}}$ by adding

$$\lambda_{\boldsymbol{p}}(y_{\boldsymbol{p}} - z(\boldsymbol{p})) + \sigma(y_{\boldsymbol{p}} - z(\boldsymbol{p}))^2 \tag{7.101}$$

to $D_{\boldsymbol{p}}(y_{\boldsymbol{p}})$ and chose the optimum as the best new sample.

7.6 Experiments

For all experiments presented in this chapter a few hundred proposals are generated using the methods described in Section 7.5. The proposals are then fused using binary fusion if nothing else is stated.

The results are presented either using *surface plots* or *disparity maps*. In a surface plot the estimated tangent planes are used to build a surface representation of the scene. In a disparity map, each pixel in the reference image is encoded by its estimated disparity. To enhance the contrast in the disparity maps, the jet colormap, given in Figure 7.8, is used.

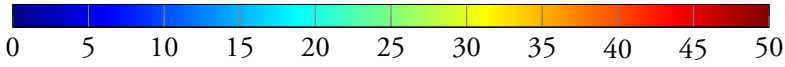


Figure 7.8: The jet colormap used to display disparity maps in this section.

7.6.1 Rectified cameras

Second-order regularization

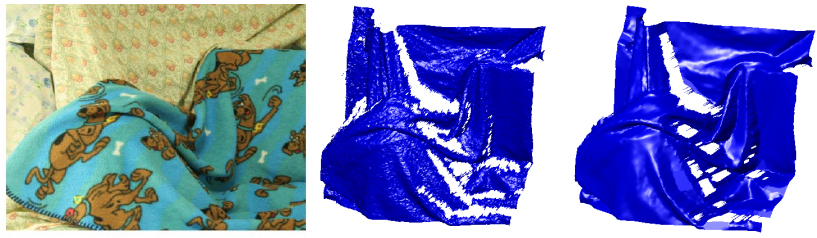
The first experiment uses the objective function E_{ncc} in (7.33) on the Middlebury dataset from Scharstein and Szeliski (2003)). The dataset consists of several image sequences. The name of each image sequence will be given in small caps. Normalized cross correlations are calculated using all images in each sequence. Figures 7.9 and 7.10 show the resulting surface plots for two different sequences. The effects of the regularization term can be seen by comparing the surface generated from the data term without regularization (b) and the one with regularization (c). The data term is particularly weak in the BOWLING sequence because of the large textureless region.

In Figure 7.11 the disparity map for the proposed second-order stereo regularizer is compared to a standard first-order stereo regularizer. (The difference between the regularization models is shown in Figure 7.2.) Both methods use the data term of the objective function E_{photo}, defined in (7.39), and the same parameters to define the regularization weights. The resulting disparity map for the proposed second-order stereo regularizer is clearly better. More examples, highlighting the quality gains using second-order stereo regularizers are given in Woodford et al. (2009).

Comparison to Woodford et al. (2009)

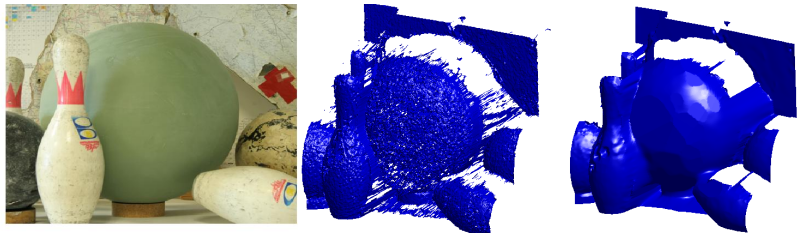
In this experiment the objective function E_{photo}, as defined in (7.39), is compared to the objective function used by Woodford et al. (2009), which will simply be referred to as *woodford*. For each binary fusion move, "improve" is used to label all unlabeled variables after using roof duality, as discussed in Woodford et al. (2009).

In Figure 7.12 both objective functions are initialized with the same random solution with fronto-parallel tangent planes. After that, each SegPln proposal is fused one at a time for each objective function. For each binary



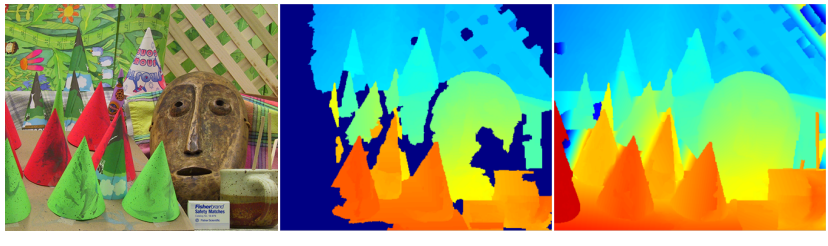
(a) Reference image (b) Only data term (c) Using regularization

Figure 7.9: CLOTH. Estimated surfaces using the objective function E_{ncc} with $\mu = 40$. In (b) no regularization is used ($\tau = 0$) and in (c) the proposed regularization is added ($\tau = 1$).



(a) Reference image (b) Only data term (c) Using regularization

Figure 7.10: BOWLING. Estimated surfaces using the objective function E_{ncc} with $\mu = 40$. In (b) no regularization is used ($\tau = 0$) and in (c) the proposed regularization is added ($\tau = 1$).



Reference image Standard stereo regularization Proposed stereo regularization

Figure 7.11: CONES. Comparing standard first-order stereo regularization to the proposed second-order stereo regularization. The disparity maps are estimated by fusing the 14 SegPln proposals.

fusion move, the number of unlabeled variables for roof duality is counted and given in Table 7.2.

Note that the binary fusion move, for the proposed method, is only submodular if each proposal function is *one* tangent plane, see Proposition 7.6. The SegPln proposals are piecewise planar, hence the regularization at transitions between tangent planes may not be submodular.

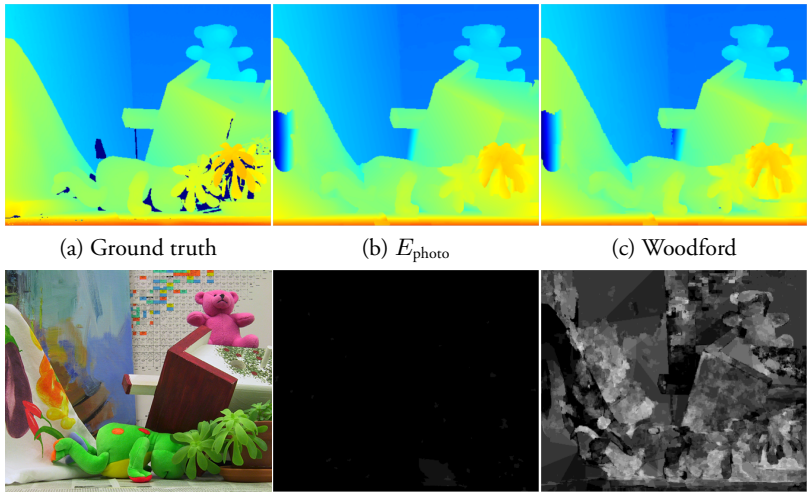
The proposed regularization is also tested on the full pipeline of Woodford et al. (2009) which uses three types of proposals (SegPln, SameUni and Smooth). The results on all of the sequences in Middlebury (Scharstein and Szeliski (2003)) are given in Table 7.3 and the execution times are given in Table 7.4.

Simultaneous fusion

This experiment demonstrates the usefulness of performing simultaneous fusion compared to iterative binary fusion. To quantitatively evaluate simultaneous fusion the full Middlebury stereo dataset is used (Hirschmüller and Scharstein (2007); Scharstein and Pal (2007); Scharstein and Szeliski (2002); Scharstein and Szeliski (2003)), consisting of 40 stereo image pairs. The objective function E_{photo} in (7.39) is used for all experiments. For every sequence, the SegPln proposals and 300 equally distanced fronto-parallel proposals are generated.

For iterative binary fusion all proposals are iteratively fused, using roof duality in a random order. This is repeated until all proposals have failed to update the solution in a fusion move. The simultaneous fusion move is optimized using TRW-s for either 3000 iterations or until the relative duality gap is less than 0.001. Simultaneous fusion is expected to be slower than iterative binary fusion, since it solves a much harder problem. To make the comparison fair, the extra time for iterative binary fusion is spent fusing random tangent planes with constant disparity. The final solution is called *extra*.

Quantitative results for ℓ_1 regularization are given in Table 7.5. Note that the ground truth lacks normal directions; it is not possible to calculate the objective value for the ground truth disparities. An example highlighting the quality difference between simultaneous and iterative binary fusion is given in Figure 7.13. The difference is larger when the number of proposals is low. When the number of proposals is very large the binary fusion



(a) Ground truth (b) E_{photo} (c) Woodford

(d) Reference image (e) Unlabeled E_{photo} (f) Unlabeled Woodford

Figure 7.12: TEDDY. (b-c) Disparity maps after fusing the SegPln proposals. (e-f) The unlabeled variables summed over all binary fusion moves. A white pixel corresponds to failed fusion for every single proposal.

	TSUKUBA	VENUS	TEDDY	CONES
E_{photo}	0.065%	0.0264%	0.127%	0.0847%
Woodford	30.0%	30.6%	27.6%	27.3%

Table 7.2: The percentage of variables unlabeled after binary fusion for the SegPln proposals.

	TSUKUBA			VENUS			
	Non occ	All	Disc	Non occ	All	Disc	
E_{photo}	4.49	5.52	12.3	0.298	0.648	3.99	
Woodford	4.83	5.99	13.9	0.536	0.921	6.39	
	TEDDY			CONES			Average
	Non occ	All	Disc	Non occ	All	Disc	
E_{photo}	7.71	11.2	17.8	9.78	15.4	18.3	8.95
Woodford	8.16	11.8	19.3	9.74	15.6	18.4	9.63

Table 7.3: Scores on Middlebury using the same proposals, lower is better. All values are % of pixels being ≥ 1 pixel incorrect for each of the three classes. The classes are **non occ**luded regions, **all** pixels and regions near depth **disc**ontinuities.

	TSUKUBA	VENUS	TEDDY	CONES	Average
E_{photo}	21.3	25.5	29.4	36.5	28.2
Woodford	106	139	143	181.0	142
Ratio	4.96	5.47	4.87	4.96	5.07

Table 7.4: Execution time in seconds, using the convergence criteria in Woodford et al. (2009).

performs surprisingly well for many of the sequences. The resulting disparity maps are almost as good as those of simultaneous fusion, which can be verified to be near optimal, due to the lower bound being almost tight. For some sequences the lower objective value clearly improves the resulting disparity map, see Figure 7.14. For other sequences the improvement is more subtle, see Figure 7.15.

Experiments are also performed using ℓ_2 regularization, quantitative results are given in Table 7.6. Examples of improvement can be seen in Figures 7.16 and 7.17.

The quality of the solution for iterative and simultaneous fusion as a function of execution time is given in Figure 7.18. Iterative binary fusion achieves lower objective values faster, but in the long run simultaneous fusion produces the best solutions.

7.6.2 Regular cameras

In this section the depth functions for two non-rectified image sequences are estimated, using the objective function E_{ncc} in (7.33). For the specific choices of μ and τ see the figure captions. The results are given in Figures 7.19 and 7.20. Both of these sequences are part of a real outdoor dataset, and as a preprocessing step the background sky is removed. In both cases, nine neighboring images are used to compute the normalized cross correlations.

Local refinement

This experiment highlights the usefulness of having proposals generated by local refinement. Two approaches are compared; fusion moves with sampled planar proposals with and without local refinement. Since the problem is not convex, convergence of ADMM is not guaranteed for fixed σ. Therefore, σ starts at a low value (0.1 in the implementation) and is slowly increased in each iteration to a high value (10 in the implementation). The specific update rule is

$$\sigma^{k+1} = \eta\sigma^k, \tag{7.102}$$

where η is determined such that σ^k is 10 in the last iteration.

In Figure 7.21 the result is given for the smooth and highly textured surfaces of ÉGLISE DU DÔME, and in Figure 7.22 the result is given for the

Sequence	Iterative (Iter)		Simultaneous (Sim)			Sim / Iter		Sim/extra
	Iterations	Time	Iterations	Time	Rel. gap	Obj.	Time	Obj.
ALOE	3278	16.4	3000	358.8	0.001	0.994	21.83	0.996
ART	3546	23.1	3000	384.5	0.002	0.992	16.67	0.994
BABY1	2458	13.0	3000	342.3	0.036	1.006	26.25	1.006
BABY2	2862	14.8	3000	342.1	0.016	0.988	23.11	0.989
BABY3	3045	17.3	3000	361.7	0.021	0.976	20.94	0.976
BARN1	4353	22.2	3000	370.2	0.000	0.999	16.66	0.999
BARN2	2897	14.6	3000	366.5	0.001	0.997	25.06	0.997
BOOKS	3683	21.3	3000	383.7	0.014	0.996	18.00	0.998
BOWLING1	2126	14.8	3000	345.4	0.009	0.992	23.30	0.993
BOWLING2	3028	17.0	3000	356.1	0.002	0.995	20.94	0.996
BULL	3896	19.2	1302	163.8	0.000	0.997	8.52	0.997
CLOTH1	3614	17.3	3000	341.0	0.001	0.989	19.76	0.991
CLOTH2	3356	18.7	3000	352.4	0.020	0.988	18.87	0.990
CLOTH3	2967	14.0	3000	341.3	0.001	0.978	24.36	0.980
CLOTH4	3940	20.6	3000	353.9	0.001	0.988	17.20	0.990
COMPUTER	3382	20.1	3000	363.1	0.002	0.992	18.09	0.994
CONES	4180	23.6	3000	373.9	0.007	1.002	15.83	1.002
DOLLS	3219	19.4	3000	375.7	0.003	0.993	19.33	0.995
DRUMSTICKS	2844	17.9	3000	379.6	0.014	1.007	21.26	1.009
DWARVES	3388	19.3	3000	369.2	0.026	1.012	19.14	1.014
FLOWERPOTS	3103	16.8	3000	356.3	0.023	0.993	21.21	0.994
LAMPSHADE1	2498	15.9	1425	173.7	0.000	0.976	10.91	0.976
LAMPSHADE2	1935	13.1	3000	352.7	0.011	0.991	26.87	0.992
LAUNDRY	4047	24.1	3000	365.4	0.006	0.995	15.17	0.998
MAP	2704	4.5	3000	133.9	0.001	0.999	29.64	0.999
MIDD1	3627	36.0	3000	378.4	0.013	0.972	10.52	0.973
MIDD2	3082	35.8	3000	371.2	0.002	0.980	10.38	0.981
MOEBIUS	4261	26.7	3000	377.3	0.003	0.990	14.12	0.992
MONOPOLY	3577	47.3	3000	359.9	0.009	1.004	7.60	1.004
PLASTIC	2522	20.3	1551	181.6	0.000	0.980	8.93	0.980
POSTER	2798	15.1	3000	375.3	0.008	1.003	24.83	1.003
REINDEER	3708	23.3	3000	369.0	0.034	1.009	15.82	1.011
ROCKS1	2578	13.6	3000	351.5	0.011	0.989	25.86	0.990
ROCKS2	3365	16.1	3000	353.3	0.001	0.965	21.93	0.966
SAWTOOTH	3302	16.2	3000	369.8	0.003	0.999	22.79	1.000
TEDDY	4386	23.8	3000	379.6	0.012	1.006	15.97	1.007
TSUKUBA	2562	9.0	3000	246.9	0.001	0.995	27.41	0.996
VENUS	2697	14.2	3000	373.3	0.000	0.996	26.21	0.997
WOOD1	2555	16.2	914	123.5	0.000	0.989	7.63	0.989
WOOD2	2958	21.0	3000	355.8	0.000	1.000	16.98	1.000
Geom. mean	3150.9	18.1	2753.6	322.9	0.002	0.993	17.83	0.994

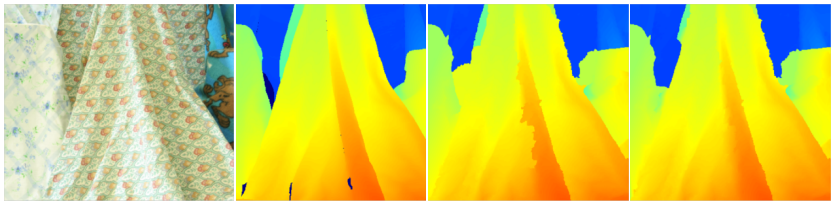
Table 7.5: Experiments on the Middlebury dataset using ℓ_1 regularization. Execution time is measured in minutes.

Sequence	Iterative (Iter)		Simultaneous (Sim)			Sim / Iter		Sim/extra
	Iterations	Time	Iterations	Time	Rel. gap	Obj.	Time	Obj.
ALOE	5113	26.4	3000	391.7	0.003	0.992	14.84	0.995
ART	4675	28.0	3000	421.4	0.075	1.054	15.04	1.058
BABY1	5776	30.2	3000	375.3	0.012	0.992	12.42	0.995
BABY2	4401	23.5	3000	369.5	0.050	0.967	15.75	0.971
BABY3	4382	24.8	3000	393.8	0.025	0.979	15.85	0.983
BARN1	4581	24.0	3000	406.2	0.001	0.998	16.93	0.999
BARN2	4159	21.8	3000	394.7	0.016	1.012	18.12	1.013
BOOKS	4931	29.6	3000	419.5	0.025	0.988	14.18	0.992
BOWLING1	4200	29.4	3000	377.1	0.007	0.969	12.83	0.971
BOWLING2	5123	31.4	3000	396.1	0.109	1.074	12.63	1.076
BULL	5185	27.2	3000	403.7	0.002	0.995	14.85	0.995
CLOTH1	4820	23.9	3000	382.8	0.034	1.030	16.03	1.035
CLOTH2	4230	24.0	3000	390.0	0.121	1.087	16.25	1.090
CLOTH3	4539	22.2	3000	381.2	0.004	0.990	17.16	0.994
CLOTH4	4192	23.3	3000	393.7	0.004	0.978	16.92	0.982
COMPUTER	5191	31.4	3000	403.5	0.025	1.002	12.83	1.006
CONES	6697	38.4	3000	418.8	0.016	1.006	10.90	1.007
DOLLS	5855	33.4	3000	422.5	0.123	1.116	12.66	1.119
DRUMSTICKS	4379	29.0	3000	427.3	0.027	1.011	14.71	1.015
DWARVES	4883	30.9	3000	424.4	0.068	1.020	13.75	1.022
FLOWERPOTS	5746	31.7	3000	390.9	0.109	1.045	12.33	1.047
LAMPSHADE1	4900	31.7	3000	388.7	0.108	1.036	12.27	1.037
LAMPSHADE2	3035	21.3	3000	389.3	0.124	1.055	18.28	1.057
LAUNDRY	6379	39.6	3000	411.0	0.100	1.068	10.37	1.071
MAP	5041	8.7	3000	154.2	0.002	0.994	17.66	0.997
MIDD1	4512	45.2	3000	412.5	0.046	0.994	9.13	0.996
MIDD2	6907	65.4	3000	407.9	0.008	0.967	6.24	0.971
MOEBIUS	50622	294.1	3000	421.5	0.076	1.062	1.43	1.064
MONOPOLY	4153	56.0	3000	402.0	0.054	1.032	7.18	1.033
PLASTIC	4884	41.8	3000	394.6	0.046	0.968	9.44	0.968
POSTER	4449	25.6	3000	415.2	0.026	1.017	16.21	1.019
REINDEER	5324	31.4	3000	409.2	0.037	1.011	13.03	1.014
ROCKS1	51868	267.1	3000	386.8	0.007	0.985	1.45	0.988
ROCKS2	4865	25.6	3000	409.3	0.021	0.995	15.98	0.997
SAWTOOTH	4834	25.3	3000	405.3	0.004	0.999	16.01	1.000
TEDDY	5921	34.6	3000	421.5	0.062	1.049	12.20	1.050
TSUKUBA	6969	24.6	3000	269.4	0.010	0.992	10.96	0.994
VENUS	5841	32.7	3000	404.0	0.004	0.997	12.35	0.998
WOOD1	5826	37.1	3000	414.3	0.001	0.982	11.17	0.983
WOOD2	4491	31.0	3000	382.5	0.021	1.016	12.34	1.017
Geom. mean	5584.0	32.7	3000.0	387.9	0.020	1.012	11.88	1.015

Table 7.6: Experiments on the Middlebury dataset using ℓ_2 regularization. Execution time is measured in minutes.

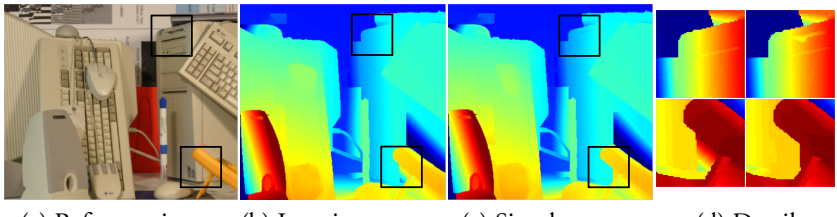
(a) Reference image (b) Iterative binary fusion, objective value: 38164.6 (c) Simultaneous fusion, objective value: 37955

Figure 7.13: BABY2. Surface view of a toy example on using ℓ_1 regularization, where only the 14 SegPln proposals are fused.



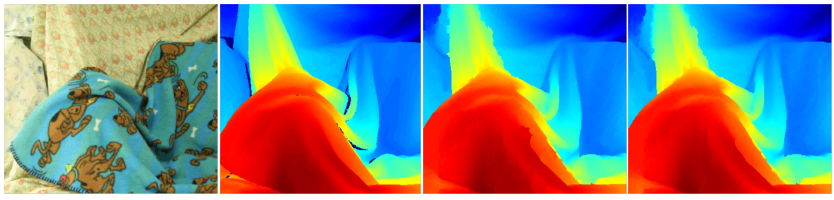
(a) Reference image (b) Ground truth (c) Iterative, obj. value: 4977 (d) Simultaneous, obj. value: 4918

Figure 7.14: CLOTH4. Estimated disparity maps using ℓ_1 regularization for iterative binary fusion and simultaneous fusion.



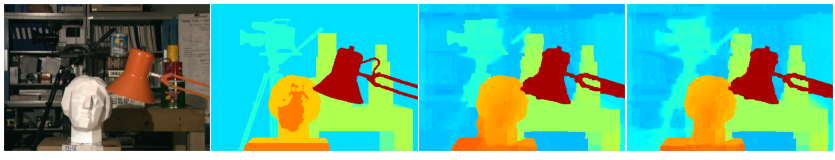
(a) Reference image (b) Iterative, obj. value: 5314 (c) Simultaneous, obj. value: 5272 (d) Details

Figure 7.15: COMPUTER. Estimated disparity maps using ℓ_1 regularization for iterative binary fusion and simultaneous fusion.



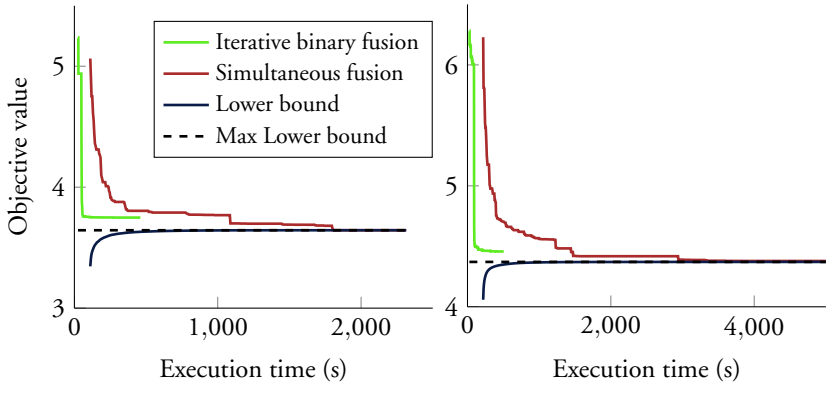
(a) Reference image (b) Ground truth (c) Iterative obj. value: 36626 (d) Simultaneous obj. value: 36180

Figure 7.16: CLOTH3. Estimated disparity maps using ℓ_2 regularization for iterative binary fusion and simultaneous fusion.



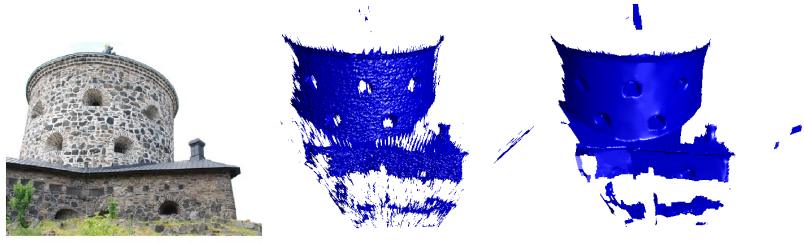
(a) Reference image (b) Ground truth (c) Iterative obj. value: 15560 (d) Simultaneous obj. value: 15260

Figure 7.17: TSUKUBA . Estimated disparity maps using ℓ_2 regularization for iterative binary fusion and simultaneous fusion.



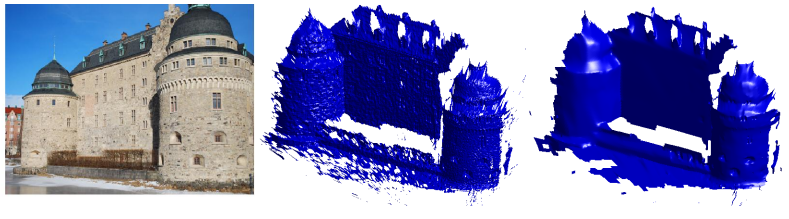
(a) BABY1. Fusing 100 proposals (b) ROCKS1. Fusing 200 proposals

Figure 7.18: Objective value as a function of time using ℓ_1 regularization.



(a) Reference image (b) Only data term (c) Using regularization

Figure 7.19: CASTLE. Estimated surfaces using the objective function E_{ncc} with $\mu = 10$. In (b) no regularization is used ($\tau = 0$) and in (c) the proposed regularization is added ($\tau = 1$).



(a) Reference image (b) Only data term (c) Using regularization

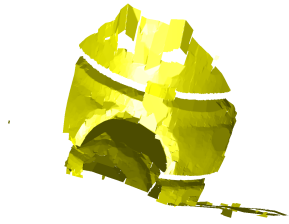
Figure 7.20: ÖREBRO. Estimated surfaces using the objective function E_{ncc} with $\mu = 10$. In (b) no regularization is used ($\tau = 0$) and in (c) the proposed regularization is added ($\tau = 1$).

non-smooth untextured surfaces of NIJO CASTLE. Note that, in addition to ambiguous texture, the NIJO CASTLE sequence contains people that have walked around between images making the data term incorrect at these positions. This is not handled in any special way other than applying more regularization. In all of these cases, the fusion moves provide solutions that approximate the underlying surfaces well. However, the planar nature of the proposals gives the appearance of piecewise planarity. In contrast, with local refinement the resulting surfaces have a much smoother appearance and at the same time capture fine details better. In addition the local refinement also repairs some defects, most likely caused by insufficient sampling, such as the hole visible on the roof of the Nijo castle gate.

7.7 Conclusions

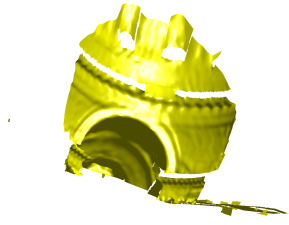
In this chapter, a method for second-order stereo regularization was introduced. In contrast to popular approaches where third-order terms are used for representing second-order surface derivatives, the proposed method uses pairwise interactions with 3D labels. It is shown how to efficiently optimize the model via binary fusion moves and how to extend this to simultaneous fusion, where more than two proposals are fused at the same time. In a number of experiments, the advantages of the proposed method are shown.

(a) Reference image



(b) Result without local refinement, objective value: 14028

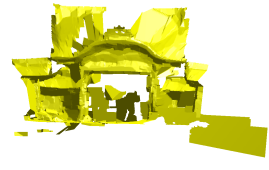
(c) Local refinement with texture



(d) Result with local refinement, objective value: 13411

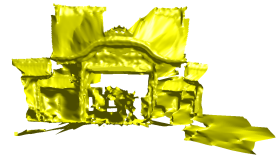
Figure 7.21: ÉGLISE DU DÔME. E_{ncc} with $\mu = 0.5$, $\tau = 0.18$

(a) Reference image



(b) Result without local refinement, objective value: 5471

(c) Local refinement with texture



(d) Result with local refinement, objective value: 5254

Figure 7.22: NIJO CASTLE. E_{ncc} with $\mu = 0.5$ and $\tau = 0.67$

Bibliography

Amini, A., T. Weymouth, and R. Jain (1990). "Using Dynamic Programming for Solving Variational Problems in Vision." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Cited on page 98.

Beasley, J. and N. Christofides (1989). "An Algorithm for the Resource Constrained Shortest Path Problem." In: *Networks*. Cited on pages 99 and 109.

Bendtsen, C. and O. Stauning (1996). *FADBAD, A Flexible C++ Package for Automatic Differentiation*. Technical report. Department of Mathematical Modelling, Technical University of Denmark. Cited on pages 100 and 111.

Bertsekas, D. (1999). *Nonlinear Programming*. Athena Scientific. Cited on page 23.

Billionnet, A. and M. Minoux (1985). "Maximizing a Supermodular Pseudoboolean Function: A Polynomial Algorithm for Supermodular Cubic Functions." In: *Discrete Applied Mathematics*. Cited on page 27.

Birchfield, S. and C. Tomasi (1999). "Multiway Cut for Stereo and Motion with Slanted Surfaces." In: *International Conference on Computer Vision*. Kerkyra, Greece, IEEE. Cited on page 155.

Boros, E. and P. Hammer (2002). "Pseudo-Boolean Optimization." In: *Discrete Applied Mathematics*. Cited on page 31.

Boros, E., P. Hammer, and G. Tavares (2006). *Preprocessing of Unconstrained Quadratic Binary Optimization*. Technical report. RUTCOR Research Report 10-2006. Cited on page 58.

Boyd, S., N. Parikh, E. Chu, B. Peleato, and J. Eckstein (2011). "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers." In: *Foundation and Trends in Machine Learning*. Cited on pages 25 and 182.

Boyd, S. and L. Vandenberghe (2004). *Convex Optimization*. Cambridge University Press. Cited on page 15.

Boyd, S., L. Xiao, and A. Mutapcic (2003). *Subgradient Methods*. Technical report. Stanford University. URL: http://www.stanford.edu/class/ee392o/subgrad_method.pdf (visited on 10/24/2014). Cited on pages 17 and 20.

Boykov, Y. and V. Kolmogorov (2003). "Computing Geodesics and Minimal Surfaces via Graph Cuts." In: *International Conference on Computer Vision*. Nice, France, IEEE. Cited on page 139.

Boykov, Y. and V. Kolmogorov (2004). "An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision." In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on*. Cited on page 28.

Boykov, Y., O. Veksler, and R. Zabih (2001). "Fast Approximate Energy Minimization via Graph Cuts." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Cited on pages 27, 54, 155, 162, and 180.

Bronstein, A., M. Bronstein, and R. Kimmel (2006). "Generalized Multidimensional Scaling: A Framework for Isometry-Invariant Partial Surface Matching." In: United States National Academy of Sciences. Cited on page 98.

Bruckstein, A., A. Netravali, and T. Richardson (2001). "Epi-convergence of Discrete Elastica." In: *Applicable Analysis*. Cited on pages 12 and 72.

Caldwell, T. (1961). "On Finding Minimum Routes in a Network with Turn Penalties." In: *Communications of the ACM*. Cited on page 99.

Carreira, J. and C. Sminchisescu (2010). "Constrained Parametric Min-Cuts for Automatic Object Segmentation." In: *Conference on Computer Vision and Pattern Recognition*. San Francisco, USA, IEEE. Cited on pages 3, 83, and 96.

Carstensen, P. (1983). "The Complexity of Some Problems in Parametric Linear and Combinatorial Programming." PhD thesis. University of Michigan. Cited on page 90.

Casta, C., P. Clarysse, J. Schaerer, and J. Pousin (2009). *Evaluation of the Dynamic Deformable Elastic Template Model for the Segmentation of the Heart in MRI Sequences*. Technical report. URL: http://hdl.handle.net/10380/3072 (visited on 10/24/2014). Cited on page 149.

Chan, T. and L. Vese (2001). "Active Contours Without Edges." In: *IEEE Transactions on Image Processing*. Cited on page 83.

Chen, C., B. He, Y. Ye, and X. Yuan (2014). "The Direct Extension of ADMM for Multi-block Convex Minimization Problems is Not Necessarily Convergent." In: *Mathematical Programming*. Cited on page 25.

Cohen, L. and R. Kimmel (1997). "Global Minimum for Active Contour Models: A Minimal Path Approach." In: *International Journal of Computer Vision*. Cited on page 98.

Constantinides, C., Y. Chenoune, N. Kachenoura, E. Roullot, E. Mousseaux, A. Herment, and F. Frouin (2009). *Semi-automated Cardiac Segmentation on Cine Magnetic Resonance Images using GVF-Snake Deformable Models*. Technical report. URL: http://hdl.handle.net/10380/3108 (visited on 10/24/2014). Cited on page 149.

Delong, A. and Y. Boykov (2009). "Globally Optimal Segmentation of Multi-Region Objects." In: *International Conference on Computer Vision*. Kyoto, Japan. Cited on pages 135 and 140.

Dijkstra, E. (1959). "A Note on Two Problems in Connexion with Graphs." In: *Numerische mathematik*. Cited on page 107.

Eckstein, J. (2012). *Augmented Lagrangian and Alternating Direction Methods For Convex Optimization: A Tutorial and Some Illustrative Computational Results*. Technical report. RUTCOR Research Report 32-2012. Cited on page 25.

Felzenszwalb, P. and D. Huttenlocher (2006). "Efficient Belief Propagation for Early Vision." In: *International Journal of Computer Vision*. Cited on pages 43, 155, 173, 176, and 177.

Fernández-Baca, D. and S Srinivasan (1991). "Constructing the Minimization Diagram of a Two-Parameter Problem." In: *Operations Research Letters*. Cited on pages 85, 86, and 87.

Fischler, M., J. Tenenbaum, and H. Wolf (1981). "Detection of Roads and Linear Structures in Low-resolution Aerial Imagery using a Multisource Knowledge Integration Technique." In: *Computer Graphics and Image Processing*. Cited on page 98.

Frangi, A., W. Niessen, K. Vincken, and M. Viergever (1998). "Multiscale Vessel Enhancement Filtering." In: *International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI)*. Cambridge, USA, Springer. Cited on page 123.

Friman, O., C. Kühnel, and H. Peitgen (2008). "Coronary Centerline Extraction using Multiple Hypothesis Tracking and Minimal Paths." In: *International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI)*. New York, USA, Springer. Cited on page 123.

Fujishige, Satoru (2005). *Submodular Functions and Optimization*. Elsevier. Cited on page 27.

Gallo, G., M. Grigoriadis, and R. Tarjan (1989). "A Fast Parametric Maximum Flow Algorithm and Applications." In: *SIAM Journal on Computing*. Cited on page 96.

Goldberg, A., S. Hed, H. Kaplan, R. Tarjan, and R. Werneck (2011). "Maximum Flows by Incremental Breadth-First Search." In: *European Symposium on Algorithms*. Cited on page 28.

Grady, L. and M. Jolly (2008). "Weights and Topology: A Study of the Effects of Graph Construction on 3D Image Segmentation." In: *International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI)*. New York, USA, Springer. Cited on page 139.

Gulshan, V., C. Rother, A. Criminisi, A. Blake, and A. Zisserman (2010). "Geodesic Star Convexity for Interactive Image Segmentation." In: *Conference on Computer Vision and Pattern Recognition*. San Francisco, IEEE. Cited on pages 98, 99, and 104.

Hachenberger, P. and L. Kettner (2000). *3D Boolean Operations on Nef Polyhedra*. URL: http://doc.cgal.org/4.4/Manual/packages.html (visited on 10/24/2014). Cited on page 92.

Hammer, P., P. Hansen, and B. Simeone (1984). "Roof Duality, Complementation and Persistency in Quadratic 0–1 Optimization." In: *Mathematical Programming*. Cited on pages 29 and 31.

Hartley, R. and A. Zisserman (2003). *Multiple View Geometry In Computer Vision*. Cambridge University Press. Cited on page 7.

Hirschmüller, H. and D. Scharstein (2007). "Evaluation of Cost Functions For Stereo Matching." In: *Conference on Computer Vision and Pattern Recognition*. Minneapolis, USA, IEEE. Cited on page 187.

Huang, S., J. Liu, L. Lee, S. Venkatesh, L. Teo, C. Au, and W. Nowinski (2009). *Segmentation of the Left Ventricle from Cine MR Images using a Comprehensive Approach*. Technical report. URL: http://hdl.handle.net/10380/3121. Cited on page 149.

Irnich, S. and G. Desaulniers (2005). "Shortest Path Problems with Resource Constraints." In: *Column Generation*. Springer. Cited on page 98.

Ivunescu, P. (Hammer) (1965). "Some Network Flow Problems Solved with Pseudo-Boolean Programming." In: *Operations Research*. Cited on page 27.

Jolly, M. (2009). *Fully Automatic Left Ventricle Segmentation in Cardiac Cine MR Images using Registration and Minimum Surfaces*. Technical report. URL: http://hdl.handle.net/10380/3114 (visited on 10/24/2014). Cited on page 149.

Kahl, F. and J. August (2003). "Multiview Reconstruction of Space Curves." In: *International Conference on Computer Vision*. Nice, France. Cited on pages 120 and 122.

Kahl, F. and P. Strandmark (2012). "Generalized Roof Duality." In: *Discrete Applied Mathematics*. Cited on pages 29, 30, 31, and 32.

Kapoor, S. (1999). "Efficient Computation of Geodesic Shortest Paths." In: *ACM Symposium on Theory of Computing*. Association for Computing Machinery. Cited on page 99.

Kappes, J. et al. (2013). "A Comparative Study of Modern Inference Techniques For Discrete Energy Minimization Problems." In: *Conference on Computer Vision and Pattern Recognition*. Portland, USA, IEEE. Cited on page 53.

Kohli, P. and P. Torr (2006). "Measuring Uncertainty in Graph Cut Solutions – Efficiently Computing Min-marginal Energies using Dynamic Graph Cuts." In: *European Conferencfe on Computer Vision*. Graz, Austria, Springer. Cited on page 53.

Kohli, P. and P. Torr (2007). "Dynamic Graph Cuts for Efficient Inference in Markov Random Fields." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Cited on page 141.

Kolmogorov, V. (2006). "Convergent Tree-Reweighted Message Passing for Energy Minimization." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Cited on pages 45, 48, 49, 50, 51, 65, and 68.

Kolmogorov, V. (2012a). "Generalized Roof Duality and Bisubmodular Functions." In: *Discrete Applied Mathematics*. Cited on pages 29 and 31.

Kolmogorov, V. (2012b). "Minimizing a Sum of Submodular Functions." In: *Discrete Applied Mathematics*. Cited on page 27.

Kolmogorov, V., Y. Boykov, and C. Rother (2007). "Applications of Parametric Maxflow in Computer Vision." In: *International Conference on Computer Vision*. Rio de Janeiro, Brazil, IEEE. Cited on pages 3 and 83.

Kolmogorov, V. and T. Schoenemann (2012). "Generalized Sequential Tree-reweighted Message Passing." In: *arXiv preprint: 1205.6352*. URL: http://arxiv.org/abs/1205.6352 (visited on 10/24/2014). Cited on pages 48, 54, and 69.

Kolmogorov, V. and M. Wainwright (2012). "On the Optimality of Tree-reweighted Max-Product Message-Passing." In: *arXiv preprint: 1207.1395*. URL: http://arxiv.org/abs/1207.1395 (visited on 10/24/2014). Cited on page 52.

Kolmogorov, V. and R. Zabih (2002). "Multi-Camera Scene Reconstruction via Graph Cuts." In: *European Conferencfe on Computer Vision*. Copenhagen, Denmark, Springer. Cited on page 155.

Kolmogorov, V. and R. Zabih (2004). "What Energy Functions can be Minimized via Graph Cuts?" In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Cited on page 27.

Komodakis, N., N. Paragios, and G. Tziritas (2007). "MRF Optimization via Dual Decomposition: Message-Passing Revisited." In: *International Conference on Computer Vision*. Rio de Janeiro, Brazil, IEEE. Cited on page 52.

Komodakis, N., N. Paragios, and G. Tziritas (2011). "MRF Energy Minimization and Beyond via Dual Decomposition." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Cited on pages 52, 53, 54, and 69.

Krueger, M., P. Delmas, and G. Gimel'farb (2013). "Robust and Efficient Object Segmentation using Pseudo-Elastica." In: *Pattern Recognition Letters*. Cited on pages 99 and 129.

Lempitsky, V., C. Rother, S. Roth, and A. Blake (2010). "Fusion Moves for Markov Random Field Optimization." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Cited on pages 54 and 155.

Lesage, D., E. Angelini, I. Bloch, and G. Funka-Lea (2009). "A Review of 3D Vessel Lumen Segmentation Techniques: Models, Features and Extraction Schemes." In: *Medical Image Analysis*. Cited on page 98.

Li, G. and S. Zucker (2010). "Differential Geometric Inference in Surface Stereo." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Cited on page 155.

Lu, Y., P. Radau, K. Connelly, A. Dick, and G. Wright (2009). *Automatic Image-Driven Segmentation of Left Ventricle in Cardiac Cine MRI*. Technical report. URL: http://hdl.handle.net/10380/3109 (visited on 10/24/2014). Cited on page 149.

MacCormick, J. and A. Fitzgibbon (2013). "Curvature Regularization for Resolution-Independent Images." In: *International Conference on Energy Minimization Methods in Computer Vision and Pattern Recognition*. Lund, Sweden, Springer. Cited on page 99.

Marák, L., J. Cousty, L. Najman, and H. Talbot (2009). *4D Morphological Segmentation and the MICCAI LV-Segmentation Grand Challenge*. Technical report. URL: http://hdl.handle.net/10380/3085 (visited on 10/24/2014). Cited on page 149.

Mehlhorn, K. and M. Ziegelmann (2000). "Resource Constrained Shortest Paths." In: *European Symposium on Algorithms*. Saarbrücken, Germany, Springer. Cited on page 99.

Meltzer, T., A. Globerson, and Y. Weiss (2009). "Convergent Message Passing Algorithms: A Unifying View." In: *Conference on Uncertainty in Artificial Intelligence*. Montreal, Canada, AUAI Press. Cited on page 49.

Metz, C., M. Schaap, T. Van Walsum, and W. Niessen (2008). "Two Point Minimum Cost Path Approach for CTA Coronary Centerline Extraction." In: *The Insight Journal*. Cited on pages 123 and 124.

Mumford, David (1994). "Elastica and Computer Vision." In: *Algebraic Geometry and its Applications*. Springer. Cited on page 97.

Nesterov, Y. (2004). *Introductory Lectures on Convex Optimization*. Kluwer Academic Publishers. Cited on page 18.

O'Brien, S., O. Ghita, and P. Whelan (2009). *Segmenting the Left Ventricle in 3D using a Coupled ASM and a Learned Non-Rigid Spatial Model*. Technical report. URL: http://hdl.handle.net/10380/3110 (visited on 10/24/2014). Cited on page 149.

Olsson, C., J. Ulén, and Y. Boykov (2013). "In Defense of 3D-Label Stereo." In: *Conference on Computer Vision and Pattern Recognition*. Portland, USA, IEEE. (Not cited.)

Olsson, C., J. Ulén, Y. Boykov, and V. Kolmogorov (2013). "Partial Enumeration and Curvature Regularization." In: *International Conference on Computer Vision*. Sydney, Australia, IEEE. (Not cited.)

Olsson, C., J. Ulén, and A. Eriksson (2014). "Local Refinement for Stereo Regularization." In: *International Conference on Pattern Recognition*. Stockholm, Sweden, IEEE. (Not cited.)

Papadimitriou, C. and K. Steiglitz (1998). *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications. Cited on page 106.

Péchaud, M., R. Keriven, and G. Peyré (2009). "Extraction of Tubular Structures over an Orientation Domain." In: *Conference on Computer Vision and Pattern Recognition*. Miami, USA, IEEE. Cited on pages 98 and 99.

Pressley, A. (2010). *Elementary Differential Geometry*. Second Edition. Springer. Cited on pages 99 and 102.

Radau, P., Y. Lu, K. Connelly, G. Paul, A. Dick, and G. Wright (2009). *MICCAI Cardiac MR Left Ventricle Segmentation Challenge*. London, United Kingdom, Springer. URL: http://smial.sri.utoronto.ca/LV_Challenge/Home.html (visited on 10/24/2014). Cited on page 145.

Reeds, J. et al. (1990). "Optimal Paths for a Car that Goes both Forwards and Backwards." In: *Pacific Journal of Mathematics*. Cited on page 99.

Rother, C., V. Kolmogorov, V. Lempitsky, and M. Szummer (2007). "Optimizing Binary MRFs via Extended Roof Duality." In: *Conference on Computer Vision and Pattern Recognition*. Minneapolis, USA, IEEE. Cited on pages 31, 41, and 58.

Rykfors, M. (2012). "Efficient Graph Cuts for Multi-Region Segmentation." Master thesis. Lund University. Cited on page 152.

Schaap, M. et al. (2009). "Standardized Evaluation Methodology and Reference Database for Evaluating Coronary Artery Centerline Extraction Algorithms." In: *Medical Image Analysis*. Cited on pages 123, 125, and 127.

Scharstein, D. and C. Pal (2007). "Learning Conditional Random Fields for Stereo." In: *Conference on Computer Vision and Pattern Recognition*. Minneapolis, USA, IEEE. Cited on page 187.

Scharstein, D. and R. Szeliski (2002). "A Taxonomy and Evaluation of Dense Two-frame Stereo Correspondence Algorithms." In: *International Journal of Computer Vision*. Cited on page 187.

Scharstein, D. and R. Szeliski (2003). "High-Accuracy Stereo Depth Maps using Structured Light." In: *Conference on Computer Vision and Pattern Recognition*. Madison, USA, IEEE. Cited on pages 81, 185, and 187.

Schoenemann, T. et al. (2011). "The Elastic Ratio: Introducing Curvature into Ratio-Based Globally Optimal Image Segmentation." In: *IEEE Transactions on Image Processing*. Cited on page 98.

Schoenemann, T. (2013). *Generalized Sequential Tree-reweighted Message Passing: Implementation*. URL: https://github.com/Thomas1205/Optimization-Toolbox (visited on 10/24/2014). Cited on page 69.

Schoenemann, T., S. Masnou, and D. Cremers (2012). "A Linear Framework for Region-Based Image Segmentation and Inpainting Involving Curvature Penalization." In: *International Journal of Computer Vision*. Cited on page 70.

Schwarz, L., A. Mkhitaryan, D. Mateus, and N. Navab (2012). "Human Skeleton Tracking from Depth Data using Geodesic Distances and Optical Flow." In: *Image and Vision Computing*. Cited on pages 98 and 99.

Seitz, S. (2001). "The Space of All Stereo Images." In: *International Conference on Computer Vision*. Vancouver, Canada, IEEE. Cited on pages 9 and 159.

Sharir, M. and A. Schorr (1986). "On Shortest Paths in Polyhedral Spaces." In: *SIAM Journal on Computing*. Cited on page 99.

Shekhovtsov, A., P. Kohli, and C. Rother (2012). "Curvature Prior for MRF-based Segmentation and Shape Inpainting." In: *Deutsche Arbeitsgemeinschaft für Mustererkennung (DAM)*. Graz, Austria, Springer. Cited on page 72.

Simayijiang, Z., S. Backman, J. Ulén, S. Wikström, and K. Åström (2013). "Exploratory Study of EEG Burst Characteristics in Preterm Infants." In: *International Conference of the IEEE Engineering in Medicine and Biology Society*. IEEE. Osaka, Japan. (Not cited.)

Staal, J., M. Abramoff, M. Niemeijer, M. Viergever, and B. van Ginneken (2004). "Ridge Based Vessel Segmentation in Color Images of the Retina." In: *IEEE Transactions on Medical Imaging*. Cited on pages 120 and 121.

Strandmark, P. (2012). "Discrete Optimization in Early Vision." PhD thesis. Lund University. Cited on page 26.

Strandmark, P. (2013). *A Library for Unconstrained Minimization of Smooth Functions using Newton's Method or L-BFGS*. URL: https://github.com/PetterS/spii (visited on 10/24/2014). Cited on page 111.

Strandmark, P. and F. Kahl (2011). "Curvature Regularization for Curves and Surfaces in a Global Optimization Framework." In: *International Conference on Energy Minimization Methods in Computer Vision and Pattern Recognition*. Saint Petersburg, Russia, Springer. Cited on page 70.

Strandmark, P., F. Kahl, and N.C. Overgaard (2009). "Optimizing Parametric Total Variation Models." In: *International Conference on Computer Vision*. Kyoto, Japan, IEEE. Cited on page 84.

Strandmark, P., F. Kahl, and T. Schoenemann (2011). "Parallel and Distributed Vision Algorithms using Dual Decomposition." In: *Computer Vision and Image Understanding*. Cited on pages 17 and 52.

Strandmark, P., J. Ulén, and F. Kahl (2012). "HEp-2 Staining Pattern Classification." In: *International Conference on Pattern Recognition*. Tsukuba, Japan, IEEE. (Not cited.)

Strandmark, P., J. Ulén, F. Kahl, and L. Grady (2013). "Shortest Paths with Curvature and Torsion." In: *International Conference on Computer Vision*. Sydney, Australia, IEEE. (Not cited.)

Tappen, M. and W. Freeman (2003). "Comparison of Graph Cuts with Belief Propagation for Stereo, using Identical MRF Parameters." In: *International Conference on Computer Vision*. Nice, France, IEEE. Cited on page 41.

Ulén, J. and C. Olsson (2013). "Simultaneous Fusion Moves for 3D-Label Stereo." In: *International Conference on Energy Minimization Methods in Computer Vision and Pattern Recognition*. Lund, Sweden, Springer. (Not cited.)

Ulén, J., P. Strandmark, and F. Kahl (2011). "Optimization for Multi-Region Segmentation of Cardiac MRI." In: *MICCAI Workshop on Statistical Atlases and Computational Models of the Heart: Imaging and Modelling Challenges*. Toronto, Canada, Springer. (Not cited.)

Ulén, J., P. Strandmark, and F. Kahl (2013). "An Efficient Optimization Framework for Multi-Region Segmentation based on Lagrangian Duality." In: *IEEE Transactions on Medical Imaging*. (Not cited.)

Veksler, O. (2009). "Multi-label Moves for MRFs with Truncated Convex Priors." In: *International Conference on Energy Minimization Methods in Computer Vision and Pattern Recognition*. Bonn, Germany. Cited on page 55.

Vendittelli, M., J. Laumond, and C. Nissoux (1999). "Obstacle Distance for Car-like Robots." In: *IEEE Transactions on Robotics and Automation*. Cited on page 99.

Wainwright, M., T. Jaakkola, and A. Willsky (2004). "Tree Consistency and Bounds on the Performance of the Max-Product Algorithm and its Generalizations." In: *Statistics and Computing*. Cited on page 41.

Wainwright, M., T. Jaakkola, and A. Willsky (2005). "Map Estimation via Agreement on Trees: Message-Passing and Linear Programming." In: *IEEE Transactions on Information Theory*. Cited on page 49.

Wang, H. (2005). "G-wire: A Livewire Segmentation Algorithm Based on a Generalized Graph Formulation." In: *Pattern Recognition Letters*. Cited on pages 98, 99, and 129.

Weiss, Y. and W. Freeman (2001). "On the Optimality of Solutions of the Max-Product Belief-propagation Algorithm In Arbitrary Graphs." In: *IEEE Transactions on Information Theory*. Cited on page 37.

Werner, T. (2010). "Revisiting the Linear Programming Relaxation Approach to Gibbs Energy Minimization and Weighted Constraint Satisfaction." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Cited on page 48.

Wijnhout, J., D. Hendriksen, H. van Assen, and R. van der Geest (2009). *LV Challenge LKEB Contribution: Fully Automated Myocardial Contour Detection*. Technical report. URL: http://hdl.handle.net/10380/3115 (visited on 10/24/2014). Cited on page 149.

Winter, S. (2002). "Modeling Costs of Turns in Route Planning." In: *GeoInformatica*. Cited on page 99.

Woodford, O., P. Torr, I. Reid, and A. Fitzgibbon (2009). "Global Stereo Reconstruction under Second-Order Smoothness Priors." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Cited on pages 5, 81, 155, 163, 165, 166, 180, 185, 187, and 189.

Wright, S. and J. Nocedal (1999). *Numerical Optimization*. Springer. Cited on page 16.

El-Zehiry, N. and L. Grady (2010). "Fast Global Optimization of Curvature." In: *Conference on Computer Vision and Pattern Recognition*. San Francisco, USA, IEEE. Cited on pages 70, 75, and 76.

Index