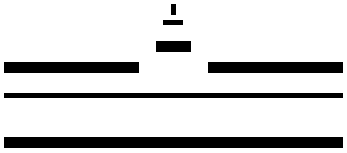


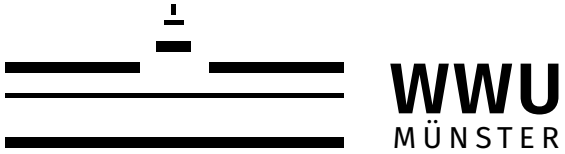
WWU
MÜNSTER

Efficient Variational Graph Methods in Imaging and 3D Data

PhD Thesis

Fjedor Gaede

– 2020 –



Fach: Mathematik

Efficient Variational Graph Methods in Imaging and 3D Data

Inaugural Dissertation

zur Erlangung des Doktorgrades der Naturwissenschaften

– Dr. rer. nat. –

im Fachbereich Mathematik und Informatik
der Mathematisch-Naturwissenschaftlichen Fakultät
der Westfälischen Wilhelms-Universität Münster

eingereicht von
Fjedor Gaede
aus
Münster
– 2020 –

Dekan:	Prof. Dr. Xiaoyi Jiang Westfälische Wilhelms-Universität Münster Münster, DE
Erster Gutachter:	Prof. Dr. Martin Burger Friedrich-Alexander-Universität Erlangen-Nürnberg Erlangen, DE
Zweiter Gutachter:	Prof. Dr. Xiaoyi Jiang Westfälische Wilhelms-Universität Münster Münster, DE
Tag der mündlichen Prüfung:	04.08.2020
Tag der Promotion:	04.08.2020

Abstract

In this thesis we derive, study and apply efficient graph methods for a variety of variational problems in imaging and data processing. The proposed algorithms are based on the recently developed Cut-Pursuit algorithm that can efficiently solve a class of variational problems on graphs. The goal of this work is to expand the Cut-Pursuit algorithm to multi-dimensional isotropic total variation regularizations and to apply it to image data and large 3D point clouds. Furthermore, we investigate the application of the Cut-Pursuit algorithm to TV-regularized Positron Emission Tomography (PET) reconstruction and substantially decrease the number of required full PET operator evaluations during the reconstruction process, replacing them with smaller sparse matrix representations of the full operator. In addition, we extend the so-called Fully4D algorithm to reconstruct spatio-temporal PET data under the influence of dynamics by adding a total variation regularization and deriving an optimization strategy. This work is divided into three parts.

In the first part, we introduce the general graph setting and the mathematical foundation of variational methods and solve these with graph variants of a well-known first-order primal-dual algorithm. Afterwards, we derive an expansion of the Cut-Pursuit algorithm to a multi-dimensional version that is isotropic in the dimensions of the data. Additionally, we develop a modified Cut-Pursuit algorithm for solving minimal partition problems and discuss the oversegmentation properties of this method. We show the efficient optimization of image and point cloud data experimentally and, moreover, recover a recent debiasing strategy to reduce the bias of the total variation, e.g., the loss of contrast in the imaging setting.

In the second part, we introduce the basics of PET and additional TV regularization. We discuss the temporal dependence of the measured PET data and the influence of *motion* and *dynamics* on the corresponding reconstructions. In order to reconstruct a time-dependent image and the associated dynamics we extend the Fully4D algorithm by adding TV regularization. We develop a corresponding optimization algorithm and show numerically that the TV regularization yields more satisfying results than the unregularized method.

In the last part, we tailor the Cut-Pursuit algorithm to TV-regularized PET reconstruction and investigate the gain in efficiency originating from the reduction of the complexity of the PET operator. This is achieved through the construction of a reduced sparse matrix representative of the full operator using a graph discretization provided by Cut-Pursuit. We show numerically that Cut-Pursuit does not only require far fewer expensive full PET operator evaluations than classical methods, but also reaches comparable results in only a few iterations.

Acknowledgements

The last three years might be the hardest but most memorable years of my life. I am very glad that I had the opportunity to be part of the infamous "Dönerbude" until the very end. As now this thesis comes to an end it is time to thank the people that I met in the last years who made this journey so memorable. Let us hope that this gets not to cheesy. I would like to thank:

- my supervisor *Martin Burger* for giving me my first ever student job in his work group during my Bachelor and introducing me to the fascinating field of applied mathematics and numerics. Thank you for giving me the opportunity and freedom to work on the projects that I found most interesting and for always providing good advice. It was a real pity that you left Münster and you will be missed, but I wish you all the best for the upcoming years in Erlangen.
- *Julian Rasch* for answering all my questions multiple times, listening to hundreds of desperate and weird voice messages filled with nonsense and pseudo-math, and taking long walks discussing about my problems. Thank you for proofreading this thesis, revealing all my flaws and telling me to be less negative. Ok, let me say this as it is: Couldn't have done this without you, *dude*! Looking forward to make music and eating Döner with you again.
- *Jonas Geiping* for taking the first steps with me together and now being the greatest help in reviewing this work that I could imagine. Thanks for our endless hours at the work stations in the way too airless computer lab fighting for Matlab licenses on our first project. Thanks for not losing touch when you moved to Siegen and still working with me on interesting topics. It was nice to read your even more elaborated criticism of my english scribble after you lived in the US for half a year and I will miss to read "meh" as a remark on my writing skills.
- *Eva-Maria Brinkmann* for recruiting Jonas and me on our first skiing seminar with your pure kindness. In retrospective, I think without this conversation I would not be here writing this thesis right now. Thank you for proofreading this work in great detail and being so helpful all the time during the last three years. I really miss seeing you in the institute every day, but I am glad that we are still in contact.
- *Daniel Tenbrinck* for introducing me to graph methods, being a great supervisor for my master thesis and always having a good time. I really appreciate all the opportunities you gave me during the last years and the time we spend together in Münster, Erlangen and on the conferences. I wish you and your family all the best in Erlangen and I hope we will meet again very soon. But please no liquor from a spoon and no "car boms" next time.
- *Meike Kinzel* for being a great co-worker and for having so many laughs that I really needed during the hard times in the last three years. Thank you for staying with me in Münster and get through this together. I guess, I will never again dance in a farewell video, or

swear in one. Thank you for pushing me to be part of this. Also, for making the office feel like christmas all year long. I am really grateful for the last spend years. *astronaut emoji(U+1F468)*

- *Ramona Sasse* for taking too long coffee breaks with me, talking about your pony, having the greatest MS Paint skills and being the only person that is really honest about what it is like to be a PhD student. Thank you for letting us show you the nice and fabulous road of being sarcastic.
- *Frank Wübbeling* for interesting discussions about PET and answering all my questions. Thank you for still hosting the skiing seminar and still being one of the best Looping Louie players in the world. I really enjoyed all of our non-mathematical chats and that you are always in for a good laugh. I really appreciate all the times you helped me out and it meant a lot to me.
- *Ulrich Bötcher* for being a great office mate in the last few weeks and helping me so much with my implementation problems. Without our pair-programming sessions I would still be searching for that `C++` bug and having a broken table of content. Thank you for all the nice lunchtimes at the Mensa and the after-work beer in the Finne.
- *Janic Feocke* for providing me with this template, for helping me fix my problems and making me buy a Switch. Your Matlab framework still serves me very well and was a real time saver in the end. Thank you for making the conference in Berlin bearable and for catching some fresh air. Well, and thank you for being so tall. I guess we would have never found the key out of the bunker.
- *Caro Dirks* for being always so positive and supportive. I really liked the discussions about the definitions of cake and pie. Talking about food, your christmas tiramisu is the best!
- *Ina Humpert* for watering the plants that I did not know that they were part of our office. Thank you for keeping the skiing seminar alive even though I could not take part this time and for always bringing people together. Thank you for all of the great drawings, the cartoons of us and for tracking our state of physical and mental condition on the Sims-esque mood board.
- *Loic Landrieu* for coming to Erlangen to talk with me about Cut-Pursuit and introducing this interesting topic. It was nice to meet you and to watch you experience the greatness of Feuerzangenbowle.
- to everyone in the Mensa Whatsapp group for the unspoken agreement to only communicating using emojis.
- to each and everyone I got to meet and had a great time with at and after work. Especially, *Hendrik Dirks, Lena Frerking, Stefan Wierling, Manni, Bernhard Schmitzer, Lynn Frohwein, Leonard Kreutz, Camille Sutour, Tim Keil, Martin Benning* and *Claudia Giesbert*.
- meiner Mutter *Barbara Langhorst* ohne die ich es nicht bis hierhin geschafft hätte. Du warst immer da für mich und hast mir unfassbar viel Unterstützung in den letzten zehn Jahren zukommen lassen. Ich danke dir für alles.
- my brothers *Lars* and *Ole Gaede* for just being the greatest brothers I could imagine.
- thank you to all of my friends that always could distract me from the cold and harsh world of mathematics.

- *Marie & Toulouse* for being the best and most fluffy isolation buddies I can think of. You have shown me that a day is only an accumulation of multiple naps in different places. Yes, this is for my cats.
- And last and most importantly *Dana Hetland* for doing everything you could to keep me sane in the last 8 weeks of this work. You took so much load of my back that I cannot thank you enough for. You are the best social isolation I could think of.

This work was supported by the Bundesministerium für Bildung und Forschung under the project id 05M16PMB (MED4D) and the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 777826 (NoMADS).

Contents

1 Introduction

The focus of this work is to discuss and apply efficient graph methods for a family of variational problems in the context of images, 3D point cloud data and medical imaging. For this purpose we introduce finite weighted graphs that are defined as

$$G = (V, E, w),$$

where V is a finite set of vertices and E is the set of weighted edges connecting these vertices. With this graph structure we can describe any kind of data by assigning a vertex to every data point and connecting related points using some measure of relation. An image, for example, can be given a graph structure by assigning a vertex to each pixel and then connecting the respective vertex to the vertices of its four direct neighbors.

We introduce how to state variational problems on finite weighted graphs describing some given data, and show that it is straightforward to solve them with common optimization strategies, e.g., *first-order primal-dual* algorithms [17]. Primarily, we work on denoising problems that are regularized by the total variation (TV) [72, 14] and in particular, by the total variation on graphs [27, 28]. Using the graph structure, denoising methods can be applied to any data type described by a graph. For example, a central topic of this work is to use TV regularization to denoise point cloud data that can be described by a spatial neighborhood graph, i.e., a k-NN-graph [9, 23].

Working on graphs opens the doors for the application of well-known graph optimization methods such as graph cut methods solving maximum flow problems (cf. [10]). A recently proposed efficient algorithm to solve total variation-regularized problems was proposed by Landrieu et. al. in [46, 47]: The Cut-Pursuit algorithm. Essentially, the algorithm can be described as a coarse-to-fine strategy that starts with a coarse representation of the graph and gets finer in each iteration. This is done by successively computing graph cuts refining the discretization and solving a reduced version of the variational problem on these refinements. The efficiency of this algorithm arises from the low complexity of the problem on the coarse discretization. Even complex problems can be solved efficiently. Therefore, this algorithm creates new application opportunities for variational methods that are otherwise too complex to apply to real-world problems. In this work our focus is to expand the field of applications of the Cut-Pursuit algorithm and apply it to complex problems, i.e., large point cloud data

denoising and sparsification on graphs and TV-regularized PET reconstruction. This work is divided into three parts.

In the first part, we introduce the general graph setting, define mathematical operators on graphs and line out how to state variational problems on graphs. We then apply the well-known first-order primal-dual algorithm from [17] to graphs and, moreover, show how to solve binary partition problems on graphs using a graph cut (cf. [10]). Then we derive the Cut-Pursuit algorithm, discuss different variants and its convergence. The main contribution of this part is the extension of the anisotropic Cut-Pursuit algorithm to a channel-isotropic version and the analysis of its properties. In the numerical section, we apply the Cut-Pursuit algorithm to grayscale and color images as well as to 3D point cloud data. We show that this method is very efficient in contrast to a traditional primal-dual algorithm that uses the full graph structure directly. An outstanding property of the Cut-Pursuit algorithm is that we do not only receive the solution of the problem, but additionally, also a coarse representation of the solution. This enables us to directly apply a debiasing method (cf. [12]) to the solution, which is a beneficial and powerful property.

We also introduce a Cut-Pursuit strategy to optimize minimal partition problems such as the piecewise constant Mumford-Shah functional [19, 57], which can be defined in the discrete setting on a graph as an L_0-TV regularization problem. This problem is highly non-convex and we contribute different strategies to tackle this. In the corresponding numerical experiments, we discuss different aspects of this algorithm in detail. We investigate the energy minimization properties of different strategies and show that all yield satisfying results, both visually and in terms of energy minimization.

Afterwards, we re-interpret the solution of the Cut-Pursuit algorithm for minimal partitions as a coarse discretization of the graph, and argue that this reveals its potential as a superpixel method. To this end, we compare the method to the popular superpixel method SLIC [1]. We will see that the Cut-Pursuit outperforms the SLIC method in terms of boundary exactness and adaptivity to local details. In addition, we will furthermore see that it is robust in presence of high noise in the data.

In the next part, we introduce the inverse problem of Positron Emission Tomography (PET) and how to reconstruct images from PET data. Mathematically, it can be described by the forward model

$$\mathcal{K}u = g$$

where u is the image that we seek, g is the given PET data and $\mathcal{K}$ the forward PET operator, that can be essentially understood as a Radon transform for a certain set of lines. The PET operator $\mathcal{K}$, in fact, computes line integrals through the image u. The goal of the reconstruction is then to receive the image u from the given data g, which can be done by solving a corresponding variational problem. In this work, we focus on the reconstruction of PET data with incorporated TV regularization and we introduce a primal-dual algorithm and the forward-backward EM-TV algorithm [74] to optimize this problem.

We also discuss the time dependency of the gathered information and that these are distorted by *dynamics* and *motion*. To alleviate the negative influence of the dynamics the *Fully4D* algorithm was introduced in [70]. This method divides the reconstruction into multi-

ple time dependent basis functions that describe the dynamics in the data, and corresponding spatial weights to describe the spatial influence of these dynamics. We derive this method in detail in this work. The contribution of this part is that we add a TV regularization to the spatial weights and a Tikhonov regularization [86] to the basis functions and present a corresponding optimization algorithm. We show numerically that applying the Fully4D algorithm with incorporated TV regularization yields satisfying and better results than the original Fully4D algorithm. Additionally, we also show the influence of additional motion in the data.

In the last part of this work, we combine the ideas from the former parts by applying the Cut-Pursuit algorithm to TV-regularized PET reconstruction. The main problem when applying TV regularization to the PET reconstruction arises in the optimization process, where we have to evaluate the full PET operator $\mathcal{K}$ multiple times per iteration, which is very costly. We present a hybrid-gradient primal-dual algorithm and the forward-backward EM-TV [74] algorithm to optimize this problem on graphs. For a more efficient method, we apply the optimization strategy of Cut-Pursuit to the TV-regularized PET problem. We learn that for the reduced problem we only need a reduced operator based on $\mathcal{K}$. This reduced operator can be stored as a sparse matrix and be computed from the full operator $\mathcal{K}$ with only one evaluation, and can then be used to optimize the reduced problems efficiently. The main contribution of this part is to show how to efficiently construct the reduced operator, the convergence is still guaranteed based on the results from [46]. In the numerical section, we will see that the Cut-Pursuit algorithm only requires a few full operator evaluations to yield equivalently solutions as the direct methods yield for several thousands of full operator evaluations. This is especially relevant in problems with high resolution, and therefore, very costly PET operators $\mathcal{K}$.

2 Mathematical Preliminaries

In this chapter, we shortly state different mathematical foundations and notations that we use throughout this work. As they do not build on each other we write them in key notes and refer to them when we use them. Afterwards, we define the weighted and unweighted L_2-data-terms and the Kullback-Leibler data-term and analyse their derivative and derive the corresponding proximity operator. For the Kullback-Leibler data-term we also state the convex conjugate. For more insights we refer to [11, 14, 6, 71].

2.1 Mathematical Statements and Notations

- **Convex Conjugate**
  Let X be a Hilbert space and X^* its dual space. For a function $f : X \to \mathbb{R} \cup \{-\infty, +\infty\}$ the convex conjugated $f^* : X^* \to \mathbb{R} \cup \{-\infty, +\infty\}$ is given by

$$f^*(x^*) := \sup_{x \in X} \{\langle x^*, x\rangle - f(x)\}. \tag{2.1}$$

- **Proximity Operator:**
  Let X be a Hilbert space and $F : X \to \mathbb{R} \cup \{+\infty\}$ a proper, lower-semicontinuous, convex function. Let $\tau > 0$ be some constant. Then the *proximity operator* is given as

$$\text{prox}_{\tau F}(y) = \underset{x \in X}{\text{argmin}} \left\{ F(x) + \frac{1}{2\tau}\|x - y\|_2^2 \right\}. \tag{2.2}$$

- **Directional Derivative:**
  Let $J : \mathcal{H}(V) \to \mathbb{R}$ be a functional. Then the *directional derivative* at a point $f \in \mathcal{H}(V)$ in direction $\vec{d} \in \mathcal{H}(V)$ is defined as

$$J'(f; \vec{d}) \;=\; \lim_{t \to 0} \frac{J(f + t\vec{d}) - J(f)}{t} \tag{2.3}$$

  if the limit exists.

- **(Absolutely) Homogeneous Functions**:
  Let $\Phi \colon X \to Y$ be a mapping between two vector spaces and $t \in \mathbb{R}$ some scalar. The

function Φ is called *p-homogeneous* with $p > 0$ if the following holds:

$$\Phi(tx) = t^p \Phi(x). \tag{2.4}$$

Moreover, the function Φ is called *absolutely p-homogeneous* if the following holds:

$$\Phi(tx) = |t|^p \Phi(x). \tag{2.5}$$

- **Karush-Kuhn-Tucker (KKT) Conditions:**
  Taken from [15, 39] we state the following modified Karush-Kuhn-Tucker (KKT) conditions.

  Proposition 2.1. (Karush-Kuhn-Tucker (KKT) conditions)
  Let $F : \mathcal{H}(V) \to \mathbb{R}$ be a continuous differentiable functional that embeds a function $x \in \mathcal{H}(V)$ into $\mathbb{R}$. Assume we want to optimize the constraint problem given as

$$\begin{aligned} &\operatorname*{argmin}_{x \in \mathcal{H}(V)} F(x) \\ &\quad h_i(x) \leq 0, \quad \text{for } 1 \leq i \leq l \end{aligned} \tag{2.6}$$

  with h_i continuous differentiable functions. Then there exist a set of Lagrange multiplier $\{\lambda_1, \ldots, \lambda_l\}$ with $\lambda_i \geq 0$ such that the stationary points of (2.6) satisfy the following conditions

$$0 \in \partial F(x) + \sum_{i=1}^{m} \lambda_i \partial h_i(x), \tag{2.7}$$

$$0 = \lambda_i h_i(x). \tag{2.8}$$

- **Diagonal Matrix:**
  Let $D \in \mathbb{R}^{n \times n}$ be a diagonal matrix with diagonal elements $D_{ii} = d_i$ and $D_{ij} = 0$ for $i \neq j$. Then we denote this by
  $$D = \operatorname{diag}(d_i)_{i \in [1,n]} = \operatorname{diag}(d_i)_n.$$

- **(Diagonal) Weighted Scalar Product:**
  Let $f, g \in \mathbb{R}^N$ be some vector and $W \in R^{N \times N}$ be a *diagonal* weight matrix. Then the *weighted scalar product* denoted by $\langle \cdot, \cdot \rangle_W$ is given as

$$\langle f, g \rangle_W = \langle Wf, g \rangle = \langle f, Wg \rangle. \tag{2.9}$$

- **(Diagonal) Weighted L_2-Norm:**
  Let $f \in \mathbb{R}^N$ be some vector and $W \in R^{N \times N}$ be a *diagonal* weight matrix. Then the *weighted L_2-norm* denoted by $\| \cdot \|_W$ is given as

$$\|f\|_{2,W} = \sqrt{\langle f, f \rangle_W} = \sqrt{\langle Wf, f \rangle}. \tag{2.10}$$

- **The p-Laplace Operator:**
  Let f be twice-differentiable real-valued function, ∇f the gradient of f and $\nabla\cdot$ the divergence operator. Then the continuous Laplace operator is defined as

$$\Delta f = \nabla \cdot \nabla f.$$

  This is a second-order differential operator.
  hallo (2.40) The p-Laplace operator is a non-linear generalization of the Laplace operator with $1 < p < \infty$. It is defined as

$$\Delta_p u := \nabla \cdot (\|\nabla u\|_2^{p-2} \nabla u). \tag{2.11}$$

- **Disjoint Union:**
  Let $I \subset \mathbb{N}$. The set X is a *disjoint union* of a family of disjoint $(X_i)_{i\in I}$ of subsets $X_i \subset X$ for $i \in I$. Then we can write

$$X = \dot{\bigcup}_{i\in I} X_i. \tag{2.12}$$

  Note that the $X_i \cap X_j = \emptyset$ pair-wise for $i \neq j$.
- **Partitions:**
  We will call a disjoint decomposition of some vertex set V into a finite number of segments A_i a partition Π. It is defined as

$$\Pi \;:=\; \{A_i \subset V \mid i \in I = \{1, \dots, m\},\ A_i \cap A_j = \emptyset \text{ for } i \neq j,\ V = \dot{\cup}_{i=1}^{m} A_i\}. \tag{2.13}$$

- **Number of Elements in Set**
  Let A be a finite set with N entries. Then we denote the number of elements in A by

$$|A| = \#\{i \in A\} = N. \tag{2.14}$$

- **Vectors:**
  In this work we denote (vertical) arrays or vectors by

$$v = (v_i)_{i=1}^{N} \in \mathbb{R}^N \tag{2.15}$$

  with entries $v_i \in \mathbb{R}$. We use this also for stacking vectors.
  Let $v_1, \dots, v_M \in \mathbb{R}^N$ be a sequence of vectors of arbitrary length N. Then we write

$$(v_i)_{i=1}^{N} \in \mathbb{R}^{NM} \tag{2.16}$$

  as the vector of the stacked vectors v_i.
  Additionally, we also use this for matrices $A_i \in \mathbb{R}^{M\times L}$ of the same size and denote this stacked matrices as

$$(A_i)_i \in \mathbb{R}^{NM\times L}. \tag{2.17}$$

- **Characteristic function:**
  Let X be some space and $\Omega \subset X$ a given subset. Then the *characteristic function* is defined as 0 everywhere inside of Ω and as ∞ every where else. This is defined by the following function:

$$\delta_\Omega(x) = \begin{cases} 0, & \text{if } x \in \Omega \\ \infty, & \text{else.} \end{cases} \tag{2.18}$$

- **Indicator Function:**
  Let X be some space and $\Omega \subset X$ a given subset. Then the *indicator function* is defined as 1 everywhere inside of Ω and as 0 every where else. This is defined by the following function:

$$(\mathbf{1}_A)_i = \begin{cases} 1, & \text{if } i \in A \\ 0, & \text{else.} \end{cases} \tag{2.19}$$

- **Vector of Ones:**
  Let $N \in \mathbb{N}$ be some value. Then we denote a vector that is of the size N and filled with only 1 as

$$\mathbf{1}_N = (1)_{i=1}^N. \tag{2.20}$$

- **Kronecker Product:**
  The Kronecker product $\otimes$ can be applied to matrices $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{l\times k}$ which yields

$$A \otimes B = (a_{ij}B)_{i,j} \in \mathbb{R}^{ml\times nk}. \tag{2.21}$$

- **Rows and Columns of Matrices:**
  Let $A \in \mathbb{R}^{m\times n}$ be a matrix. Then we denote with

$$A_{[i,:]}$$

  the i'th row and with

$$A_{[:,j]}$$

  the j'th columns fo A.

2.2 Analysis of Different Data-Terms

In this work we feature different data-terms that we analyse in this section. For that reason, we show some properties of the terms, derive their derivatives and the corresponding proximity operator as defined in (2.2).

2.2.1 The L_2-Data-Term

One of the most common terms to use as a data-term in image processing is the L_2-data-term, which is the data term in the well-known ROF problem [72].

Definition 2.2 (L_2-Data-Term)**.**
Let X be a Hilbert space and $g \in X$ be some given data. Then the L_2-data-term $D(\cdot, g) : \mathcal{H}(V) \to \mathbb{R}$ *is a functional defined as*

$$D(f, g) = \frac{1}{2}\|f - g\|_2^2 = \frac{1}{2}\sum_{u \in V}(f(u) - g(u))^2 \tag{2.22}$$

for any $f \in X$.

This data-term is also separable over the index set V.

Proposition 2.3 (Separable L_2-Data-Term)**.**
Let X be a Hilbert space, $g \in X$ be some measured data and D an L_2-data-term defined in Definition 2.2. Then D is separable over the index set V, i.e., $D(f, g) = \sum_{u \in V} D_u(f_u, g_u)$.

Proof. Let $D_u(f_u, g_u) = \frac{1}{2}(f(u) - g(u))^2$ then the following holds

$$\begin{aligned} D(f, g) &= \frac{1}{2}\|f - g\|_2^2 = \frac{1}{2}\sum_{u \in V}(f(u) - g(u))^2 \\ &= \sum_{u \in V} D_u(f_u, g_u). \end{aligned}$$

□

Also we want to state the derivative of the L_2-data-term for later purposes.

Proposition 2.4 (Derivative of the L_2-Data-Term)**.**
Let X be a Hilbert space and D an L_2-data-term defined in Definition 2.2. Then the derivative of D is given as

$$\nabla D(f, g) = \frac{\partial D(f, g)}{\partial f} = f - g. \tag{2.23}$$

The proximity operator defined in (2.2) is given as follows.

Proposition 2.5 (Proximity Operator of L_2-Data-Term)**.**
Let everything be defined as in Definition 2.2 and $\tau > 0$ some scalar. Then the proximity operator of the L_2-data-term D is given as

$$\operatorname{prox}_{\tau D}(h) = (I + \tau)^{-1}(h + \tau g). \tag{2.24}$$

Proof.
The proximity operator defined in (2.2) of D is given as

$$\operatorname{prox}_{\tau H}(h) = \underset{f \in \mathcal{H}(V)}{\operatorname{argmin}} \left\{ \frac{1}{2}\|f - g\|_2^2 + \frac{1}{2\tau}\|f - h\|_2^2 \right\}. \tag{2.25}$$

Taking the derivative from Proposition 2.4 and isolating f in the optimality condition yields

$$f = (I + \tau)^{-1}(h + \tau g) \tag{2.26}$$

from which follows the given proximity operator. □

2.2.2 The Weighted L_2 Data-Term

The (diagonal) weighted L_2-data-term is defined as the L_2-data-term in Definition 2.2, but instead of a common L_2-norm one applies a *weighted L_2-norm* with a diagonal weighting matrix W as defined in (2.10). Then the data-term is given as follows.

Definition 2.6 (Diagonal Weighted L_2-Data-Term)**.**
Let $f, g \in \mathbb{R}^N$ and $W \in \mathbb{R}^{N\times N}_{\geq 0}$ a diagonal *weighting matrix. Then the* diagonal weighted L_2-data-term *is given as*

$$D_W(f,g) = \frac{1}{2}\|f-g\|^2_{2,W} = \frac{1}{2}\langle W(f-g), f-g\rangle. \tag{2.27}$$

The derivative for this data-term is given as follows.

Proposition 2.7 (Derivative of Diagonal Weighted L_2-Data-Term)**.**
Let $f, g \in \mathbb{R}^N$ and $W \in \mathbb{R}^{N\times N}$ a diagonal weighting matrix. Let D_W be defined as in (2.27). Then the derivative is given as:

$$\nabla D_W(f,g) = \frac{\partial D_W(f,g)}{\partial f} = W(f-g). \tag{2.28}$$

Proof.
By standard differentiation rules, abbreviating $\frac{\partial}{\partial f} = \partial_f$ and using the linearity of ∂_f we can reformulate

$$\begin{aligned}
\frac{\partial D_W(f,g)}{\partial f} &= \frac{1}{2}\partial_f\langle W(f-g), f-g\rangle \\
&= \frac{1}{2}\left(\langle \partial_f W(f-g), f-g\rangle + \langle W(f-g), \partial_f f - g\rangle\right) \\
&= \frac{1}{2}\left(W(f-g) + W(f-g)\right) \\
&= W(f-g).
\end{aligned} \tag{2.29}$$

□

Next we state and derive the proximity operator as defined in (2.2) for the weighted L_2-data-term.

Proposition 2.8 (Proximity Operator of Diagonal Weighted L_2-Data-Term)**.**
Let everything be defined as in Definition 2.6 and a given $\tau > 0$. Then the proximity operator of the diagonal weighted L_2-data-term D_W is given as

$$\operatorname{prox}_{\tau D_W}(h) = (I + \tau W)^{-1}(h + \tau W g). \tag{2.30}$$

Proof.
The corresponding proximity operator by

$$\operatorname{prox}_{\tau D_W}(h) = \operatorname*{argmin}_{f\in\mathbb{R}^N}\left\{ \frac{1}{2\tau}\|f-h\|_2^2 + \frac{1}{2}\|f-g\|^2_{2,W} \right\}. \tag{2.31}$$

Taking the derivative of D_W from (2.29) reformulating the optimality condition of (2.30) yields

$$\begin{aligned} 0 &= \frac{1}{\tau}(f-h) + Wf - Wg \\ 0 &= f - h + \tau Wf - \tau Wg \\ (I+\tau W)f &= h + \tau Wg \\ f &= (I+\tau W)^{-1}(h+\tau Wg) \end{aligned}$$

which is the stated proximity operator. □

Note that the proximity operator can be computed directly since $I + \tau W$ is a diagonal matrix, and thus, the inverse is just the inverse of every entry on the diagonal.

2.2.3 Kullback-Leibler Data-Term

In this section, we investigate the basic properties of the Kullback-Leibler data-term. To that end, we take the *Kullback-Leibler divergence* [45] that measures the *directed divergence* between two discrete probability distributions P and Q as

$$KL(P,Q) = \sum_{j=1}^{L} P_j \log\left(\frac{P_j}{Q_j}\right). \tag{2.32}$$

The Kullback-Leibler (KL) data-term then given by the following definition.

Definition 2.9 (Kullback-Leibler Data-Term)**.**
Let $f \in \mathbb{R}^N$, $g \in \mathbb{R}^M$ and a matrix $K \in \mathbb{R}^{M\times N}$ with $Kf > 0$ everywhere. Then the Kullback-Leibler data-term *is defined as*

$$\begin{aligned} D_{KL}(Kf,g) &= \sum_{j=1}^{M} (Kf)_j - g_j - g_j \ \log\left((Kf)_j\right) && (2.33)\\ &= \langle Kf - g - g \ \log\left(Kf\right), \mathbf{1}_M \rangle && (2.34) \end{aligned}$$

with $\mathbf{1}_M = (1)_{i=1}^M$ as defined in (2.20).

The derivative of the KL data-term is given as follows.

Proposition 2.10 (Derivative of Kullback-Leibler Data-Term)**.**
All assumptions of Definition 2.9 are valid. Then for any $i \in \{1,\dots,N\}$ we can compute the pointwise derivative of D_{KL} as:

$$\frac{\partial D_{KL(f,g)}}{\partial f_i} = \sum_{j=1}^{M} K_{ji} - K_{ji}\frac{g_j}{\sum_{i=1}^{M} K_{ji} f_i}. \tag{2.35}$$

The gradient of D_{KL} is then defined as

$$\nabla D_{KL} = K^* \mathbf{1}_V - K^* \frac{g}{Kf}.$$

Proof.
We compute the derivative pointwise for every $j \in \{1, \dots, M\}$ with the abbreviation $\frac{\partial}{\partial f_i} = \partial_{f_i}$. Let us state two derivatives we make use of in the following deduction

$$\partial_{f_i}(Kf)_j = \partial_{f_i} \sum_{k=1}^{N} K_{jk} f_k = K_{ji}, \tag{2.36}$$

$$\partial_{f_i} \log((Kf)_j) = \frac{1}{(Kf)_j} \partial_{f_i}(Kf)_j = \frac{K_{ji}}{(Kf)_j}. \tag{2.37}$$

Then we can compute

$$\begin{aligned} \frac{\partial D_{KL}(Kf, g)}{\partial f_i} &= \partial_{f_i} \sum_{j=1}^{M} (Kf)_j - \partial_{f_i} g_j \log\left((Kf)_j\right) \\ &= \sum_{j=1}^{M} K_{ji} - g_j \frac{K_{ji}}{(Kf)_j} \\ &= \sum_{j=1}^{M} K_{ji} - \sum_{j=1}^{M} K_{ji} \frac{g_j}{(Kf)_j} \\ &= (K^* \mathbf{1})_i - (K^* \frac{g}{Kf})_i. \end{aligned} \tag{2.38}$$

With this we directly deduce the given gradient stated above by writing it in a vectorized form. □

In this work we use the proximity operator of the convex conjugate of the KL data-term which is given as follows.

Proposition 2.11 (Convex Conjugate of the KL data-term)**.**
The convex conjugate as defined in (2.1) *of the Kullback-Leibler data-term is given for some given data* $g \in \mathbb{N}^M$ *by*

$$D^*_{KL}(x) = \sum_{j=1}^{M} g_j \left(\log(g_j) - \log(1 - x_j) - 1\right) \tag{2.39}$$

with $x_j < 1$ *for* $1 \leq j \leq M$*.*

Proof.
To compute the convex conjugate that is defined in (2.1) of D_{KL} we have to solve

$$\begin{aligned} D^*_{KL}(x^*) &:= \sup_{x \in \mathbb{R}^M} \left\{ \langle x^*, x \rangle - D_{KL}(x, g) \right\} \\ &= \sup_{x \in \mathbb{R}^M} \left\{ \langle x^*, x \rangle - \left[\sum_{j=1}^{M} x_j - g_j - g_j \ \log\left(x_j\right) \right] \right\} \\ &= \sup_{x \in \mathbb{R}^M} \left\{ \langle x^*, x \rangle + \sum_{j=1}^{M} g_j - x_j + g_j \ \log\left(x_j\right) \right\}. \end{aligned} \tag{2.40}$$

Computing the pointwise derivative of the term in (2.40) and reformulating the optimality

condition yields

$$\begin{aligned} 0 &= x_j^* - 1 + \frac{g_j}{x_j} \\ \Leftrightarrow \quad x_j &= \frac{g_j}{1 - x_j^*} \end{aligned} \tag{2.41}$$

for $x_j \neq 1$. This is vectorized just

$$x = \frac{g}{\mathbf{1} - x^*} \tag{2.42}$$

with $x_j^* \neq 1$ for all $1 \leq j \leq M$. To insert x^* into the term (2.40) we have to also assume that $x_j^* < 1$ everywhere, since $g_j > 0$ and log is not defined for values smaller than 0. Hence, we assume $x_j^* < 1$ for all $1 \leq j \leq M$ and plug it into (2.40) which yields

$$\begin{aligned} D_{KL}^*(x^*) &= \langle x^*, \frac{g}{\mathbf{1} - x^*} \rangle + \sum_{j=1}^{M} g_j - \frac{g_j}{1 - x_j^*} + g_j log\left(\frac{g_j}{1 - x_j^*}\right) \\ &= \sum_{j=1}^{M} \frac{g_j x_j^*}{1 - x_j^*} - \frac{g_j}{1 - x_j^*} + g_j log\left(\frac{g_j}{1 - x_j^*}\right) \\ &= \sum_{j=1}^{M} g_j \left(\frac{x_j^*}{1 - x_j^*} - \frac{1}{1 - x_j^*} + \log\left(\frac{g_j}{1 - x_j^*}\right)\right) \\ &= \sum_{j=1}^{M} g_j \left(\log\left(g_j\right) - \log(1 - x_j^*) - 1\right) \end{aligned} \tag{2.43}$$

where we used that the following holds

$$\frac{x_j^*}{1 - x_j^*} - \frac{1}{1 - x_j^*} = \frac{x_j^* - 1}{1 - x_j^*} = -\frac{1 - x_j^*}{1 - x_j^*} = -1. \tag{2.44}$$

□

With this one can compute the proximity operator for the convex conjugate of the KL data-term.

Proposition 2.12 (Proximity Operator of the Convex Conjugate)**.**
Let the conditions of Proposition 2.11 hold, let $g \in \mathbb{R}^M$ be a given data set and $\tau > 0$ a scalar. Then the proximity operator of D_{KL}^* can be derived as

$$\text{prox}_{\tau D_{KL}^*}(x) = \frac{1}{2}\left(x + 1 - \sqrt{(x-1)^2 + 4\tau g}\right). \tag{2.45}$$

Proof.
This is a special case of the proximity operator stated in [66, Appendix A.2]. □

Part I
Graph Cut-Pursuit

3
Introduction to Graph Processing

Graph data structures play an important role in many different fields of research today, e.g., image processing [27, 33], machine learning [95, 4, 13], or network analysis [49, 58, 77]. Their key advantage is that they allow to model and process discrete data of arbitrary topology. Recently, there has been a strong effort to translate well-studied tools from applied mathematics to finite weighted graphs, e.g., variational methods and partial differential equations (e.g. [27, 28, 51]). This enables one to apply common tools to many new application areas that cannot be tackled directly by traditional data modeling techniques, i.e., grids and finite elements. One example we investigate in this work is unstructured point cloud data which can be handled by the use of graphs. Furthermore, graphs allow to exploit repetitive patterns or self-similarity in the data by building connections between related data points. Hence, they can be used to process both local as well as non-local problems in the same unified framework. Due to the abstract nature of the graph structure one may build hierarchical graphs to represent whole sets of entities by a single vertex, e.g., image regions consisting of neighboring pixels [55]. These coarse data representations allow for very efficient optimization techniques as we discuss in Chapter 4 later in this work.

In this chapter, we introduce finite weighted graphs. In particular, we explain how to describe them mathematically, translate them into the setting of variational methods and use them to solve optimization problems.

3.1 Finite Weighted Graphs

The mathematical foundations for finite weighted graphs have been introduced in [27, 32, 33]. In these works the authors developed a common consent on basic concepts and definitions to build a general graph framework and the corresponding discrete mathematics. In the following, we recall these basic concepts and the respective mathematical notation, which we need throughout this work.

3.1.1 Basic Graph Terminology

We start with the definition of a finite weighted graph.

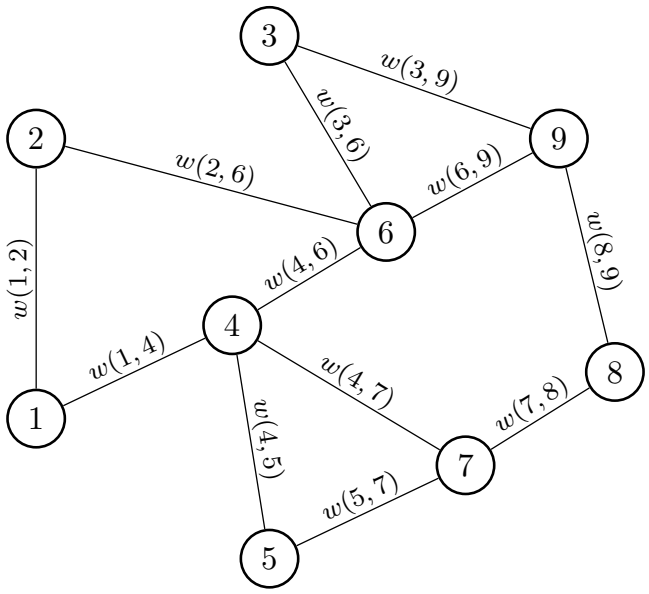


Figure 3.1.: An example of an undirected finite weighted graph $G = (V, E, w)$ with the vertex set $V = \{1, 2, \ldots, 9\}$, an edge set $E = \{(1, 2), (1, 4), (2, 6), \ldots\}$ and some weighting function w.

Definition 3.1 (Finite Weighted Graph)**.**
A finite weighted graph G *is defined as a triple* $G = (V, E, w)$*, where*

- $V = \{1, \ldots, N\}, N \in \mathbb{N}$*, is a finite set of* N *(consecutive) indices that we denote as the* vertex set*,*
- $E \subset V \times V$ *is a finite set of (directed)* edges *connecting vertices in* V*, and*
- $w \colon E \to \mathbb{R}^+$ *is a non-negative* weight function *defined on the edges in* E*.*

A simple finite weighted example graph can be seen in Figure 3.1. For given application data each *vertex* $u \in V$ typically models an entity in the data structure, e.g., an element of a finite set, a pixel in an image, or a node in a network. It is important to distinguish between abstract data entities modeled by graph vertices and attributes associated with them. The latter can be modeled by introducing vertex functions as defined in Definition 3.3 later on.

An *edge* $(u, v) \in E$ between an origin node $u \in V$ and an end node $v \in V$ models a relationship between two entities, e.g., geometric adjacency, entity interactions, or similarity depending on the associated attributes. In our case, we consider graphs with *undirected* edges

Definition 3.2 (Undirected Finite Weighted Graph)**.**
Let $G = (V, E, w)$ *be a finite weighted graph. The graph* G *- and thus the edge set* E *- are called undirected if*

$$(u, v) \in E \Leftrightarrow (v, u) \in E, \qquad \forall (u, v) \in E. \tag{3.1}$$

Additionally, for every edge $(u, v) \in E$ *we have that the weights are symmetric, i.e.,* $w(u, v) = w(v, u)$*.*

Let us introduce some further designations and notations. Let us denote a directed edge set for E as $E_d = \{(u, v) \in E \mid u < v\}$. A node $v \in V$ is called a *neighbor* of the node $u \in V$

if there exists an edge $(u, v) \in E$. For this relationship we use the abbreviation $v \sim u$, which reads as "v is a neighbor of u". If on the other hand v is not a neighbor of u, we use $v \not\sim u$. We define the *neighborhood* $\mathcal{N}(u)$ of a vertex $u \in V$ as $\mathcal{N}(u) \coloneqq \{v \in V : v \sim u\}$. The *degree* of a vertex $u \in V$ is defined as the total number of its neighbors $\deg(u) = |\mathcal{N}(u)|$.

3.1.2 Vertex and Edge Functions

Let us introduce *graph functions* to relate the abstract structure of a finite graph to some given data. More precisely, we introduce vertex and edge functions. A *vertex function* is defined as follows.

Definition 3.3 (Vertex Function)**.**
Let $G = (V, E, w)$ be a finite weighted graph. Then a vertex function *is defined as a function*

$$f : V \to \mathbb{R}^d \tag{3.2}$$

that maps every vertex $u \in V$ to a d-dimensional vector $f(u) \in \mathbb{R}^d$ for $d \in \mathbb{N}$.

Moreover, let $\mathcal{H}(V; \mathbb{R}^d) = \{f : V \to \mathbb{R}^d\}$ be the space of these vector-valued vertex functions. We know that the vertices V are in fact a consecutive set of indices from 1 to N, and thus, we can represent a vertex function $f \in \mathcal{H}(V; \mathbb{R}^d)$ by a vector (or matrix) $\vec{f} \in \mathbb{R}^{N \times d}$. As a consequence, we can endow $\mathcal{H}(V; \mathbb{R}^d)$ with the following inner product

$$\langle f, g \rangle_{\mathcal{H}(V;\mathbb{R}^d)} = \sum_{u \in V} \langle f(u), g(u) \rangle_{\mathbb{R}^d}. \tag{3.3}$$

Then $\mathcal{H}(V; \mathbb{R}^d)$ is a *Hilbert space*. For the sake of simplicity we denote $\mathcal{H}(V; \mathbb{R}^d)$ as $\mathcal{H}(V)$ in the following.

Remark 3.4. *Note that we use a vertex function $f \in \mathcal{H}(V; \mathbb{R}^d)$ and its alternative representation as a matrix $\vec{f} \in \mathbb{R}^{N \times d}$ interchangeably in this work and denote both as f.*

With the definition of the Hilbert space $\mathcal{H}(V)$ and the associated inner product (3.3) we can define the well-known concepts of L^p- and L^∞-norm for functions $f \in \mathcal{H}(V)$.

Definition 3.5 (L^p-Norm of Vertex Functions)**.**
Let G be a finite weighted graph as in Definition 3.1 and $f \in \mathcal{H}(V)$ be a vertex function as in Definition 3.3. Then the L^p-norms on graphs are defined as

$$\begin{aligned} \|f\|_p &= \Big(\sum_{u \in V} \|f(u)\|^p \Big)^{1/p}, && \text{for } 1 \leqslant p < \infty, \\ \|f\|_\infty &= \max_{u \in V} (\|f(u)\|), && \text{for } p = \infty. \end{aligned} \tag{3.4}$$

Similar to vertex functions we also define vector-valued *edge functions*.

Definition 3.6 (Edge Functions)**.**
Let G be a graph as in Definition 3.1 and E its corresponding edge set. Then an edge function

is defined as

$$F : E \to \mathbb{R}^m \tag{3.5}$$

that maps each edge $(u, v) \in E$ *to an* m*-dimensional vector* $F(u, v) \in \mathbb{R}^m$ *with* $m \in \mathbb{N}$.

Note that the dimensions d and m of the image spaces of the vertex functions and the edge functions, respectively, can be chosen arbitrarily as any positive integer. This solely depends on the application. However, as before, we set $\mathcal{H}(E; \mathbb{R}^m)$ as the space of edge functions and abbreviate $\mathcal{H}(E; \mathbb{R}^m)$ by $\mathcal{H}(E)$. The space $\mathcal{H}(E)$ is a Hilbert space endowed with the inner product

$$\langle F, G \rangle_{\mathcal{H}(E)} = \sum_{(u,v)\in E} \langle (F(u, v), G(u, v)) \rangle_{\mathbb{R}^m}, \tag{3.6}$$

for any $F, G \in \mathcal{H}(E)$. Analogously to the definition of the L^p-norm for vertex functions in Definition 3.5, we can define the L^p-norm for edge functions.

Definition 3.7 (L^p-Norm for Edge Functions)**.**
Let $G = (V, E, w)$ *be a finite weighted graph and* $F \in \mathcal{H}(E)$ *be an edge function as in Definition 3.6. Then the* L^p*-norms on edges are defined as*

$$\begin{aligned} \|F\|_p &= \Big(\sum_{(u,v)\in E} \|F(u, v)\|^p \Big)^{1/p}, \quad \text{for } 1 \le p < \infty \,, \\ \|F\|_\infty &= \max_{(u,v)\in E} (\|F(u, v)\|), \qquad \text{for } p = \infty \,. \end{aligned} \tag{3.7}$$

To finish this introduction, let us take a look a the weights w in Definition 3.1. The weights w model the significance of a relationship between two connected vertices $(u, v) \in E$ with respect to an application dependent criterion. For this reason, one introduces a *weight function* $w \in \mathcal{H}(E; \mathbb{R}_+)$. Often, the weight function is chosen as a similarity function based on the attributes of the modeled entities, i.e., by the evaluation of associated vertex functions. In real-world applications, the weighting function w is chosen such that it takes high values if it corresponds to important edges, i.e., high similarity of the involved vertices, and low values for less important ones. In many applications, one normalizes the values of the weight function such that $w \colon E \to [0, 1]$.

Remark 3.8. *Note that a natural extension of the weight function to the full set* $V \times V$ *is given by defining* $w(u, v) = 0$*, if* $v \not\sim u$ *or* $u = v$ *for any* $u, v \in V$*. Then the edge set of the graph can simply be characterized as* $E = \{(u, v) \in V \times V \mid w(u, v) > 0\}$.

In this work, we only consider *undirected* finite weighted graphs as in Definition 3.2. Thus, we only use *symmetric* weighting functions, i.e., weighting functions that satisfy $w(u, v) = w(v, u)$, for all $(u, v) \in E$. Further, we exclude self-loops in the graph, i.e., $u \not\sim u$ for all $u \in V$. Let us introduce some discrete mathematics in the following subsection.

3.1.3 First-Order Partial Difference Operators on Graphs

Using the basic concepts from the previous subsections, we are able to introduce the needed mathematical tools to translate standard differential operators from the continuous setting to finite weighted graphs. The fundamental elements for this translation are first-order partial difference operators on graphs, which have been initially proposed in [27, 33]. In the following, we assume that the considered graphs are connected and undirected with neither self-loops nor multiple edges between vertices. We define a discrete derivative on a graph as follows:

Definition 3.9 (Weighted Partial Difference)**.**
Let $G = (V, E, w)$ be an undirected finite weighted graph as defined in Definition 3.2 and let $f \in \mathcal{H}(V)$ be a vertex function. Then we define the weighted partial difference *of f at a vertex $u \in V$ along the edge $(u, v) \in E$ as:*

$$\partial_v f(u) \;=\; \sqrt{w(u,v)}\,(f(v) - f(u)) \;. \tag{3.8}$$

We denote these weighted partial differences also as finite differences*.*

Remark 3.10.
Analogous to the continuous definition of directional derivatives the graph derivatives have the following properties

$$\begin{aligned} \partial_v f(u) &= -\partial_u f(v), \quad \forall (u,v) \in E \\ \partial_u f(u) &= \vec{0}, \\ f(u) = f(v) &\Rightarrow \partial_v f(u) = \vec{0}. \end{aligned}$$

Based on the definition of weighted partial differences in Definition 3.9 one can straightforwardly introduce the *weighted gradient operator* on graphs.

Definition 3.11 (Weighted Gradient Operator)**.**
Let G be a graph as in Definition 3.9. Then we call the operator

$$\nabla_w : \mathcal{H}(V) \to \mathcal{H}(E), \tag{3.9}$$

the linear weighted gradient operator *on the graph G. It is defined as the vector of weighted finite difference on each edge $(u, v) \in E$ given as*

$$(\nabla_w f)(u,v) \;=\; \partial_v f(u), \quad \forall (u,v) \in E. \tag{3.10}$$

We call $\nabla_w f$ the weighted gradient *of f on G.*

In this work, we want to apply the concept of variational methods to the finite weighted graph setting. In this context, adjoint operators are of importance. For any linear operator $A : \mathcal{H}(V) \to \mathcal{H}(E)$ on a finite weighted graph, its *adjoint operator* $A^* : \mathcal{H}(E) \to \mathcal{H}(V)$ is characterized by

$$\langle Af, G\rangle_{\mathcal{H}(E)} = \langle f, A^*G\rangle_{\mathcal{H}(V)}, \quad \forall f \in \mathcal{H}(V), G \in \mathcal{H}(E).$$

Then the adjoint of the weighted gradient operator $\nabla_w : \mathcal{H}(V) \to \mathcal{H}(E)$ is given as $\nabla_w^* \colon \mathcal{H}(E) \to \mathcal{H}(V)$ which is a linear operator for which holds

$$\langle \nabla_w f, G\rangle_{\mathcal{H}(E)} = \langle f, \nabla_w^* G\rangle_{\mathcal{H}(V)} \quad \forall f \in \mathcal{H}(V), G \in \mathcal{H}(E).$$

The adjoint gradient operator can be deduced as in [27] and is computed as follows:

Definition 3.12 (Adjoint Gradient Operator)**.**
For an undirected graph G given as in Definition 3.2, the weighted adjoint operator ∇_w^* *applied to an edge function $G \in \mathcal{H}(E)$ evaluated at some vertex $u \in V$ is given as*

$$(\nabla_w^* G)(u) \;=\; \sum_{v \sim u} \sqrt{w(u,v)}(G(v,u) - G(u,v)). \tag{3.11}$$

With these definitions at hand, it is easy to derive the *weighted divergence operator* on a graph via the adjoint operator as $\operatorname{div}_w \coloneqq -\nabla_w^*$. Thus, we define:

Definition 3.13 (Weighted Divergence)**.**
For an undirected graph G given as in Definition 3.2, the weighted divergence operator *applied to an edge function $G \in \mathcal{H}(E)$ at a vertex $u \in V$ is given as*

$$(\operatorname{div}_w G)(u) \;=\; \sum_{v \sim u} \sqrt{w(u,v)}(G(u,v) - G(v,u)). \tag{3.12}$$

The weighted divergence on a graph measures the net outflow of an edge function in each vertex of the graph.

3.1.4 Total Variation Regularization on Graphs

In the following, we want to introduce a discrete generalization of the well-known Total Variation (TV) regularization. We do not offer any further introduction to this topic and refer to the introductory chapter *A guide to the TV zoo* in [14] by Martin Burger and Stanley Osher.

In the course of this work, we make use of the generalized TV regularization that we denote as p-q-TV regularization. To introduce this, we first define a family of *discrete p-q-norms* .

Definition 3.14 (Discrete p-q-Norm of Edge Functions)**.**
Let G be an undirected graph as in Definition 3.2, $p, q \geq 1$, and $F \in \mathcal{H}(E; \mathbb{R}^m)$ an edge function. Then the p-q-norm of the edge function F is defined as

$$\|F\|_{p;q} = \left[\sum_{(u,v)\in E} \|F(u,v))\|_q^p \right]^{\frac{1}{p}}. \tag{3.13}$$

Before introducing the p-q-TV regularization on graphs, let us address this subject by investigating the TV regularization in the context of images and then moving over to the setting of graphs. In most imaging applications using the TV regularization, one distinguishes between *anisotropic* and *isotropic* TV regularization. Let us assume that we use forward differences for the discretization of the gradient. Then, for a given $m \times n$ image u, the anisotropic TV regularization is given as

$$\|\nabla u\|_{1,1} = \sum_{i=1}^{m}\sum_{j=1}^{n} \sqrt{(u_{i,j+1} - u_{i,j})^2} + \sqrt{(u_{i+1,j} - u_{i,j})^2}$$

and the isotropic TV regularization as

$$\|\nabla u\|_{1,2} = \sum_{i=1}^{m}\sum_{j=1}^{n} \sqrt{(u_{i,j+1} - u_{i,j})^2 + (u_{i+1,j} - u_{i,j})^2}.$$

The isotropic regularization is defined as a term that penalizes the length of the gradient in each pixel, while the anisotropic case is decoupled for every finite forward difference. Hence, when we transfer this to the graph setting, this yields for a given finite weighted graph $G = (V, E, w)$ and a 1-dimensional vertex function $f \in \mathcal{H}(V; \mathbb{R})$ that the anisotropic TV regularization is defined as

$$\|\nabla f\|_{1,1} = \sum_{u\in V}\sum_{v\sim u} \sqrt{w(u,v)}\sqrt{(f(v) - f(u))^2}$$

and the isotropic case as

$$\|\nabla f\|_{1,2} = \sum_{u\in V} \sqrt{\sum_{v\sim u} w(u,v)(f(v) - f(u))^2}.$$

Since we have already introduced d-dimensional vertex functions, it is of interest to also define TV regularization terms for these functions. As we have said before, we call the TV regularization isotropic if we penalize not a single partial derivative over one edge, but multiple edges at once. With this idea in mind, we introduce four different variants of TV regularization on graphs in the following.

Definition 3.15 (Different TV Regularization Terms on Graphs)**.**
Let $G = (V, E, w)$ be a finite weighted graph and $f \in \mathcal{H}(V; \mathbb{R}^d)$ some d-dimensional vertex function. Then we can define the following TV regularization terms:

Anisotropic TV Regularization:

$$\|\nabla_w f\| = \sum_{u \in V} \sum_{v \sim u} \sum_{j=1}^{d} w(u,v)^{\frac{1}{2}} |f(v)_j - f(u)_j| \tag{3.14}$$

Spatially-Isotropic TV Regularization:

$$\|\nabla_w f\| = \sum_{u \in V} \sum_{j=1}^{d} \Big(\sum_{v \sim u} w(u,v)(f(v)_j - f(u)_j)^2 \Big)^{\frac{1}{2}} \tag{3.15}$$

Channel-Spatially-Isotropic TV Regularization:

$$\|\nabla_w f\| = \sum_{u \in V} \Big(\sum_{v \sim u} \sum_{j=1}^{d} w(u,v)(f(v)_j - f(u)_j)^2 \Big)^{\frac{1}{2}} \tag{3.16}$$

Channel-Isotropic TV Regularization:

$$\begin{aligned} \|\nabla_w f\| &= \sum_{u \in V} \sum_{v \sim u} \Big(w(u,v) \sum_{j=1}^{d} (f(v)_j - f(u)_j)^2 \Big)^{\frac{1}{2}} \\ &= \sum_{u \in V} \sum_{v \sim u} w(u,v)^{\frac{1}{2}} \ \|f(v) - f(u)\|_2 \end{aligned} \tag{3.17}$$

Note that we call the isotropy over the dimensions *channel-isotropy* since in the context of images the term *dimensions* refers to the image space and not to the value space. In fact, a $1D$ signal, a grayscale image and a 3D volume image are all 1-dimensional vertex functions on a graph that encodes the multi-dimensional structures of the respective data accordingly. On the other hand, in the graph setting, an RGB image is given as a 3-dimensional vertex function $f \in \mathcal{H}(V; \mathbb{R}_+^d)$ on a grid graph describing the image. The dimensions are denoted as red-channel, green-channel and blue-channel. In reminiscence to this notation, we call dimensions of the vertex function also *channels* from now on, and hence, the isotropic TV terms over the channels *channel-isotropic TV regularization*. We call the isotropy over the edges in this work *spatial-isotropy* since in the image context it covers the derivatives in the spatial directions.

Remark 3.16.
Both isotropic regularizer defined in (3.16) and (3.17) are important for denoising images, but not further considered in this work for the following reasons: In the next chapter, we introduce the Cut-Pursuit algorithm where we use graph cuts that minimize energies that incorporate TV-type regularizers. It is not easy - or maybe even not possible - to build a graph cut algorithm to solve a spatial isotropic regularization problem. Thus, we cannot derive a Cut-Pursuit algorithm for such regularizers.

Now let us finally introduce the p-q-TV regularization based on p-q-norms defined in Definition 3.14 and the weighted gradient operator ∇_w for $p, q \geq 1$.

Definition 3.17 (Discrete p-q-TV Regularization)**.**
Let G be an undirected graph as in Definition 3.2, $p, q \geq 1$, and $f \in \mathcal{H}(V; \mathbb{R}^d)$ a vertex function. Then the p-q-TV regularization *is defined as*

$$\begin{aligned} \|\nabla_w f\|_{p;q} &= \left[\sum_{(u,v)\in E} \|\nabla_w f(u,v))\|_q^p \right]^{\frac{1}{p}} \\ &= \left[\sum_{u\in V} \sum_{v\sim u} w(u,v)^{\frac{p}{2}} \|f(v) - f(u)\|_q^p \right]^{\frac{1}{p}} \\ &= \left[\sum_{u\in V} \sum_{v\sim u} w(u,v)^{\frac{p}{2}} \left(\sum_{j=1}^{d} |f(v)_j - f(u)_j|^q \right)^{\frac{p}{q}} \right]^{\frac{1}{p}}. \end{aligned} \tag{3.18}$$

The advantage of using the general p-q-TV (3.18) is that this term captures different regularization terms as the anisotropic TV regularization ($p = q = 1$) we state in (3.14), the channel-isotropic TV ($p = 1, q = 2$) from (3.17) and the classical Tikhonov regularization ($p = q = 2$). These regularization terms are widely used for denoising monochromatic and also vector-valued signals, see e.g. [27, 56, 11] and references therein. It is also used to combining image reconstruction with denoising as we do in Chapter 10 for reconstructing data from Positron Emission Tomography (PET). Depending on the choice of the parameters $p, q \geq 1$, the introduced regularization has different properties that we will analyse later on.

3.2 Graph p-q-Laplace Operator

In the following, we provide a small course about the topic of p-q-Laplace operators. The continuous p-Laplace operator in (2.11) is an example of a second-order differential operator for which we can define a discrete counterpart on finite weighted graphs. It allows for the translation of various partial differential equations to the graph setting and it has been used for many image processing applications(cf. [27, 33, 52, 51]). For a detailed discussion of the graph p-Laplacian and its variants we refer, to [28].

Based on the first-order partial differential operators introduced in (3.10) and (3.12), we are able to formally derive a family of graph p-q-Laplace operators $\Delta_{w,p,q} \colon \mathcal{H}(V) \to \mathcal{H}(V)$ by computing the derivative of the p-q-TV regularizer as defined in (3.18) in its differentiable parts. In the following, we derive a multi-dimensional version of the p-Laplacian introduced in [27] and extend it to a p-q Laplace operator. For the sake of simplicity, we assume that we work on an undirected finite weighted graph $G = (V, E, w)$ as defined in Definition 3.2. Note that this implies that the weight function $w \in \mathcal{H}(E)$ is symmetric, i.e., $w(u, v) = w(v, u)$.

Definition 3.18 (The p-q-Laplace Operator)**.**
Let G be an undirected graph as in Definition 3.2 and $f \in \mathcal{H}(V)$ a vertex function. Moreover, let $|\nabla_w f(u,v)|$ denote the pointwise absolute value in the gradient $\nabla_w f(u,v)$ and $\cdot$ be a pointwise product between vectors. Then the weighted p-q-Laplace operator *is defined as*

$$\Delta_{w,p,q} f = \frac{1}{2} \operatorname{div}_w \left(\|\nabla_w f\|_q^{p-q} \nabla_w f \cdot |\nabla_w f|^{q-2} \right) \tag{3.19}$$

In Appendix 3.A, we show that one can reformulate the p-q-Laplace operator in the following way for every $u \in V$ and $1 \leq j \leq d$

$$\begin{aligned} &(\Delta_{w,p,q} f)(u)_j \\ &\qquad = \sum_{v \sim u} w(u,v)^{\frac{p}{2}} \|f(v) - f(u)\|_q^{p-q} |f(v)_j - f(u)_j|^{q-2} \cdot (f(v)_j - f(u)_j). \end{aligned} \tag{3.20}$$

In Appendix 3.B, we derive that the p-q-Laplace operator is equivalent to the derivative of the differentiable parts of the p-q-TV regularizer. We consider two special cases that we face several times in this work. The first special case with $p = q$ is the multi-dimensional *anisotropic* p-Laplacian given as

$$\Delta_{w,p}^{a} f(u) = \sum_{v \sim u} w(u,v)^{\frac{p}{2}} \nabla_w f(u,v) \cdot |\nabla_w f(u,v)|^{p-2}. \tag{3.21}$$

The second case for $q = 2$ is given as the multi-dimensional *(channel)-isotropic* p-Laplacian

$$\Delta_{w,p}^{i} f(u) = \sum_{v \sim u} w(u,v)^{\frac{p}{2}} \|f(v) - f(u)\|_2^{p-2} \nabla_w f(u,v). \tag{3.22}$$

For $p = q = 2$ we obtain a notion of a classical linear operator known as the un-normalized graph Laplacian, now in multiple dimensions, as

$$\Delta_w f(u) \;=\; \sum_{v \sim u} w(u,v) \left(f(v) - f(u) \right).$$

Remark 3.19.
Note that in the terminology of [28] *both variants of the p-Laplacian would be called* anisotropic *since the authors discussed only the one-dimensional case of vertex and edge functions. In our more general case we relate the term* isotropic *to the coupling of coordinates along all dimensions, which we call* channel-isotropy *as we have already pointed out in Definition 3.15 and the following. Also note that in the anisotropic case the inner terms decouple and allow for an pairwise independent computation.*

3.3 Optimization Problems on Graphs

Thus far, we have created the basis for transferring common mathematical formulations and operations into a unified graph setting. In this section, we consider and solve variational problems on graphs. An exhaustive introduction into variational problems can be found in [11, Chapter 4.2]. As a case study for algorithms that can solve variational problems on graphs we use the very common primal-dual algorithm introduced first in the work of Pock,

Bischof, Cremers and Chambolle [63] for minimizing a convex relaxation of the Mumford-Shah functional and then followed up in a more general framework by Chambolle and Pock [17]. The primal-dual algorithm solves problems given as

$$\operatorname*{argmin}_{f \in X} \; F(Kf) + H(f) \tag{3.23}$$

with X, Y two real finite-dimensional vector spaces, a continuous linear operator $K\colon X \to Y$, and two proper, convex and lower-semicontinuous functionals F and H. To translate problems of this kind onto a finite weighted graph $G = (V, E, w)$ as defined in Definition 3.1 we first set $X = \mathcal{H}(V)$ - as defined in Section 3.1.2. We do not specify Y here, since it depends on the setting we are dealing with. However, let us mention that in most cases in this work Y is equal to the Hilbert space of edge functions $\mathcal{H}(E)$. The continuous linear operator K is then given as $K\colon \mathcal{H}(V) \to Y$. Thus, the corresponding saddle-point problem is given as

$$\min_{f \in \mathcal{H}(V)} \min_{y \in Y} \langle Kf, y \rangle + H(f) - F^*(y) \tag{3.24}$$

where $F^*\colon Y \to \mathbb{R}_+$ denotes the convex conjugate functional of F, as defined in (2.1), which maps edge functions to a positive real value. Consequently, we can directly obtain the following graph version of the primal-dual hybrid gradient algorithm in [17].

Algorithm 3.20 (Primal-Dual Hybrid Gradient Algorithm on Graphs)**.**
Let G be an undirected finite weighted graph as in Definition 3.2. Choose step sizes τ, $\sigma > 0, \theta \in [0, 1]$. Initialize $f^0 \in \mathcal{H}(V)$ and $y^0 \in Y$. Set $\bar{f}^0 = f^0$. Then the primal-dual algorithm on the graph G to solve (3.23) is given as

$$\begin{cases} y^{n+1} = \operatorname{prox}_{\sigma F^*}(y^n + \sigma K \bar{f}^n) \\ f^{n+1} = \operatorname{prox}_{\tau H}(f^n - \tau K^* y^{n+1}) \\ \bar{f}^{n+1} = f^{n+1} + \theta(f^{n+1} - f^n) \end{cases} \tag{3.25}$$

with the proximity operators prox *as defined in (2.2).*

In this work, we denote the primal-dual hybrid gradient algorithm as primal-dual algorithm for simplicity. With this primal-dual algorithm we can solve any multi-dimensional variational problem of the form (3.23) on a given graph G. All of the proofs and methods to determine or update the step sizes τ and σ are analog in the graph setting. Nevertheless, choosing the step sizes as constant values can be tricky in a graph setting. According to [17], the step sizes $\tau, \sigma > 0$ should be chosen such that

$$\tau\sigma\|K\|^2 < 1. \tag{3.26}$$

If we assume that the graph G has a fixed structure, e.g., a grid graph representing an image, and Y is somehow related to $\mathcal{H}(E)$, then we map into a set of edge functions that are always structured the same for different instances of data. Thus, the operator norm $\|K\|$ has always the same value since it only depends on the structure of G and not on the data. Accordingly, the value for $\|K\|$ only has to be computed once and can be used for every subsequent

application. On the other hand, if we have some unstructured data, e.g., point clouds or some graph built from pixel or patch similarities, the resulting graphs vary depending on the data and the chosen method to build the graphs. Consequently, the edge set E and the space of edge functions $\mathcal{H}(E)$ differ extremely for every graph. Thus, the condition number $\|K\|$ of the operator is different for every graph, and one has to compute it for every application on new data. There exist good methods to approximate the condition like the power iteration, but this entails some extra computational costs one would like to prevent. Additionally, a constant step size might also be an inefficient choice if some vertices have only a few neighbors and some have many. We see this later on in the numerics section when we use the gradient operator, i.e., $Y = \mathcal{H}(E), \mathbf{K} = \nabla_w$, and solve an ROF problem on graphs as we state in Section 3.3.1. To prevent these inefficient constant step sizes, it is a straightforward idea to compute an adaptive step size for every node in V and every edge in E such that it better reflects the given graph structure. A method to do this was introduced in [62] called *diagonal preconditioning*, where the condition of every node and every edge is incorporated in the step sizes. This method can be directly applied to the primal-dual algorithm on graphs as defined in Algorithm 3.20. As we have pointed out in Section 3.1.2, we can represent any vertex function $f \in \mathcal{H}(V)$ as a vector $f \in \mathbb{R}^{N\times d}$ with N as the number of vertices. Moreover, we let $Y \cong \mathbb{R}^M$, which corresponds to the case of $Y = \mathcal{H}(E)$ with M denoting the number of edges. Thus, we can describe any continuous linear operator $K\colon \mathcal{H}(V) \to Y$ as a matrix $K \in \mathbb{R}^{M\times N}$ and define a diagonal preconditioning as follows.

Definition 3.21 (Diagonal Preconditioning)**.** *Let $T = diag(\tau) \in \mathbb{R}_+^{N\times N}$ with $\tau = (\tau_1, \cdots, \tau_N)$ and $\Sigma = diag(\sigma) \in \mathbb{R}_+^{M\times M}$ with $\sigma = (\sigma_1, \cdots, \sigma_M)$. Then analogously to* [62] *the individual step-sizes are given as*

$$\tau_i = \frac{1}{\sum_{j=1}^{M} |K_{ji}|^{2-\alpha}}, \quad \sigma_j = \frac{1}{\sum_{i=1}^{N} |K_{ji}|^{\alpha}}, \tag{3.27}$$

with $\alpha \in [0, 2]$.

Note that in this work we denote $\mathbb{R}_+ = \mathbb{R}_{\geq 0}$. For more insights into convergence and rationale for this appraoch please consult [62]. For a better understanding, let us apply this to the case where $Y = \mathcal{H}(E)$, M the number of edges in G and $K = \nabla_w$. The operator K can be interpreted as a weighted differential operator matrix $\mathcal{D} \in \mathbb{R}^{M\times N}$ representing the graph operator ∇_w, which means that $\mathcal{D}f = \nabla_w f$. Since E is finite, we can find a corresponding edge $e_i = (u_i, v_i) \in E$ for every $i \in \{1, \ldots, M\}$ and define $\mathcal{D}$ as follows

$$\mathcal{D}_{i,\tilde{u}} = \begin{cases} \sqrt{w(u_i, v_i)}\ , & \tilde{u} = u_i, \\ -\sqrt{w(v_i, u_i)}, & \tilde{u} = v_i, \\ 0, & \text{else.} \end{cases} \tag{3.28}$$

As we can see, the number of entries in a column for a given vertex $u \in V$ depends on the number of neighbors. Thus, for graphs with a rather inhomogeneous structure, e.g., a symmetrized k-nearest neighbors graph (cf. e.g. [9, 23]) on unstructured point cloud data, the norm of the operator might not be a good choice for the step sizes τ and σ in (3.20) as it might be too conservative for most vertices. Applying preconditioning often is a good measure to

enhance the convergence speed of the algorithm. In order to apply the preconditioning scheme from Definition 3.21, we have to compute the row and column sums of the absolute values in $\mathcal{D}$. Assuming that $w(u,v) = w(v,u)$ for all $(u,v) \in E$ the component-wise preconditioners for $\mathcal{D}$ are then given by

$$\begin{aligned} \tau_u &= \frac{1}{\sum_{i=1}^{M} |\mathcal{D}_{i,u}|^{2-\alpha}} = \frac{1}{\sum_{v \sim u} w(u,v)^{\frac{2-\alpha}{2}}}, \quad \forall u \in V \\ \sigma_i &= \frac{1}{\sum_{u \in V} |\mathcal{D}_{i,u}|^{\alpha}} = \frac{1}{2w(u,v)^{\frac{\alpha}{2}}}, \quad \forall i \in \{1,\dots,M\} \end{aligned} \tag{3.29}$$

for any $\alpha \in [0,2]$. This leads to the diagonal preconditioners

$$\begin{aligned} T &= \operatorname{diag}(\tau_1,\dots,\tau_N) \\ \Sigma &= \operatorname{diag}(\sigma_1,\dots,\sigma_M). \end{aligned} \tag{3.30}$$

As we can see, the preconditioning for the primal update T takes the number of edges and their weights directly into account, and thus, improves the condition number of very inhomogeneous graph problems significantly.

This gives us all the tools we need to solve a multi-dimensional variational problem given as in (3.23) with a primal-dual algorithm on any finite weighted graph G.

3.3.1 ROF on Graphs

In this subsection, we introduce the well-known Rudin-Osher-Fatemi (ROF) denoising problem from [72] on graphs. It is a variational problem with an L_2-data-term and a TV regularization. In this section, we have already derived all tools needed to state the minimization problem on a graph. Let $G = (V,E,w)$ be an undirected finite weighted graph as in Definition 3.2 and $g \in \mathcal{H}(V;\mathbb{R}^d)$ some given d-dimensional noisy data. Let the regularizer be the general p-q-TV regularizer from (3.17). Then the general p-q-ROF problem on the graph G to denoise g is given as

$$\operatorname*{argmin}_{f \in \mathcal{H}(V)} \frac{1}{2}\|f-g\|_2^2 + \frac{\alpha}{2p}\|\nabla_w f\|_{p;q}^p, \tag{3.31}$$

where the regularization parameter $\alpha > 0$. Note that α is divided by 2 since we are working on undirected graphs for which every edge is summed up twice, once for each direction. This is necessary to compare the results of (3.31) with the results of the classical ROF problem applied to an image where the gradient is defined by finite differences, which is in fact like a directed graph. To put this problem into relation with the general problem in (3.23) we set $Y = \mathcal{H}(E)$, $K = \nabla_w$, $H = \frac{1}{2}\|\cdot - g\|_2^2$ and $F = \frac{\alpha}{2p}\|\cdot\|_{p;q}$. To state the primal-dual algorithm on graphs for problem (3.31), we have to compute the proximity operators for F^* and H. Since H is a squared L_2-norm, we can compute the proximity operator easily as we have shown in Section 2.2.1 in Proposition 2.5. For the functional $F = \frac{\alpha}{2}\|\cdot\|_{p;q}$, we have derived the different proximity operators for different combinations of p and q in Appendix 3.C. To be more general, we state the primal-dual algorithm with $\operatorname{prox}_{\sigma(\frac{\alpha}{2p}\|\cdot\|_{p;q})^*}(y)$ as a representation for the different proximity operators and point to the corresponding definition in Appendix 3.C.

At this point, we can state the primal-dual algorithm to solve the general ROF optimization problem in (3.31) on graphs and also incorporate the diagonal preconditioning stated in (3.30).

Algorithm 3.22 (ROF Primal-Dual Algorithm on Graphs)**.**
Let G be an undirected finite weighted graph as in Definition 3.2. Choose $\theta \in [0,1]$ and compute diagonal preconditioners T, Σ as in (3.30). Initialize $f^0 \in \mathcal{H}(V)$ and $y^0 \in \mathcal{H}(E)$. Set $\bar{f}^0 = f^0$. Then the primal-dual algorithm on the graph G to solve the ROF denoising problem in (3.22) is given as:

$$\begin{cases} y^{n+1} = \operatorname{prox}_{\Sigma(\frac{\alpha}{2p}\|\cdot\|_{p;q})^*}(y^n + \Sigma\nabla_w \bar{f}^n) \\ f^{n+1} = (I+T)^{-1}(f^n + T(\operatorname{div}_w(y^n) + g)) \\ \bar{f}^{n+1} = f^{n+1} + \theta(f^{n+1} - f^n) \end{cases} \tag{3.32}$$

with the proximity operators $\operatorname{prox}_{\sigma(\frac{\alpha}{2p}\|\cdot\|_{p;q})^*}$ as defined in Appendix 3.C for selected p and q.

Remark 3.23.
Note that in Algorithm 3.22 one could still try to solve the problem with fixed step sizes τ and σ if the graph structure is in some sense significantly homogeneous.

3.3.2 Weighted ROF on Graphs

In this work we will discuss several problems where a weighted L_2-data-term is incorporated, e.g., the reduced problem of the Cut-Pursuit algorithm we introduce in the next chapter and the TV-step in the forward-backward EM-TV algorithm to reconstruct TV regularized PET data in Section 10.2. Hence, we want to discuss the weighted L_2-data-term in this section and derive a primal-dual algorithm to solve a weighted ROF problem. The weighted L_2-data-term including a diagonal weight matrix $W \in \mathbb{R}^{N\times N}$ is given as

$$\frac{1}{2}\|f-g\|_{2;W}^2 = \frac{1}{2}\langle W(f-g), f-g\rangle. \tag{3.33}$$

Before we conclude this chapter, we want to state the primal-dual algorithm for the *weighted ROF* problem on graphs

$$\operatorname*{argmin}_{f\in\mathcal{H}(V)} \frac{1}{2}\|f-g\|_{2,W}^2 + \frac{\alpha}{2p}\|\nabla_w f\|_{p;q}^p \tag{3.34}$$

for a given diagonal matrix $W \in \mathbb{R}^{N\times N}$ and some data $g \in \mathbb{R}^N$. From Proposition 2.8 we know that the proximity operator of the weighted L_2-data-term is given as

$$\operatorname{prox}_{\tau D_W}(f) = (I+\tau W)^{-1}(f+\tau W g) \tag{3.35}$$

for some $g \in \mathcal{H}(V)$. Thus, we can replace the primal proximity operator in Algorithm 3.22 by (3.35) and get the following primal-dual algorithm:

Algorithm 3.24 (Weighted ROF Primal-Dual Algorithm on Graphs).
Let every assumption be as in Algorithm 3.22 and in addition let $W \in \mathbb{R}^{N\times N}$ be a diagonal matrix. Then the primal-dual algorithm to solve (3.34) is given as

$$\begin{cases} y^{n+1} = \operatorname{prox}_{\Sigma(\frac{\alpha}{2p}\|\cdot\|_{p;q}^p)^*}(y^n + \Sigma\nabla_w \bar{f}^n) \\ f^{n+1} = (I + TW)^{-1}(f^n + TW(\operatorname{div}_w(y^{n+1}) + g)) \\ \bar{f}^{n+1} = f^{n+1} + \theta(f^{n+1} - f^n) \end{cases} \tag{3.36}$$

with the proximity operators $\operatorname{prox}_{\sigma(\frac{\alpha}{2p}\|\cdot\|_{p;q}^p)^*}$ as defined in Appendix 3.C for selected p and q.

3.4 Graph Cuts for Energy Minimization

In this section, we want to introduce the ideas that Kolmogorov et. al. discuss in [43] to minimize an energy functional on a graph G for a labeling function $\mathbf{1}_B \in \{0,1\}^{|V|}$ given as

$$\mathcal{J}(\mathbf{1}_B) = \sum_{u\in V} \mathcal{D}_u(\mathbf{1}_B(u)) + \sum_{(u,v)\in E_d} \mathcal{R}_{u,v}(\mathbf{1}_B(u), \mathbf{1}_B(v)) \tag{3.37}$$

with the *directed edge set* $E_d = \{(u,v) \in E \mid u < v\}$. The data-term $\mathcal{D}_u$ is the cost of assigning a label to $u \in V$ and $\mathcal{R}_{u,v}$ measures the cost of $u, v \in V$ to have the same or different labeling. Optimizing this for $\mathbf{1}_B$ would generate a *binary splitting* of the vertices in V into two sets $B \subset V$ and $B^c \subset V$. In [43] they suggest to do so using a minimum s-t-cut, which is a minimum graph cut between two terminals s and t in a directed graph. They state that if an energy $\mathcal{J}$ is *graph-representable* (cf. [43, Lemma 3.2]) one can compute the exact minimum of J in polynomial time by computing the s-t-cut. In [43, Theorem 4.1] they state that for a binary problems as in (3.37) the energy $\mathcal{J}$ is *graph-representable* if, and only if, the following holds

$$\mathcal{R}_{u,v}(0,0) + \mathcal{R}_{u,v}(1,1) \leq \mathcal{R}_{u,v}(0,1) + \mathcal{R}_{u,v}(1,0). \tag{3.38}$$

If this inequality holds $\mathcal{J}$ is called regular. To build the graph G_{flow} - which we denote here *flow graph*, because we solve it by computing the maximum flow - that represents the energy $\mathcal{J}$ one first introduces two *terminals* to V, i.e., s as the *source* and t as the *sink*, and store them in a new vertex set

$$V_{flow} = V \cup \{s,t\}. \tag{3.39}$$

To compute the new edge set E_{flow} we follow the instructions in [43, Section 4.1]. For every vertex $u \in V$ they propose to add an edge (s,u) from the source s to a node u with capacity

$$c(s,u) = \mathcal{D}_u(1) - \mathcal{D}_u(0)$$

if $\mathcal{D}_u(0) < \mathcal{D}_u(1)$ or an edge (u,t) from a node u to the sink t with the capacity

$$c(u,t) = \mathcal{D}_u(0) - \mathcal{D}_u(1)$$

if $\mathcal{D}_u(1) < \mathcal{D}_u(0)$. Every edge $(u, v) \in E$ is assigned with the weight

$$c(u, v) = \mathcal{R}_{u,v}(0, 1) + \mathcal{R}_{u,v}(1, 0) - \mathcal{R}_{u,v}(0, 0) - \mathcal{R}_{u,v}(1, 1). \tag{3.40}$$

Hence, we get the following edge set for the flow graph

$$E_{flow} = E \cup \{(s, u) \mid \mathcal{D}_u(0) < \mathcal{D}_u(1)\} \cup \{(u, t) \mid \mathcal{D}_u(1) < \mathcal{D}_u(0)\} \tag{3.41}$$

and the weights of the flow graph that we call *capacities* from now on are given as

$$\begin{cases} c(s, u) = \mathcal{D}_u(1) - \mathcal{D}_u(0), & \text{if } \mathcal{D}_u(0) < \mathcal{D}_u(1), \\ c(u, t) = \mathcal{D}_u(0) - \mathcal{D}_u(1), & \text{if } \mathcal{D}_u(1) < \mathcal{D}_u(0), \\ c(u, v) = \mathcal{R}_{u,v}(0, 1) + \mathcal{R}_{u,v}(1, 0) & \\ \qquad - \mathcal{R}_{u,v}(0, 0) - \mathcal{R}_{u,v}(1, 1), & \forall\ (u, v) \in E. \end{cases} \tag{3.42}$$

The flow graph $G_{flow} = (V_{flow}, E_{flow}, c)$ is the graph representation of the energy functional $\mathcal{J}$ in (3.37). Thus, by the statement from above computing a minimum s-t-cut or a maximum flow from s to t gets the optimum of the energy $\mathcal{J}$. Since the solution $\mathbf{1}_B^*$ assigns to every vertex $u \in V$ a 1 if $u \in B$ and a 0 if $u \in B^c$ we can find a binary partition of the vertex set V by B. Note that B does not have to be connected and can consist of multiple connected components.

Appendix

3.A Reformulating p-q-Laplace Operator

In this section, we derive the weighted graph p-q-Laplace operator defined as in (3.18). All computations done below are based on the definition of the divergence in Definition 3.13 and the weighted gradient operator ∇_w in Definition 3.11. Additionally, it is assumed that we have an undirected graph as in Definition 3.2 with symmetric weights, i.e., $w(u,v) = w(v,u)$, and thus $\partial_u f(v) = -\partial_v f(u)$ as described in Section 3.1.2. Then the p-q-Laplace operator can be reformulated as follows:

$$
\begin{aligned}
\Delta_{w,p,q} f(u) &= \frac{1}{2} \operatorname{div}_w \left(\|\nabla_w f\|_q^{p-q} \left(\nabla_w f_j |\nabla_w f_j|^{q-2} \right)_{j=1}^{d} \right) \\
&\overset{(3.11)}{=} -\frac{1}{2} \sum_{v \sim u} \sqrt{w(u,v)} \|\nabla_w f(u,v)\|_q^{p-q} \\
&\qquad \left(\nabla_w f(v,u)_j |\nabla_w f(v,u)_j|^{q-2} - \nabla_w f(u,v)_j |\nabla_w f(u,v)_j|^{q-2} \right)_{j=1}^{d} \\
&= \sum_{v \sim u} \sqrt{w(u,v)} \|\nabla_w f(u,v)\|_q^{p-q} \left(\nabla_w f(u,v)_j |\nabla_w f(u,v)_j|^{q-2} \right)_{j=1}^{d} \\
&= \sum_{v \sim u} \sqrt{w(u,v)} \|w(u,v)^{\frac{1}{2}} (f(v) - f(u))\|_q^{p-q} \\
&\qquad \left(w(u,v)^{\frac{1}{2}} (f(v)_j - f(u)_j) |w(u,v)^{\frac{1}{2}} (f(v)_j - f(u)_j)|^{q-2} \right)_{j=1}^{d} \\
&= \sum_{v \sim u} w(u,v)^{\frac{p}{2}} \|f(v) - f(u)\|_q^{p-q} \left(|f(v)_j - f(u)_j|^{q-2} (f(v)_j - f(u)_j) \right)_{j=1}^{d}.
\end{aligned}
$$

Hence, for every $u \in V$ and $j \in \{1, \dots, d\}$ we can obtain the following pointwise formulation

$$
\Delta_{w,p,q} f(u)_j = \sum_{v \sim u} w(u,v)^{\frac{p}{2}} \|f(v) - f(u)\|_q^{p-q} |f(v)_j - f(u)_j|^{q-2} (f(v)_j - f(u)_j). \tag{3.43}
$$

3.B Derivative of the p-q-TV Regularizer

In this appendix section, we want to show that the derivative of the differentiable parts of the p-q-TV regularization in Definition 3.17 is equivalent to the p-q-Laplace operator from Definition 3.18. Note that for different selections of p and q the differentiability differs. Since

we have that

$$\begin{aligned}\frac{\partial}{\partial f(u)_j}\|\nabla_w f\|_{p;q}^p &= \frac{\partial}{\partial f(u)_j}\sum_{(u,v)\in E} w(u,v)^{\frac{p}{2}}\left(\sum_{j=1}^{d}|f(v)_j - f(u)_j|^q\right)^{\frac{p}{q}} \\ &= \sum_{(u,v)\in E} w(u,v)^{\frac{p}{2}}\frac{\partial}{\partial f(u)_j}\|f(v)-f(u)\|_q^p\end{aligned} \tag{3.44}$$

we see that we can compute the derivative for each edge $(u,v)\in E$ and $1\leq j\leq d$ combination individually and that the differentiability is also pointwise. Thus, we can split the sum up into differentiable and non-differentiable parts. It is important to note that if we compute the derivative as in (3.44) for every edge $(u,v)\in E$ there also exists the opposite edge $(v,u)\in E$ with $w(u,v)=w(v,u)$. Thus, we have to compute the derivative of $\partial_{f(u)_j}\|f(u)-f(v)\|_q^p$ and $\partial_{f(u)_j}\|f(v)-f(u)\|_q$. We define the set of directed edges as

$$E_d = \{(u,v)\in E \mid u < v\}.$$

Note that as $V=\{1,\dots,N\}$ is a index set we can compare the index associated with the vertices $u,v\in V$. Let us assume that the term $\|f(v)-f(u)\|_q^p$ is differentiable for a selected edge $(u,v)\in E_d$. Then the derivative along the edge (u,v) can be computed as follows:

$$\begin{aligned}\frac{\partial}{\partial f(u)_j}\|f(v)-f(u)\|_q^p &= \frac{\partial}{\partial f(u)_j}\left(\sum_{j=1}^{d}|f(v)_j-f(u)_j|^q\right)^{\frac{p}{q}} \\ &= \frac{p}{q}\left(\sum_{j=1}^{d}|f(v)_j-f(u)_j|^q\right)^{\frac{p}{q}-1}\frac{\partial}{\partial f(u)_j}|f(v)_j-f(u)_j|^q \\ &= \frac{p}{q}\|f(v)-f(u)\|_q^{p-q}\frac{\partial}{\partial f(u)_j}|f(v)_j-f(u)_j|^q\end{aligned} \tag{3.45}$$

where we used that $\frac{p}{q}-1=\frac{p-q}{q}$. For the derivative of the right term, we assume for now that $f(v)_j\neq f(u)_j$, which implies differentiability at this point. Then applying the chain rule multiple times yields

$$\begin{aligned}
&\frac{\partial}{\partial f(u)_j}|f(v)_j - f(u)_j|^q \\
&\quad = q\ |f(v)_j - f(u)_j|^{q-1}\frac{\partial}{\partial f(u)_j}|f(v)_j - f(u)_j| \\
&\quad = q\ |f(v)_j - f(u)_j|^{q-1}\frac{f(v)_j - f(u)_j}{|f(v)_j - f(u)_j|}\frac{\partial}{\partial f(u)_j}(f(v)_j - f(u)_j) \\
&\quad = q\ |f(v)_j - f(u)_j|^{q-2}(f(v)_j - f(u)_j)(-1) \\
&\quad = q\ |f(v)_j - f(u)_j|^{q-2}(f(u)_j - f(v)_j).
\end{aligned} \tag{3.46}$$

Thus, we can show that if $f(v) \neq f(u)$ and $f(v)_j \neq f(u)_j$ for some $1 \leq j \leq d$, the derivative is given as

$$\begin{aligned}
&\frac{\partial}{\partial f(u)_j}\|f(v) - f(u)\|_q^p \\
&\quad = p\ \|f(v) - f(u)\|_q^{p-q}|f(v)_j - f(u)_j|^{q-2}(f(u)_j - f(v)_j).
\end{aligned} \tag{3.47}$$

For every edge $(v,u) \in E$ that is the opposite edge $(u,v) \in E_d$ we can compute the derivative easily and see that

$$\partial_{f(u)_j}\|f(u) - f(v)\|_q^p = \partial_{f(u)_j}\|f(v) - f(u)\|_q^p. \tag{3.48}$$

Thus, for each two-sided edge we have the same result and can compute the final derivative of the differentiable parts of R given as

$$(\nabla R_S)_{u,j} = 2p \sum_{(u,v)\in E_d} w(u,v)^{\frac{p}{2}}|f(v)_j - f(u)_j|^{q-2}\frac{f(u)_j - f(v)_j}{\|f(v) - f(u)\|_q^{q-p}}. \tag{3.49}$$

The 2 here comes from the two-sided edges and that we have the same derivative in both directions. As stated above, if $f(v) = f(u)$, we would divide by zero if $p < q$ or if $f(v)_j = f(u)_j$ for some $1 \leq j \leq d$, we divide by zero for $q < 2$. Thus, we have to investigate these settings individually.

Let $f(v) = f(u)$ and $p < q$ then we can use the directional differentiability and compute the directional derivative at $f(u) - f(v) = 0$ for any direction $\gamma \in \mathbb{R}^d$ as

$$\begin{aligned}\lim_{\varepsilon \to 0} \frac{\|f(v) - f(u) + \varepsilon\gamma\|_q^p - \|f(v) - f(u)\|_q^p}{\varepsilon} \\ = \lim_{\varepsilon \to 0} \frac{\|\varepsilon\gamma\|_q^p}{\varepsilon} = \lim_{\varepsilon \to 0} \frac{\varepsilon^p \|\gamma\|_q^p}{\varepsilon} \\ = \lim_{\varepsilon \to 0} \varepsilon^{p-1} \|\gamma\|_q^p = \begin{cases} 0, & \text{if } p > 1 \\ \|\gamma\|_q^p, & \text{if } p = 1. \end{cases}.\end{aligned}$$

Hence, we can deduce that as long as $1 < p < q$ the term in (3.44) is differentiable in $f(v) - f(u) = 0$ with the derivative equals 0, since it is a continuous function. If $p = 1 < q$ it is *non-differentiable* for $f(u) = f(v)$.

The second setting is if $f(v)_j = f(u)_j$ for some $1 \leq j \leq d$ and $q < 2$. Then the derivative $\frac{\partial}{\partial f(u)_j} |f(v) - f(u)|^q$ cannot be computed as in (3.46), since we would divide by zero. We again have to compute the directional derivatives and check if they are all equal to 0. Let $\gamma \in \mathbb{R}$ be some value, then we compute

$$\begin{aligned}\lim_{\varepsilon \to 0} \frac{|f(v)_j - f(u)_j + \varepsilon\gamma|^q - |f(v)_j - f(u)_j|^q}{\varepsilon} \\ = \lim_{\varepsilon \to 0} \frac{|\varepsilon\gamma|^q}{\varepsilon} \\ = \lim_{\varepsilon \to 0} \frac{\varepsilon^q |\gamma|^q}{\varepsilon} \\ = \lim_{\varepsilon \to 0} \varepsilon^{q-1} |\gamma|^q = \begin{cases} 0, & \text{if } q > 1, \\ |\gamma|^q & \text{if } q = 1. \end{cases}\end{aligned}$$

From this we can deduce that for $q > 1$ the term in (3.44) is differentiable in $f(v)_j - f(u)_j = 0$ and has the derivative 0. For $q = 1$, which implies that $p = 1$ as we only look at $1 \leq p \leq q$, it is non-differentiable if $f(v)_j = f(u)_j$.

With these different settings in mind we can compute the differentiable parts of the p-q-TV regularization in the following and state the corresponding Laplace operators.

Let $\mathbf{p = q = 1}$:
This is the *anisotropic* TV regularization that is non-differentiable for edges $(u, v) \in E$ and $1 \leq j \leq d$ such that $f(v)_j = f(u)_j$. Let $S_{1;1}^j = \{(u, v) \in E_d \times \{1, \ldots, d\} \mid f(v)_j \neq f(u)_j\}$ be the set where R is differentiable for $p = q = 1$. We deduce from (3.49) that this setting yields for $u \in V$ and $1 \leq j \leq d$

$$(\nabla R_S)_{u,j} = 2 \sum_{(u,v) \in S_{1;1}^j} \sqrt{w(u, v)} \frac{(f(u)_j - f(v)_j)}{|f(v)_j - f(u)_j|}.$$

Then we get the following Laplace operator as defined in Appendix 3.A for $p = q = 1$

$$(\Delta_{w,1,1} f)_{u,j} = \frac{1}{2}(\nabla R_S)_{u,j} = \sum_{(u,v)\in S_{1;1}^j} \sqrt{w(u,v)}\ \mathrm{sign}(f(u)_j - f(v)_j). \tag{3.50}$$

Note that we drop the non-differentiable parts here.

Let $\mathbf{p = 1 < q < 2}$:
This regularizer is non-differentiable at the edges $(u, v) \in E$ for which $f(v)_j = f(u)_j$ with $1 \leq j \leq d$. Thus, we can define the set where the p-q-TV term is differentiable as $S_{1;q,j} = \{(u, v) \in E_d \mid f(u)_j \neq f(v)_j\}$. For the differentiable part $S_{1;q,j}$, we thus obtain the following derivative by plugging into (3.49)

$$(\nabla R_S)_{u,j} = 2 \sum_{(u,v)\in S_{1;q,j}} \sqrt{w(u,v)} |f(v)_j - f(u)_j|^{q-2} \frac{f(u)_j - f(v)_j}{\|f(v) - f(u)\|_q^{q-1}}. \tag{3.51}$$

Taking into account Appendix 3.A for $p = 1 < q < 2$ the Laplace operator is equal to the derivative of the differentiable parts $S_{1;q}$ given as

$$\Delta_{w,1,q} f = \frac{1}{2} \nabla R_{S_{1;q}}. \tag{3.52}$$

Let $\mathbf{p = 1, q \geq 2}$:
As long as $p = 1$ this regularizer is non-differentiable at the edges $(u, v) \in E$ for which $f(v) = f(u)$. Thus, we can define the set where the p-q-TV term is differentiable as $S_{1;q} = \{(u, v) \in E_d \mid f(u) \neq f(v)\}$. For the differentiable part $S_{1;q}$, we thus obtain the following derivative by plugging into (3.49) as

$$(\nabla R_S)_{u,j} = 2 \sum_{(u,v)\in S_{1;q}} \sqrt{w(u,v)} |f(v)_j - f(u)_j|^{q-2} \frac{f(u)_j - f(v)_j}{\|f(v) - f(u)\|_q^{q-1}}. \tag{3.53}$$

Taking into account Appendix 3.A for $p = 1, q \geq 2$ the Laplace operator is equal to the derivative of the differentiable parts $S_{1;q}$ given as

$$\Delta_{w,1,q} f = \frac{1}{2} \nabla R_{S_{1;q}}. \tag{3.54}$$

Let $\mathbf{1 < p,\ q \geq 2,\ p \leq q}$:
In this case the p-q-TV regularizer R is *differentiable* for every edge $(u, v) \in E$ and every $1 \leq j \leq d$. Then the derivative is given by (3.49) if $f(u)_j \neq f(v)_j$ and equal to zero if $f(v)_j = f(u)_j$ for some edge $(u, v) \in E$ and dimension $1 \leq j \leq d$. Thus, we can introduce the set of $\mathcal{M}_j = \{(u, v) \in E_d \mid f(u)_j \neq f(v)_j\}$ which are the edges where it is not 0. This yields

$$(\nabla R_S)_{u,j} = (\nabla R)_{u,j} = 2 \sum_{(u,v)\in \mathcal{M}_j} w(u,v)^{\frac{p}{2}} |f(v)_j - f(u)_j|^{q-2} \frac{f(u)_j - f(v)_j}{\|f(v) - f(u)\|_q^{q-p}} \tag{3.55}$$

for which we can again see that

$$\Delta_{w,p,q} f = \frac{1}{2} \nabla R$$

The special case for $p = q = 2$ is the well-known standard Laplace operator that is given as

$$\Delta_w f(u) = \sum_{(u,v) \in E} w(u,v)(f(u) - f(v)). \tag{3.56}$$

Everything combined, we can finally deduce using the computations of Appendix 3.A and (3.49) that

$$(\Delta_{w,p,q} f)_{u,j} = \frac{1}{2} (\nabla R_S)_{u,j}. \tag{3.57}$$

This means that the weighted p-q-Laplace is exactly the same as $\frac{1}{2}$ times the derivative of the differentiable parts of the weighted p-q-TV regularizer.

3.C Proximity Operator of the p-q-TV Regularizer

In this appendix section, we aim to compute the convex conjugate $F^* = (\frac{\alpha}{2} \| \cdot \|_{p;q})^*$. As shown in [81], the dual norm of the norm $\| \cdot \|_{p;q}$ is given by $\| \cdot \|_{p^*;q^*}$ with $\frac{1}{p} + \frac{1}{p^*} = 1$, $\frac{1}{q} + \frac{1}{q^*} = 1$, and $p, q \geq 1$. Hence, it follows that

$$(\beta \| \cdot \|_{p;q})^*(y) \;=\; \delta_{B_{p^*;q^*}(\beta)} \;=\; \begin{cases} 0, & \|y\|_{p^*;q^*} \leq \beta \\ \infty, & \text{else} \end{cases}$$

with $y \in \mathcal{H}(E)$.

We recall that the proximity operator of the characteristic function δ_C over a convex set $C \subset X$ is a projection, which is given as

$$\begin{aligned} \operatorname{prox}_{\tau \delta_C}(z) \;&=\; \operatorname*{argmin}_{x \in X} \left\{ \frac{1}{2\tau} \|x - z\|_2^2 + \delta_C(x) \right\} \\ &=\; \operatorname*{argmin}_{x \in C} \left\{ \frac{1}{2\tau} \|x - z\|_2^2 \right\} \;=:\; \operatorname{proj}_C(z). \end{aligned} \tag{3.58}$$

Consequently, the proximity operator for $C = B_{p^*;q^*}(\beta)$ is the projection of an element $z \in \mathcal{H}(E)$ onto the p^*, q^*-ball of radius β. Thus, we get

$$\operatorname{prox}_{\tau F^*}(z) \;=\; \operatorname{proj}_{B_{p^*;q^*}(\beta)}(z) \tag{3.59}$$

In the following we see that for different combinations of p and q we get different proximity operators. We distinguish between three cases.

Case 1: $q > 1, p = 1$

For the special case $p = 1$ we get $p^* = \infty$, and thus the dual norm becomes $\|y\|_{\infty,q^*} = \max_{(u,v)\in E} \left\{ \|y(u,v)\|_{q^*} \right\}$. Then

$$B_{\infty;q^*} = \left\{ y \in X^* \mid \|y\|_{\infty,q^*} \leq \alpha \right\} = \left\{ y \in X^* \mid \|y(u,v)\|_{q^*} \leq \alpha, \forall (u,v) \in E \right\}$$

The proximity operator for $p = 1$, $q > 1$ and every $(u,v) \in E$ is just a projection of every $y(u,v)$ onto the ball $B_{q^*}(\alpha)$.

In conclusion we get the proximity operator of F^* as

$$\begin{aligned} \operatorname{prox}_{\tau F^*}(z) &= \operatorname*{argmin}_{y\in X^*} \left\{ \frac{1}{2\tau}\|y - z\|_2^2 + F^*(y) \right\} \\ &= \operatorname{proj}_{B_{\infty,q^*}(\alpha)}(z) \\ &= \Big(\frac{\alpha \; z(u,v)}{\max(\alpha, \|z(u,v)\|_{q^*})} \Big)_{(u,v)\in E}. \end{aligned}$$

Case 2: $q = 1, p = 1$

When $q = 1$ and $p = 1$, then $q^* = \infty$ and $p^* = \infty$. Then the ball becomes

$$\begin{aligned} B_{\infty;\infty}(\alpha) &= \{ y \in \mathcal{H}(E) \mid \|y(u,v)\|_\infty \leq \alpha, \forall (u,v) \in E \} \\ &= \{ y \in \mathcal{H}(E) \mid |y(u,v)_j| \leq \alpha, \forall (u,v) \in E, j \in [1,d] \}, \end{aligned}$$

from which it follows that the proximity operator becomes

$$\begin{aligned} \operatorname{prox}_{\tau F^*}(z) &= \operatorname{proj}_{B_{\infty;\infty}(\alpha)}(z) \\ &= \left(\frac{\alpha \; z(u,v)_j}{\max(\alpha, |z(u,v)_j|)} \right)_{(u,v,j)\in E\times[1,d]}. \end{aligned}$$

Case 3: $q \geq 1, 1 < p < \infty$

In this special case, one has to compute the projection onto the p^*, q^*-ball numerically as there does not exist any known closed-form solution, except for the case of $p = 2$. Since the constraints are smooth one is able to use a standard Newton method for computing this projection. Note that usually this case is not relevant in most applications from imaging or machine learning in contrast to cases 1 and 2 above.

Case 4: $q \geq 1, p = \infty$

In this case one has to project onto $1, q^*$-balls. To compute these projections there exist efficient numerical algorithms, see, e.g., [24, 80].

4

Cut-Pursuit

In the former chapters we have introduced variational problems on graphs and a graph variant of the well-known primal-dual algorithm in Algorithm 3.20 to solve convex optimization problems on graphs, e.g. ROF in Algorithm 3.22, weighted ROF Algorithm 3.24. This enables us to solve convex problems for any given data structure which can be represented by a finite weighted graph. This opens up the application of convex problems in many fields like e.g. (unorganized) point clouds, networks, non-local data using a unified framework. The downside of using graphs for processing data is that the efficiency of graph algorithms is highly dependent on the number of vertices *and* the number of edges. This means that if we have a huge set of data points and a densely connected graph iterative graph algorithms tend to scale in complexity proportional to the complexity of the graph. Hence, using primal-dual on graphs to solve convex problems tend to be inefficient for complex graphs as we show in numerical experiments in Section 6.2. On the implementation side it is additionally not trivial to store and compute the graph gradient efficiently, since some data points might be connected with other points that are not in the same memory location such that look up is inefficient. Consequently, when we compare the computational times of the implementations of a primal-dual algorithm on images with finite differences versus the primal-dual on graphs for the same image, the finite difference primal-dual is more effective in most cases due to these problems. To tackle these inefficiencies Landrieu et. al. introduce the *Cut-Pursuit algorithm* in [46] to solve variational problems on graphs efficiently. More details on Cut-Pursuit can be found in [47]. In this chapter, we derive and understand Cut-Pursuit by extending the family of solvable problems to multi-dimensional, channel-isotropic regularizers. We introduce a generic Cut-Pursuit algorithm and derive the special case of the p-q-TV regularizer for different choices of p, q.

Throughout this chapter we work on undirected finite weighted graphs $G = (V, E, w)$ as defined in Definition 3.2. The general variational problem that we want to solve is to minimize the energy functional

$$\operatorname*{argmin}_{f \in \mathcal{H}(V)} \left\{ J(f) = D(f, g) + \frac{\alpha}{2} R(f) \right\} \tag{4.1}$$

on the set $\mathcal{H}(V)$ for which $\alpha > 0$ is a fixed regularization parameter The functional D is a

differentiable, convex data fidelity term with the original data given as g. We study a more general problem then the authors in [46] by introducing multiple dimensions and also give the possibility to have (channel-)isotropic regularizers (cf. Section 3.1.4). We discuss two types of regularization namely *anisotropic* and *(channel-)isotropic* regularization.

Definition 4.1 (Generic Anisotorpic Regularizer)**.**
Let $G = (V, E, w)$ be a finite weighted graph and $f \in \mathcal{H}(V)$ a vertex function. Than the anisotropic regularizer *is defined as*

$$\begin{aligned} R_a(f) &= \sum_{(u,v)\in E} \Phi_a(w(u,v), f(v) - f(u)) \\ &= \sum_{(u,v)\in E} \sum_{j=1}^{d} \Phi_a^j(w(u,v), f(v)_j - f(u)_j) \end{aligned} \tag{4.2}$$

with $\Phi_a^j(x, y)$ a absolutely λ-homogenous function in the first and absolutely p-homogenous functional in the second argument, i.e., $\Phi_a^j(kx, ty) = |k|^\lambda |t|^p \Phi_a^j(x, y)$, with $\Phi_a^j(x, 0) = 0$ and which is non-differentiable for $y = 0$, but directional differentiable everywhere. Here $\Phi_a = \sum_{j=1}^{d} \Phi_a^j$.

Definition 4.2 (Generic Channel-Isotorpic Regularizer)**.**
Let $G = (V, E, w)$ be a finite weighted graph and $f \in \mathcal{H}(V)$ a vertex function. Than the channel-isotropic regularizer *is defined as*

$$R_i(f) = \sum_{(u,v)\in E} \Phi_i(w(u,v), f(v) - f(u)) \tag{4.3}$$

with $\Phi_i(x, y)$ a absolutely λ-homogenous function in the first and absolutely p-homogenous functional in the second argument, i.e., $\Phi_i(kx, ty) = |k|^\lambda |t|^p \Phi_i(x, y)$, with $\Phi_i(x, 0) = 0$ and which is non-differentiable for $y = 0$, but directional differentiable everywhere.

We subsume both regularizers under the notation R and specify when needed. Note that for the special case for $d = 1$ and setting $\Phi(w(u,v), f(v) - f(u)) = w(u,v)\phi(f(v) - f(u))$ we have the same conditions as given in [46]. Thus, we have not only extended the problem with a multi-dimensional isotropic regularizer, but also have more suitable functions Φ_i and Φ_a that fulfill these conditions. Additionally, note that both regularizers discriminate discontinuities in an anisotropic and isotropic fashion respectively. For $p > 1$ they are also differentiable in y everywhere.

Lemma 4.3 (p-homogenous Φ with $p > 1$)**.**
Let $\Phi(x)$ be an absolutely p-homogenous function as defined in (2.5) with $p > 1$. Then Φ is differentiable in 0, and thus, differentiable everywhere.

Proof.
Let $v \in \mathbb{R}^d$ be any direction. Then every directional derivative is computed as

$$\begin{aligned}\lim_{\varepsilon \to 0} \frac{\Phi(0+\varepsilon v) - \Phi(0)}{\varepsilon} &= \lim_{\varepsilon \to 0} \frac{\Phi(\varepsilon v)}{\varepsilon} = \lim_{\varepsilon \to 0} \frac{\varepsilon^p \Phi(v)}{\varepsilon} \\ &= \lim_{\varepsilon \to 0} \varepsilon^{p-1} \Phi(v) = 0.\end{aligned}$$

Since every direction yields the directional derivative as 0 and Φ is continuous it is differentiable in 0 and $\Phi'(0) = 0$. □

This lemma is directly applicable to the definition of Φ_i and Φ_a that are p-homogeneous in the second argument. Before continuing we state that the anisotropic problem can be decoupled over the d channel dimensions while preserving the solution.

Lemma 4.4 (Decoupeling of Anisotropic Problem)**.**
Solving the anisotropic problem

$$f^* = \operatorname*{argmin}_{f \in \mathcal{H}(V;\mathbb{R}^d)} \sum_{j=1}^{d} D_j(f) + \frac{\alpha}{2} \sum_{(u,v)\in E} \Phi_a^j(w(u,v), f(v)_j - f(u)_j) \tag{4.4}$$

is equivalent to solving the problem for each dimension j separately as

$$f_j^* = \operatorname*{argmin}_{f_j \in \mathcal{H}(V;\mathbb{R})} D_j(f) + \frac{\alpha}{2} \sum_{(u,v)\in E} \Phi_a^j(w(u,v), f(v)_j - f(u)_j). \tag{4.5}$$

Then the solution of the overall problem can be computed as $f^* = (f_j^*)_{j=1}^d$.

Remark 4.5.
In one dimension for $d = 1$ the isotropic problem is equivalent to the anisotropic problem. By Lemma 4.4 we see that we can solve the overall anisotropic problem by computing it for each dimension separately. Thus, if we derive the Cut-Pursuit algorithm for the multi-dimensional isotropic case we can directly deduce the Cut-Pursuit algorithm for the multi-dimensional anisotropic case.

To this end, we state the necessary ideas and derive the formulations for the multi-dimensional isotropic case in the following sections and in the end show the anisotropic case as a special case. We show in Appendix 4.A that the p-q-TV regularizer is a special case of the general regularizer Definition 4.1 and Definition 4.2, and hence, the former lemmas hold for these.

In the following sections we give an exhaustive derivation of the Cut-Pursuit algorithm for the general minimization problem (4.1). We show the key idea of exploiting the piecewise constant structure of the solutions of (4.1) and derive an efficient alternating algorithm to solve these. We also show how this more general Cut-Pursuit can be applied to ROF problems with a p-q-TV regularization.

4.1 Introduction to Partitioning and Reduced Problems

As we have already discussed in former sections, especially in Section 3.3,applying a direct method on complex graph structures can lead to inefficient implementations and long computational run times. The main idea of [46] comes from the fact that the solution of an ROF denoising problem tends to be piecewise constant. Let us assume we have some given undirected graph $G = (V, E, w)$ as was defined in Definition 3.2 and some given vertex function $f \in \mathcal{H}(V)$ that is piecewise constant on n segments $A_i \subset V$. We assume to know the segments A_i here beforehand. Let $\Pi = \{A_i \subset V \mid V = \dot{\bigcup}_{i=1}^{n} A_i\}$ be a *partition* of V and the *segments* $A_i, A_j \in \Pi$ pairwise disjoint, i.e., $A_i \cap A_j = \emptyset$. Defining the set $V_\Pi = \{1, \dots, n\}$ as the numeration of the segments in Π we can define *reduced (vertex) functions* $c \in \mathcal{H}(V_\Pi)$. Then we can find for every reduced function $c \in \mathcal{H}(V_\Pi)$ a function $f_c \in \mathcal{H}(V)$ such that

$$f(u) = c(A) = c_A, \quad \forall u \in A, \quad \forall A \in \Pi. \tag{4.6}$$

Note that $c = (c_A)_{A\in\Pi} \in \mathbb{R}^{N_\Pi}$ with $N_\Pi = n = |\Pi|$ as the number of segments. We define the set of piecewise constant functions in $\mathcal{H}(V)$ that are constant on segments in Π as

$$\mathcal{S}_\Pi := \{f_c \in \mathcal{H}(V) \mid \exists c \in \mathcal{H}(V_\Pi) \text{ s.t. } f(u) = c_A, \ \forall u \in A, \ \forall A \in \Pi\}. \tag{4.7}$$

Thus, we for every function that is piecewise constant on the segments of Π we can find a reduced function in $\mathcal{H}(V_\Pi)$ and vice versa, i.e.,

$$\mathcal{S}_\Pi \cong \mathcal{H}(V_\Pi). \tag{4.8}$$

We denote a piecewise constant vertex function on V that is related to a function $c \in \mathcal{H}(V_\Pi)$ as $f_c \in \mathcal{H}(V)$. Note that partition Π and V_Π are actually isomorphic since every segment A_i is unique in Π, and thus corresponds to a unique vertex $i \in V_\Pi$. So we just refer to Π from now on.

Let $f_c \in \mathcal{S}_\Pi$ and its related reduced function $c \in \mathcal{H}(\Pi)$. Let $\mathbf{1}_A \in \{0,1\}^N$ a vector as defined in (2.19) which is 1 where the index is in A and 0 else. Then the following identity holds

$$f_c = \sum_{A\in\Pi} \mathbf{1}_A c_A. \tag{4.9}$$

This sum can be realized using a matrix-vector multiplication of the matrix

$$P = (\mathbf{1}_{A_i}, \dots, \mathbf{1}_{A_n}) \in \{0,1\}^{N\times N_\Pi} \tag{4.10}$$

and with c - which is actually a vector in $\mathbb{R}^{N_\Pi}$ - yields

$$Pc = \sum_{A\in\Pi} \mathbf{1}_A c_A \in \mathcal{S}_\Pi. \tag{4.11}$$

We call this matrix operator P the *expansion operator* that maps - or expands - functions living in $\mathcal{H}(\Pi)$ to their related piecewise constant functions in the parent set $\mathcal{H}(V)$. This

notation enables us to use the following properties of P.

Lemma 4.6 (Properties of the Expansion Operator P)**.**
Let $\Pi = \{A_i \subset V \mid V = \dot{\bigcup}_{i=1}^{n} A_i\}$ be a partition of V as defined above and let the expansion operator P be defined as in (4.10). Then the following properties hold:

$$Pc \in \mathcal{H}(V), \tag{i}$$

$$P^*P = \operatorname{diag}(|A_i|, \dots, |A_{N_\Pi}|), \tag{ii}$$

$$P^*\nu = \left(\nu_A^B\right)_{A\in\Pi} \text{ for any } \nu \in \mathbb{R}^{N\times d} \text{ with } \nu_A^B = \sum_{u\in A} \nu(u) \in \mathbb{R}^d. \tag{iii}$$

We call Pc an *expansion* of reduced functions from $\mathcal{H}(\Pi)$ to a piecewise constant function $f_c \in \mathcal{H}(V)$, and P^*f a *reduction* of $f \in \mathcal{H}(V)$ to the reduced space $\mathcal{H}(\Pi)$ that sums up the values in every segment. Note that for applying the reduction P^* to a function $f \in \mathcal{H}(V)$ it does not have to be piecewise constant. We can plug these piecewise constant functions in $\mathcal{S}_\Pi$ into the general data-term of the (4.1) and reformulate them to only depend on the reduced function.

To see this on a simple example let us apply this on our special case with the L_2-data-term from Definition 2.2 and plug in piecewise constant function $f_c \in \mathcal{S}_\Pi$. We show the result by computing it for the more general case of a weighed L_2-data-term as introduced in Definition 2.6 in Appendix 4.B.1 which is given as

$$D_W(f_c, g) = \frac{1}{2}\|f_c - g\|_{2,W}^2 = \frac{1}{2}\|c - g_r\|_{2,W_P}^2 = D_{P^*WP}(c, g_r) \tag{4.12}$$

with $g_r = (P^*WP)^{-1}P^*Wg$ and $W_P = P^*WP$. For the unweighted L_2-data-term D with $W = I$ the reduced data is given as the mean over the data in each segment:

$$g_r = \left(\frac{\sum_{u\in A} g(u)}{|A|}\right)_{A\in\Pi}. \tag{4.13}$$

The question that arises here is if we could also build a *reduced graph* $G_r = (\Pi, E_r, w_r)$ on the partition Π that corresponds to the original graph G and the full given problem. Indeed, the construction of the reduced graph G_r depends on the regularizer R of the general minimization problem from (4.1). To find the correct reduced graph corresponding to the problem let us plug in a piecewise constant function $Pc = f_c \in \mathcal{S}_\Pi$ into R which yields the following.

Lemma 4.7.
Let $G = (V, E, w)$ be a finite weighted graph and Π a partition. Moreover, we have the expansion operator P given as in (4.10). Then for any piecewise constant function $f_c \in S_\Pi$ on Π with $Pc = f_c$ the regularizer $R(f_c)$ is given as

$$\begin{aligned} R(Pc) &= \sum_{(u,v)\in E} \Phi(w(u,v), (Pc)(v) - (Pc)(u)) \\ &= \sum_{\substack{(A,B)\in\Pi\times\Pi \\ A\cap B=\emptyset}} \sum_{\substack{(u,v)\in E \\ u\in A, v\in B}} \Phi(w(u,v), c_B - c_A). \end{aligned} \tag{4.14}$$

Proof.
Let us consider this for every edge $(u, v) \in E$ separately. First let us assume that we have $(u, v) \in E$ with $u, v \in A$ for some $A \in \Pi$. Then we have with $\Pi(w(u, v), 0) = 0$ that we define in Definition 4.1 and Definition 4.2 that

$$\Phi(w(u,v), (Pc)(v) - (Pc)(u)) = \Phi(w(u,v), c_A - c_A) = \Pi(w(u,v), 0) = 0.$$

Hence, we can drop every edge $(u, v) \in E$ that is inside of a segment, i.e., $(u, v) \in A \times A \cap E$ for all $A \in \Pi$. Hence, we only sum over edges between segments which is formally given as $(u, v) \in (A \times B) \cap E$ for any tuple $(A, B) \in \Pi \times \Pi$ with $A \neq B$. □

We see that this regularizer only depends on the reduced function $c \in \mathcal{H}(\Pi)$ and also sums up over all edges between two segments $A, B \in \Pi$ with $A \cap B = \emptyset$. To this end, let us define the set of edges E_{AB} between segments A and B as

$$E_{AB} = \{(u, v) \in E \mid u \in A, v \in B\}. \tag{4.15}$$

Consequently, we do not lose the connections between the segments and also not the weight information of edges in E_{AB} as we see in (4.14). Additionally, we can deduce from (4.14) that all edges in a segment $A \in \Pi$ have no influence on the solution, since their values are 0, and thus, dropped out from the summation. Only the edges E_{AB} between different segments $A, B \in \Pi$ are relevant. By these observations we can deduce that the reduced graph G_r has the *reduced edge set* E_r that is given as

$$E_r = \{(A, B) \in \Pi \times \Pi \mid (A \times B) \cap E \neq \emptyset\}. \tag{4.16}$$

The reduced edge set defines an edge for every pair of different segments that have at least one edge between them. Rewriting (4.14) with E_r and E_{AB} we can define the reduced regularizer $R_r : \mathcal{H}(\Pi) \to \mathbb{R}$ for R as

$$R_r(c) = R(Pc) = \sum_{(A,B)\in E_r} \sum_{(u,v)\in E_{AB}} \Phi(w(u,v), c_B - c_A). \tag{4.17}$$

The last and final step to get a reduced graph and the reduced problem is to define a reduced weight w_r that fits to the reduced regularizer. By definition Φ is λ-homogenous in the first and p-homogenous in the second argument, thus we can reformulate the reduced regularizer as follows

$$\begin{aligned} R_r(c) &= \sum_{(A,B)\in E_r} \sum_{(u,v)\in E_{AB}} w(u,v)^\lambda \Phi(1, c_B - c_A) \\ &= \sum_{(A,B)\in E_r} \Phi(1, (\sum_{(u,v)\in E_{AB}} w(u,v)^\lambda)^{\frac{1}{p}}(c_B - c_A)) \\ &= \sum_{(A,B)\in E_r} \Phi(1, w_r(A,B)(c_B - c_A)) \\ &= \Phi_r(\nabla_{w_r} c) \end{aligned} \tag{4.18}$$

for the *reduced weighting function* w_r defined as

$$w_r(A,B) = \big[\sum_{(A,B)\in E_r} w(u,v)^\lambda \big]^{\frac{1}{p}}. \tag{4.19}$$

With these information we can write the *reduced graph* (for Φ_i that is λ-homogenous in the first argument) as

$$\begin{aligned} G_r &= (\Pi, E_r, w_r), \\ E_r &= \{(A,B) \in \Pi \times \Pi \mid (A \times B) \cap E \neq \emptyset\}, \\ w_r(A,B) &= \big[\sum_{(u,v)\in E_{AB}} w(u,v)^\lambda \big]^{\frac{1}{p}}. \end{aligned} \tag{4.20}$$

We have shown that the *reduced energy* is given as

$$J_r(c) = D_r(c) + \frac{\alpha}{2} R_r(c) \tag{4.21}$$

on the reduced graph G_r, and is equal to the full energy J of the function $f_c = Pc$ for every $f_c \in \mathcal{S}_\Pi$, i.e.,

$$J_r(c) = J(f_c). \tag{4.22}$$

Hence, we have deduced how to derive the reduced graph G_r and the corresponding reduced energy J_r for any partition Π on a graph G. In the following proposition we gather all of these information, and furthermore state that solving the minimization problem over the set of piecewise constant functions on Π in $\mathcal{S}_\Pi$ is equivalent to solve the reduce problem with the reduced energy J_r.

Proposition 4.8 (From Piecewise Constant Functons to the Reduced Setting)**.**
Let $G = (V, E, w)$ be an undirected graph as in Definition 3.2 and Π some partition of the vertex set V. and define the corresponding reduced graph $G_r = (\Pi, E_r, w_r)$ *as in* (4.20) *with a* reduced edge set E_r *and a* reduced weighting w_r*. For any piecewise constant function $f_c \in \mathcal{S}_\Pi$ exists a* reduced function $c \in \mathcal{H}(\Pi)$ *living on the reduced graph G_r. Then for any $f_c = Pc \in \mathcal{S}_\Pi$ the reduced* (4.21) *and full energy* (4.1) *are equal, i.e.*

$$J(f_c) = J_r(c). \tag{4.23}$$

Additionally, the minimization over all piecewise constant functions in $\mathcal{S}_\Pi$ over Π is equivalent to the minimization over the reduced functions in $\mathcal{H}(\Pi)$, i.e.

$$f_c^* \in \operatorname*{argmin}_{f_c \in \mathcal{S}_\Pi} J(f_c) \;\Leftrightarrow\; c^* \in \operatorname*{argmin}_{c \in \mathcal{H}(\Pi)} J_r(c) \tag{4.24}$$

with $f_c^* = Pc^*$.

In Section 4.4.1 we apply the results from above onto a weighted p-q-ROF problem on graphs.

The method of splitting the original vertex set V into some partition and defining a reduced setting for a given problem enables us to make certain assumptions about solutions of the general problems on graphs (4.1), and the p-q-ROF problems in particular. Let us assume for now that we know the solution f^* the general problem (4.1) for some given data g and a selected regularization parameter $\alpha > 0$. Then we know that f^* can be assumed to be piecewise constant, and thus, we can find a partition Π and a function $c^* \in \mathcal{H}(\Pi)$ such that $Pc^* = f^*$. With Proposition 4.8 we can directly deduce that if we minimize the reduced energy J_r from (4.21) for functions in $\mathcal{H}(\Pi)$ we get the reduced solution c^* by minimization, i.e.

$$c^* = \operatorname*{argmin}_{c \in \mathcal{H}(\Pi)} J_r(c) \tag{4.25}$$

with $f^* = Pc^*$. Thus, if we know the partition Π corresponding to a piecewise constant solution the problem, we can solve the reduced problem instead of the full problem and get the full solution by applying the expansion operator P. Since Π has equal or less segments than V has vertices, it is more efficient to solve the reduced problem and to compute the same solution than solving on the vertex set V. However, in practice it is impossible to compute the correct partition Π for a given α before solving the problem. Hence, it would be desirable to have an optimization algorithm that generates partitions automatically.

Let us again assume that we have some given partition Π of V and the corresponding reduced graph G_r to a full graph G. Solving the reduced problem (4.25) on G_r yields a solution $c^* \in \mathcal{H}(\Pi)$. Let $f^* \in \mathcal{H}(V)$ be the solution to the corresponding full ROF problem (4.1) on G. If the selected partition Π does not describe all of the piecewise constant segments in f^* we cannot find a function $c \in \mathcal{H}(\Pi)$ such that $Pc = f^*$. Thus, the energy of the full problem is lower than the energy of the reduced problem, because otherwise Pc would be a

better solution than the real solution f^*, which is not possible. Hence, we can deduce that for any Π and corresponding reduced graph G_r holds

$$J(f^*) \leq J_r(c^*). \tag{4.26}$$

If $J(f^*) = J_r(c^*)$, we can build a solution with P, otherwise we cannot. Thus, if Π is not selected correctly we see that

$$J(f^*) < J_r(c^*). \tag{4.27}$$

Hence, the solution on Π cannot improve the original energy functional J any further. The idea to get closer to the real optimum would be to split Π up into more segments, such that the energy can be decreased even more. As we are already working on graphs we use graph cut methods, i.e., maxflow algorithms as introduced in Section 3.4.

4.2 Finer Partitioning via Graph Cuts

The aim of the following subsection is to state an energy functional that can be represented by a flow graph as we described in Section 3.4 and decreases the energy $J_r(c^*)$ given by the reduced result c^* by applying a graph cut. This then leads to a binary partitioning $B \subset V$ that we can use to refine the current partition Π.

4.2.1 Refining the Partition

In Section 3.4 we have recapped the ideas of [43] to build a flow graph to represent a binary labeling energy and minimize it using graph cuts. Assuming we have found the correct binary energy problem for a given partition Π and have computed a binary partitioning $B \subset V$ we could go ahead and build a new partition Π_{new} by splitting the segments $A \in \Pi$ via B by $A \cap B$, $A \cap B^c$ and taking the connected components as new segments in Π_{new}. Then we build a new reduced graph as we did in the former section and compute a reduced solution to the problem which will give us a better energy than of the former solution on Π did which is the aim of this subsection.

Starting with a current partition Π and a solution to the problem (4.1) f_Π we aim to find an energy functional that decreases the energy $J(f_\Pi)$ by a binary split $B \in V$. As above let us assume we have given an undirected graph $G = (V, E, w)$. Then we can solve the reduced problem (4.25) by minimizing J_r from (4.67) over $\mathcal{H}(\Pi)$ and find a solution $c^* \in \mathcal{H}(\Pi)$, i.e.,

$$c^* \in \operatorname*{argmin}_{c \in \mathcal{H}(\Pi)} J_r(c). \tag{4.28}$$

Let $f_\Pi = Pc^*$ be the piecewise representation of the reduced solution c^* in $\mathcal{H}(V)$. Then we already know that

$$J_r(c^*) = J(f_\Pi). \tag{4.29}$$

Let $f^* \in \mathcal{H}(V)$ the real solution of the general problem (4.1) and assume that $J(f^*) < J(f_\Pi)$.

This means that the solution c^* of the reduced problem (4.25) is not the optimal solution, and thus, the partition Π does not represent the solution f^* of the general problem. To get a finer and better partition, we want to find a binary splitting $B \subset V$ which is represented by $\mathbf{1}_B$ as defined in (2.19) such that the energy decreases. The easiest way to compute the new partition Π_{new} from B would be to take the intersections $A \cap B$ and $A \cap B^c$ as new segments for every $A \in \Pi$. Then we define the new partition as

$$\Pi_{new} = \{A \cap B, A \cap B^c \mid A \in \Pi\}. \tag{4.30}$$

The functions that are piecewise constant on Π_{new} are defined as

$$\tilde{f}_B = \sum_{A \in \Pi} (\mathbf{1}_{A \cap B} d_{A \cap B} + \mathbf{1}_{A \cap B^c} d_{A \cap B^c}) \in \mathcal{S}_{\Pi_{new}} \tag{4.31}$$

with any values $d_{A \cap B}, d_{A \cap B^c} \in \mathbb{R}^d$ as constants, and the corresponding function $d \in \mathcal{H}(\Pi_{new})$. Plugging these functions into the energy functional J yields

$$\begin{aligned} J(\tilde{f}_B) = D\Big(\sum_{A \in \Pi} (\mathbf{1}_{A \cap B} d_{A \cap B} + \mathbf{1}_{A \cap B^c} d_{A \cap B^c}), g \Big) \\ + \frac{\alpha}{2} R\Big(\sum_{A \in \Pi} (\mathbf{1}_{A \cap B} d_{A \cap B} + \mathbf{1}_{A \cap B^c} d_{A \cap B^c}) \Big). \end{aligned} \tag{4.32}$$

Finding an optimal partitioning described by $B \subset V$ such that the energy of $J(\tilde{f}_B)$ decreases the most compared to the current energy $J(f_\Pi)$ can be done by finding an optimal $B \subset V$ that minimizes

$$\begin{aligned} &\operatorname*{argmin}_{B \in \mathcal{P}(V)} \; J(\tilde{f}_B) - J(f_\Pi) \\ \Leftrightarrow \quad &\operatorname*{argmin}_{B \in \mathcal{P}(V)} \; D(\tilde{f}_B, g) - D(f_\Pi, g) + \frac{\alpha}{2} \; \Big(R(\tilde{f}_B) - R(f_\Pi) \Big) \end{aligned} \tag{4.33}$$

with some $d_{A \cap B}$ and $d_{A \cap B^c}$ that we assume to be fix for now. The problem that we face in this work later on is that we actually do not know the values of $d_{A \cap B}$ and $d_{A \cap B^c}$ and we would have to optimize these values in (4.33) to assure that the energy decreases, i.e., $J(\tilde{f}_B) \leq J(f_\Pi)$. We show in Appendix 4.C in detail how to select $d_{A \cap B}$, $d_{A \cap B^c}$ such that we can assure the decrease.

For the sake of clarity let us start by selecting one segment $A \in \Pi$ and studying the one-dimensional case for $d = 1$. Then $d_{A \cap B}, d_{A \cap B^c} \in \mathbb{R}$ for all $A \cap B$, $A \cap B^c \in \Pi_{new}$. As we are in the one-dimensional case here we know that the new values in $A \cap B$ and $A \cap B^c$ can be described by a deviation from the former solution c_A. Also it can be assumed that if we induce a split the one part will increase the current value while the other one decreases. Let us assume that the distance of deviation in both parts is the same and given by $\varepsilon > 0$. Then we can define the new values as

$$d_{A \cap B} = c_A + \varepsilon, \quad d_{A \cap B^c} = c_A - \varepsilon \tag{4.34}$$

for every $A \in \Pi$ and *one* $\varepsilon > 0$. Then we can rewrite $\tilde{f}_B$ as

$$\tilde{f}_B = \sum_{A \in \Pi} \left(1_{A \cap B}(c_A + \varepsilon) + \mathbf{1}_{A \cap B^c}(c_A - \varepsilon)\right). \tag{4.35}$$

This would split the segment into two new sets of vertices. The one set is where increasing the current value c_A by ε yields a lower energy and the other set is where decreasing the value lowers the energy. Even a very small $\varepsilon > 0$ can show the trend of the vertices in A to prefer deviating in the positive or negative direction from c_A. Hence, we can compute the split that yields B even for very small $\varepsilon > 0$ correctly. Then we can build the partition Π_{new} from B and Π and the corresponding reduced graph G_r. On this reduced graph we can solve a reduced ROF problem which would give a solution that has a better energy than f_Π.

For multi-dimensional data with $d > 1$ this problem is not as easy as for $d = 1$. Note that this case is only interesting for the (channel)-isotropic case of the regularizer Definition 4.2 since for the anisotropic regularizer Definition 4.1 we know from Lemma 4.4 that we can solve it by applying the ideas for the case $d = 1$ for every dimension. Thus, we will focus only on the isotropic regularizer R_i in Definition 4.2 from now on. As before, we want to deviate from the former solution f_Π on Π, but in higher dimensions we do have to choose a direction in which the deviation happens. In fact, we have infinity directions to choose from. We assume for now that we can choose such a direction for $d > 1$. Let $\varepsilon > 0$ and two directions for every segment $A \in \Pi$ as $\bar{\gamma}_B^A, \bar{\gamma}_{B^c}^A \in \mathbb{R}^d$ with the property

$$\|\bar{\gamma}_B^A + \bar{\gamma}_{B^c}^A\|_q = 1. \tag{4.36}$$

Let us combine ε and the directions for simpler notation as

$$\gamma_B^A = \frac{\varepsilon}{2}\bar{\gamma}_B^A, \quad \gamma_{B^c}^A = \frac{\varepsilon}{2}\bar{\gamma}_{B^c}^A \tag{4.37}$$

Then we define the function $\tilde{f}_B$ as follows

$$\tilde{f}_B = \sum_{A \in \Pi} \left(1_{A \cap B}(c_A + \gamma_B^A) + \mathbf{1}_{A \cap B^c}(c_A - \gamma_{B^c}^A)\right). \tag{4.38}$$

Note, that in the one-dimensional case obviously $\bar{\gamma}_B^A = \bar{\gamma}_{B^c}^A = \frac{1}{2}$, such that $\|\bar{\gamma}_B^A + \bar{\gamma}_{B^c}^A\|_q = \|\frac{1}{2} + \frac{1}{2}\|_q = 1$.

In Appendix 4.C we reformulate the energy functional $J(\tilde{f}_B)$ and see that solving (4.33) is the same as minimizing the directional derivative of J into the direction $\bar{\boldsymbol{\gamma}}_{\boldsymbol{B}}$. We also have derived the following about the directional derivative of the regularizer.

Lemma 4.9 (Partition Problem 1-homogenous Φ_i)**.**
When Φ_i is 1-homogenous in the second argument we have seen in Appendix 4.C that the directional derivative is given as

$$R'_{S^c}(f_\Pi; \bar{\boldsymbol{\gamma}}_{\boldsymbol{B}}) = \sum_{A \in \Pi} \sum_{(u,v) \in (A \times A) \cap S^c} |\mathbf{1}_{A \cap B}(v) - \mathbf{1}_{A \cap B}(u)| \Phi_i\big(w(u,v), (\bar{\gamma}_B^A + \bar{\gamma}_{B^c}^A)\big). \tag{4.39}$$

Let us state the results of Appendix 4.C in the following proposition.

Proposition 4.10 (Partition Problem).
Let Π be the current partition of V and $c \in \mathcal{H}(\Pi)$ a vertex function on the reduced graph $G_r = (\Pi, E_r, w_r)$. Let $f_\Pi \in \mathcal{S}_\Pi$ be a vertex function that is piecewise constant on the segments in Π and is given as $f_\Pi = Pc$. Let $g \in \mathcal{H}(V)$ be a vertex function representing the given data. Also let $\bar{\gamma}_B^A, \bar{\gamma}_{B^c}^A \in \mathbb{R}_+^d$ two descent directions for each segment $A \in \Pi$ with length 1. Let R_S be the differentiable and R_{S^c} be the non-differentiable parts of R. Then a subset $B^* \in \mathcal{P}(V)$ that minimizes (4.33) can be found by solving

$$B^* \in \operatorname*{argmin}_{B \in \mathcal{P}(V)} \left\{ \langle \nabla D(f_\Pi, g) + \frac{\alpha}{2} \nabla R_S(f_\Pi), \bar{\boldsymbol{\gamma}}_{\boldsymbol{B}} \rangle + \frac{\alpha}{2} R'_{S^c}(f_\Pi; \bar{\boldsymbol{\gamma}}_{\boldsymbol{B}}) \right\}. \tag{4.40}$$

with

$$\bar{\boldsymbol{\gamma}}_{\boldsymbol{B}} = \sum_{A \in \Pi} \mathbf{1}_{A \cap B} (\bar{\gamma}_B^A + \bar{\gamma}_{B^c}^A).$$

Proof. See Appendix 4.C for in-depth details. □

Once again we want to apply these general results to the p-q-TV regularizer from Definition 3.17 with $1 \leq p \leq q$. As we know from Appendix 3.B the p-q-TV is differentiable everywhere if $1 < p \leq q$, and thus, $\nabla R_S = \nabla R$. As the p−q−TV term is also absolutely p-homogeneous we can take the directional derivative from Lemma 4.9. The non-differentiable part for the isotropic case, i.e., $p = 1$, $q > 1$ is given as $S^c = \{(u, v) \in E_d \mid f(u) = f(v)\} = \{(u, v) \in E_d \cap (A \times A) \mid A \in \Pi\}$ which yields

$$\begin{aligned} R'_{S^c}(f_\Pi; \bar{\boldsymbol{\gamma}}_{\boldsymbol{B}}) &= 2 \sum_{A \in \Pi} \sum_{(u,v) \in (A \times A) \cap S^c} \sqrt{w(u,v)} \left\| \gamma_B^A + \gamma_{B^c}^A \right\|_q |\mathbf{1}_{A \cap B}(v) - \mathbf{1}_{A \cap B^c}(u)| \\ &= 2 \sum_{(u,v) \in S^c} \sqrt{w(u,v)} \, |\mathbf{1}_{A \cap B}(v) - \mathbf{1}_{A \cap B^c}(u)|. \end{aligned} \tag{4.41}$$

Note that the term $\|\gamma_B^A + \gamma_{B^c}^A\|_q = 1$ due to the assumptions in (4.36). Again the directional derivative is equal for both directions of the edges, and thus, we add the 2. The gradient of the differentiable part of R we computed in Appendix 3.B in (3.53) for $p = 1$, $q > 1$ is given as

$$\nabla R_S(f) = 2 \sum_{(u,v) \in S_{1;q}} \sqrt{w(u,v)} |f(v)_j - f(u)_j|^{q-2} \frac{f(u)_j - f(v)_j}{\|f(v) - f(u)\|_q^{q-1}}.$$

As we will investigate the channel-isotropic case for $p = 1$, $q = 2$ has the gradient

$$(\nabla R_S(f))_u = \Big(\sum_{(u,v) \in S_{1;2}} \sqrt{w(u,v)} \frac{f(u) - f(v)}{\|f(v) - f(u)\|_2} \Big)$$

we have the partition problem given as

$$\begin{aligned} \operatorname*{argmin}_{B \in \mathcal{P}(V)} \sum_{A \in \Pi} \sum_{u \in A} &\langle (\nabla D(f_\Pi) + \frac{\alpha}{2} \nabla R_S(f_\Pi))_u, \bar{\gamma}_B^A + \bar{\gamma}_{B^c}^A \rangle_{\mathbb{R}^d} \mathbf{1}_B(u) \\ &+ \alpha \sum_{(u,v) \in S^c} \sqrt{w(u,v)} |\mathbf{1}_B(v) - \mathbf{1}_B(u)|. \end{aligned} \tag{4.42}$$

For the anisotropic case we have

$$\begin{aligned}\operatorname*{argmin}_{B\in\mathcal{P}(V)} \sum_{j=1}^{d}\sum_{A\in\Pi}\sum_{u\in A}(\nabla D(f_\Pi)+\frac{\alpha}{2}\nabla R_S(f_\Pi))_{u,j}\mathbf{1}_B(u) \\ +\alpha\sum_{(u,v)\in S^c}\sqrt{w(u,v)}|\mathbf{1}_B(v)-\mathbf{1}_B(u)|\end{aligned} \tag{4.43}$$

which follows that we can again solve the problem for each dimension j separately as stated in Lemma 4.4.

Having derived the *partition problem* (4.40) to refine the partition Π accordingly to the given energy J we still have to find a solution somehow. This is where the minimal graph cut comes in that we introduced in Section 3.4, where we explained how one can compute the solution for certain binary minimization problems via graph cuts.

4.2.2 Solving the Partition Problem

In this subsection we translate the general partition problem from (4.40) such that it corresponds to the binary partitioning energy (3.37) from Section 3.4. Then we can find a graph representation for the energy and solve it via a minimal cut algorithm.

We start by stating the partition energy from (4.40)

$$\mathcal{J}(\mathbf{1}_B)=\langle\nabla D(f_\Pi)+\frac{\alpha}{2}\nabla R_S(f_\Pi),\bar{\boldsymbol{\gamma}}_{\boldsymbol{B}}\rangle+\frac{\alpha}{2}R'_{S^c}(f_\Pi;\bar{\boldsymbol{\gamma}}_{\boldsymbol{B}})\ . \tag{4.44}$$

For computations let us denote $J_S=D+\frac{\alpha}{2}R_S$. Starting with the scalar product we see that

$$\begin{aligned}\langle\nabla D(f_\Pi)+\frac{\alpha}{2}\nabla R_S(f_\Pi),\bar{\boldsymbol{\gamma}}_{\boldsymbol{B}}\rangle &= \sum_{A\in\Pi}\sum_{u\in A}\langle\nabla J_S(f_\Pi)_u,\mathbf{1}_B(u)(\bar{\gamma}^A_{B^c}+\bar{\gamma}^A_{B^c})\rangle_{\mathbb{R}^d} \\ &= \sum_{A\in\Pi}\sum_{u\in A}\langle\nabla J_S(f_\Pi)_u,\bar{\gamma}^A_{B^c}+\bar{\gamma}^A_{B^c}\rangle_{\mathbb{R}^d}\,\mathbf{1}_B(u)\end{aligned}$$

which can be interpreted as the data-term in (3.37) that is pointwise decomposable for every $u\in V$ and depends on the labeling $\mathbf{1}_B(u)$, i.e., $\mathcal{D}_u(\mathbf{1}_B(u))=\langle\nabla J_S(f_\Pi)_u,\bar{\gamma}^A_{B^c}+\bar{\gamma}^A_{B^c}\rangle_{\mathbb{R}^d}\,\mathbf{1}_B(u)$.

Lemma 4.11 (Capacities for Vertical Edges)**.**
Note that by the above definition $\mathcal{D}_u(0)=0$ for every setting. Consequently, to compute the flow graph capacities as in (3.42) we only have to check if $\mathcal{D}_u(1)=\langle\nabla J_S(f_\Pi)_u,\bar{\boldsymbol{\gamma}}_{\boldsymbol{B}}(u)\rangle$ is positive or negative.

Additionally, note that if Φ_i is absolutely p-homogeneous with $p>1$ then it is already differentiable everywhere and $\nabla R_S=\nabla R$. Thus, we will assume $p=1$ in the following.

For the non-differentiable part R_{S^c} of the absolutely p-homogeneous regularizer with $p=1$ we derived in Appendix 4.C that the directional derivative is given as

$$R'_{S^c}(f_\Pi;\bar{\boldsymbol{\gamma}}_{\boldsymbol{B}})=\sum_{(u,v)\in S^c}|\mathbf{1}_{A\cap B}(v)-\mathbf{1}_{A\cap B}(u)|\Phi_i\big(w(u,v),(\bar{\gamma}^A_B+\bar{\gamma}^A_{B^c})\big)$$

which is decoupled pointwise for every edge $(u,v) \in S^c$. Thus, we can set

$$\sum_{(u,v)\in S^c} \mathcal{R}_{u,v}(\mathbf{1}_B(u), \mathbf{1}_B(v)) = \frac{\alpha}{2} R'_{S^c}(f_\Pi; \bar{\boldsymbol{\gamma}}_{\boldsymbol{B}})$$

where we define for all edges $(u,v) \in S^c$

$$\mathcal{R}_{u,v}(\mathbf{1}_B(u), \mathbf{1}_B(v)) = |\mathbf{1}_{A\cap B}(v) - \mathbf{1}_{A\cap B}(u)| \Phi_i\big(w(u,v), (\bar{\gamma}_B^A + \bar{\gamma}_{B^c}^A)\big). \tag{4.45}$$

and for all $(u,v) \in S$ we set

$$\mathcal{R}_{u,v}(\mathbf{1}_B(u), \mathbf{1}_B(v)) = 0.$$

Hence, we have found a binary label energy functional $\mathcal{J}(\mathbf{1}_B)$ such that we can represent it by a graph from Section 3.4. To do so, we have to state the flow graph $G_{flow} = (V_{flow}, E_{flow}, c)$ with terminals s and t, and the capacities as defined as in (3.42). We state the three general flow graphs in the following for anisotropic and isotropic non-differentiable cases and the differentiable case.

Let $\mathbf{p = 1}$ and anisotropic $\boldsymbol{\Phi_a^j}$:
By former computation we know that this is the *anisotropic non-differentiable* TV regularization with $S_j = \{(u,v) \in E \mid f(v)_j \neq f(u)_j\}$. From Lemma 4.4 we know that we can decouple the anisotropic problems over the dimensions and also for the anisotropic partition problem (4.43). So we can state and solve a flow graph for each dimension j separately. Additionally, we know that $\gamma_B^A + \gamma_{B^c}^A = 1$ in the one-dimensional case. Thus, we compute a flow graph for every dimension as

$$G_{flow,j} = (V_{flow}, E_{flow}, c_j).$$

As stated in Lemma 4.11 for the vertical edges in the flow graph we have to investigate if $(\nabla J_S(f_\Pi)))_{u,j}$ is positive or negative. For the horizontal edges we have to compute (3.40) with (4.45) which is then given for $(u,v) \in S_j^c$ as

$$\begin{aligned} c_j(u,v) &= \mathcal{R}_{u,v}(0,1) + \mathcal{R}_{u,v}(1,0) - \mathcal{R}_{u,v}(0,0) - \mathcal{R}_{u,v}(1,1) \\ &= \frac{\alpha}{2}\big(\Phi_a^j(w(u,v),1) + \Phi_a^j(w(u,v),1) - 0 - 0\big) \\ &= \alpha \Phi_a^j(w(u,v),1) \end{aligned} \tag{4.46}$$

and for $(u,v) \in S_j$ as

$$c_j(u,v) = 0.$$

Then we can define the edge set of horizontal edges as $(S_{1;1}^j)^c$ and the horizontal edges capacities as in (4.46). Then the capacities are given as

$$\begin{cases} c_j(s,u) = \quad \nabla J_S(f_\Pi)_{u,j}, & \text{if } \nabla J_S(f_\Pi)_{u,j} \geq 0, \\ c_j(u,t) = -\nabla J_S(f_\Pi)_{u,j}, & \text{if } \nabla J_S(f_\Pi)_{u,j} < 0, \\ c_j(u,v) = \alpha \Phi_a^j(w(u,v),1), & \forall\ (u,v) \in S_j^c. \end{cases} \tag{4.47}$$

The flow graph then is given by the edge set

$$E_{flow,j} = \{(u,v) \in S_j^c \cup \{s\} \times V \cup V \times \{t\} \mid c(u,v) > 0\}.$$

Then we have the flow graph $G_{flow,j}$ that represents the energy for the anisotropic problem. The optimization can be done by computing a s-t cut between source s and sink t.

Let $\mathbf{p = 1}$ and isotropic $\mathbf{\Phi_i}$:
This is the *non-differentiable isotropic* case where the set of the differentiable part is given as $S = \{(u,v) \in E \mid f(u) \neq f(v)\}$. In this case the dimensions are coupled, and thus, we only have to compute one flow graph for this setting. For the vertical edges to the terminals we have to check if $D_u(1) = \langle \nabla J_S(f_\Pi), \bar{\gamma}_B^A + \bar{\gamma}_{B^c}^A \rangle_{\mathbb{R}^d}$ is positive or negative for $u \in A$ for every $A \in \Pi$. The horizontal edges are given by S^c and the capacity is analogously to (4.46) given as

$$c(u,v) = \alpha \Phi_i(w(u,v), 1). \tag{4.48}$$

Then the capacities of the flow graph are given as

$$\begin{cases} c(s,u) = \quad \langle \nabla J_S(f_\Pi)_u, \bar{\gamma}_B^A + \bar{\gamma}_{B^c}^A \rangle_{\mathbb{R}^d}, & \text{if } \langle \nabla J_S(f_\Pi)_u, \bar{\gamma}_B^A + \bar{\gamma}_{B^c}^A \rangle_{\mathbb{R}^d} \geq 0, \\ c(u,t) = -\langle \nabla J_S(f_\Pi)_u, \bar{\gamma}_B^A + \bar{\gamma}_{B^c}^A \rangle_{\mathbb{R}^d}, & \text{if } \langle \nabla J_S(f_\Pi)_u, \bar{\gamma}_B^A + \bar{\gamma}_{B^c}^A \rangle_{\mathbb{R}^d} < 0, \\ c(u,v) = \alpha \Phi_i(w(u,v), 1), & \forall\ (u,v) \in S^c. \end{cases} \tag{4.49}$$

The flow edge set is given as

$$E_{flow} = \{(u,v) \in S^c \cup \{s\} \times V \cup V \times \{t\} \mid c(u,v) > 0\}.$$

Then we have a flow graph $G_{flow} = (V_{flow}, E_{flow}, c)$ given that represents the isotropic partition problem for $p = 1 < q$.

Let $\mathbf{p > 1}$:
Let us investigate the anisotropic and isotropic case at once. As we know when $p > 1$ then the regularizer R is differentiable everywhere for the anisotropic and isotropic case. Thus, we do have $\nabla J_S = \nabla J$ and $\mathcal{R}_{u,v} = 0$ for all $(u,v) \in E$. With this and the former results we can directly see that we only have to investigate in the isotropic case if $(\nabla J(f_\Pi))_u$ is positive or negative for every $u \in V$. Thus, the isotropic flow graph capacities are given as

$$\begin{cases} c(s,u) = \quad \langle \nabla J_S(f_\Pi)_u, \bar{\gamma}_B^A + \bar{\gamma}_{B^c}^A \rangle_{\mathbb{R}^d}, & \text{if } \langle \nabla J_S(f_\Pi)_u, \bar{\gamma}_B^A + \bar{\gamma}_{B^c}^A \rangle_{\mathbb{R}^d} \geq 0, \\ c(u,t) = -\langle \nabla J_S(f_\Pi)_u, \bar{\gamma}_B^A + \bar{\gamma}_{B^c}^A \rangle_{\mathbb{R}^d}, & \text{if } \langle \nabla J_S(f_\Pi)_u, \bar{\gamma}_B^A + \bar{\gamma}_{B^c}^A \rangle_{\mathbb{R}^d} \geq 0, \\ c(u,v) = 0. \end{cases} \tag{4.50}$$

Thus, the flow edge set is just given as

$$E_{flow} = \{(u,v) \in \{s\} \times V \cup V \times \{t\} \mid c(u,v) > 0\}.$$

In the anisotropic case we get the analog one-dimensional flow graph capacities as in (4.50) with $\bar{\gamma}_B^A + \bar{\gamma}_{B^c}^A = 1$ and for every dimension $1 \leq j \leq d$ separately. The flow graph $G_{flow,j}$ is then given for every $1 \leq j \leq d$ as

$$\begin{cases} c_j(s,u) = \quad \nabla J_S(f_\Pi)_{u,j}, & \text{if } \nabla J_S(f_\Pi)_{u,j} \geq 0, \\ c_j(u,t) = -\nabla J_S(f_\Pi)_{u,j}, & \text{if } \nabla J_S(f_\Pi)_{u,j} \geq 0, \\ c_j(u,v) = 0. \end{cases} \tag{4.51}$$

Thus, the flow edge set is just given as

$$E_{flow,j} = \{(u,v) \in \{s\} \times V \cup V \times \{t\} \mid c(u,v) > 0\}.$$

Finally, we have found a way to solve the partition problem from (4.40) via a very efficient graph cut algorithm for the general regularizer R. With all of the derivations from above we can also compute the corresponding flow graphs for the p-q-TV regularizer in Section 4.4.2. Computing a subset B and split the former partition Π into new segments with B enables us to compute a new reduced solution on the new set Π_{new}. Now we know how to solve the partition problem with a given direction for every segment in the former partition Π. In the next subsection we investigate the importance of choosing the direction correctly

4.2.3 Direction for Isotropic Cut-Pursuit

Let us assume that we use some arbitrary method that computes the directions $\bar{\gamma}_B^A, \bar{\gamma}_{B^c}^A$ for every segment $A \in \Pi$. The question that arises here is what we can conclude if the solution B^* of the partition problem Proposition 4.10 is $B^* = \emptyset$ or $B^* = V$ for the given directions $\bar{\gamma}_B^A, \bar{\gamma}_{B^c}^A$. In the one-dimensional channel-anisotropic case we reached the optimum since we cannot decrease the energy by the graph cut anymore (cf. [46]). In the multi-dimensional channel-isotropic case this can also mean that for this particular direction we cannot find a non-trivial cut, but that there still might exist other directions that would yield a finer partition. Hence, we have to check for all directions and extend the statement from [46, Proposition 3] for the optimum in the anisotropic case to the multi-dimensional channel-isotropic case in the following.

Proposition 4.12.
Let G be a finite weighted graph and $J\colon \mathcal{H}(V) \to \mathbb{R}$ be proper and convex. Then we can follow that

$$f^* \in \operatorname*{argmin}_{f \in \mathcal{H}(V)} J(f) \quad \text{if and only if} \quad \min_{B \in \mathcal{P}(V)} J'(f^*; \bar{\boldsymbol{\gamma}}_{\boldsymbol{B}}^*) = 0, \ \forall \gamma_A \in \mathbb{R}^d, A \in \Pi^*$$

with $\bar{\boldsymbol{\gamma}}_{\boldsymbol{B}}^* = \sum_{A \in \Pi^*} \mathbf{1}_{A \cap B} \bar{\gamma}^A$.

Proof. The proof can be done analogously to the proof in [46, Proposition 3]. As we are considering convex functionals in this work one can also use the statement in [6, Proposi-

tion 17.3] for further insight of the relation of optimal solution and the directional derivative being 0 in all directions. □

Hence, we know that we found the optimum of the problem if for every direction no cut can be computed by solving the partition problem. This also means that if we only consider a fixed direction and the partition problem does yield a trivial cut, i.e., $B = \emptyset$, $B = V$, we cannot deduce that this is the optimum. Hence, we face the problem that we have to optimize over the directions or to have sound heuristics for selecting the directions with some rationale.

In fact, the condition in the isotropic case that the algorithm reached the optimal partition is that it cannot compute a new partition in any direction in $\mathbb{R}^d$. When computing the partition problem before we used the trick that

$$\mathbf{1}_{A\cap B}\bar{\gamma}_B^A - \mathbf{1}_{A\cap B^c}\bar{\gamma}_{B^c}^A = \mathbf{1}_{A\cap B}(\bar{\gamma}_B^A + \bar{\gamma}_{B^c}^A) - \mathbf{1}_A\bar{\gamma}_{B^c}^A, \tag{4.52}$$

and hence, we could drop the term $\mathbf{1}_A\bar{\gamma}_{B^c}^A$ from the optimization, since it is independent on B. As we now optimize over $\bar{\gamma}_{B^c}^A$ we cannot drop it from the overall minimization problem, and consequently, have to solve

$$\operatorname*{argmin}_{B\in\mathcal{P}(V),\gamma_B^A,\gamma_{B^c}^A} \langle \nabla J_S(f_\Pi, g), \sum_{A\in\Pi} 1_{A\cap B}\bar{\gamma}_B^A - \mathbf{1}_{A\cap B^c}\bar{\gamma}_{B^c}^A\rangle + \frac{\alpha}{2}R'_{S^c}(f_\Pi; \bar{\boldsymbol{\gamma}}_{\boldsymbol{B}}) \tag{4.53}$$

with $\nabla J_S(f_\Pi, g) = \nabla D(f_\Pi, g) + \frac{\alpha}{2}\nabla R_S(f_\Pi)$ and $\bar{\boldsymbol{\gamma}}_{\boldsymbol{B}} = \sum_{A\in\Pi} 1_{A\cap B}(\bar{\gamma}_B^A + \bar{\gamma}_{B^c}^A)$. We only provide a simple alternating algorithm here and postpone this problem to a future work. Let us simplify here by setting $\bar{\gamma}_A = \bar{\gamma}_B^A + \bar{\gamma}_{B^c}^A$ and also $\|\bar{\gamma}_A\|_2 = 1$. The alternating algorithm is then given by fixing B and optimize for $\bar{\gamma}_A$ for every $A \in \Pi$ and fix the directions $\bar{\gamma}_A$ and optimize for B via a graph cut. When updating for B we can again use the trick from (4.52) and solve the equivalent partition problem (4.42) with the updated directions. When updating for the direction $\bar{\gamma}_A$ we can drop the regularization since it is independent on the directions. Then the alternating algorithm reads

$$\begin{cases} B^{n+1} &= \operatorname*{argmin}_{B\in\mathcal{P}(V)} \langle \nabla J_S(f_\Pi, g), \sum_{A\in\Pi} \mathbf{1}_{A\cap B}\bar{\gamma}_A^n\rangle \\ &\qquad\qquad +\alpha \sum_{(u,v)\in S^c} \sqrt{w(u,v)}\ |\mathbf{1}_B(v) - \mathbf{1}_B(u)| \\ \bar{\gamma}_A^{n+1} &= \operatorname*{argmin}_{\|\bar{\gamma}_A\|_2=1} \sum_{u\in A}\langle (\nabla J_S(f_\Pi, g))_u, (\mathbf{1}_B(u) - \mathbf{1}_{B^c}(u))\bar{\gamma}_A\rangle, \quad \forall A \in \Pi. \end{cases} \tag{4.54}$$

The partition problem once again can be solved by stating the flow graph as in a former section and then solve via a graph cut. The constraint minimization problem below can be solved via a Lagrange multiplier method [7]. We postpone the exact computation to (4.81) where we solve this problem for a weighted p-q-ROF problem. However, for the first computation of B we have to select an initial direction $\bar{\gamma}_A^0$ for every segment. This can be done by selecting some direction or just using a random direction.

In Section 6.3, we compare different approaches for different data sets with heuristically

computed directions and optimized directions like above. In the next section we finally state the Cut-Pursuit algorithm that solves the problem (4.1).

4.3 Cut-Pursuit Algorithm

In the former sections we learn how to solve a problem as (4.1) on a partition Π of V, that the reduced solutions are equal to the real solutions if the Π is chosen correctly, and how to split the Π into a new partition using a graph cut and the energy of the last solution. We also know that we have to compute directions for every segment $A \in \Pi$ if we are working with isotropic regularizer in multi-dimensional spaces.

Let $G = (V, E, w)$ be an undirected finite weighted graph as in Definition 3.2, $g \in \mathcal{H}(V; \mathbb{R}^d)$ some d-dimensional data. A general minimization problem is given as in (4.1) with the energy functions J that consist of a differentiable data-term D and a directional differentiable regularizer R as defined in Definition 4.1 for the anisotropic and in Definition 4.2 for the isotropic regularization. In the following, we show an alternating algorithm that in each iteration splits the current partition into a finer partition depending on the former solution and solves the reduced problem (4.1) on the corresponding reduced graph. Note that we assume the directions, in which the partition is refined by the partition problem, is chosen optimal. These alternating steps are done until convergence, which means that the energy of the reduced problem cannot be improved using a graph cut in any direction. This is what we state in Proposition 4.12 where we show that we reached the optimum if $\min_{B \in \mathcal{P}(V)} J'(f_\Pi; \bar{\boldsymbol{\gamma}}^*_{\boldsymbol{B}}) = 0$ for every direction.

We initialize the first partition with one segment that holds all nodes, i.e., $\Pi^0 = \{V\}$, which implies that the initial reduced graph G_r^0 has one node representing $\{V\}$ and no edges. Thus, the initial solution $c^0 \in \mathcal{H}(\Pi^0)$ is the result on one node. Since there are no edges in the graph the solution is in fact computed by minimizing the data-term D on a constant function, i.e.,

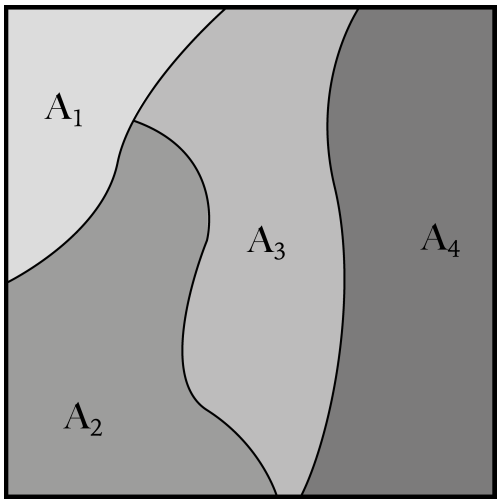
$$c^0 = \operatorname*{argmin}_{c \in \mathbb{R}^c} \; D(Pc).$$

Note that this means that every node in V has the same value for initilaization, i.e., $f^0 = P^0 c^0$. Computing this can be minimized due to the differentiability of D and that we have to solve it on one solely vertex. In terms of an L_2-data-term this is just given as the mean value over the image.

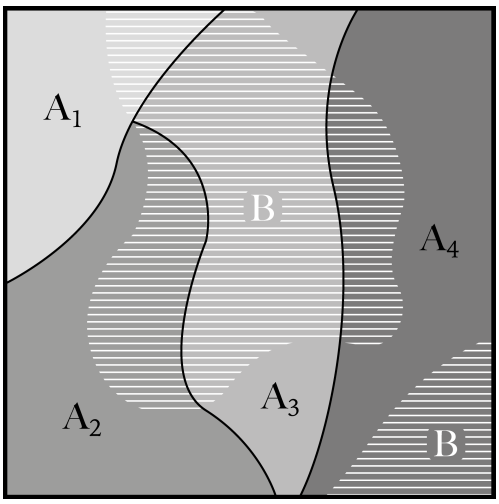
Let us assume we are in the k'th alternating step and a partition Π^k, a reduced solution c^k and corresponding $f^k = P^k c^k$ are given. Then we start by computing the directions $\bar{\gamma}_B^{k+1}$ and $\bar{\gamma}_{B^c}^{k+1}$ for every segment $A \in \Pi^k$. These are then stored in $\bar{\boldsymbol{\gamma}}_{\boldsymbol{B}}{}^{k+1}$ and used to compute the new split B^{k+1} by solving

$$B^{k+1} \in \operatorname*{argmin}_{B \in \mathcal{P}(V)} \langle \nabla D(f_{\Pi^k}) + \frac{\alpha}{2} \nabla R_S(f_{\Pi^k}), \bar{\boldsymbol{\gamma}}_{\boldsymbol{B}} \rangle + \frac{\alpha}{2} R'_{S^c}(f_{\Pi^k}; \bar{\boldsymbol{\gamma}}_{\boldsymbol{B}}).$$

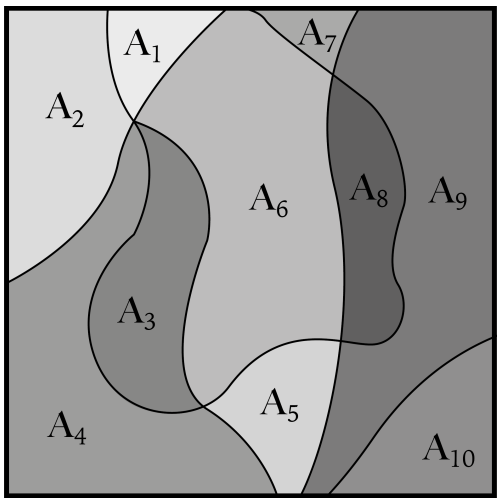
As we have shown in Section 4.2.2, we can solve this for the different anisotropic and isotropic regularizations of the general problem (4.1) using an s-t graph cut applied to the flow graphs we state in (4.47), (4.49), (4.50), (4.51) representing the different energies. With the set B^{k+1} we can move on and build the new partition P^{k+1}. There exist different methods to do so.



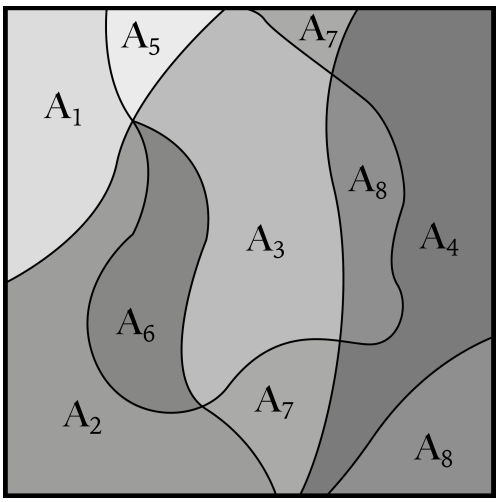
(a) Initial partition $\Pi = \{A_i, A_j, A_3, A_4\}$.



(b) Steepest binary partition where $B \in \mathcal{P}(V)$ is visualized by the white dashed set.



(c) Resulting partition $\Pi_{new} = \{A_i | i \in [1, 10]\}$ generated by the steepest binary cut and selecting connected components as the new partitions A_i.



(d) Resulting partition $\Pi_{new} = \{A_i | i \in [1, 8]\}$ generated by the steepest binary cut and selecting the new partition as $\Pi_{new} = (\Pi \cap B) \cup (\Pi \cap B^c)$.

Figure 4.1.: Illustration of two different methods to generate a new partition Π_{new} from a given partition Π and the set $B \in \mathcal{P}(V)$

In this work we use the connected components of $A \cap B$ and $A \cap B^c$ for all $A \in \Pi^k$. Thus, the new partition is given as

$$\Pi^{k+1} = \{\text{connComp}(A \cap B), \text{connComp}(A \cap B) \mid A \in \Pi^k\}. \tag{4.55}$$

Actually, this is a valid idea since we compute a split that can decrease the energy on the a new partition with segments as $A \cap B$ and $A \cap B^c$ for all $A \in \Pi$. Hence, the energy on a finer partition consisting of all connected component can only be less or equal to the one on the coarser partition. Nevertheless, one can also decide to set the new partition as $\{A \cap B, A \cap B^c \mid A \in \Pi^k\}$. In Figure 4.1 we see the difference between using connected

components or just using the plain splits as a new partition. The expansion operator for Π^{k+1} is given by

$$P^{k+1} = (\mathbf{1}_A)_{A\in\Pi}\,.$$

With these we construct the new reduced graph G_r^{k+1} as in (4.20) and solve the reduced problem (4.25) that is given as

$$c^{k+1} = \operatorname*{argmin}_{c\in\mathcal{H}(\Pi^{k+1})} D(P^{k+1}c) + \frac{\alpha}{2}R(P^{k+1}c) \tag{4.56}$$

via a primal-dual algorithm on graphs as we derived in Algorithm 3.20. Then we can compute $f_{\Pi^{k+1}} = P^{k+1}c^{k+1}$ and start from the top. We summarize this algorithm in for the isotropic case in Algorithm 1.

Let us take a quick look on how to use Cut-Pursuit for the anisotropic problems with Φ_a^j as in Definition 4.1. As in Lemma 4.4, we can decouple the problem for multi-dimensional anisotropic functionals for each dimension j. Thus, we can apply the Cut-Pursuit algorithm from Algorithm 1 for each dimension, since it in one dimension channel-isotropy and channel-anisotropy are obviously equivalent. For each evaluation of the Cut-Pursuit we get a one-dimensional result f_j^* and a partition Π_j^* representing the piecewise constant parts. The partitions $\Pi_j^* = \{A_i^j \mid 1 \le i \le N_j\}$ of every dimension $1 \le j \le d$ cannot be assumed to be equivalent, since the solution f_j^* is computed decoupled from the other channels, and thus, the piecewise constant segments differ between dimensions. The partition Π^* to represent f^* then is the set of all possible intersections of combinations of segments in the different partitions Π_j^*. To be more precise let us assume that we have a two-dimensional data given and therefore computed two solutions f_1^* and f_2^* with their representing partitions Π_1^* and Π_2^*. We assume that there exists at least on combination of segments such that $A_i \in \Pi_1^*$ and $A_j \in \Pi_2^*$ are not equal, i.e., $A_i \cap A_j \neq A_i$ and $A_i \cap A_j \neq A_j$, and that $A_i \cap A_j \neq \emptyset$. Let us consider a node $u \in A_i \cap A_j$. For this node we know that $f_j^*(v) = f_j^*(u)$ for every $v \in A_j$ for $j \in \{1,2\}$. But we also know that for example $j = 1$ we have that $f_1^*(u) \neq f_1^*(v)$ for $v \in A_j \setminus A_i \cap A_j$, due to the definition of the partitions. The same holds for $j = 2$ where $f_2^*(u) \neq f_2^*(v)$ for $v \in A_i \setminus A_i \cap A_j$. Since $f^*(u) = (f_1^*(u), f_2^*(u))$ we can deduce from the pointwise observations that $f^*(u) = f^*(v)$ for $u, v \in A_i \cap A_j$ and $f^*(u) \neq f^*(v)$ for $u \in A_i \cap A_j$ and $v \in A_i \setminus A_i \cap A_j$ or $v \in A_j \setminus A_i \cap A_j$. Thus, we see that f^* is piecewise constant on the intersection of the segments $A_i \in \Pi_1^*$ and $A_j \in \Pi_2^*$. This can be now done for every $u \in V$ and also extended to more dimensions. Let $\prod_{j=1}^d \Pi_j^* = \Pi_1^* \times \dots \Pi_d^*$. Then the anisotropic solution yields the partition Π^* as

$$\Pi_a^* = \{\ \cap_{j=1}^d A_j \mid (A_i, \dots, A_d) \in \prod_{j=1}^d \Pi_j^*\ \}. \tag{4.57}$$

In the following section we apply the Cut-Pursuit algorithm on the (weighted) ROF problem (3.34) and derive the Cut-Pursuit algorithm for the different settings.

Algorithm 1: Isotropic Cut-Pursuit Algorithm.

Input:
$G = (V, E, w) \leftarrow$ Undirected finite weighted graph
$g \leftarrow$ Some given d-dimensional data in $\mathcal{H}(V; \mathbb{R}^d)$
Initialization:
$\Pi^0 \leftarrow \{V\}$
$c^0 \leftarrow \operatorname{argmin}_{c \in \mathbb{R}^d} \; D(P^0 c)$
Method:
while $J'(f_{\Pi^k}) < 0$ **do**
  $\bar{\boldsymbol{\gamma}}_{\boldsymbol{B}}{}^{k+1} \leftarrow$ compute a direction for every $A \in \Pi$ heuristically or by solving (4.53) (e.g. algorithm (4.54)).
  $G_{flow} \leftarrow$ buildFlowGraph(G,f_{Π^k},$\bar{\boldsymbol{\gamma}}_{\boldsymbol{B}}{}^{k+1}$,$g$,$\alpha$) for given methods, (4.49), (4.50)
  $B^{k+1} \leftarrow$ computeMaxFlow(G_{flow}) via maxflow algorithm, that solves (4.40) (cf. [10], Section 3.4).
  $\Pi^{k+1} \leftarrow \{\text{connComp}(A \cap B), \text{connComp}(A \cap B) \mid A \in \Pi\}$.
  $P^{k+1} \leftarrow (\mathbf{1}_A)_{A \in \Pi^{k+1}}$
  $G_r = (V_r, E_r, w_r) \leftarrow$ with $V_r = \Pi^{k+1}$, E_r, w_r as in (4.20).
  $g_r \leftarrow$ reduced data according to data-term, e.g. (4.12).
  $c^{k+1} = \operatorname{argmin}_{c \in \mathcal{H}(\Pi^{k+1})} \; J(P^{k+1} c, g_r) \leftarrow$ reduced problem as in (4.21). Solve with method, e.g. the primal-dual algorithm on graphs as stated in Algorithm 3.20.
Result: The solution f^* for (4.1) and the corresponding partition Π^* representing the piecewise constant parts of f^*.

Algorithm 2: Anisotropic Cut-Pursuit Algorithm.

Input:
$G = (V, E, w) \leftarrow$ Undirected finite weighted graph
$g \leftarrow$ Some given d-dimensional data in $\mathcal{H}(V; \mathbb{R}^d)$
Initialization:
$\Pi^0 \leftarrow \{V\}$
$c^0 \leftarrow \operatorname{argmin}_{c \in \mathbb{R}^d} \; D(P^0 c)$
Method:
for $j \in \{1, \ldots, d\}$ **do**
  $f_j^* \leftarrow$ Use the one-dimensional form of Algorithm 1 on each dimension j separately.
$\Pi^* \leftarrow$ Compute as in (4.57)
Result: The solution f^* for the anisotropic (4.1) and the corresponding partition Π^*representing the piecewise constant segments of f^*.

4.4 Cut-Pursuit Algorithm for ROF Problems

In this section, we want to apply the derived Cut-Pursuit algorithm from the former section onto the (weighted) ROF problem in (3.34). In the numerics in Chapter 6 we show the application of Cut-Pursuit to different data and investigate the run time and other properties.

Again we work with an undirected finite weighted graph and some given d-dimensional data $g \in \mathcal{H}(V;\mathbb{R}^d)$. In this section, we are interested in solving the family of weighted p-q-ROF problems on graph given as

$$\operatorname*{argmin}_{f\in\mathcal{H}(V)} \left\{ J(f) = \frac{1}{2}\|f-g\|_{2,W}^2 + \frac{\alpha}{2p}\|\nabla_w f\|_{p;q}^p \right\} \tag{4.58}$$

where $p,q \geq 1$ and $W \in \mathbb{R}_{\geq 0}^{|V|\times|V|}$ is some diagonal weighting matrix. We investigate these since they are a simple generalization of the common ROF problem where the data-term is an L_2-data-term which is equivalent as the weighting $W = I$.

To apply the Cut-Pursuit algorithm to this problem we have to deduce the reduced graph and the reduced problem for any given partition Π and solve it with the primal-dual algorithm on graphs we introduce in Algorithm 3.24 for weighted ROF problems. Afterwards, we state the partition problem for (4.58) and give the flow graphs for different settings of p and q.

But first let us show that the p-q-TV regularization is actually a special case of the general anisotropic and channel-isotropic regularizer in Definition 4.1 and Definition 4.2 respectively. It also follows that the statements of Lemma 4.3 and Lemma 4.4 hold. As we have seen in Section 3.3.1, the p-q-TV term is anisotropic and channel-isotropic for different p,q combinations. The p-q-TV regularizer can be rewritten as

$$\begin{aligned} R(f) &= \frac{1}{p}\sum_{u\in V}\sum_{v\sim u} w(u,v)^{\frac{p}{2}} \left(\sum_{j=1}^{d} |f(v)_j - f(u)_j|^q\right)^{\frac{p}{q}} \\ &= \sum_{(u,v)\in E} \Phi(w(u,v), f(v)-f(u)) \end{aligned} \tag{4.59}$$

with

$$\Phi(w(u,v), f(v)-f(u)) = \frac{1}{p} w(u,v)^{\frac{p}{2}} \left(\sum_{j=1}^{d} |f(v)_j - f(u)_j|^q\right)^{\frac{p}{q}}. \tag{4.60}$$

Note that Φ is $\frac{p}{2}$-homogenous in the first argument and p-homogenous in the second argument. We already introduced a primal-dual algorithm on graphs in Algorithm 3.22 to solve the graph problem (4.58) using the proximity operators we state in Appendix 3.C.

4.4.1 Reduced ROF Problem

As we have derived the reduced graph setting for the general problem in Section 4.1 let us apply it to the ROF problem (4.58) where R is the p-q-TV regularizer as in (4.59). We see in (4.59) that Φ is $\frac{p}{2}$-homogenous in the first and p-homogenous in the second argument,

and thus, we can deduce directly from (4.19) the *reduced weighting* $w_r\colon E_r \to \mathbb{R}_+$ given as

$$w_r(A,B) = \big[\sum_{(u,v)\in E_{AB}} w(u,v)^{\frac{p}{2}}\big]^{\frac{1}{p}}, \quad \forall (A,B)\in E_r. \tag{4.61}$$

The reduced regularizer for p-q-TV is then computed by applying (4.18) to the Φ from (4.59) as

$$\begin{aligned} R_r(c) &= \sum_{(A,B)\in E_r} \Phi(1, w_r(A,B)(c_B - c_A)) \\ &= \frac{1}{p}\sum_{(A,B)\in E_r} w_r(A,B)\|c_B - c_A\|_q^p \\ &= \Phi_r(\nabla_{w_r} c) = \|\nabla_{w_r} c\|_{p;q}^p \end{aligned} \tag{4.62}$$

with the reduced function $\Phi_r(x) = \|x\|_{p;q}^p$. With these deductions we can define the *reduced energy* J_r corresponding to the full energy J from (4.58) using the reduced form of the (weighted) L_2-data-term in (4.12) and the reduced p-q-TV term in (4.66) which yields

$$J_r(c) = \frac{1}{2}\|c - g_r\|_{2,W_P} + \frac{\alpha}{2p}\|\nabla_{w_r} c\|_{p;q}^p = J(f_c) \tag{4.63}$$

with $W_P = P^* W P$. In conclusion we found the reduced graph G_r and the reduced problem J_r for any partition Π describing all functions in $\mathcal{S}_\Pi$ that are piecewise constant on Π. We sum this results up in the following proposition.

Proposition 4.13 (Reduced p-q-TV Regularizer)**.**
Let $G = (V, E, w)$ be a finite weighted graph. Define a reduced graph as $G_r = (\Pi, E_r, w_r)$ for the partition Π of V with the reduced edge set *E_r defined as*

$$E_r = \{(A,B)\in \Pi\times\Pi \mid (A\times B)\cap E \neq \emptyset\} \tag{4.64}$$

and the defined reduced weighting *as $w_r\colon E_r \to \mathbb{R}_+$ as*

$$w_r(A,B) = \big(\sum_{(u,v)\in E_{AB}} w(u,v)^{\frac{p}{2}}\big)^{\frac{1}{p}}, \quad \forall (A,B)\in E_r. \tag{4.65}$$

Let $f_c \in \mathcal{S}_\Pi$ with $f_c = \sum_{A\in\Pi} c_A 1_A$ and $c = (c_A)_{A\in\Pi} \in \mathcal{H}(\Pi)$. Then the following identity holds

$$\|\nabla_w f\|_{p;q} = \|\nabla_{w_r} c\|_{p;q}. \tag{4.66}$$

The reduced energy *functional for (4.58) on G_r is then given as*

$$J_r(c) = \frac{1}{2}\|c - g_r\|_{2,W_P} + \frac{\alpha}{2p}\|\nabla_{w_r} c\|_{p;q}^p \tag{4.67}$$

with $W_P = P^* W P$ and $g_r = (P^* W P)^{-1} P^* W g$.

In order to solve the reduced problem (4.67) on G_r we use the primal-dual algorithm on graphs we introduce in Algorithm 4.14 for weighted ROF problems. As we see here, the

reduced weighted L_2-data-term is in fact a differently weighted L_2-data-term with the weight W_P. For the unweighted L_2-data-term with $W = I$ this leads to the diagonal weighting $W_P = \operatorname{diag}(|A|)_{A\in\Pi}$ which is a weighting by the sizes of the segments in Π. The primal-dual algorithm Algorithm 3.24 for weighted ROF problems yields the primal dual step for with the weighting matrix W_P as

$$c^{n+1} = (I + T_r W_P)^{-1}(c^n + T_r W_P(\operatorname{div}_{w_r}(y^{n+1}) + g_r)) \tag{4.68}$$

$$\Leftrightarrow \quad c^{n+1} = (I + T_r W_P)^{-1}(c^n + T(W_P \operatorname{div}_{w_r}(y^{n+1}) + P^* W g)). \tag{4.69}$$

The diagonal matrix T_r is the diagonal preconditioning matrix from (3.30) for the reduced weighting w_r. Note that this preconditioning is even more effective on reduced graphs as on normal graph structures. The reduced graph also suffers from a very inhomogeneous structure since every segment can have arbitrary numbers of neighboring segments. Moreover, the reduced weights are the sum over the weights between these neighboring segments in the original graph. Thus, the weighting is also very inhomogeneous, e.g. in the case of images it is the length of the boundary between segments. This leads to a very slow convergence of the reduced problems when applying a global constant step size, since for some partitions the step is too large while for others it is too small. This makes the convergence very inconsistent. The diagonal preconditioning alleviates these problems. In the numerics section we will also use a adaptive diagonal preconditioning introduced in [34] that we will not go into detail here. Additionally, to this let us emphasize that the primal-dual algorithm does not only reduce the number of primal parameters we have to optimize, but furthermore also reduces the number of parameters of the dual, since $y \in \mathcal{H}(E_r)$. Hence, the updates for primal and dual are both more efficient on the reduced graph than on the original full graph. Finally, the primal-dual algorithm for a reduced ROF problem is stated in the following.

Algorithm 4.14 (Reduced Weighted ROF Primal-Dual)**.**
Let $G = (V, E, w)$ be an undirected finite weighted graph and $G_r = (\Pi, E_r, w_r)$ be a reduced graph as defined in Proposition 4.16 for some partition Π of V. Let $W \in \mathbb{R}^{N\times N}_{\geq 0}$ be a diagonal matrix and $W_P = P^* W P$ its reduced relative. Let T_r, Σ_r be the reduced diagonal precondition matrices defined in (3.30) for the reduced weighting w_r. Then the primal-dual algorithm to solve the minimization of the function in (4.67) is given as

$$\begin{cases} y^{n+1} = \operatorname{prox}_{\Sigma_r(\frac{\alpha}{2p}\|\cdot\|^p_{p;q})^*}(y^n + \Sigma_r \nabla_{w_r} c^n) \\ c^{n+1} = (I + T_r W_r)^{-1}(f^n + T_r(W_P \operatorname{div}_{w_r}(y^{n+1}) + W P^* g)) \\ \bar{c}^n + 1 = c^{n+1} + \theta(c^{n+1} - c^n) \end{cases} \tag{4.70}$$

with the proximity operators $\operatorname{prox}_{\sigma(\frac{\alpha}{2p}\|\cdot\|^p_{p;q})^*}$ as defined in Appendix 3.C for selected p and q.

With the ability to solve the reduced problems let us deduce the partition problem and corresponding flow graphs for the weighted ROF problem in (4.58).

4.4.2 Partition Problem of the ROF Problem

To compute the flow graph for the partition problem we have to investigate the differentiable (weighted) L_2-data-term and the differentiable and non-differentiable parts of the p-q-TV regularization term that we already computed in Appendix 3.B. The derivative of the weighted L_2-data-term is given in Proposition 2.7 as

$$\nabla D(f, g) = W(f - g).$$

For the p-q-TV regularization term we state in (4.66) that it is absolutely $\frac{p}{2}$-homogeneous in the first and p-homogeneous in the second argument. In Appendix 3.B we derive the differentiable and non-differentiable parts of different settings of p and q. Since, $p \leq q$ in this work and p-homogeneous function are differentiable for $p > 1$ as we state in Lemma 4.3 we see that for $1 < p \leq q$ the p-q-TV term is *differentiable everywhere*. For $p = q = 1$ we have a *non-differentiable anisotropic* setting and for $p = 1 < q$ we have a *non-differentiable isotropic* setting. Note that for this particular p-q-TV term we can only have anisotropy for $p = q = 1$. Hence, we can directly apply the already derived flow graphs in (4.47) for $p = q = 1$, in (4.49) for $p = 1 < q$ and in (4.50) for $1 < p \leq q$. Let us state the flow graphs for the different settings explicitly. More detailed information about the derivatives can be found in Appendix 3.B.

Let $\mathbf{p = q = 1}$:
Since this is the non-differentiable anisotropic case we state the flow graph for every $1 \leq j \leq d$ individually. The gradient of the data-term is given as

$$\nabla D(f, g)_j = W(f_j - g_j).$$

As we state in Appendix 3.B the non-differentiable part of the anisotropic regularizer is if $f(v)_j = f(u)_j$ for some $(u, v) \in E$. Thus, $S_j = \{(u, v) \in E_d \mid f(u)_j \neq f(v)_j\}$ where $E_d = \{(u, v) \in E \mid u < v\}$ is the set of directed edges and we give the derivative in (3.50) as

$$\nabla R_{S_j}(f)_{u,j} = 2 \sum_{(u,v)\in S_j} \sqrt{w(u, v)}\, \operatorname{sign}(f(u)_j - f(v)_j).$$

Since, the anisotropic regularizer is defined in (4.59) for $p = q = 1$ as $\Phi_a^j(w(u, v), y) = w(u, v)|y|$ we can deduce from (4.47) we set $c(u, v) = \Phi_a^j(w(u, v), 1) = \sqrt{w(u, v)}$ for every $(u, v) \in S_j^c$. Thus, with

$$\nabla J_S(f_\Pi)_{u,j} = W_{uu}(f_\Pi(u)_j - g_{uj}) + \alpha \sum_{(u,v)\in S_j} \sqrt{w(u, v)}\, \operatorname{sign}(f_\Pi(u)_j - f_\Pi(v)_j)$$

the capacity c_j of the flow graph $G_{flow,j}$ is given as

$$\begin{cases} c_j(s,u) = \quad \nabla J_S(f_\Pi)_{u,j}, & \text{if } \nabla J_S(f_\Pi)_{u,j} \geq 0, \\ c_j(u,t) = -\nabla J_S(f_\Pi)_{u,j}, & \text{if } \nabla J_S(f_\Pi)_{u,j} < 0, \\ c_j(u,v) = \alpha\sqrt{w(u,v)}, & \forall\ (u,v) \in S_j^c. \end{cases} \tag{4.71}$$

Let $\mathbf{p = 1 < q}$:
For $q > 1$ we consider channel-isotropic regularization which is non-differentiable for $p = 1 < q$ when $f(u) = f(v)$ for some $(u,v) \in E$. The set where it is differentiable is given as $S = \{(u,v) \in E_d \mid f(u) \neq f(v)\}$. In the isotropic setting we have to choose some directions $\bar{\gamma}_B^A, \bar{\gamma}_{B^c}^A \in \mathbb{R}^d$ with $\|\bar{\gamma}_B^A + \bar{\gamma}_{B^c}^A\|_q = 1$ for every segment $A \in \Pi$. Let us only focus on $q = 2$ here since it is the case that we will investigate in this work. The gradient of the data-term is again given as above. The gradient of the differentiable part of the p-q-TV regularizer we compute in Appendix 3.B is stated in (3.53) and is given for $q = 2$ as

$$(\nabla R_S)_u = 2 \sum_{(u,v)\in S} \sqrt{w(u,v)}\ \frac{f(u) - f(v)}{\|f(v) - f(u)\|_2}. \tag{4.72}$$

Then we deduce that

$$\langle \nabla J_S(f_\Pi)_u, \bar{\gamma}_B^A + \bar{\gamma}_{B^c}^A \rangle_{\mathbb{R}^d} = \langle \nabla W_{uu}(f(u) - g(u)) + \alpha \sum_{(u,v)\in S} \sqrt{w(u,v)} \frac{f(u) - f(v)}{\|f(v) - f(u)\|_2}, \bar{\gamma}_B^A + \bar{\gamma}_{B^c}^A \rangle_{\mathbb{R}^d}.$$

The directional derivative of the non-differentiable parts is given as

$$R'_{S^c}(f; \bar{\boldsymbol{\gamma}}_{\boldsymbol{B}}) = 2 \sum_{(u,v)\in S^c} \Phi_i(w(u,v), 1) = 2 \sum_{(u,v)\in S^c} w(u,v)^{\frac{1}{2}} |\mathbf{1}_B(v) - \mathbf{1}_B(u)| \tag{4.73}$$

which leads to $c(u,v) = \alpha\sqrt{w(u,v)}$. The capacities for the flow graph are then given as

$$\begin{cases} c(s,u) = \quad \langle \nabla J_S(f_\Pi)_u, \bar{\gamma}_B^A + \bar{\gamma}_{B^c}^A \rangle_{\mathbb{R}^d}, & \text{if } \langle \nabla J_S(f_\Pi)_u, \bar{\gamma}_B^A + \bar{\gamma}_{B^c}^A \rangle_{\mathbb{R}^d} \geq 0, \\ c(u,t) = -\langle \nabla J_S(f_\Pi)_u, \bar{\gamma}_B^A + \bar{\gamma}_{B^c}^A \rangle_{\mathbb{R}^d}, & \text{if } \langle \nabla J_S(f_\Pi)_u, \bar{\gamma}_B^A + \bar{\gamma}_{B^c}^A \rangle_{\mathbb{R}^d} < 0, \\ c(u,v) = \alpha\sqrt{w(u,v)}, & \forall\ (u,v) \in S^c. \end{cases} \tag{4.74}$$

Let $\mathbf{1 < p \leq q}$:
In this case we have an regularizer that is differentiable everywhere, i.e., $\nabla R_S = \nabla R$. Hence, the derivative is given as in (3.55) and is equal 0 when $f(u) = f(v)$. It is *anisotropic* for $p = q$ and isotropic for $p < q$. Let us state the flow graph for the well-known case of $p = q = 2$. Then the derivative is given as

$$(\nabla R)_u = 2 \sum_{(u,v)\in E_d} w(u,v)(f(u) - f(v)). \tag{4.75}$$

Then we can deduce that we have for $u \in A$

$$\langle \nabla J(f)_u, \bar{\gamma}_B^A + \bar{\gamma}_{B^c}^A \rangle = \langle W_{uu}(f(u) - g(u)) + \alpha \sum_{(u,v)\in E_d} w(u,v)(f(u) - f(v)), \bar{\gamma}_B^A + \bar{\gamma}_{B^c}^A \rangle.$$

In the flow graph we do not have any horizontal edges between nodes in the graph, since there is no non-differentiable part. Thus, the flow graph only has connections to the terminals. The capacities are given as

$$\begin{cases} c(s,u) = \quad \langle \nabla J_S(f_\Pi)_u, \bar{\gamma}_B^A + \bar{\gamma}_{B^c}^A \rangle_{\mathbb{R}^d}, & \text{if } \langle \nabla J_S(f_\Pi)_u, \bar{\gamma}_B^A + \bar{\gamma}_{B^c}^A \rangle_{\mathbb{R}^d} \geq 0, \\ c(u,t) = -\langle \nabla J_S(f_\Pi)_u, \bar{\gamma}_B^A + \bar{\gamma}_{B^c}^A \rangle_{\mathbb{R}^d}, & \text{if } \langle \nabla J_S(f_\Pi)_u, \bar{\gamma}_B^A + \bar{\gamma}_{B^c}^A \rangle_{\mathbb{R}^d} < 0, \\ c(u,v) = 0, & \forall\ (u,v) \in E. \end{cases} \tag{4.76}$$

For this case one does not have to compute a graph cut since there are no edges between nodes to cut. In fact, it is just a segmentation of the nodes by checking if $\langle \nabla J_S(f_\Pi)_u, \bar{\gamma}_B^A + \bar{\gamma}_{B^c}^A \rangle_{\mathbb{R}^d}$ is positive or negative.

4.4.3 Directions for the ROF Partition Problem

In order to compute a splitting B by minimizing of the partition problem, we have to select feasible directions. In numerical experiments in Section 6.3 we show the influence of different heuristically selected directions. Nevertheless, we also provide an algorithm to optimize over the directions. The optimization problem is again given as

$$\operatorname*{argmin}_{B\in\mathcal{P}(V),\gamma_B^A,\gamma_{B^c}^A} \langle \nabla J_S(f_\Pi, g), \sum_{A\in\Pi} 1_{A\cap B}\bar{\gamma}_B^A - 1_{A\cap B^c}\bar{\gamma}_{B^c}^A \rangle + \frac{\alpha}{2} R'_{S^c}(f_\Pi; \bar{\boldsymbol{\gamma}}_{\boldsymbol{B}}) \ . \tag{4.77}$$

with $\nabla J_S(f_\Pi, g) = W(f_\Pi - g) + \frac{\alpha}{2}\nabla R_S(f_\Pi)$ and $\bar{\boldsymbol{\gamma}}_{\boldsymbol{B}} = \sum_{u\in A} \mathbf{1}_{A\cap B}(\bar{\gamma}_B^A + \bar{\gamma}_{B^c}^A)$. Let $\bar{\gamma}_A = \bar{\gamma}_B^A + \bar{\gamma}_{B^c}^A$, and hence, $\|\bar{\gamma}_A\|_2 = 1$. Then we can deduce that the partition problem of the alternating algorithm in (4.54) is the same as in the former section for the different settings of p and q. Note that the directional derivative of the non-differentiable part of the regularizer in (4.73) can be dropped for optimization since it does not depend on directions and only on B. The constraint problem to update the direction $\bar{\gamma}_A \in \mathbb{R}^d$ for every $A \in \Pi$ is then given as

$$\bar{\gamma}_A^{n+1} = \operatorname*{argmin}_{\|\bar{\gamma}_A\|_2=1} \sum_{u\in A} \langle W_{uu}(c_A - g(u)) + \frac{\alpha}{2}\nabla R_S(f_\Pi)_u, (\mathbf{1}_B(u) - \mathbf{1}_{B^c}(u))\bar{\gamma}_A \rangle \tag{4.78}$$

This can be solved by applying the Lagrange multiplier method. Let us set

$$\begin{aligned} \nu_A^B &= \sum_{u\in A} W_{uu}(c_A - g(u)) + \frac{\alpha}{2}\nabla R_S(f_\Pi)_u(\mathbf{1}_B(u) - \mathbf{1}_{B^c}(u)) \\ &= \langle \nabla J_S(f_\Pi, g), \mathbf{1}_{A\cap B} - \mathbf{1}_{A\cap B^c} \rangle \end{aligned}$$

then we have to minimize

$$\operatorname*{argmin}_{\|\bar{\gamma}_A\|_2} \langle \bar{\gamma}_A, \nu_A^B \rangle$$

for which the Lagrangian function minimization is given as

$$\underset{\bar{\gamma_A},\nu_A^B}{\operatorname{argmin}} \{\ \mathcal{L}(\bar{\gamma}_A,\lambda) = \langle \bar{\gamma}_A, \nu_A^B\rangle + \lambda(\langle \bar{\gamma}_A, \bar{\gamma}_A\rangle - 1)\ \} \tag{4.79}$$

Which can be solved by computing the optimality condition for both arguments which follows that

$$\bar{\gamma}_A = \pm \frac{\nu_A^B}{\|\nu_A^B\|_2}. \tag{4.80}$$

With $|\mathbf{1}_{A\cap B}(u) - \mathbf{1}_{A\cap B}(u)| = 1$ for every $u \in A$ and $|\mathbf{1}_{A\cap B}(u) - \mathbf{1}_{A\cap B}(u)| = 0$ for every $u \in V \setminus A$ we can deduce that

$$\|\nu_A^B\|_2 = (\sum_{u\in A} \nabla J_S(f_\Pi, g)_u^2)^{\frac{1}{2}} = \|\nabla J_S(f_\Pi, g)\cdot \mathbf{1}_A\|_2 = \|\nabla J_S(f_\Pi, g)_{|A}\|_2.$$

Then we can write down the alternating algorithm to solve the partition problem as

$$\begin{cases} B^{n+1} &= \underset{B\in\mathcal{P}(V)}{\operatorname{argmin}} \langle \nabla J_S(f_\Pi, g), \sum_{A\in\Pi} \mathbf{1}_{A\cap B}\bar{\gamma}_A^n\rangle \\ &\qquad\qquad +\alpha \sum_{(u,v)\in S^c} \sqrt{w(u,v)}\ |\mathbf{1}_B(v) - \mathbf{1}_B(u)| \\ \bar{\gamma}_A^{n+1} &= \dfrac{\langle \nabla J_S(f_\Pi, g), \mathbf{1}_{A\cap B} - \mathbf{1}_{A\cap B^c}\rangle}{\|\nabla J_S(f_\Pi, g)_{|A}\|_2}, \quad \forall A \in \Pi. \end{cases} \tag{4.81}$$

Note that we dropped the $\pm$ because in the partition problem to compute B we get the same solution for using the positive or negative direction.

Finally, we have derived how to construct the flow graphs and how to solve the reduced problems for the weighted p-q-ROF problem (4.58). Hence, we can apply the Cut-Pursuit algorithm Algorithm 1 to solve (4.58) for any given graph G and corresponding d-dimensional data set g which we state explicitly in Algorithm 3. We show in the numerical experiments presented in Chapter 6 different results by applying the Cut-Pursuit algorithm for solving ROF problems to grayscale and RGB images, and to 2D and 3D point cloud data. In the next chapter we introduce a different Cut-Pursuit algorithm that uses the same idea but solves optimization problems with an L_0-TV regularization.

Algorithm 3: Isotropic Cut-Pursuit Algorithm for (weighted) ROF Problems.

Input:
$G = (V, E, w) \leftarrow$ Undirected finite weighted graph
$g \leftarrow$ Some given d-dimensional data in $\mathcal{H}(V; \mathbb{R}^d)$
Initialization:
$\Pi^0 \leftarrow \{V\}$
$c^0 \leftarrow \operatorname{argmin}_{c \in \mathbb{R}^d} \; D(P^0 c)$
Method:
while $J'(f_{\Pi^k}) < 0$ **do**
  $\bar{\boldsymbol{\gamma}}_{\boldsymbol{B}}{}^{k+1} \leftarrow$ compute a direction for every $A \in \Pi$ heuristically or by solving (4.77) (e.g. algorithm (4.81)).
  $G_{flow} \leftarrow$ buildFlowGraph(G,f_{Π^k},$\bar{\boldsymbol{\gamma}}_{\boldsymbol{B}}{}^{k+1}$,$g$,$\alpha$, p, q) depending on p, q with (4.71), (4.74) or (4.76)
  $B^{k+1} \leftarrow$ computeMaxFlow(G_{flow}) via maxflow algorithm, that solve (4.40) (cf. [10], Section 3.4).
  $\Pi^{k+1} \leftarrow \{\text{connComp}(A \cap B), \text{connComp}(A \cap B) \mid A \in \Pi\}$.
  $P^{k+1} \leftarrow (\mathbf{1}_A)_{A \in \Pi^{k+1}}$
  $G_r = (V_r, E_r, w_r) \leftarrow$ with $V_r = \Pi^{k+1}$, E_r, w_r as in Proposition 4.16.
  $g_r \leftarrow$ reduced data according to data-term, e.g. (4.12).
  $c^{k+1} = \operatorname{argmin}_{c \in \mathcal{H}(\Pi^{k+1})} \; J(P^{k+1} c, g_r) \leftarrow$ reduced problem as in (4.63). Solve with method, e.g. the primal-dual algorithm on graphs as stated in Algorithm 4.14.
Result: The solution f^* for (4.58) and the corresponding partition Π^* representing the piecewise constant parts of f^*.

Appendix

4.A General Regularizer for ROF

In this appendix section, we show that the p-q-TV regularization is a special case of the Definition 4.1 and Definition 4.2 such that the statements of Lemma 4.3 and Lemma 4.4 hold. In particular, we focus on optimizing the following family of variational denoising problems for a fixed regularization parameter $\alpha > 0$

$$\underset{f \in \mathcal{H}(V)}{\operatorname{argmin}} \ \frac{1}{2}\|f - g\|_2^2 + \frac{\alpha}{2p}\|\nabla_w f\|_{p;q}^p \tag{4.82}$$

for $q, p \geq 1$ using the notation introduced in Section 3.1.2. As we have seen in Section 3.3.1 the p-q-TV term is channel-anisotropic and -isotropic for different p, q combinations. The p-q-TV regularizer can be rewritten as

$$\begin{aligned} R(f) &= \frac{1}{p} \sum_{u \in V} \sum_{v \sim u} w(u,v)^{\frac{p}{2}} \left(\sum_{j=1}^{d} |f(v)_j - f(u)_j|^q \right)^{\frac{p}{q}} \\ &= \sum_{(u,v) \in E} \Phi(w(u,v), f(v) - f(u)) \end{aligned} \tag{4.83}$$

with

$$\Phi(w(u,v), f(v) - f(u)) = \frac{1}{p} w(u,v)^{\frac{p}{2}} \left(\sum_{j=1}^{d} |f(v)_j - f(u)_j|^q \right)^{\frac{p}{q}}. \tag{4.84}$$

Note that Φ is $\frac{p}{2}$-homogenous in the first argument and p-homogenous in the second argument. We already introduced a primal-dual algorithm on graphs in Algorithm 3.22 to solve the problem (4.82) using the proximity operators in Appendix 3.C.

4.B Reduced Terms

In this appendix we derive different reduced terms that we use in the context of Cut-Pursuit algorithms, i.e. the (weighted) L_2-data-term and the p-q-TV regularizer.

4.B.1 Reduced (Weighted) L_2-Data-Term

In this subsection we state how the weighted L_2-data-term on an undirected graph $G = (V, E, w)$ as defined in Definition 3.2 is reduced on some given partition Π for V in the following proposition.

Proposition 4.15 (Piecewise Constant of (Weighted) L_2-Data-Term)**.**

Let $g \in \mathcal{H}(V)$ be some given data and $D(\cdot, g)$ be a weighted L_2-data-term as in Definition 2.6 with a diagonal weighting matrix $W \in \mathbb{R}^{N\times N}$. Plugging in any piecewise constant function $f_c = Pc \in \mathcal{S}_\Pi$ yields that minimizing $D_W(f_c, g)$ is equivalent to minimizing $D_{P^*WP}(c, g_r)$ with $g_r = (P^*WP)^{-1}P^*Wg$.

For $W = I$ we get the L_2-data-term and the equivalence of minimizing $D(f_c, g)$ and $D_{P^*P}(c, g_r)$ with $(g_r)_A = \dfrac{\sum_{u\in A} g(u)}{|A|}$ for every $A \in \Pi$.

Proof.
We start by write out the following

$$\begin{aligned} D_W(f_c, g) &= \frac{1}{2}\|f_c - g\|^2_{2,W} \\ &= \frac{1}{2}\langle W(Pc - g), Pc - g\rangle \\ &= \frac{1}{2}\langle W(Pc - g), Pc - g\rangle \\ &= \frac{1}{2}\big(\underbrace{\langle WPc, Pc\rangle}_{(1)} - \underbrace{\langle WPc, g\rangle}_{(2)} - \underbrace{\langle Pc, Wg\rangle}_{(3)} + \langle Wg, g\rangle\big). \end{aligned} \tag{4.85}$$

We will look at each of the 3 terms separately and try to reformulate them in some smart way. Note that the last term does not depend on c, and thus can be exchanged as we want to when minimizing since it is a constant value.

Reformulating (1):

$$\langle WPc, Pc\rangle = \langle P^*WPc, c\rangle. \tag{4.86}$$

Reformulating (2):

For this computation we use that $P^*WP = \operatorname{diag}(\sum_{i\in A} w_i)_{A\in\Pi}$ is a diagonal matrix. Then trying to reformulate such that it fits to (4.86)

$$\begin{aligned} \langle WPc, g\rangle &= \langle c, P^*Wg\rangle \\ &= \langle (P^*WP)^{-1}(P^*WP)c, P^*Wg\rangle \\ &= \langle P^*WPc, (P^*WP)^{-1}P^*Wg\rangle \\ &= \langle P^*WPc, g_r\rangle \end{aligned} \tag{4.87}$$

We denote $g_r = (P^*WP)^{-1}P^*Wg = \left(\dfrac{\sum_{i\in A} w_i g_i}{\sum_{i\in A} w_i}\right)_{A\in\Pi}$. Subtracting the reformulations of (2) in (4.87) from (1) in (4.86) we get

$$\langle P^*WPc, g\rangle - \langle P^*WPc, g_r\rangle = \langle P^*WPc, c - g_r\rangle. \tag{4.88}$$

This looks very similar to the definition of a weighted L_2-data-term as defined in Definition 2.6 and can be realized by subtracting

$$\langle P^*WPg_r, c - g_r\rangle = \langle P^*WPg_r, c\rangle - \langle P^*WPg_r, g_r\rangle.$$

The right term can be just added since it is only dependent on g_r, and hence a constant. Consequently, we have to reformulate (3) from (4.85) to $\langle P^*WPg_r, c\rangle$ as we do in the following.

Reformulating (3):

$$\langle P^*WPg_r, c\rangle = \langle (P^*WP)(P^*WP)^{-1}P^*Wg, c\rangle = \langle P^*Wg, c\rangle = \langle Pc, Wg\rangle. \tag{4.89}$$

Finally, when we combine the reformulations in (4.86), (4.87) and (4.89) and add $\langle P^*WPg_r, g_r\rangle$ we get the following

$$\begin{aligned} D_{P^*WP}(c, g_r) &= \frac{1}{2}\langle P^*WP(c - g_r), c - g_r\rangle \\ &= \frac{1}{2}(\langle P^*WPc, c\rangle - \langle P^*WPc, g_r\rangle - \langle P^*WPg_r, c\rangle + \langle P^*WPg_r, g_r\rangle). \end{aligned} \tag{4.90}$$

This concludes the general proof. Let us study the special case of $W = I$. Then the reduced data g_r becomes

$$g_r = \left(\frac{\sum_{u\in A} g(u)}{\sum_{u\in A} 1}\right)_{A\in\Pi} = \left(\frac{\sum_{u\in A} g(u)}{|A|}\right)_{A\in\Pi} \tag{4.91}$$

which is the *mean value* of the data g on every segment $A \in \Pi$. As we know from Lemma 4.6 $P^*P = \mathrm{diag}(|A|)_{A\in ß}$ a diagonal matrix. Minimizing the L_2-data-term given as

$$D(Pc, g) = \frac{1}{2}\|Pc - g\|_2^2 \tag{4.92}$$

is then equivalent to minimizing

$$\begin{aligned} D_{P^*P}(c, g_r) &= \frac{1}{2}\langle P^*P(c - g_r), c - g_r\rangle = \frac{1}{2}\langle \mathrm{diag}(|A|)_{A\in\Pi}(c - g_r), c - g_r\rangle \\ &= \frac{1}{2}\|c - g_r\|_{\mathrm{diag(|A|),2}}. \end{aligned} \tag{4.93}$$

□

4.B.2 Reduced p-q-TV Regularizer

In this subsection we will show the reduced p-q-TV regularizer for a given graph $G = (V, E, w)$ and partition Π of V. Before let us define

$$E_{AB} = \{(u, v) \in E \mid u \in A, v \in B\} \tag{4.94}$$

as the edges in E that are between two segments A and B. The following proposition yields how we can compute the $\mathrm{p-q-}TV$ regularizer in a reduced setting.

Proposition 4.16 (Reduced p-q-TV Regularizer)**.**
Let $G = (V, E, w)$ be a finite weighted graph. Define a reduced graph as $G_r = (\Pi, E_r, w_r)$ for

the partition Π *of* V *with the* reduced edge set E_r *defined as*

$$E_r = \{(A, B) \in \Pi \times \Pi \mid (A \times B) \cap E \neq \emptyset\} \tag{4.95}$$

and the defined reduced weighting *as* $w_r \colon E_r \to \mathbb{R}_+$ *as*

$$w_r(A, B) = \sum_{(u,v)\in E_{AB}} w(u, v)^{\frac{p}{2}}, \quad \forall (A, B) \in E_r. \tag{4.96}$$

Let $f_c \in \mathcal{S}_\Pi$ *with* $f_c = \sum_{A\in\Pi} c_A 1_A$ *and* $c = (c_A)_{A\in\Pi} \in \mathcal{H}(\Pi)$. *Then the following identity holds*

$$\|\nabla_w f\|_{p;q} = \|\nabla_{w_r} c\|_{p;q}. \tag{4.97}$$

Proof.
Let $f_c \in \mathcal{S}_\Pi$ as defined in (4.7) and $c \in \mathcal{H}(\Pi)$ such that $f_c = Pc = \sum_{A\in\Pi} \mathbf{1}_A c_A$. Then the p-q-TV regularizer applied to f_c can be reformulated as follows:

$$\begin{aligned} \|\nabla_w f\|_{p;q} &= \left[\sum_{u\in V} \sum_{v\sim u} w(u, v)^{\frac{p}{2}} \left(\sum_{j=1}^{d} |f_c(v)_j - f_c(u)_j|^q \right)^{\frac{p}{q}} \right]^{\frac{1}{p}} \\ &= \left[\sum_{A\in\Pi} \sum_{u\in A} \sum_{(u,v)\in E} w(u, v)^{\frac{p}{2}} \left(\sum_{j=1}^{d} |f_c(v)_j - f_c(u)_j|^q \right)^{\frac{p}{q}} \right]^{\frac{1}{p}}. \end{aligned} \tag{4.98}$$

The value of the term

$$w(u, v)^{\frac{p}{2}} \left(\sum_{j=1}^{d} |f_c(v)_j - f_c(u)_j|^q \right)^{\frac{p}{q}} \tag{4.99}$$

for some vertex $u \in A$ and segment $A \in \Pi$ is different if $v \in A$ or if $v \in B \in \Pi$ with $B \neq A$. For $v \in A$ we know that $f(v) = f(u) = c_A$. Plugging this into the term (4.99) yields

$$w(u, v)^{\frac{p}{2}} \left(\sum_{j=1}^{d} |c_{Aj} - c_{Aj}|^q \right)^{\frac{p}{q}} = 0. \tag{4.100}$$

On the other hand let $u \in A$ and $v \in B$ with $A, B \in \Pi$ and $A \neq B$ then $f(u) = c_A$ and $f(v) = c_B$. The term in (4.99) becomes for this choice

$$w(u, v)^{\frac{p}{2}} \left(\sum_{j=1}^{d} |c_{Bj} - c_{Aj}|^q \right)^{\frac{p}{q}}. \tag{4.101}$$

Using these definitions of E_{AB} in (4.15) and of the reduced edge set E_r in (4.95) to reformulate

(4.98) into the following:

$$\begin{aligned}\|\nabla_w f\|_{p;q} &= \left[\sum_{(A,B)\in E_r} \sum_{(u,v)\in E_{A,B}} w(u,v)^{\frac{p}{2}} \left(\sum_{j=1}^{d} |c_{Bj} - c_{Aj}|^q \right)^{\frac{p}{q}} \right]^{\frac{1}{p}} \\ &= \left[\sum_{(A,B)\in E_r} \left(\sum_{j=1}^{d} |c_{Bj} - c_{Aj}|^q \right)^{\frac{p}{q}} \sum_{(u,v)\in E_{A,B}} w(u,v)^{\frac{p}{2}} \right]^{\frac{1}{p}} \\ &= \left[\sum_{(A,B)\in E_r} \|c_B - c_A\|_q^p \sum_{(u,v)\in E_{A,B}} w(u,v)^{\frac{p}{2}} \right]^{\frac{1}{p}}.\end{aligned} \tag{4.102}$$

By using the definition of the *reduced weighting* as defined in (4.103) given as

$$w_r(A,B) = \sum_{(u,v)\in E_{AB}} w(u,v)^{\frac{p}{2}}, \quad \forall (A,B) \in E_r \tag{4.103}$$

we can finally deduce

$$\|\nabla_w f\|_{p;q} = \left[\sum_{(A,B)\in E_r} w_r(A,B) \|c_B - c_A\|_q^p \right]^{\frac{1}{p}} = \|\nabla_{w_r} c\|_{p;q}. \tag{4.104}$$

Note that the reduced weighting w_r depends on the selected p. □

4.C Derivation of Partition Problem

In this appendix we reformulate the minimization problem in (4.33). To do so, we start by stating the same initial situation as in Section 4.2.1.We have a current partition Π of V and aim to find a set $B \subset V$ to refine Π by computing $A \cap B$ and $A \cap B^c$ for every $A \in \Pi$ and define a new partition

$$\Pi_{new} = \{A \cap B, A \cap B^c \mid A \in \Pi\}. \tag{4.105}$$

Let $c \in \mathcal{H}(\Pi)$ be a solution of the former reduced ROF problem on Π and $f_\Pi = Pc = \sum_{A\in\Pi} 1_A c_A$ be the corresponding piecewise constant function in $\mathcal{S}_\Pi$. Assume now we have some binary splitting of the partition Π described by $B \in V$ and a corresponding piecewise constant function in $\mathcal{H}(\Pi_{new})$ given as

$$\tilde{f}_B = \sum_{A\in\Pi} (1_{A\cap B} d_{A\cap B} + 1_{A\cap B^c} d_{A\cap B^c}) \tag{4.106}$$

with some vectors $d_{A\cap B}, d_{A\cap B^c} \in \mathbb{R}^d$ for any combination of $A \in \Pi$ and $B \subset V$. Note that this function is piecewise constant on $A \cap B \subset A$ and $A \cap B^c \subset A$ for every $A \in \Pi$, and thus

$\tilde{f}_B \in \mathcal{S}_{\Pi_{new}}$. The energy functional becomes

$$J(\tilde{f}_B) = D\Big(\sum_{A\in\Pi}\left(1_{A\cap B}d_{A\cap B} + 1_{A\cap B^c}d_{A\cap B^c}\right), g\Big) + \frac{\alpha}{2}R\Big(\sum_{A\in\Pi}\left(1_{A\cap B}d_{A\cap B} + 1_{A\cap B^c}d_{A\cap B^c}\right)\Big).$$

We now want to find a partitioning described by $B \subset V$ such that the energy of $J(\tilde{f}_B)$ decreases the most compared to the current energy $J(f)$. Thus, we optimize to find the set $B \subset V$ that minimizes

$$\underset{B\subset V}{\operatorname{argmin}}\ D(\tilde{f}_B, g) - D(f_\Pi, g) + \frac{\alpha}{2}\ \Big(R(\tilde{f}_B) - R(f_\Pi)\Big) \tag{4.107}$$

with fixed vectors for $d_{A\cap B}$ and $d_{A\cap B^c}$ as we already stated in (4.33). If we argue in Section 4.2.1 we use for every segment $A \in \Pi$ two directions $\bar{\gamma}_B^A, \bar{\gamma}_{B^c}^A$ with $\|\bar{\gamma}_B^A\|_2 = 1$ and $\|\bar{\gamma}_{B^c}^A\|_2 = 1$ and a small perturbation $\varepsilon > 0$. Then we define for every $A \in \Pi$

$$\gamma_B^A = \varepsilon\bar{\gamma}_B^A, \quad \gamma_{B^c}^A = \varepsilon\bar{\gamma}_{B^c}^A. \tag{4.108}$$

Then we set

$$d_{A\cap B} = c_A + \gamma_B^A, \quad d_{A\cap B^c} = c_A - \gamma_{B^c}^A \tag{4.109}$$

for every $A \in \Pi$. Then the function (4.106) becomes

$$\begin{aligned}\tilde{f}_B &= \sum_{A\in\Pi}\left(\mathbf{1}_{A\cap B}d_{A\cap B} + \mathbf{1}_{A\cap B^c}d_{A\cap B^c}\right)\\ &= \sum_{A\in\Pi}\Big(\mathbf{1}_{A\cap B}(c_A + \gamma_B^A) + \mathbf{1}_{A\cap B^c}(c_A - \gamma_{B^c}^A)\Big)\\ &= \sum_{A\in\Pi}\Big(\mathbf{1}_A c_A + \mathbf{1}_{A\cap B}\gamma_B^A - \mathbf{1}_{A\cap B^c}\gamma_{B^c}^A\Big)\end{aligned} \tag{4.110}$$

We can simplify (4.110) even more and make it only depending on $\mathbf{1}_B$ using the equivalence

$$\mathbf{1}_V = \mathbf{1}_B + \mathbf{1}_{B^c} \ \Leftrightarrow\ \mathbf{1}_{B^c} = \mathbf{1}_V - \mathbf{1}_B.$$

This implies that for every $A \in \Pi$ we can deduce

$$\mathbf{1}_A = \mathbf{1}_{A\cap B} + \mathbf{1}_{A\cap B^c} \ \Leftrightarrow\ \mathbf{1}_{A\cap B^c} = \mathbf{1}_A - \mathbf{1}_{A\cap B}.$$

Then we can rewrite for every $A \in \Pi$ that

$$\begin{aligned}\mathbf{1}_{A\cap B}\gamma_B^A - \mathbf{1}_{A\cap B^c}\gamma_{B^c}^A &= \mathbf{1}_{A\cap B}\gamma_B^A - (\mathbf{1}_A - \mathbf{1}_B)\gamma_{B^c}^A\\ &= \mathbf{1}_{A\cap B}(\gamma_B^A + \gamma_{B^c}^A) - \mathbf{1}_A\gamma_{B^c}^A\\ &= \varepsilon(\mathbf{1}_{A\cap B}(\bar{\gamma}_B^A + \bar{\gamma}_{B^c}^A) - \mathbf{1}_A\bar{\gamma}_{B^c}^A)\\ &= \varepsilon\bar{\boldsymbol{\gamma}}^{\boldsymbol{A}}\end{aligned}$$

and define the directions

$$\varepsilon\bar{\boldsymbol{\gamma}} = \varepsilon \sum_{A\in\Pi} \bar{\boldsymbol{\gamma}}^{\boldsymbol{A}}$$

Consequently, these rewriting follows that we actually are investigating the problem

$$\operatorname*{argmin}_{B\in\mathcal{P}(V)} J(f+\varepsilon\bar{\boldsymbol{\gamma}}) - J(f). \tag{4.111}$$

Dividing a functional by some scalar does not change the solution, thus we can state the following identity:

$$\operatorname*{argmin}_{B\in\mathcal{P}(V)} J(f+\varepsilon\bar{\boldsymbol{\gamma}}) - J(f) = \operatorname*{argmin}_{B\in\mathcal{P}(V)} \frac{J(f+\varepsilon\bar{\boldsymbol{\gamma}}) - J(f)}{\varepsilon}. \tag{4.112}$$

We know that $J(f) = D(f) + \frac{\alpha}{2}R(f)$ is directional differentiable due to the differentiable term D and the directional differentiable term R. Hence, taking the limit with $\varepsilon \to 0$ the minimization functional yields the directional derivative as defined in (2.3) as

$$J'(f;\bar{\boldsymbol{\gamma}}) = \lim_{\varepsilon\to 0} \frac{J(f+\varepsilon\bar{\boldsymbol{\gamma}}) - J(f)}{\varepsilon}.$$

The solution of the minimization in (4.112) is equal to the minimizer in the limit with $\varepsilon \to 0$. Thus, we can deduce that

$$\begin{aligned}\operatorname*{argmin}_{B\in\mathcal{P}(V)} J(f+\varepsilon\bar{\boldsymbol{\gamma}}) - J(f) &= \operatorname*{argmin}_{B\in\mathcal{P}(V)} \lim_{\varepsilon\to 0} \frac{J(f+\varepsilon\bar{\boldsymbol{\gamma}}) - J(f)}{\varepsilon} \\ &= \operatorname*{argmin}_{B\in\mathcal{P}(V)} J'(f;\bar{\boldsymbol{\gamma}}).\end{aligned}$$

As a result we have derived that we have to minimize the directional derivative of the energy J into a direction certain $\bar{\boldsymbol{\gamma}}$ that depends on B and is provided with two a-priori selected direction $\bar{\gamma}^A_B, \bar{\gamma}^A_{B^c}$ for every segment $A \in \Pi$ to find a set $B \in \mathcal{P}(V)$ that splits up the current partition Π correctly. To wrap this derivation up let us reformulate the directional derivative in even more detail. In the following we will define

$$\begin{aligned}\bar{\boldsymbol{\gamma}} &= \bar{\boldsymbol{\gamma}}_{\boldsymbol{B}} + \boldsymbol{r} \\ \bar{\boldsymbol{\gamma}}_{\boldsymbol{B}} &= \sum_{A\in\Pi} \mathbf{1}_{A\cap B}(\bar{\gamma}^A_B + \bar{\gamma}^A_{B^c}) \\ \bar{\boldsymbol{r}} &= -\sum_{A\in\Pi} \mathbf{1}_A \bar{\gamma}^A_{B^c}\end{aligned} \tag{4.113}$$

Let us first take a look at the *differentiable* data-term D. Since it is differentiable the directional derivative is directly given as

$$D'(f_\Pi;\bar{\boldsymbol{\gamma}}) = \langle \nabla D(f_\Pi), \bar{\boldsymbol{\gamma}}\rangle = \langle \nabla D(f_\Pi), \bar{\boldsymbol{\gamma}}_{\boldsymbol{B}}\rangle + \langle \nabla D(f_\Pi), \bar{\boldsymbol{r}}\rangle.$$

where $\bar{\boldsymbol{\gamma}}_{\boldsymbol{B}}$ is depending on B, while $\bar{\boldsymbol{r}}$ is the rest that is not depending on B. Since $\bar{\boldsymbol{r}}$ is a constant we can drop it for the minimization with respect to B. For the data-term we then

would just minimize

$$D'(f;\bar{\boldsymbol{\gamma}}_{\boldsymbol{B}}).$$

We compute the directional derivative for the different isotropic regularizer Φ_i since - as we pointed out before - the anisotropic regularizer can be treated as a special case. Due to the definition of R_i as sums over the functions Φ_i in Definition 4.2 the regularizer can be separated into differentiable and non-differentiable parts. In fact, the differentiability depends directly on the choice of Φ_a and Φ_i. As we only inspect the isotropic regularizer we set $R = R_i$. Let $S = \{(u,v) \in E \mid f(u) \neq f(v)\}$ be the where R is differentiable by definition in Definition 4.2. Hence, R is non-differentiable in S^c. We can split the directional derivative of R up into

$$R'(f_\Pi;\bar{\boldsymbol{\gamma}}) = \langle \nabla R_S(f_\Pi), \bar{\boldsymbol{\gamma}}\rangle + R'_{S^c}(f_\Pi;\bar{\boldsymbol{\gamma}}) \tag{4.114}$$

with R_S the differentiable and R_{S^c} the non-differentiable part. Note that the gradient of R_S is pointwise for every $u \in V$, and thus $\nabla R_S(f) \in \mathbb{R}^N$. With the same argument as in (4.114) we can exchange the direction for the differentiable part R_S with $\bar{\boldsymbol{\gamma}}_{\boldsymbol{B}}$, i.e.,

$$\langle \nabla R_s(f_\Pi), \bar{\boldsymbol{\gamma}}_{\boldsymbol{B}}\rangle.$$

To reformulate the non-differentiable part we have to check for the different anisotropic R_a and isotropic R_i definitions of R in Definition 4.1 and Definition 4.2 respectively. We assume that $f(u) = f(v)$ if, and only if, $u, v \in A$ for some $A \in \Pi$. Then we can deduce the following:

$$\begin{aligned}
\bar{\boldsymbol{\gamma}}_{\boldsymbol{B}}(v) &= \mathbf{1}_{A\cap B}(v)(\bar{\gamma}^A_B + \bar{\gamma}^A_{B^c}),\\
\bar{\boldsymbol{\gamma}}_{\boldsymbol{B}}(u) &= \mathbf{1}_{A\cap B}(u)(\bar{\gamma}^A_B + \bar{\gamma}^A_{B^c})\\
\bar{\boldsymbol{\gamma}}_{\boldsymbol{B}}(v) - \bar{\boldsymbol{\gamma}}_{\boldsymbol{B}}(u) &= (\mathbf{1}_{A\cap B}(v) - \mathbf{1}_{A\cap B}(u))(\bar{\gamma}^A_B + \bar{\gamma}^A_{B^c})\\
\bar{\boldsymbol{r}}(v) - \bar{\boldsymbol{r}}(u) &= \gamma^A_{B^c} - \gamma^A_{B^c} = 0.
\end{aligned}$$

Then for with $\Phi_i(w(u,v),0) = 0$ we can compute

$$\begin{aligned}
R'_{S^c}(f_\Pi;\bar{\boldsymbol{\gamma}}) &= \lim_{\varepsilon\to 0} \sum_{(u,v)\in S^c} \frac{\Phi_i\big(w(u,v), \varepsilon(\bar{\boldsymbol{\gamma}}_{\boldsymbol{B}}(v) - \bar{\boldsymbol{\gamma}}_{\boldsymbol{B}}(u) + \bar{\boldsymbol{r}}(v) - \bar{\boldsymbol{r}}(u))\big)}{\varepsilon}\\
&= \lim_{\varepsilon\to 0} \sum_{(u,v)\in S^c} \frac{\Phi_i\big(w(u,v), \varepsilon(\mathbf{1}_{A\cap B}(v) - \mathbf{1}_{A\cap B}(u))(\bar{\gamma}^A_B + \bar{\gamma}^A_{B^c})\big)}{\varepsilon}\\
&= \lim_{\varepsilon\to 0} \sum_{(u,v)\in S^c} \frac{\varepsilon^p\Phi_i\big(w(u,v), (\mathbf{1}_{A\cap B}(v) - \mathbf{1}_{A\cap B}(u))(\bar{\gamma}^A_B + \bar{\gamma}^A_{B^c})\big)}{\varepsilon}\\
&= \lim_{\varepsilon\to 0} \varepsilon^{p-1} \sum_{(u,v)\in S^c} \Phi_i\big(w(u,v), (\mathbf{1}_{A\cap B}(v) - \mathbf{1}_{A\cap B}(u))(\bar{\gamma}^A_B + \bar{\gamma}^A_{B^c})\big).
\end{aligned}$$

Since the rest $\bar{\boldsymbol{r}}$ dropped in the computation this is equivalent for the direction $\bar{\boldsymbol{\gamma}}_{\boldsymbol{B}}$, i.e., $R'_{S^c}(f_\Pi;\bar{\boldsymbol{\gamma}}) = R'_{S^c}(f_\Pi;\bar{\boldsymbol{\gamma}}_{\boldsymbol{B}})$. We set $p \geq 1$ by definition. Thus, we see that for $p > 1$ we get

$R'_{S^c}(f_\Pi; \bar{\gamma}) = 0$, which is analog to what we see in Appendix 3.B. For $p = 1$ we deduce

$$R'_{S^c}(f_\Pi; \bar{\gamma}_{\boldsymbol{B}}) = \sum_{(u,v)\in S^c} |\mathbf{1}_{A\cap B}(v) - \mathbf{1}_{A\cap B}(u)| \Phi_i\big(w(u,v), (\bar{\gamma}^A_B + \bar{\gamma}^A_{B^c})\big) \tag{4.115}$$

since Φ_i is 1-homogeneous in the second argument.

Finally, we have rewritten the former partition problem (4.107) into the following new partition problem

$$\operatorname*{argmin}_{B\in\mathcal{P}(V)} \left\{ \langle \nabla D(f_\Pi) + \frac{\alpha}{2} \nabla R_S(f_\Pi), \bar{\gamma}_{\boldsymbol{B}} \rangle + \frac{\alpha}{2} R'_{S^c}(f_\Pi; \bar{\gamma}_{\boldsymbol{B}}) \right\}. \tag{4.116}$$

5

Cut-Pursuit for Minimal Partition Problems

In this chapter, we want to introduce a family of *(general) minimal partition problems* and present multiple Cut-Pursuit algorithms to find solutions of these problems. This family of problems is related to the piecewise constant Mumford-Shah [19, 57] minimization problem that is given on a finite weighted graph $G = (V, E, w)$ as

$$\underset{C\in\mathcal{P}(E),f\in\mathcal{H}(V)}{\operatorname{argmin}} \sum_{u\in V}(f(u)-g(u))^2 + \alpha|C| \tag{5.1}$$

where $|C|$ is the length of all boundaries in $f \in \mathcal{H}(V)$ over the edge set E and $\alpha > 0$ is a regularization parameter. The length of all boundaries is equivalent to the total number *jumps* in f over edges the edges E. We denote every edge $(u, v) \in E$ as a *jump* if $f(v) \neq f(u)$. The *jump set* $\mathcal{J}$ of $f \in \mathcal{H}(V)$ is then the set of all edges where a jump is, i.e.,

$$\mathcal{J} = \{(u, v) \in E \mid f(v) \neq f(u)\}.$$

In Figure 5.1 we can see the jump set of a TV solution as every green peak that corresponds to the absolute value of each jump in the solution. Hence, we can deduce that $|C| = |\mathcal{J}|$. Let us introduce the following measure.

Definition 5.1 (L_0-"norm")**.**
The L_0-"norm" for some $x \in \mathbb{R}^{n\times d}$ as

$$\|x\|_0 = \sum_{i=1}^{n} \begin{cases} 1, & \text{if } \|x_i\| > 0, \\ 0, & \text{else.} \end{cases} \tag{5.2}$$

In Figure 5.2 we visualize this norm in 1D, i.e., $x \in \mathbb{R}$. Note that this function is highly non-convex and not continuous. With this measure we can tell if a vertex function $f \in \mathcal{H}(V)$ has a jump over an edge $(u, v) \in E$ when $\|f(v) - f(u)\|_0 = 1$ and no jump when it is equal to 0. Then we can reformulate the regularization term of the Mumford-Shah problem as

$$|C| = |\mathcal{J}| = \sum_{(u,v)\in E} \|f(v) - f(u)\|_0.$$

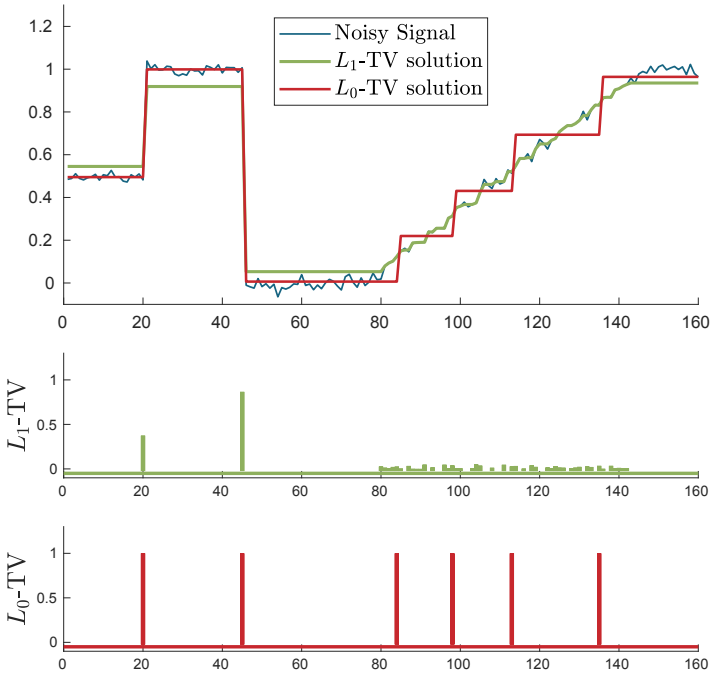


Figure 5.1.: Illustration of a noisy 1D signal denoised with L_0-TV and L_1-TV and their corresponding jump sets and costs per jump.

Since we are working on finite weighted graphs we also want to incorporate a weighting function to this regularization and define the following.

Definition 5.2 (Isotropic Weighted L_0-TV)**.**
Let $G = (V, E, w)$ be a finite weighted graph and $f \in \mathcal{H}(V)$. Then the isotropic L_0-TV *regularizer is defined as*

$$R_{i,0}(f) = \sum_{(u,v)\in E} \sqrt{w(u,v)}\|f(v) - f(u)\|_0. \tag{5.3}$$

We denote this as the *(channel-)isotropic weighted L_0-TV* due to its relation to the weighted channel-isotropic TV regularization on graphs in (3.17). To distinguish between L_0-TV and that TV regularizer we denote it as L_1-TV regularization. We can define such a regularization also in an anisotropic form as in (3.14).

Definition 5.3 (Anisotropic Weighted L_0-TV)**.**
Let $G = (V, E, w)$ be a finite weighted graph and $f \in \mathcal{H}(V)$. Then the anisotropic L_0-TV *regularizer is defined as*

$$R_{a,0}(f) = \sum_{(u,v)\in E} \sum_{j=1}^{d} \sqrt{w(u,v)}\|f(v)_j - f(u)_j\|_0. \tag{5.4}$$

As the anisotropic case is again a special variant of the isotropic case as we discuss in

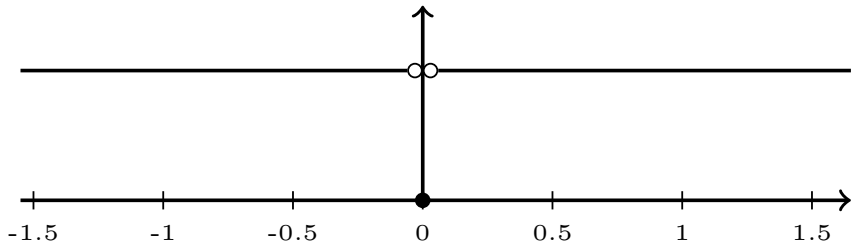


Figure 5.2.: The 1D L_0 function defined in (5.1).

Lemma 4.4 we only consider the isotropic case here. Then we can state the weighted variant of the problem (5.1) with the definitions from above on the graph G as

$$\underset{f\in\mathcal{H}(V)}{\operatorname{argmin}} \ \frac{1}{2}\|f-g\|_2^2 + \frac{\alpha}{2}\sum_{(u,v)\in E}\sqrt{w(u,v)}\|f(v)-f(u)\|_0. \tag{5.5}$$

We denote this as the L_0-ROF problem. To make this more general we replace the L_2-data-term with a continuously differentiable data-term D and then get

$$\underset{f\in\mathcal{H}(V)}{\operatorname{argmin}} \ D(f,g) + \frac{\alpha}{2}\sum_{(u,v)\in E}\sqrt{w(u,v)}\|f(v)-f(u)\|_0. \tag{5.6}$$

We call this the *(general) minimal partition problem* in this work due to its relation to the minimal partition problem in (5.1).

Let us compare this problem to the L_1-TV-regularized problem we introduce in (3.14) and (3.17). In Figure 5.1 we plot the L_0-TV (red line) and L_1-TV solution (green line) to the noisy input signal (blue line). Additionally, we illustrate the jump sets of both solutions in the two bottom plots with the height of the peaks as the cost of the jumps. In the L_1-TV regularization a jump costs the height of the jump while in the L_0-Regularizer every jump independently of its height costs the weight of the edge. Extending this considerations to an image the weighted L_1-TV measures the weighted height of the jumps for each boundary in the image and the weighted L_0-TV measures the weighted length of the boundary. Hence, regularizing with these terms yields to different penalizations. The L_1-TV term penalizes a jump by its weighted height, and hence, it is as costly to have successive small jumps in the solutions as one large jump. This is what we can see in the L_1-TV plot in Figure 5.1 on the right side where the smooth - but noisy - part of the data is described with multiple small jumps. On the other side the L_0-TV regularization penalizes any jump, and hence, tries to suppress as many jumps as possible to describe the data. This is the reason for only 4 jumps to describe this part. Another artifact that appears in L_1-TV solutions is the *loss of contrast* as we can see in Figure 5.1 where the height of the piecewise constant solutions does not fit to the data. This emerges from the penalization of the height of the jump which dominates over the data-term here, and hence, reduces the height of the jump between the piecewise constant functions. The L_0-TV-regularized solution on the other hand does only care about the presence of a jump which lets the data-term take over of the actual value of the segment.

This leads to no loss of contrast. Note that we show in Section 6.4 that Cut-Pursuit to solve L_1-TV problems yields the possibility to apply debiasing to the solution. These observations and considerations can be directly applied to images and point clouds as we see in Section 6.4.

We now understand what to expect from this regularization and have to face the problem that this is a highly non-convex optimization problem. Consequently, applying the primal-dual algorithm from Algorithm 3.20 that is designed for convex problems would not hold. Hence, let us point out just a few methods that tackle this problem with different approaches.

One approach to tackle this problem is to solve it with some graph cut approaches. This is for example done in [92] and [5] for piecewise constant Mumford-Shah problems. In [35] Grady et. al. introduce a method to even solve the piecewise-smooth Mumford-Shah problem. Even though it is a non-convex problem, and thus the primal-dual algorithm cannot be applied directly onto this problem, there exist work to solve this problem with a modified primal-dual algorithm. In [64] Pock et. al. introduce a convex relaxation to the unweighted L_0-TV regularization and apply the primal-dual algorithm for optimization. Strekalovskiy and Cremers introduce in [83] a primal-dual algorithm to solve the problem by rewriting the proximity operator of the convex conjugate of the L_0-norm by the Moreau's identity (cf. [71]). In both works they are investigating a piecewise-smooth Mumford-Shah problem which can be reformulated into a piecewise constant one. Landrieu et. al. also introduce a Cut-Pursuit related algorithm to solve the non-convex minimal partition problem with a L_0-TV regularization in [46]. As the idea is still inspired and created using the ideas of the Cut-Pursuit algorithm for (isotropic) TV problems it is in fact much harder to solve due to the high non-convexity. Thus, they introduce a different algorithm to find the best binary splitting with an alternating algorithm. It consists of a graph cut step to solve a given partition problem and then updates the values in the newly introduced segments. Then it uses these updated values to compute new graph cuts until a optimal cut was found for the partition problem. Additionally, it is hard to find a global optimum of the problem such that they also introduce different merging and splitting methods to get a better solution. Note that the data-term D for their problem has to be separable over the nodes, i.e., $D(f) = \sum_{u \in V} D_u(f(u))$. In the following section we introduce the generic Cut-Pursuit algorithm we use in this work.

5.1 Minimal Partition Cut-Pursuit Algorithm

In this section, we want to introduce a generic Cut-Pursuit method to solve the general minimal partition problem (5.6) that is closely related to the method in [46]. We call this algorithm L_0-Cut-Pursuit in this work. Again we extend their method to an isotropic setting and also state different optimization strategies.

Let us again solely consider the isotropic case since the isotropic case is a special case of the anisotropic case. The minimization problem we are interested in to solve here is given as

$$\operatorname*{argmin}_{f \in \mathcal{H}(V)} \; D(f, g) + \frac{\alpha}{2} \sum_{(u,v) \in E} \sqrt{w(u,v)} \|f(v) - f(u)\|_0 \tag{5.7}$$

where D is continuously differentiable. We call this problem the *minimal partition problem* from now on. To derive the L_0-Cut-Pursuit algorithm we want to apply the same procedure

as in Section 4.2.1 by assuming that we have given some data g on a finite weighted graph G, are in the k'th iteration of the algorithm where we have given a partition Π^k of V with a corresponding reduced graph G_r^k. We have also given the solution $f_{\Pi^k} = P_k c^k$ of the reduced problem. Let us introduce the family of functions depending on some splitting $B \in \mathcal{P}(V)$ and some values $d_{A\cap B}, d_{A\cap B^c} \in \mathbb{R}^d$ with $d_{A\cap B} \neq d_{A\cap B^c}$ as

$$f_B = \sum_{A\in\Pi} \mathbf{1}_{A\cap B} d_{A\cap B} + \mathbf{1}_{A\cap B^c} d_{A\cap B^c}.$$

The aim is again to find a B such that the energy $J(f_B) < J(f_{\Pi^k})$. Additionally, we assume that for any $(A_i, A_j) \in E_r$ we have that $d_{A_i\cap B} \neq d_{A_j\cap B}$, $d_{A_i\cap B} \neq d_{A_j\cap B^c}$, $d_{A_i\cap B^c} \neq d_{A_j\cap B}$ and $d_{A_i\cap B^c} \neq d_{A_j\cap B^c}$. The partition problem to find a split $B \in \mathcal{P}(V)$ is then given as

$$\operatorname*{argmin}_{B\in\mathcal{P}(V)} \; D(f_B, g) + \frac{\alpha}{2} \sum_{(u,v)\in E} \sqrt{w(u,v)}\; \|f_B(v) - f_B(v)\|_0. \tag{5.8}$$

We again aim to solve this problem using an efficient graph cut and start by computing the L_0-TV regularization applied to a piecewise constant function in $\mathcal{H}(\Pi^k)$.

Lemma 5.4.
Let $G = (V, E, w)$ be a finite weighted graph and Π a given partition and f_B as above. Moreover, let $S^c = \{(u, v) \in E \mid f_\Pi(u) = f_\Pi(v)\}$. Then we can rewrite the L_0-TV regularizer as

$$R_{i,0}(f_B) = 2 \sum_{(u,v)\in S^c} \sqrt{w(u,v)}\; |\mathbf{1}_B(v) - \mathbf{1}_B(u)|. \tag{5.9}$$

Proof.
Let us investigate the L_0-TV regularizer $R_{i,0}$ by plugging in f_B. From (5.3) we first get the following

$$\begin{aligned} R_{i,0}(f_B) &= \sum_{(u,v)\in E} \sqrt{w(u,v)}\; \|f_B(v) - f_B(u)\|_0 \\ &= 2 \sum_{(u,v)\in E_d} \sqrt{w(u,v)} \|f_B(v) - f_B(u)\|_0 \end{aligned}$$

where $E_d = \{(u, v) \in E \mid u < v\}$ is the directed edge set. We also denote the set of edges $E_{A_1A_2} = \{(u, v) \in E, u \in A_i, v \in A_j\}$ as the set of directed edges in E that are between the segments A_i and A_j in Π^k. Let us consider the two cases where $u, v \in A$ with $A \in \Pi^k$ and $u \in A_i, v \in A_j$ for $A_i \neq A_j \in \Pi$.
Let us start with $u, v \in A$ for some $A \in \Pi$. Then we see that

$$f_B(v) - f_B(u) = (\mathbf{1}_{A\cap B}(v) - \mathbf{1}_{A\cap B}(u)) d_{A\cap B} + (\mathbf{1}_{A\cap B^c}(v) - \mathbf{1}_{A\cap B^c}(u)) d_{A\cap B^c}.$$

Further, we can deduce that

$$\mathbf{1}_{A\cap B}(v) - \mathbf{1}_{A\cap B}(u) = \begin{cases} 1, & \text{if } u \in B^c, v \in B, \\ -1, & \text{if } u \in B, v \in B^c, \\ 0, & \text{if } u, v \in B \text{ or } u, v \in B^c \end{cases} \tag{5.10}$$

$$= -(\mathbf{1}_{A\cap B^c}(v) - \mathbf{1}_{A\cap B^c}(u))$$

which then yields

$$f_B(v) - f_B(u) = (\mathbf{1}_{A\cap B}(v) - \mathbf{1}_{A\cap B}(u))(d_{A\cap B} - d_{A\cap B^c}).$$

Additionally, we see that we can deduce from (5.10) that

$$\|\mathbf{1}_{A\cap B}(v) - \mathbf{1}_{A\cap B}(u)\|_0 = \begin{cases} 1, & \text{if } u \in B, v \in B^c \text{ or } u \in B^c, v \in B, \\ 0, & \text{if } u, v \in B \text{ or } u, v \in B^c \end{cases} \tag{5.11}$$

$$= |\mathbf{1}_{A\cap B}(v) - \mathbf{1}_{A\cap B}(u)|$$

Plugging this into the L_0-norm and using the fact that $\|d_{A\cap B} - d_{A\cap B^c}\|_0 = 1$ since $d_{A\cap B} - d_{A\cap B^c} \neq 0$ we get

$$\begin{aligned} \|f_B(v) - f_B(v)\|_0 &= \|(\mathbf{1}_{A\cap B}(v) - \mathbf{1}_{A\cap B}(u))(d_{A\cap B} - d_{A\cap B^c})\|_0 \\ &= |\mathbf{1}_{A\cap B}(v) - \mathbf{1}_{A\cap B^c}(u)| \; \|d_{A\cap B} - d_{A\cap B^c}\|_0 \\ &= |\mathbf{1}_{A\cap B}(v) - \mathbf{1}_{A\cap B^c}(u)| \end{aligned} \tag{5.12}$$

for $u, v \in A$ for some $A \in \Pi^k$.

Let us observe edges $(u, v) \in E$ with $u \in A_i, v \in A_j$ for $A_i, A_j \in \Pi^k$ with $A_i \neq A_j$. Then the difference from above is given as

$$f_B(v) - f_B(u) = \mathbf{1}_{A_j\cap B}(v)d_{A_j\cap B} - \mathbf{1}_{A_i\cap B}(u)d_{A_i\cap B} + \mathbf{1}_{A_j\cap B^c}(v)d_{A_j\cap B^c} - \mathbf{1}_{A_i\cap B^c}(u)d_{A_i\cap B^c}.$$

It is easy to see with the assumptions from above that this is always non-zero, and thus $\|f_B(v) - f_B(u)\|_0 = 1$ in this case. Since this is a constant we can drop this for minimization. With these considerations we can deduce with $S = \{(u, v) \in E_d \mid f_\Pi(u) \neq f_\Pi(v)\} = \{(u, v) \in E_d \mid u \in A_i, v \in A_j, A_i \in \Pi, A_j \neq A_i\}$ and with its complement $S^c = \{(u, v) \in E_d \mid u, v \in A, A \in \Pi^k\}$ that the regularizer is given as

$$R_{i,0}(f_B) = 2 \sum_{(u,v)\in S^c} \sqrt{w(u, v)} \; |\mathbf{1}_B(v) - \mathbf{1}_B(u)| \tag{5.13}$$

which is exactly the term as for the non-differentiable part of the regularizer in the partition problem (4.39) of the Cut-Pursuit algorithm for L_1-TV regularization.

□

Using the result of this lemma we can rewrite the partition problem from (5.8) as the partition problem

$$\operatorname*{argmin}_{B\in\mathcal{P}(V)} D(f_B, g) + \alpha \sum_{(u,v)\in S^c} \sqrt{w(u,v)}\ |\mathbf{1}_B(v) - \mathbf{1}_B(u)|. \tag{5.14}$$

The regularization term in this problem is submodular as in the partition problem of the Cut-Pursuit algorithm to solve L_1-TV problems, and hence, can be solved using a graph cut algorithm. The question is now how we can reformulate the data-term in some manner to apply a graph cut to solve the whole problem. Let us try to apply the same ideas as in Chapter 4 to find a directional derivative. To this end, we assumed in Section 4.2.1 that we can rewrite $d_{A\cap B} = c_A + \varepsilon\bar{\gamma}^A_B$ and $d_{A\cap B^c} = c_A - \varepsilon\bar{\gamma}^A_{B^c}$ for every segment $A \in \Pi^k$ for $c_A = f^k_\Pi(u),\ u \in A$ and some $\varepsilon > 0$. For simplicity we only use one direction for each segment, i.e., $\bar{\gamma}^A = \bar{\gamma}^A_B = \bar{\gamma}^A_{B^c}$ with $\|\bar{\gamma}^A\|_2 = 1$. Then we see that the split functions are given as

$$\begin{aligned} f_B &= \sum_{A\in\Pi^k} \mathbf{1}_{A\cap B}(c_A + \varepsilon\bar{\gamma}^A) + \mathbf{1}_{A\cap B^c}(c_A - \varepsilon\bar{\gamma}^A) \\ &= f^k_\Pi + \varepsilon \sum_{A\in\Pi^k} (\mathbf{1}_{A\cap B} - \mathbf{1}_{A\cap B^c})\bar{\gamma}^A \\ &= f^k_\Pi + \varepsilon\bar{\boldsymbol{\gamma}}_{\boldsymbol{B}} \end{aligned} \tag{5.15}$$

for any $\varepsilon > 0$. As in Section 4.2.1 we could subtract $D(f^k_\Pi)$ from (5.14) and divide by $\varepsilon > 0$ without changing the solution. In this case doing so leads to

$$\operatorname*{argmin}_{B\in\mathcal{P}(V)} \frac{D(f^k_\Pi + \varepsilon\bar{\boldsymbol{\gamma}}_{\boldsymbol{B}}, g) - D(f^k_\Pi, g)}{\varepsilon} + \frac{\alpha}{\varepsilon} \sum_{(u,v)\in S^c} w(u,v)|\mathbf{1}_B(v) - \mathbf{1}_B(u)|. \tag{5.16}$$

As we see here we cannot pull the ε out of the regularizer $R_{i,0}(f + \varepsilon\bar{\boldsymbol{\gamma}}_{\boldsymbol{B}})$, since it is a positive constant, and thus, the L_0-norm of it is equal 1. The regularizer would tend to infinity for ε to zero and cannot deduce the directional derivative of the data-term D as we did in Section 4.2.1. Consequently, the optimization does not only depend on the choice of B but also on the choice of $\varepsilon > 0$. To tackle this problem we optimize over $\varepsilon > 0$ and B. The general partition problem (5.14) than has to optimize over the values $d_{A\cap B}, d_{A\cap B^c} \in \mathbb{R}^d$ for each segment $A \in \Pi$ and over $B \in \mathcal{P}(V)$. This optimization problem is then given with $d_B = (d_{A\cap B}, d_{A\cap B^c})_{A\in\Pi^k} \in \mathbb{R}^{|\Pi^k|\times d}$ as

$$\begin{aligned} \operatorname*{argmin}_{d_B\in\mathbb{R}^{|\Pi^k|\times d}, B\in\mathcal{P}(V)} \quad & D\Big(\sum_{A\in\Pi^k} \mathbf{1}_{A\cap B} d_{A\cap B} + \mathbf{1}_{A\cap B^c} d_{A\cap B^c}, g\Big) \\ & + \alpha \sum_{(u,v)\in S^c} \sqrt{w(u,v)}\ |\mathbf{1}_B(v) - \mathbf{1}_B(u)|. \end{aligned} \tag{5.17}$$

This problem is much more complex cannot be solved via a single graph cut, since it is no binary partition problem anymore. Thus, we propose the most straightforward way to solve this problem by alternating between fixing B and solving for $d_{A\cap B}, d_{A\cap B^c}$ for all $A \in \Pi^k$ and fixing these values and solving for $B \in \mathcal{P}(V)$. This alternating optimization is given by the

following algorithm

$$\begin{cases} B^{n+1} = \underset{B\in\mathcal{P}(V)}{\operatorname{argmin}} & D(\sum_{A\in\Pi^k} \mathbf{1}_{A\cap B} d^n_{A\cap B^n} + \mathbf{1}_{A\cap B^c} d^n_{A\cap (B^n)^c}, g) \\ & \qquad +\alpha \sum_{(u,v)\in S^c} \sqrt{w(u,v)}\, |\mathbf{1}_B(v) - \mathbf{1}_B(u)| \\ d^{n+1} = \underset{d_{B^{n+1}}}{\operatorname{argmin}} & D(\sum_{A\in\Pi^k} \mathbf{1}_{A\cap B^{n+1}} d_{A\cap B^{n+1}} + \mathbf{1}_{A\cap (B^{n+1})^c} d_{A\cap (B^{n+1})^c}, g). \end{cases} \tag{5.18}$$

Since we assume that D is a continuously differentiable data-term the minimization of solely the data-term yields a solution by applying gradient descent strategies. The question is if the minimization over B has a solution. To compute this with a graph cut the partition problem has to be submodular as stated in [46]. The regularizer is submodular, and thus, we have to assume that the data-term is also submodular. In [46] the authors restrict the data-term to be separable over the nodes, since then this problem is always submodular. An example for such a data-term is the (weighted) L_2-data-term. In fact, it is not necessary for the data-term to be submodular using a linearization of the data-term D. Since D is continuously differentiable we can compute the linearization as

$$D(f_{\Pi^k} + \varepsilon\bar{\boldsymbol{\gamma}}_{\boldsymbol{B}}, g) - D(f_{\Pi^k}, g) \approx \langle \nabla D(f_{\Pi^k}, g), \varepsilon\bar{\boldsymbol{\gamma}}_{\boldsymbol{B}}\rangle.$$

Then we can replace the data-term in (5.16) by this linearization and get the following partition problem

$$\underset{\varepsilon>0, B\in\mathcal{P}(V)}{\operatorname{argmin}} \langle \nabla D(f_{\Pi^k}, g), \bar{\boldsymbol{\gamma}}_{\boldsymbol{B}}\rangle + \frac{\alpha}{\varepsilon} \sum_{(u,v)\in S^c} w(u,v)|\mathbf{1}_B(v) - \mathbf{1}_B(u)|. \tag{5.19}$$

This problem then is separable over the vertices which follows submodularity, and hence, we can solve this problem with a graph cut. This opens up the application of this algorithm to a whole family of data-terms, e.g., the Kullback-Leibler data-term Definition 2.9 that is not separable, but continuously differentiable. We can state a similar alternating algorithm as in (5.18) for non-separable continuously differentiable data-terms using the linearization.

Using the alternating algorithm (5.18) is, due to the multiple applications of the graph cut to find a certain B, more costly then solving the problem via a single graph cut. Nevertheless, we already face this problem in (4.54) to solve the partition problem for the isotropic case where we have to optimize over directions and the cut. Note that decreasing the energy the most by a cut does not imply that the solution in the end is the best possible solution due to the high non-convexity. We see this also in the numerics in Chapter 7 where choosing a direction that is more conservative, but fits the given data, yields better results than minimizing the partition problem as much as possible in every alternating update.

As we have derived a method to solve the partition problem, let us derive the reduced problem for (5.6). Let us assume in the following that we have computed B^{k+1} as the solution of (5.17). Then we can state the new partition Π^{k+1} as the connected components of $A\cap B^{k+1}$ and $A\cap (B^{k+1})^c$ for every $A\in\Pi^k$ as we did in (4.55). This is reasonable since it enables the reduced problem to find a better solution on the finer partition. The reduced regularization term is given in the following statement.

Proposition 5.5 (Reduced L_0-TV Regularizer)**.**
Let $G = (V, E, w)$ be a finite weighted graph. Then the reduced L_0-TV regularizer R_0^r for $c \in \mathcal{H}(\Pi)$ with $f_\Pi = P_\Pi c$ is given as

$$R_0^r(c) = R_0(f_\Pi) = \sum_{(A_i,A_j)\in E_r} w_r(A_i, A_j) \tag{5.20}$$

with the reduced graph defined as

$$\begin{aligned} G_r &= (\Pi, E_r, w_r), \\ E_r &= \{(A_i, A_j) \in E \mid (A_i \times A_j) \cap E \neq \emptyset, A_i, A_j \in \Pi\}, \\ w_r(A_i, A_j) &= \sum_{(u,v)\in E_{A_1 A_2}} \sqrt{w(u, v)}, \quad \forall (A_i, A_j) \in E_r. \end{aligned} \tag{5.21}$$

Proof.
Before computing the regularizer let us take a closer look at piecewise constant function $f_c = \sum_{A\in\Pi} \mathbf{1}_A c_A$. We can deduce two cases from this for the regularizer. First let $u, v \in A$ for some $A \in \Pi$. Then $\|f_c(v) - f_c(u)\|_0 = \|c_A - c_A\|_0 = 0$. The second case is if $u \in A_i$ and $v \in A_j$ with $(u, v) \in E$. Then $\|f_c(v) - f_c(u)\|_0 = \|c_{A_j} - c_{A_i}\|_0 = 1$ since $c_{A_j} \neq c_{A_i}$. Then we can deduce that the regularizer can be computed as

$$\begin{aligned} R_0(f_c) &= \sum_{(u,v)\in E} \sqrt{w(u, v)}\ \|f_c(v) - f_c(u)\|_0 \\ &= \sum_{(A_i,A_j)\in E_r} w_r(A_i, A_j)\|c_{A_j} - c_{A_i}\|_0 \\ &= \sum_{(A_i,A_j)\in E_r} w_r(A_i, A_j) = R_0^r(c) \end{aligned} \tag{5.22}$$

□

Then the reduced formulation of the minimal partition problem (5.6) on the reduced graph G_r^{k+1} is given as

$$\operatorname*{argmin}_{c\in\mathbb{R}^{|\Pi^{k+1}|\times d}} D(P_{\Pi^{k+1}} c, g) + \frac{\alpha}{2} \sum_{(A_i,A_j)\in E_r} w_r(A_i, A_j). \tag{5.23}$$

Solving this reduced non-convex problem is still a hard task, but on this low dimensional level it is much more efficient. As we have pointed out in Section 4.3 we can use any method that can solve the problem on the full graph G also directly on the reduced graph G_r using the reformulated problem (5.23). In this work we compare different methods to solve the reduced problem. With these considerations we can state a L_0-Cut-Pursuit algorithm in Algorithm 4 for the (isotropic) minimal partition problem (5.6) that is related to the Cut-Pursuit algorithm in Section 4.3.

The minimal partition problem is highly non-convex which implies that there exists an unknown amount of local minima and that the convergence to a global optimum is not clear. It is not possible to prove the convergence to a global optimum of any of the proposed

Algorithm 4: Generic Isotropic L_0-Cut-Pursuit Algorithm.

Input:
$G = (V, E, w) \leftarrow$ Undirected finite weighted graph
$g \leftarrow$ Some given d-dimensional data in $\mathcal{H}(V; \mathbb{R}^d)$
$n \leftarrow$ Number of inner iterations
Initialization:
$\Pi^0 \leftarrow \{V\}$
$c^0 \leftarrow \operatorname{argmin}_{c \in \mathbb{R}^d} \; D(P^0 c)$
Method:
while $\Pi^{k+1} \neq \Pi^k$ **do**
  $B^{k+1} \leftarrow$ Apply the alternating algorithm (5.18) for n iterations. Solve partition problem with graph cuts (cf. [10], Section 3.4).
  $\Pi^{k+1} \leftarrow \{\text{connComp}(A \cap B), \text{connComp}(A \cap B) \mid A \in \Pi^k\}$.
  $P^{k+1} \leftarrow (\mathbf{1}_A)_{A \in \Pi}$
  $G_r^{k+1} = (V_r, E_r, w_r) \leftarrow$ with $V_r = \Pi$, E_r, w_r as in (5.21).
  $c^{k+1} = \operatorname{argmin}_{c \in \mathcal{H}(\Pi)} \; J(P^{k+1} c, g) \leftarrow$ reduced problem as in (5.23).
Result: The local optimum f^* for (5.6) and the corresponding partition Π^* representing the piecewise constant parts of f^*.

algorithm in [46], [83] or Algorithm 4. In [46, Appendix E, E.1.3] the authors prove that their algorithm converges to a local optimum. The same results hold for this algorithm, if we can find a B as a minimizer of the partition problem and also find a minimizer for the reduced problem. We see in the following that not every proposed optimization strategy can guarantee the convergence to a local minimizer. Nevertheless, we will see in the numerics in Chapter 7 that the algorithm decreases the energy efficiently and yields visually desired results. Note that the only quality measure we have at hand in this case is the energy value provided by a solution.

In the following sections we propose different methods to solve the partition problem in (5.17) and how to solve the reduced problem (5.23). Afterwards we apply considerations of the special case with an L_2-data-term. In the numerics section Chapter 7 we will show the results of the different methods.

5.1.1 Solving the Partition Problems

In the former section we have introduced a partition problem in (5.17) to split a given partition Π into a finer set by finding a function

$$f_B = \sum_{A \in \Pi} \mathbf{1}_{A \cap B} d_{A \cap B} + \mathbf{1}_{A \cap B^c} d_{A \cap B^c}$$

where $d_{A \cap B}, d_{A \cap B^c} \in \mathbb{R}^d$ for $A \in \Pi$. We discussed that to solve the partition (5.17) we have to optimize over a cut B and the corresponding values $d_{A \cap B}, d_{A \cap B^c}$ what can be done applying an alternating algorithm as (5.18). We have already talked about that we can also approximate the values $d_{A \cap B}$ and $d_{A \cap B^c}$ by rewriting them as in (5.15) by a given direction γ_A and a step size ε. Then the partition problem (5.17) is optimizing over ε and B. We can also give every segment $A \in \Pi$ its own step size ε_A such that the optimization in (5.17)

is over B and each of the ε_A. In the following paragraphs we want to state the alternating algorithms explicitly for clarification as we apply them in the numerics in Chapter 7 and investigate their different behaviors.

1. Optimizing the Values:
The first and most straightforward way is to solve the partition problem (5.17) is the already introduced alternating algorithm (5.18). Optimization over the data-term can be done analytically or via first-order methods due to the continuous differentiability. The optimization over the data-term is solved on $2|\Pi|$ variables. We again state this algorithm here

$$\begin{cases} B^{n+1} = \underset{B\in\mathcal{P}(V)}{\operatorname{argmin}} & D(\sum_{A\in\Pi^k} \mathbf{1}_{A\cap B} d^n_{A\cap B^n} + \mathbf{1}_{A\cap B^c} d^n_{A\cap (B^n)^c}, g) \\ & \qquad +\alpha \sum_{(u,v)\in S^c} \sqrt{w(u,v)}\ |\mathbf{1}_B(v) - \mathbf{1}_B(u)| \\ d^{n+1} = \underset{d_{B^{n+1}}}{\operatorname{argmin}} & D(\sum_{A\in\Pi^k} \mathbf{1}_{A\cap B^{n+1}} d_{A\cap B^{n+1}} + \mathbf{1}_{A\cap (B^{n+1})^c} d_{A\cap (B^{n+1})^c}, g). \end{cases} \tag{5.24}$$

If we apply this algorithm we will always find a new partition as long as a cut can decrease the energy. Hence, with this strategy to solve the minimal partition problem we can guarantee the convergence as long as we find a minimizer for the reduced problem.

2. Optimizing Step Size:
The second and third strategies both originate from the ideas of approximating the values on the new segments by

$$d_{A\cap B} = c_A + \varepsilon\bar{\gamma}^A, d_{A\cap B^c} = c_A - \varepsilon\bar{\gamma}^A$$

for every segment $A \in \Pi$. We only focus on one direction per segment $A \in \Pi$ instead of two for practical reasons. The directions $\bar{\gamma}^A$ are computed or selected in some manner for each partition and have length 1, i.e., $\|\bar{\gamma}^A\|_2 = 1$. This algorithmic strategy aims to find a scalar $\varepsilon > 0$ that optimizes the data-term when plugging in

$$f_B = f + \varepsilon \sum_{A\in\Pi} (\mathbf{1}_{A\cap B} - \mathbf{1}_{A\cap B^c})\bar{\gamma}^A = f + \varepsilon\bar{\boldsymbol{\gamma}}_{\boldsymbol{B}}.$$

Hence, the algorithm from (5.24) for this optimization strategy becomes

$$\begin{cases} B^{n+1} & = \underset{B\in\mathcal{P}(V)}{\operatorname{argmin}}\ D(f + \varepsilon^n\bar{\boldsymbol{\gamma}}_{\boldsymbol{B}}, g) + \alpha \sum_{(u,v)\in S^c} \sqrt{w(u,v)}\ |\mathbf{1}_B(v) - \mathbf{1}_B(u)| \\ \varepsilon^{n+1} & = \underset{\varepsilon>0}{\operatorname{argmin}}\ D(f + \varepsilon\bar{\boldsymbol{\gamma}}_{\boldsymbol{B^{n+1}}}, g). \end{cases} \tag{5.25}$$

The update of $\varepsilon > 0$ can be solved efficiently since it only has to be solved for *one* variable.

Note, that it is also possible to not optimize over $\varepsilon > 0$ at all and just select some appropriate value. Then one also only has to compute the graph cut once. We show in the numerics in Chapter 7 that it is possible to argue that it is not necessary in practice to optimize this to achieve desirable results.

For this strategy - and also for the next - we cannot guarantee the convergence since it could provide a trivial cut, i.e., $B = \emptyset$, $B = V$, even though the energy is not minimized. This is the same problem as in Section 4.2.3 where we show that one has to show for all directions that the cut is trivial and not only for the chosen direction. This would imply to also optimize over the directions which would be in fact just strategy 1. Hence, we just use this method since it is more efficient to use this strategy than the strategy in 1. Also we see in Chapter 7 that this strategy and the next strategy still can decrease the energy and yield visually good results.

3. Updating the Step Sizes for each Segment:
The last method is to not only use *one* $\varepsilon > 0$, but find a scalar $\varepsilon_A > 0$ for every segment $A \in \Pi$. Then we optimize over the following family of functions

$$f_B = f + \sum_{A \in \Pi} (\mathbf{1}_{A \cap B} - \mathbf{1}_{A \cap B^c}) \varepsilon_A \bar{\gamma}^A)$$

where we again have to pre-compute the directions $\bar{\gamma}^A$, but now have to update ε_A for every segment $A \in \Pi$. Then the alternating algorithm to solve the partition problem becomes

$$\begin{cases} B^{n+1} & = \underset{B \in \mathcal{P}(V)}{\operatorname{argmin}} \; D(f + \sum\limits_{A \in \Pi} (\mathbf{1}_{A \cap B} - \mathbf{1}_{A \cap B^c}) \varepsilon_A^n \bar{\gamma}^A, g) \\ & \qquad\qquad + \alpha \sum\limits_{(u,v) \in S^c} \sqrt{w(u,v)} \; |\mathbf{1}_B(v) - \mathbf{1}_B(u)| \\ (\varepsilon_A^n)_{A \in \Pi} & = \underset{\substack{\varepsilon_A > 0 \\ A \in \Pi}}{\operatorname{argmin}} \; D(f + \sum\limits_{A \in \Pi} (\mathbf{1}_{A \cap B} - \mathbf{1}_{A \cap B^c}) \varepsilon_A \bar{\gamma}^A, g). \end{cases} \tag{5.26}$$

This algorithm is more complex than optimizing only over one ε in (5.25), but might yield better solutions due to the higher flexibility.

Note that we can use the algorithms (5.25) and (5.26) to optimize the partition problem of the linearized data-term in (5.21).

As we can see in all of these optimization strategies the problems do *not* incorporate any edges between the segments, i.e., edges in $S(f_{\Pi^k})$. Hence, the splitting is completely decoupled from the other sets and the optimization can be done for every segment individually. This also sets up the possibility for efficient parallelized implementation strategies, since the graph cut algorithms cannot be parallelized efficiently without loss. We postpone this implementation problem to the future.

We show the influence of the proposed methods on the quality of the solutions in Chapter 7. In the following section we take a closer look at how to find solutions for the reduced problem (5.23).

5.1.2 Solving the Reduced Minimal Partition Problem

In this section, we want to find a solution to the reduced minimal partition problem given in (5.23). We present three different strategies in this section. The first one is to compute the solution of the reduced problem by applying the primal-dual method from [83]. The second

method is a merging strategy that merges the segments of the partitions as long as the energy decreases. The third one is to assume that on the reduced problem the regularizer does not have so much influence on the solution and can be dropped for minimization.

1. Solve with Fast MS:
In the first setting we assume that it might happen that the solution c^* of (5.23) has the same value in neighboring segments, i.e., $c^*_{A_i} = c^*_{A_j}$ for some $(A_i, A_j) \in E_r$. Then we have to solve the problem for which we use the fast Mumford-Shah minimization from [83] where they use the Moreau's identity

$$\operatorname{prox}_{\sigma,R^*}(y) = y - \sigma \operatorname{prox}_{\frac{1}{\sigma},R}(y/\sigma)$$

to state the proximity operator as

$$\operatorname{prox}_{\sigma,R^*_{i,0}}(y) = \begin{cases} y, & \text{if } |y| \leq \sqrt{2\alpha\sigma} \\ 0, & \text{else.} \end{cases} \tag{5.27}$$

We can exchange the dual proximity operator in (3.20) with this one to get a primal-dual algorithm to solve (5.23). We only consider the special case with an L_2-data-term here for which the primal-dual algorithm on graphs to solve the reduced minimal partition problem is given as follows.

Algorithm 5.6 (Fast Mumford-Shah Primal-Dual Algorithm on Graphs)**.**
Let the assumptions be the same as in Algorithm 3.22. Then the primal-dual algorithm to solve (5.17) *with an L_2-data-term is given as*

$$\begin{cases} \tilde{y}^{n+1} = y^n + \sigma \nabla_w \bar{f}^n \\ y^{n+1}_{u,v} = \begin{cases} \tilde{y}^{n+1}_{u,v}, & \text{if } |\tilde{y}^{n+1}_{u,v}| \leq \sqrt{2\alpha\sigma}, \\ 0, & \text{else,} \end{cases} \qquad \forall (u,v) \in E \\ f^{n+1} = (I+\tau)^{-1}(f^n + \tau(\operatorname{div}_w(y^n) + g)) \\ \bar{f}^{n+1} = f^{n+1} + \theta(f^{n+1} - f^n). \end{cases} \tag{5.28}$$

We denote this algorithm as the *Fast Mumford-Shah (MS)* algorithm in this work. With this algorithm it is possible to solve (5.17) on the full graph G and problem (5.23) on the reduced graph G_r. In [83] they could not give any convergence results, but show experimentally that it decreases the energy and yields visually good results. Hence, when using this algorithm to compute a solution for the reduced problem we can also not guarantee any convergence to a local optimum. In the numerics in Chapter 7 we show comparisons of using Fast MS on the full graph G and our proposed method. Additionally, we show the results when using this primal-dual algorithm as a solver for the reduced problem. Also note that we apply a simple merge step after the computation of the reduced solution where we merge all the segments that are connected and have the same values. This does not change the energy at all, but decreases the number of segments and enables the partition problem to form a better splitting.

2. Merging Strategy:
The key idea is that we want to merge two given sets $A_i, A_j \in \Pi$ if the energy of the reduced problem (5.23) is smaller for the union of these sets $A_i \cup A_j$. For the merging strategy we have to assume that the data-term is *pointwise separable* which implies that $D_A = \sum_{u \in A} D_u$ for $A \in \Pi$. Thus, for each edge $(A_i, A_j) \in E_r$ the reduced graph G_r we have to check if

$$D_{A_i}(c_{A_i}, g) + D_{A_j}(c_{A_j}, g) + \alpha w_r(A_i, A_j) \geq D_{A_i \cup A_j}(c_{A_i \cup A_j}, g) \tag{5.29}$$

where $c_{A_i \cup A_j}$ is the solution on the union set of A_i and A_j. If it holds the two sets are merged. When D is pointwise separable this can be computed easily by solving

$$c_{A_i \cup A_j} = \operatorname*{argmin}_c D_{A_i}(c, g) + D_{A_j}(c, g). \tag{5.30}$$

If the inequality holds we merge A_i and A_j and remove them from the partition Π and add $A_i \cup A_j$, i.e.

$$\Pi = \Pi \setminus \{A_i, A_j\} \cup \{A_i \cup A_j\}. \tag{5.31}$$

Additionally, we have to reconnect the reduced graph G_r since $A_i \cup A_j$ is now a single node and is connected to every segment A_i and A_j were connected to, i.e.

$$\begin{aligned} E_r = &\big\{(B, C) \in E_r \mid B, C \notin \{A_i, A_j\}\big\} \\ &\cup \big\{(B, A_i \cup A_j), (A_i \cup A_j, B) \mid (B, A_i) \in E_r \vee (B, A_j) \in E_r\big\}. \end{aligned} \tag{5.32}$$

The reduced weight is then just computed by setting

$$w_r(B, A_i \cup A_j) = w_r(B, A_i) + w_r(B, A_j). \tag{5.33}$$

After merging two segments into one segment we also have to check the condition in (5.29) for the newly produced edges and merge again if it makes sense in terms of the energy. In practice it is favorable to compute the values $c_{A_i \cup A_j}$ for every segment pair in the reduced edge set E_r and the *merge value* we define for each connected segments $(A_i, A_j) \in E_r$ as

$$M_{A_i, A_j} = D_{A_i}(c_{A_i}, g) + D_{A_j}(c_{A_j}, g) + \alpha w_r(A_i, A_j) - D_{A_i \cup A_j}(c_{A_i \cup A_j}, g) > 0. \tag{5.34}$$

The higher the merge value M_{A_i, A_j} is, the more the energy decreases when we merge these segments. Consequently, in practice we compute this value for every edge $(A_i, A_j) \in E_r$ and sort them from the highest to the lowest in an array. Then we would start by merging the segments with the highest merge value, e.g. merging A_i, A_j to the segment $A_i \cup A_j$ and updating Π and G_r. After merging we have to compute the merge values of the new edges in E_r and sort them into the array of sorted merge values. Then we again select the segment pair with the highest value, merge them, compute the new merge values and sort them into the array. This can be done until there is nothing left to merge, a certain improvement threshold is reached or a certain number of merges have been done. If we apply the merging until there are no segments with a merge value greater than 0 we actual reached a local optimum of the

reduced problem on the current reduced graph. We summarize this method in Algorithm 5.

Algorithm 5: Merging.

Input:
$\Pi \leftarrow$ some given partition of a vertex set V
$G_r = (\Pi, E_r, w_r) \leftarrow$ Reduced finite weighted graph for partition Π
$g \leftarrow$ Some given d-dimensional data in $\mathcal{H}(V; \mathbb{R}^d)$
$\varepsilon_{\text{thresh}} \leftarrow$ threshold for minimal merge value
$maxNumMerges \leftarrow$ maximum set numbers of merges
Initialization:
$c_{A,B} \leftarrow$ compute the solution for the merged segments for every $(A, B) \in E_r$ as in (5.30)
$M_{A,B} \leftarrow$ compute the merge value for every $(A, B) \in E_r$ as in (5.34) with $c_{A,B}$
$v \leftarrow$ sort the values of $M_{A,B}$ for every edge $(A, B) \in E_r$ from high to low and store them with their edge (A, B)
Method:
while $i < maxNumMerges \wedge \max(v) \geq \varepsilon_{thresh}$ **do**
  Merge the first segments pair (A^*, B^*) in v
  $\Pi \leftarrow$ update as in (5.31) for segment pair (A^*, B^*)
  $G_r \leftarrow$ update the edge set as in (5.32) and the weights as in (5.33) for (A^*, B^*)
  $M_{A^*,C}, M_{B^*,C} \leftarrow$ compute the merge values for the new edges in E_r as in (5.30)
  $v \leftarrow$ update v by removing the merge values of removed edges and sorting the new merge values into the array
  $i \leftarrow i + 1$
Result: $(\Pi, G_r, c) \leftarrow$ the merged partition Π and reduced graph G_r and reduced solution c.

3. Data-Term Minimization:

Let us assume that in practice the partition problem only induces new boundaries such that the energy decreases so that we can assume that there exists no edge $(A_i, A_j) \in E_r$ such that $c_{A_i} = c_{A_j}$. Then it follows that the regularizer is given as

$$R_{i,0} = \sum_{(A_i, A_j) \in E_r} w_r(A_i, A_j) \tag{5.35}$$

which does not depend on c, and thus, can be dropped for minimization. Then we would only minimize the reduced data-term

$$\underset{c \in \mathbb{R}^{|\Pi| \times d}}{\operatorname{argmin}} \; D(P_\Pi c, g) \tag{5.36}$$

which is in most applications rather easy to solve. However, if we assume that this solution is not optimal on the reduced graph we can decrease the energy by applying the merging strategy from before in Algorithm 5.

Before closing this section let us state the proposed L_0-Cut-Pursuit algorithm to minimize the partition problem (5.17) in Algorithm 6 which is a more specific version of Algorithm 4.

Algorithm 6: Proposed Isotropic L_0-Cut-Pursuit Algorithm.

Input:
$G = (V, E, w) \leftarrow$ Undirected finite weighted graph
$g \leftarrow$ Some given d-dimensional data in $\mathcal{H}(V; \mathbb{R}^d)$
$n \leftarrow$ Number of inner iterations
$useMerging \leftarrow$ boolean to activate a merging step
$resultMerging \leftarrow$ boolean to apply merging onto the result
Initialization:
$\Pi^0 \leftarrow \{V\}$
$c^0 \leftarrow \operatorname{argmin}_{c \in \mathbb{R}^d} \; D(P^0 c)$
Method:
while $\Pi^{k+1} \neq \Pi^k$ **do**
  $B^{k+1} \leftarrow$ Apply one of the alternating algorithms in (5.24), (5.25) or (5.26) for n iterations. Solve partition problem with graph cuts (cf. [10], Section 3.4).
  $\Pi^{k+1} \leftarrow \{\text{connComp}(A \cap B), \text{connComp}(A \cap B) \mid A \in \Pi^k\}$.
  $P^{k+1} \leftarrow (\mathbf{1}_A)_{A \in \Pi}$
  $G_r^{k+1} = (V_r, E_r, w_r) \leftarrow$ with $V_r = \Pi$, E_r, w_r as in (5.21).
  $c^{k+1} \leftarrow$ solve reduced problem as by applying the primal-dual proposed in (5.28) or by minimizing the data-term as in (5.36)
  if $useMerging =$ ***true*** **then**
    $(\Pi^{k+1}, G_r^{k+1}, c^{k+1}) \leftarrow$ apply merging from Algorithm 5
  $f^{k+1} \leftarrow P^{k+1} c^{k+1}$
if $resultMerging =$ ***true*** **then**
  $(\Pi^*, G_r^*, f^*) \leftarrow$ apply merging from Algorithm 5 onto Π^*, G_r^* and f^*
Result: A solution f^* for (5.6) and the corresponding partition Π^* representing the piecewise constant parts of f^*.

As we see in the numerics in Chapter 7 for the different proposed optimization strategies the algorithm stops after decreasing the energy. We show that in terms of energy and visual results the solutions of these optimization strategies are comparable to [46] and [83].

In the following section we apply this algorithm directly onto an L_2-data-term.

5.2 Cut-Pursuit for L_0-ROF

In this section, we want to apply the formerly proposed minimization strategies to the L_0-ROF problem which is given with an L_2-data-term and an L_0-TV regularization as in (5.7) on a given graph G and for some given data $g \in \mathbb{R}^{N \times d}$. This problem is formally given as

$$\operatorname*{argmin}_{f \in \mathcal{H}(V;\mathbb{R}^d)} \; \frac{1}{2}\|f - g\|_2^2 + \frac{\alpha}{2} \sum_{(u,v) \in E} \sqrt{w(u,v)} \|f(v) - f(u)\|_0. \tag{5.37}$$

For this we want to derive the algorithms for the partition problem in (5.24), (5.25) and (5.26). Afterwards we state how to compute the reduced problems as we do in Section 5.1.2. We use the same structure as in the former section and specify the results for the given optimization problem.

5.2.1 Solving the Partition Problems

We want to derive the algorithms in (5.24), (5.25) and (5.26) for problem (5.37). To do so, we start with the most generic one and derive the others as special cases.

1. Optimizing the Values:
We again assume that we have given some partition Π and corresponding reduced graph G_r for a given graph G. To derive the partition problem in (5.24) we fix some values $d_{A\cap B}, d_{A\cap B^c} \in \mathbb{R}^d$ for every segment $A \in \Pi$ and investigate the data-term D for

$$f_B = \sum_{A\in\Pi} \mathbf{1}_{A\cap B} d_{A\cap B} + \mathbf{1}_{A\cap B^c} d_{A\cap B^c} = \sum_{A\in\Pi} \mathbf{1}_{A\cap B}(d_{A\cap B} - d_{A\cap B^c}) + \mathbf{1}_A d_{A\cap B^c}. \tag{5.38}$$

Plugging this function into the data-term yields for every $u \in A$ and $A \in \Pi$

$$\begin{aligned} D_u(f_B(u), g) &= \frac{1}{2}(\mathbf{1}_{A\cap B}(u)(d_{A\cap B} - d_{A\cap B^c}) + d_{A\cap B^c} - g(u))^2 \\ &= \frac{1}{2}\mathbf{1}_{A\cap B}(u)(d_{A\cap B} - d_{A\cap B^c})^2 \\ &\quad + \mathbf{1}_{A\cap B}(u)\langle d_{A\cap B} - d_{A\cap B^c}, d_{A\cap B^c} - g(u)\rangle \\ &\quad + \frac{1}{2}(d_{A\cap B^c} - g(u))^2. \end{aligned} \tag{5.39}$$

For minimization purposes we can drop the third term since it is not dependent on B and can reformulate the data-term for $u \in A$ to

$$\begin{aligned} &\tilde{D}_u(f_B(u), g) \\ &\quad = \langle \tfrac{1}{2}(d^2_{A\cap B} - d^2_{A\cap B^c}) - (d_{A\cap B} - d_{A\cap B^c})g(u), d_{A\cap B} - d_{A\cap B^c}\rangle \mathbf{1}_{A\cap B}(u). \end{aligned} \tag{5.40}$$

Thus, we can replace the data-term in (5.24) with this data-term and compute a new partition via a graph cut. The binary split B computed by the cut is then used to compute the values for $d_{A\cap B}, d_{A\cap B^c} \in \mathbb{R}^d$ for every segment $A \in \Pi$. In this case we can compute them analytically without any iterative method. Let us derive the value $d_{A\cap B}$ for a fixed B and any segment $A \in \Pi$ by minimizing the data-term for $d_{A\cap B}$. For that reason, we have to derive the optimality condition and solve for $d_{A\cap B}$.

$$\begin{aligned} \frac{\partial}{\partial d_{A\cap B}} D(f_B) &= \frac{\partial}{\partial d_{A\cap B}} \frac{1}{2} \sum_{u\in A} (\mathbf{1}_{A\cap B}(u) d_{A\cap B} + \mathbf{1}_{A\cap B^c}(u) d_{A\cap B^c} - g(u))^2 \\ &= \sum_{u\in A} \mathbf{1}_{A\cap B}(u)\ (\mathbf{1}_{A\cap B}(u) d_{A\cap B} + \mathbf{1}_{A\cap B^c}(u) d_{A\cap B^c} - g(u)) \\ &= \sum_{u\in A\cap B} d_{A\cap B} - g(u) \\ &= d_{A\cap B}|A\cap B| - \sum_{u\in A\cap B} g(u) = 0 \\ \Leftrightarrow \quad d_{A\cap B} &= \frac{\sum_{u\in A\cap B} g(u)}{|A\cap B|}. \end{aligned} \tag{5.41}$$

Since this computation can be done analogously for $d_{A\cap B^c}$ we can directly deduce that the update is given as

$$d_{A\cap B} = \frac{\sum_{u\in A\cap B} g(u)}{|A\cap B|}, \quad d_{A\cap B^c} = \frac{\sum_{u\in A\cap B^c} g(u)}{|A\cap B^c|}. \tag{5.42}$$

In fact, we have to compute the mean value of g over the newly introduced segments $A\cap B$ and $A\cap B^c$. The alternating algorithm to find an optimal cut corresponding to (5.24) is then given as

$$\begin{cases} B^{n+1} & = \underset{B\in\mathcal{P}(V)}{\operatorname{argmin}} \sum_{A\in\Pi}\sum_{u\in A} \langle \frac{1}{2}(d^2_{A\cap B} - d^2_{A\cap B^c}) \\ & \qquad -(d_{A\cap B} - d_{A\cap B^c})g(u), d_{A\cap B} - d_{A\cap B^c}\rangle \mathbf{1}_{A\cap B}(u) \\ & \qquad +\alpha \sum_{(u,v)\in S^c} \sqrt{w(u,v)}\, |\mathbf{1}_B(v) - \mathbf{1}_B(u)| \\ d^{n+1}_{A\cap B} & = \frac{\sum_{u\in A\cap B} g(u)}{|A\cap B|}, \quad d^{n+1}_{A\cap B^c} = \frac{\sum_{u\in A\cap B^c} g(u)}{|A\cap B^c|}, \qquad \forall A\in\Pi. \end{cases} \tag{5.43}$$

Due, to the chosen L_2-data-term the only computational cost of this alternating algorithm is to compute the graph cut to compute the B multiple times.

2. Updating the Step Sizes for each Segment:

Let us deduce the algorithms for searching the step size $\varepsilon_A > 0$ in the directions $\bar{\gamma}^A$ for every segment $A\in\Pi$. As we state in the former section we set $d_{A\cap B} = c_A + \varepsilon_A\bar{\gamma}^A$ and $d_{A\cap B^c} = c_A - \varepsilon_A\bar{\gamma}^A$ and the corresponding family of functions depending on B and ε is given as

$$f_B = f_\Pi + \sum_{A\in\Pi} (\mathbf{1}_{A\cap B} - \mathbf{1}_{A\cap B^c})\varepsilon_A\bar{\gamma}^A.$$

We deduce the partition problem by computing the following identities

$$\begin{aligned} d_{A\cap B} - d_{A\cap B^c} &= 2\varepsilon_A\bar{\gamma}^A \\ d^2_{A\cap B} &= c_A^2 + 2\varepsilon_A\langle c_A, \bar{\gamma}^A\rangle + \varepsilon_A^2\langle\bar{\gamma}^A, \bar{\gamma}^A\rangle \\ d^2_{A\cap B^c} &= c_A^2 - 2\varepsilon_A\langle c_A, \bar{\gamma}^A\rangle + \varepsilon_A^2\langle\bar{\gamma}^A, \bar{\gamma}^A\rangle \\ \frac{1}{2}(d^2_{A\cap B} - d^2_{A\cap B^c}) &= 2\varepsilon_A\langle c_A, \bar{\gamma}^A\rangle. \end{aligned} \tag{5.44}$$

and plugging these into the splitting problem to solve B in (5.43) yields the update we state in (5.46). Again we can explicitly compute the update of ε_A for each segment $A\in\Pi$ by

computing the optimality condition of the data-term for a fixed B as follows

$$\begin{aligned}\frac{\partial}{\partial \varepsilon_A} D(f_B) &= \frac{\partial}{\partial \varepsilon_A} \frac{1}{2} \sum_{A\in\Pi} \sum_{u\in A} \left(c_A + (\mathbf{1}_{A\cap B}(u) - \mathbf{1}_{A\cap B^c}(u))\varepsilon_A \bar{\gamma}^A - g(u)\right)^2 \\ &= \sum_{u\in A\cap B} \langle \bar{\gamma}^A, c_A + \varepsilon_A \bar{\gamma}^A - g(u)\rangle \\ &\quad - \sum_{u\in A\cap B^c} \langle \bar{\gamma}^A, c_A - \varepsilon_A \bar{\gamma}^A - g(u)\rangle \\ &= |A|\varepsilon_A \langle \bar{\gamma}^A, \bar{\gamma}^A\rangle + (|A\cap B| - |A\cap B^c|)c_A \\ &\quad - \sum_{u\in A\cap B} g(u) + \sum_{u\in A\cap B^c} g(u) = 0 \\ \Leftrightarrow\ \varepsilon_A &= \frac{(|A\cap B| - |A\cap B^c|)c_A + \sum_{u\in A\cap B^c} g(u) - \sum_{u\in A\cap B} g(u)}{|A|} \\ &= \frac{(|A\cap B| - 1)\sum_{u\in A\cap B} g(u) - (|A\cap B^c| - 1)\sum_{u\in A\cap B^c} g(u)}{|A|}.\end{aligned} \tag{5.45}$$

Then we can deduce the following algorithm to alternatingly find an optimal partition with $\bar{\boldsymbol{\gamma}}^{\boldsymbol{\varepsilon}}_{\boldsymbol{B}} = \sum_{A\in\Pi} \mathbf{1}_{A\cap B}\varepsilon_A \bar{\gamma}^A$ as

$$\begin{cases} B^{n+1} &= \underset{B\in\mathcal{P}(V)}{\operatorname{argmin}} \langle f_\Pi - g, 2\bar{\boldsymbol{\gamma}}^{\boldsymbol{\varepsilon^n}}_{\boldsymbol{B}}\rangle + \alpha \sum_{(u,v)\in S^c} \sqrt{w(u,v)}\ |\mathbf{1}_B(v) - \mathbf{1}_B(u)| \\ \varepsilon_A^{n+1} &= \dfrac{(|A\cap B^{n+1}| - 1)\sum_{u\in A\cap B^{n+1}} g(u) - (|A\cap (B^{n+1})^c| - 1)\sum_{u\in A\cap (B^{n+1})^c} g(u)}{|A|} \\ \bar{\boldsymbol{\gamma}}^{\boldsymbol{\varepsilon^{n+1}}}_{\boldsymbol{B}} &= \sum_{A\in\Pi} \mathbf{1}_{A\cap B^{n+1}} \varepsilon_A^{n+1} \bar{\gamma}^A. \end{cases} \tag{5.46}$$

This algorithm requires given directions for each segment that have to be picked rationally for the given data. Computing these directions can be rather costly in some applications. Computing the step sizes on the other hand is very easy again as we can compute them analytically.

3. Optimizing the Step Size:
Finally, we deduce the algorithm for one constant $\varepsilon > 0$ that is the same for all segments. For this setting we have given

$$f_B = f_\Pi + \varepsilon \sum_{A\in\Pi} (\mathbf{1}_{A\cap B} - \mathbf{1}_{A\cap B^c})\bar{\gamma}^A$$

where we have set

$$d_{A\cap B} = c_A + \varepsilon \bar{\gamma}^A, \quad d_{A\cap B^c} = c_A - \varepsilon \bar{\gamma}^A, \quad \forall A \in \Pi.$$

Then again we can find identities

$$\begin{aligned} d_{A\cap B} - d_{A\cap B^c} &= 2\varepsilon\bar{\gamma}^A \\ \frac{1}{2}(d^2_{A\cap B} - d^2_{A\cap B^c}) &= 2\varepsilon\langle c_A, \bar{\gamma}^A\rangle. \end{aligned} \tag{5.47}$$

to deduce the partition problem by plugging it into (5.24) and additionally can compute the update for ε explicitly by the optimality condition

$$\begin{aligned} &\frac{\partial}{\partial\varepsilon} D(f_B) \\ &\quad= \frac{\partial}{\partial\varepsilon} \sum_{A\in\Pi}\sum_{u\in A} (c_A + \varepsilon(\mathbf{1}_{A\cap B}(u) - \mathbf{1}_{A\cap B^c}(u)) - g(u))^2 \\ &\quad= \sum_{A\in\Pi}\sum_{u\in A} (\mathbf{1}_{A\cap B}(u) - \mathbf{1}_{A\cap B^c}(u))\langle c_A + \varepsilon(\mathbf{1}_{A\cap B}(u) - \mathbf{1}_{A\cap B^c}(u)) - g(u), \bar{\gamma}^A\rangle \\ &\quad= \sum_{A\in\Pi}\Big[\sum_{u\in A\cap B} \langle c_A + \varepsilon - g(u), \bar{\gamma}^A\rangle - \sum_{u\in A\cap B^c} \langle c_A - \varepsilon - g(u), \bar{\gamma}^A\rangle\Big] \\ &\quad= \sum_{A\in\Pi} |A|\varepsilon\Big[\sum_{u\in A\cap B} \langle c_A - g(u), \bar{\gamma}^A\rangle - \sum_{u\in A\cap B^c} \langle c_A - g(u), \bar{\gamma}^A\rangle\Big] \\ &\quad= |V|\varepsilon \sum_{A\in\Pi}\Big[\sum_{u\in A\cap B} \langle c_A - g(u), \bar{\gamma}^A\rangle - \sum_{u\in A\cap B^c} \langle c_A - g(u), \bar{\gamma}^A\rangle\Big] = 0 \\ \Leftrightarrow\ &\varepsilon = \frac{\sum_{A\in\Pi}\big[\sum_{u\in A\cap B} \langle c_A - g(u), \bar{\gamma}^A\rangle - \sum_{u\in A\cap B^c} \langle c_A - g(u), \bar{\gamma}^A\rangle\big]}{|V|}. \end{aligned} \tag{5.48}$$

Plugging these observations into (5.43) with $\bar{\boldsymbol{\gamma}}_{\boldsymbol{B}} = \sum_{A\in\Pi} \mathbf{1}_{A\cap B}\bar{\gamma}^A$ yields the following algorithm

$$\begin{cases} B^{n+1} &= \underset{B\in\mathcal{P}(V)}{\operatorname{argmin}} \langle f_\Pi - g, 2\varepsilon^n\bar{\boldsymbol{\gamma}}_{\boldsymbol{B}}\rangle + \alpha\sum_{(u,v)\in S^c} \sqrt{w(u,v)}\, |\mathbf{1}_B(v) - \mathbf{1}_B(u)| \\ \varepsilon^{n+1} &= \dfrac{\sum_{A\in\Pi}\big[\sum_{u\in A\cap B^{n+1}} \langle c_A - g(u), \bar{\gamma}^A\rangle - \sum_{u\in A\cap (B^{n+1})^c} \langle c_A - g(u), \bar{\gamma}^A\rangle\big]}{|V|}. \end{cases} \tag{5.49}$$

As we have now computed the algorithms for the three partition strategies stated in Section 5.1.1 we head over to investigate how to compute the reduced problem with the different strategies we present in Section 5.1.2.

5.2.2 Solving the Reduced Minimal Partition Problem

In this section, we look at the three strategies to tackle the reduced problem. We structure this section analogously to Section 5.1.2.

1. Solve with Fast MS:
Let us assume that we have computed a $B \in \mathcal{P}(V)$ and the corresponding new partition Π^{k+1} for the former Π^k. Let the reduced graph be given as $G_r = (\Pi^{k+1}, E_r, w_r)$. The

reduced problem of (5.37) is stated as

$$\underset{c\in\mathbb{R}^{|\Pi^{k+1}|\times d}}{\operatorname{argmin}} \ \frac{1}{2}\|c-g_r\|_{2,W_P}+\alpha\sum_{(A,B)\in E_r}\sqrt{w_r(A,B)}\ \|c_B-c_A\|_0 \tag{5.50}$$

with $W_P = P^*P = \operatorname{diag}(|A|)_{A\in\Pi^{k+1}}$ and $g_r = (\frac{\sum_{u\in A} g(u)}{|A|})_{A\in\Pi^{k+1}}$ as in (4.67). The primal-dual approach we state in (5.28) can be applied by replacing the primal proximity operator with the proximity operator of (2.30).

Algorithm 5.7 (Fast MS Primal-Dual Algorithm on Reduced Graphs)**.**
Let the assumptions be the same as in Algorithm 3.22. Let Π be a partition and $G_r = (\Pi, E_r, w_r)$ the corresponding reduced graph to G. Let T, Σ be diagonal preconditioners as stated in Definition 3.21. Then the primal-dual algorithm to solve (5.23) with an L_2-data-term is given as

$$\begin{cases} \tilde{y}^{n+1} = y^n + \Sigma\nabla_{w_r}\bar{f}^n \\ y^{n+1}_{u,v} = \begin{cases} \tilde{y}^{n+1}_{u,v}, & \text{if } \ |\tilde{y}^{n+1}_{u,v}| \le \sqrt{2\alpha\sigma}, \\ 0, & \text{else,} \end{cases} \qquad \forall (u,v)\in E_r \\ f^{n+1} = (I + T\operatorname{diag}(|A|)_{A\in\Pi})^{-1}(f^n + T(\operatorname{div}_{w_r}(y^n) + \operatorname{diag}(|A|)_{A\in\Pi} g)) \\ \bar{f}^{n+1} = f^{n+1} + \theta(f^{n+1} - f^n). \end{cases} \tag{5.51}$$

Next let us investigate how the merging from the former section works here.

2. Merging Strategy:

As in the former section let us check out how we are able to merge certain partitions in Π^{k+1}. In (5.29) we state a general merging condition to determine if the energy does decrease by merging two segments $A_i, A_j \in \Pi^{k+1}$. For that reason, we have to compute the solution of the merged segment $A_i \cup A_j$ as

$$\begin{aligned} c_{A_i\cup A_j} = \underset{c\in\mathbb{R}^d}{\operatorname{argmin}} \ D_{A_i}(c,g) + D_{A_j}(c,g) &= \sum_{u\in A_i\cup A_j} (c-g(u))^2 \\ \Leftrightarrow \quad c_{A_i\cup A_j} &= \frac{\sum_{u\in A_i\cup A_j} g(u)}{|A_i|+|A_j|} \end{aligned} \tag{5.52}$$

and plug this into the merge condition in (5.29). Reformulating this yields in the end the merge condition stated in the following lemma.

Lemma 5.8. *Let $G = (V, E, w)$ be a finite weighted graph and $g \in \mathcal{H}(V;\mathbb{R}^d)$ be some data. Let Π be some partition and G_r the corresponding reduced graph to G. Let $\alpha > 0$ be the regularization parameter. Then the merging condition for (5.37) for a given edge $(A_i, A_j) \in E_r$ is given as*

$$\alpha w_r(A_i,A_j) \ \ge \ |A_i|(2c_{A_i}\bar{g}_{A_i} - c^2_{A_i}) + |A_j|(2c_{A_j}\bar{g}_{A_j} - c^2_{A_j}) - |A_i\cup A_j| c^2_{A_i\cup A_j}. \tag{5.53}$$

Proof. See Appendix 5.A. □

When we assume that the former values c_{A_i} and c_{A_j} on the former segments $A_i, A_j \in \Pi$ by computing the mean value we can also deduce the following even more compact formulation.

Lemma 5.9. *Let all conditions of Lemma 5.8 hold and assume additionally that we computed the former values c_{A_i}, c_{A_j} as the mean values over the data in A_i and A_j respectively. Then the merging condition from Lemma 5.8 condenses to*

$$\alpha w_r(A_i, A_j) \geq \frac{|A_i||A_j|}{|A_i| + |A_j|}(c_{A_i} - c_{A_j})^2. \tag{5.54}$$

Proof. See Appendix 5.A. □

This formulation is more compact and we do not have to compute the value of $c_{A_i \cup A_j}$ explicitly which saves these computations. We apply this condition in Algorithm 6 and use it as the merging strategy.

3. Data-Term Minimization:

If we drop the regularizer as we did in the former section and compute the solution by minimizing the reduced data-term as in (5.36) we have to solve

$$\operatorname*{argmin}_{c \in \mathbb{R}^{|\Pi| \times d}} \frac{1}{2}\|c - g_r\|^2_{2,W_P}$$

where the minimizer is given as $c = g_r$. Thus, solving the reduced problem by only considering the data-term yields

$$\begin{aligned} c_A &= \frac{\sum_{u \in A} g(u)}{|A|}, \ \forall A \in \Pi^{k+1} \\ f_{\Pi^{k+1}} &= \sum_{A \in \Pi^{k+1}} \mathbf{1}_A c_A = \sum_{A \in \Pi^{k+1}} \mathbf{1}_A \frac{\sum_{u \in A} g(u)}{|A|} \end{aligned} \tag{5.55}$$

which is the mean value of the given data over all segments in Π^{k+1}. In Algorithm 7 we state the complete algorithm to find a local minimum of (5.37).

Algorithm 7: Proposed Isotropic L_0-Cut-Pursuit Algorithm to solve (5.37).

Input:
$G = (V, E, w) \leftarrow$ Undirected finite weighted graph
$g \leftarrow$ Some given d-dimensional data in $\mathcal{H}(V; \mathbb{R}^d)$
$n \leftarrow$ Number of inner iterations
$useMerging \leftarrow$ boolean to activate a merging step
$resultMerging \leftarrow$ boolean to apply merging onto the result
Initialization:
$\Pi^0 \leftarrow \{V\}$
$c^0 \leftarrow \operatorname{argmin}_{c \in \mathbb{R}^d} \; D(P^0 c)$
Method:
while $\Pi^{k+1} \neq \Pi^k$ **do**
  $B^{k+1} \leftarrow$ Apply one of the alternating algorithms in (5.43), (5.46) or (5.49) for n iterations. Solve partition problem with graph cuts (cf. [10], Section 3.4).
  $\Pi^{k+1} \leftarrow \{\text{connComp}(A \cap B), \text{connComp}(A \cap B) \mid A \in \Pi^k\}$.
  $P^{k+1} \leftarrow (\mathbf{1}_A)_{A \in \Pi}$
  $G_r^{k+1} = (V_r, E_r, w_r) \leftarrow$ with $V_r = \Pi$, E_r, w_r as in (5.21).
  $c^{k+1} \leftarrow$ solve reduced problem as by applying the primal-dual proposed in (5.28) with the proximity operator (2.30) or using the minimum of the data-term given as in (5.55)
  if $useMerging =$ ***true*** **then**
    $(\Pi^{k+1}, G_r^{k+1}, c^{k+1}) \leftarrow$ apply merging from Algorithm 5 with merge condition (5.54)
  $f^{k+1} \leftarrow P^{k+1} c^{k+1}$
if $resultMerging =$ ***true*** **then**
  $(\Pi^*, G_r^*, f^*) \leftarrow$ apply merging from Algorithm 5 onto Π^*, G_r^* and f^*
Result: A local minimum f^* for (5.6) and the corresponding partition Π^* representing the piecewise constant parts of f^*.

This concludes this section. For a discussion and outlook we refer to Section 15.1.

In the following section we compare the different formularized minimization strategies from this chapter visually and based on their energy levels. We also propose an easy, very efficient variant that is reasonable in most cases.

Appendix

5.A Proof: Merging Values

In this appendix section we prove the merge conditions stated in Lemma 5.8 and Lemma 5.9. We have the same conditions and assumptions as in these Lemmas. Let Π be the partition we are looking at and G_r the corresponding reduced graph. Let us consider some edge $(A_i, A_j) \in E_r$ for which we compute the merge value for the merged segment $A_i \cup A_j$ as

$$\begin{aligned} c_{A_i \cup A_j} = \underset{c \in \mathbb{R}^d}{\operatorname{argmin}} \; D_{A_i}(c, g) + D_{A_j}(c, g) = \sum_{u \in A_i \cup A_j} (c - g(u))^2 \\ \Leftrightarrow \quad c_{A_i \cup A_j} = \frac{\sum_{u \in A_i \cup A_j} g(u)}{|A_i| + |A_j|}. \end{aligned} \tag{5.56}$$

Plugging this into the merge condition in (5.29) we can do the following reformulations

$$\begin{aligned} \sum_{u \in A_i} (c_{A_i} - g(u))^2 + \sum_{u \in A_j} (c_{A_j} - g(u))^2 + \alpha w_r(A_i, A_j) \\ - \sum_{u \in A_i \cup A_j} (c_{A_i \cup A_j} - g(u))^2 \geq 0. \end{aligned} \tag{5.57}$$

By computing the binomial formulas we deduce that

$$\sum_{u \in A_i} g(u)^2 + \sum_{u \in A_j} g(u)^2 - \sum_{u \in A_i \cup A_j} g(u)^2 = 0$$

and reformulate (5.57) as

$$\begin{aligned} |A_i| c_{A_i}^2 + |A_j| c_{A_j}^2 - |A_i \cup A_j| c_{A_i \cup A_j}^2 \\ - 2c_{A_i} \sum_{u \in A_i} g(u) - 2c_{A_j} \sum_{u \in A_j} g(u) \\ + 2c_{A_i \cup A_j} \sum_{u \in A_i \cup A_j} g(u) + \alpha w_r(A_i, A_j) \geq 0. \end{aligned} \tag{5.58}$$

By defining $\bar{g}_A = \frac{\sum_{u \in A} g(u)}{|A|}$ as the mean of the data over a segment A, and thus, $c_{A_i \cup A_j} = \bar{g}_{A_i \cup A_j}$ we can reformulate the former equation with

$$c_{A_i \cup A_j} \sum_{u \in A_i \cup A_j} g(u) = c_{A_i \cup A_j}^2 |A_i \cup A_j| \tag{5.59}$$

into the formulation that is stated in Lemma 5.8 given as

$$\alpha w_r(A_i, A_j) \;\geq\; |A_i|(2c_{A_i}\bar{g}_{A_i} - c_{A_i}^2) + |A_j|(2c_{A_j}\bar{g}_{A_j} - c_{A_j}^2) - |A_i \cup A_j|c_{A_i \cup A_j}^2.$$

When we assume that the values $c_{A_i} = \frac{\sum_{u \in A_i} g(u)}{|A_i|}$ and $c_{A_j} = \frac{\sum_{u \in A_j} g(u)}{|A_j|}$ as the mean value of the data g over the segments $A_i, A_j \in \Pi$ can deduce compact formulation as we present in Lemma 5.9. To do so we compute that

$$|A|(2c_A\bar{g}_A - c_A^2) = |A|(2c_A^2 - c_A) = |A|c_A^2,$$

and hence, we have

$$\alpha w_r(A_i, A_j) \;\geq\; |A_i|c_{A_i}^2 + |A_j|c_{A_j}^2 - |A_i \cup A_j|c_{A_i \cup A_j}^2.$$

We can also see that

$$\begin{aligned} c_{A_i \cup A_j} &= \frac{\sum_{u \in A_i} g(u) + \sum_{u \in A_j} g(u)}{|A_i| + |A_j|} \\ &= \frac{|A_i|c_{A_i} + |A_j|c_{A_j}}{|A_i| + |A_j|} \end{aligned}$$

and also compute the squared value as

$$\begin{aligned} |A_i \cup A_j|c_{A_i \cup A_j}^2 &= |A_i \cup A_j|\frac{(|A_i|c_{A_i} + |A_j|c_{A_j})^2}{(|A_i| + |A_j|)^2} \\ &= \frac{|A_i|^2c_{A_i}^2 + 2|A_i||A_j|c_{A_i}c_{A_j} + |A_j|^2c_{A_j}^2}{|A_i| + |A_j|}. \end{aligned} \tag{5.60}$$

Then we can use the expansion of a fraction by $|A_i| + |A_j|$ like

$$\begin{aligned} |A_i|c_{A_i}^2 &= \frac{|A_i|c_{A_i}^2(|A_i| + |A_j|)}{|A_i| + |A_j|} \\ &= \frac{|A_i|^2c_{A_i}^2 + |A_i||A_j|c_{A_i}^2)}{|A_i| + |A_j|} \end{aligned}$$

and the same for c_{A_j}. Subtracting (5.60) from these terms yields

$$\begin{aligned} |A_i|c_{A_i}^2 + |A_j|c_{A_j}^2 - |A_i \cup A_j|c_{A_i \cup A_j}^2 &= \frac{|A_i||A_j|(c_{A_i}^2 + c_{A_j}^2 - 2c_{A_i}c_{A_j})}{|A_i| + |A_j|} \\ &= \frac{|A_i||A_j|}{|A_i| + |A_j|}(c_{A_i} - c_{A_j})^2 \end{aligned}$$

which finally leads to the condition that we merge the segments A_i and A_j if

$$\alpha w_r(A_i, A_j) \geq \frac{|A_i||A_j|}{|A_i| + |A_j|}(c_{A_i} - c_{A_j})^2. \tag{5.61}$$

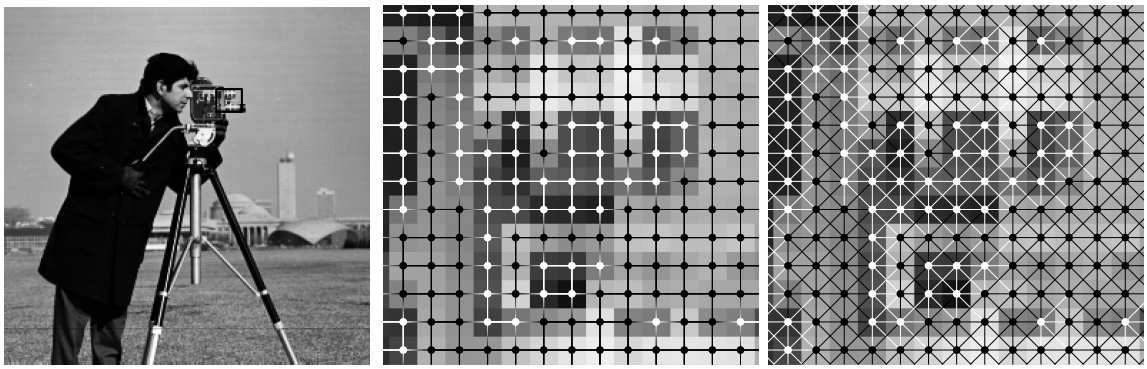
6

Numerics: Cut-Pursuit for TV Problems

In this chapter we apply the Cut-Pursuit algorithm from Algorithm 3 to solve ROF problems on graphs to 1D signals, different images and to 3D point cloud data. We start by taking an insight into the iterations steps of Cut-Pursuit and observe the evolution of the solution. Then we compare run time and convergence for different data types and investigate the convergence of the isotropic Cut-Pursuit algorithm for different direction strategies. Additionally, we want to show the remarkable and useful by-product of a simple debiasing step after the solution was computed using Cut-Pursuit .

The primal-dual algorithm from Algorithm 3.20 and the Cut-Pursuit algorithm from Algorithm 3 are implemented in an object-oriented fashion in MATLAB. It was implemented modular such that, e.g., the implementation of the primal-dual algorithm on graphs is used not only for solving the problems on the full graph, but also for the computation of the reduced solutions in the Cut-Pursuit algorithm. This makes the efficiency of the code comparable for all algorithms. In addition, the different directions for the isotropic Cut-Pursuit implementation share the same code basis and are just exchanged for different settings. To cut the flow graphs in the partition problem step of the Cut-Pursuit algorithm we use the MATLAB internal *maxflow* implementation which is a wrapper for efficient code written in `C`. Note that this implementation is only working on one CPU kernel, and hence, has certain drawbacks when the graph structure becomes very complex. We also want to point out that the implementation of algorithms on graphs is not easy due to the non-local connection of vertices. This can lead to very inefficient look-ups of variables in the memory. Additionally, one cannot use the very efficient vectorized computations that MATLAB provides as one can use, e.g., for finite differences, which leads to run time drawbacks.

We apply the algorithms on 1D signals, images and point cloud data. The 1D signal is described by a simple undirected graph that connects direct neighbors with each other. The weight is set to 1 for all edges. Note that one could also incorporate non-local edges or more neighbors in the graph.



(a) Cameraman (b) Grid graph (c) *Star-shaped graph*

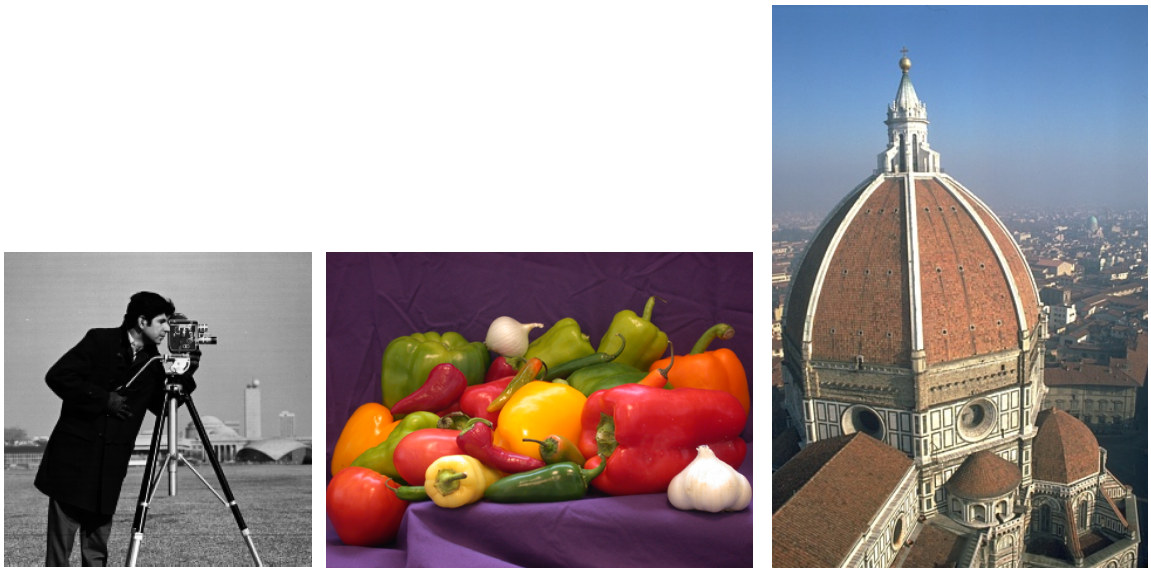
Figure 6.1.: Graph structure to represent an image.

The graph structure to describe an $m \times n$ image is given by a vertex set with $N = nm$ vertices where each vertex is assigned to one pixel, e.g., by assigning the linear index of the pixel to the vertex. To mimic discretization of gradients as finite differences we connect each vertex with the vertices of the 4 neighboring pixels of its assigned pixel. Then we have a grid graph as can be seen in the middle of Figure 6.1. We denote this as the *(grid) image graph*. Without prior knowledge of any underlying structure we set the weight for every edge to 1. As we pointed out in Section 3.1.4, we can only compute spatially-anisotropic solutions with Cut-Pursuit on the graphs. To tackle this problem and also get solutions with more isotropic appearance we can also use a more complex graph structure on images. For that structure we also incorporate the diagonal neighbors of each pixel into account as we show in the right figure in Figure 6.1. We denote this graph as the 8-*neighbor image graph* which then incorporates more spatially-isotropic information, and hence, the solution of a spatially-anisotropic problem has a more isotropic appearance. We show results in the next section even though we do not use this to greater extend in this work. The difference of grayscale and RGB images is the dimension of the vertex functions. For grayscale images we have $f\colon V \to \mathbb{R}$ and for RGB we have $f\colon V \to \mathbb{R}^3$. In Figure 6.2 we show the images that we use in this work in their original resolution. As we investigate the application of Cut-Pursuit on point cloud data later in Chapter 8 we only investigate the *Bunny* point cloud data - we visualize in Figure 6.3 - here. For point clouds we always compute a k-nearest neighbor graph (k-NN graph) (cf. [9], [23]) and set the weight of an edge $(u, v) \in E$ to

$$w(u,v) = \exp\big(-\frac{\|f(v) - f(u)\|_2^2}{2}\big).$$

This weighting incorporates that points that are far away have only little to no influence on each other while close points get assigned a high weight. Furthermore, this weight caps at 1 since all values are positive, which consequently means that all weights are between 0 and 1. The vertex functions are defined as for RGB images to be functions $f\colon V \to \mathbb{R}^3 \in \mathcal{H}(V; \mathbb{R}^d)$.

[1] Matlab built-in images
[2] http://r0k.us/graphics/kodak/



(a) Cameraman[1] 256 × 256 px

(b) Peppers[1] 384 × 512 px

(c) Florence Cathedral [54] 481 × 321 px

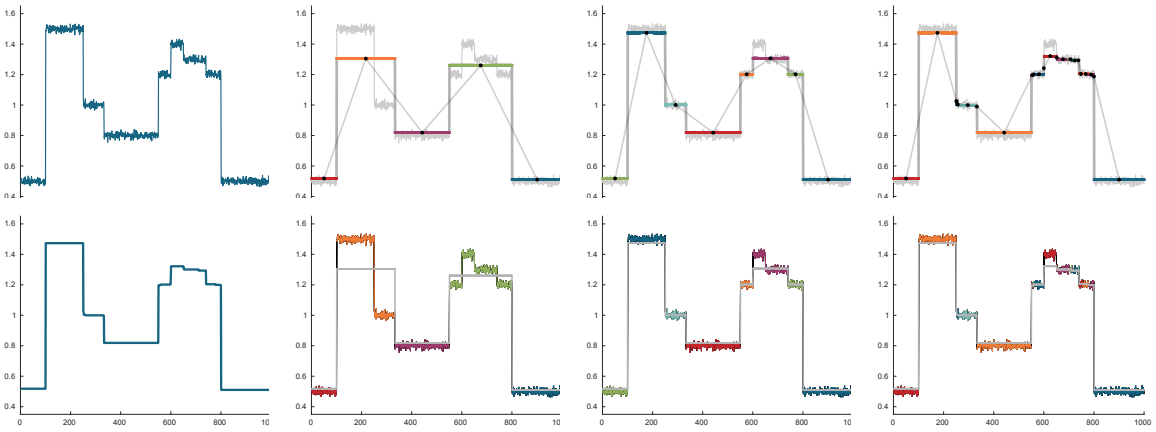
(d) Piano [75] 1920 × 2820 px

(e) Plane [54] 321 × 481 px

(f) Parrots[2] 512 × 768 px

Figure 6.2.: Images used in this numerical experiments.

Figure 6.3.: *Bunny* (35, 947 points), *Happy* (543, 652) and *Dragon* (437, 645) point cloud from [82].



(a) Noisy signal (top) and TV solution (bottom).

(b) Iteration: 1
Number of sets: 5

(c) Iteration: 2
Number of sets: 9

(d) Iteration: 5
Number of sets: 19

Figure 6.4.: Denoising of a noisy 1D signal via Cut-Pursuit. Top row shows the solution and the reduced graph at a certain iteration and the bottom row the corresponding sets labeled by color.

6.1 Iterative Behavior of the Cut-Pursuit Algorithm

In this section, we want to give an insight into the actual iterative steps of the Cut-Pursuit algorithm to get an intuition of how Cut-Pursuit processes data and what we can expect as a result. For this reason, we investigate two Cut-Pursuit applications to two toy examples. In the first one we denoise the 1D signal we present in Figure 6.4 and in the second we denoise an RGB image on a grid image graph Figure 6.5 and on an 8-neighbor image graph Figure 6.6.

First let us take a look at the 1D example in Figure 6.4. The graph of this 1D data is built by just connecting each vertex with its two neighbors and weight each edge with 1. On the left side of Figure 6.4 we see the noisy input signal and the denoised TV solution. In the following three columns we illustrate the current state of the solution in different iteration steps. In the top row we visualize the current solution and color-code the segments. We also show the reduced graph and its current values. Note that the segments in the upper and lower plots are color-coded equally. The light gray signal in the background is the noisy input data. In the bottom row we just color-code the noisy input data by their current set. This sequence illustrates that the path the solution takes is by first cutting the data into rough parts that approximate certain levels of the data. Then as the algorithm proceeds it refines these segments in each iteration, but only if it is necessary for the solution. This can be seen by looking at the larger plateaus in the converged solution that are described by only one vertex, while the parts with more details - as on the right hand side - are described by more vertices. In the end the solution can be described by 19 vertices instead of $1,000$ points.

As we have now seen the path the solution takes let us take a look at an noisy RGB image in Figure 6.5. That image consists of some colorized objects with simple shapes. In the first column we start in the top row with the initial result which is solving the reduced problem on only one vertex. If we have an L_2-data-term this is the mean value of the input image. Then a cut is computed via the partition problem. To each of the disjoint segments a vertex gets

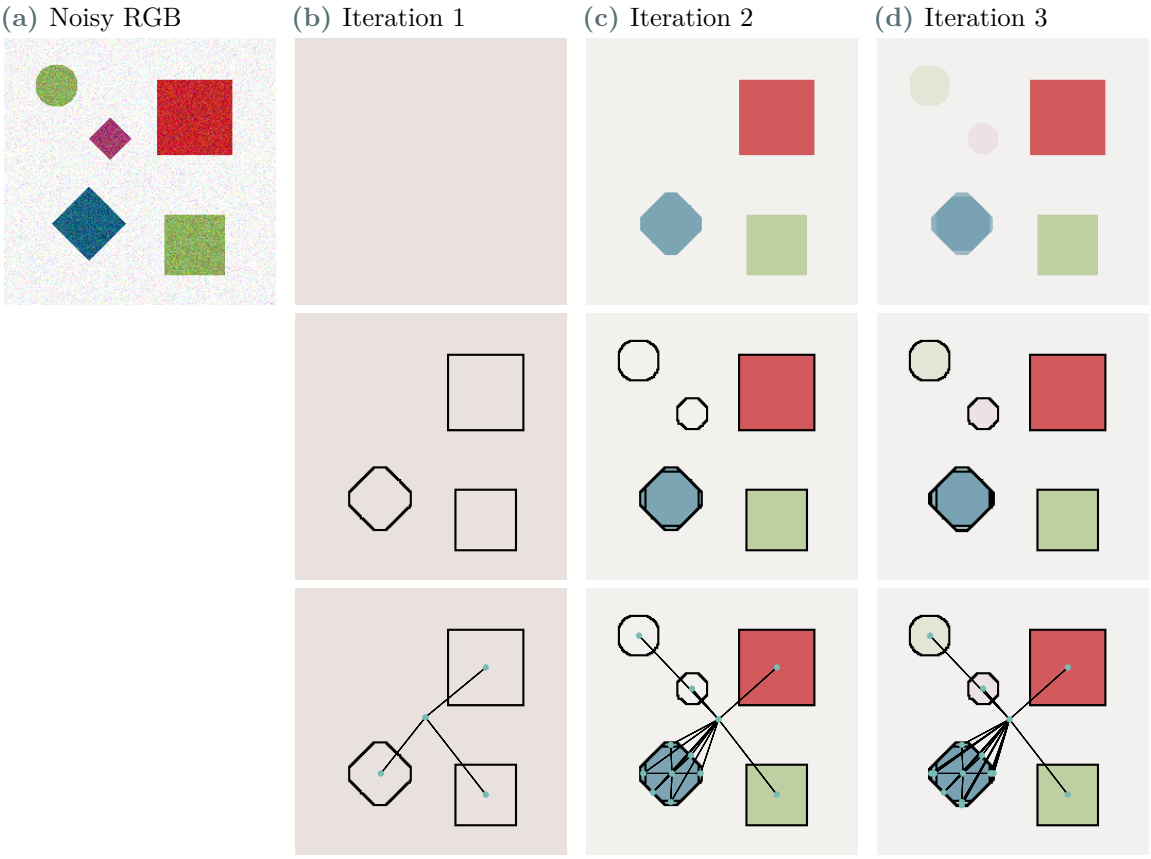


Figure 6.5.: Denoising of a noisy RGB image via isotropic Cut-Pursuit on an image graph.

assigned, and the reduced graph is built as in Proposition 4.16 where the reduced vertices are connected if the segments have boundaries in common and weighted by the length of the boundary between them. On this reduced graph we can then solve the reduced ROF problem (4.63) via Algorithm 4.14 and get the result in the third column. Then we again compute a cut which introduces the missing smaller objects and also parts of the corners of the purple rhombus. Here we can see that we only work on a spatially-anisotropic grid since the corners of the rhombus are not aligned with the axis, and hence, are not represented well by the regularization. Consequently, the rhombus is now described by multiple vertices in the reduced graph setup. On the other hand the green and the red square are both perfectly aligned with the axis which leads to no further cut in these objects. Each of them is still represented by only a single vertex in the reduced graph. In the last column we see the resulting reduced graph and solution which suffers a lot from the TV induced loss of contrast. This is a problem we face in Section 6.4.

Let us also look at Figure 6.6 where we have the exact same image as before, but in this case we use a 8-neighbor image graph to induce implicit spatial-isotropy. We see the same process as before and now have an intuition of how the algorithm works. The main difference we can observe in the intermediate results is that the rhombus objects are now described in the reduced graph by a single vertex while the squares seem to have problems at the corners. This is what we would expect from this discretization of the graph and is a visual proof of

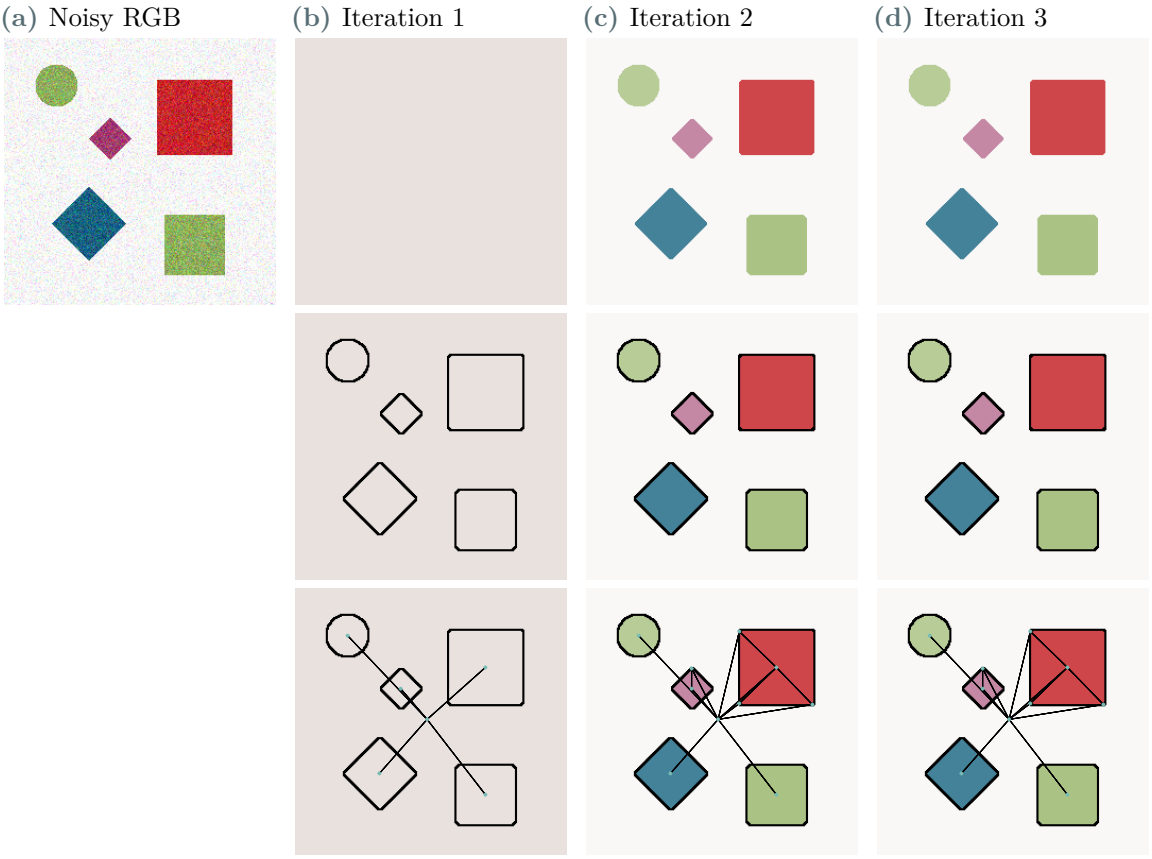


Figure 6.6.: Denoising of a noisy RGB image via Cut-Pursuit on the 8-neighbor image graph.

concept that we can produce isotropic-like results by choosing an 8-neighbor image graph and solve and spatially-anisotropic ROF problem on it. Note, that one could also increase the number of neighbors to an even higher number, e.g., such as 16 or 32. This would increase the complexity of the graph, and thus, the computational cost of the maxflow algorithm, but also induce an even more isotropic-like appearance to the solution.

Finally, let us take a look at a real-world example in Figure 6.7, where we denoised a noisy version of the *piano* RGB image. We can see that the reduced graph again has a high density in regions with more details and less density in large piecewise constant areas as we would expect from the simpler examples.

As we now built up an intuition of how the Cut-Pursuit algorithm processes the given input data and how the reduced graphs look like let us proceed and investigate the numerical convergence of the Cut-Pursuit algorithm compared to the primal-dual algorithm on graphs we defined in Algorithm 3.20 applied to the same problems.

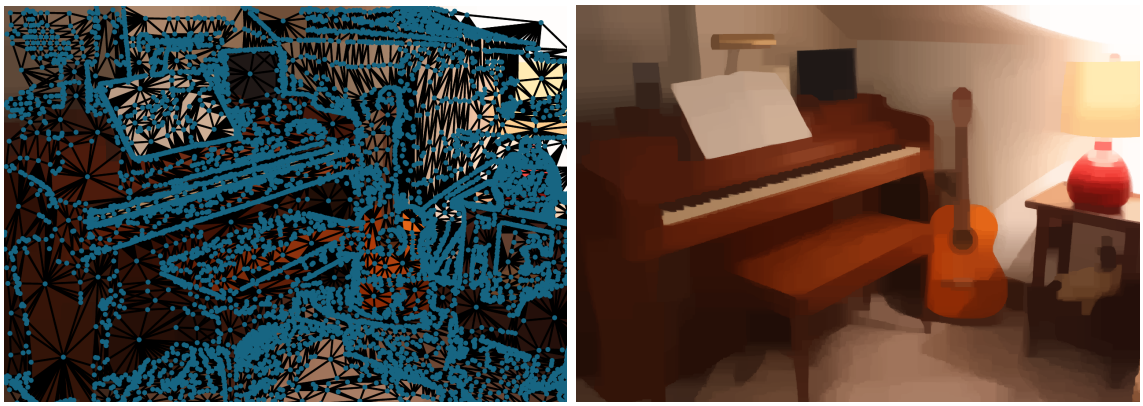


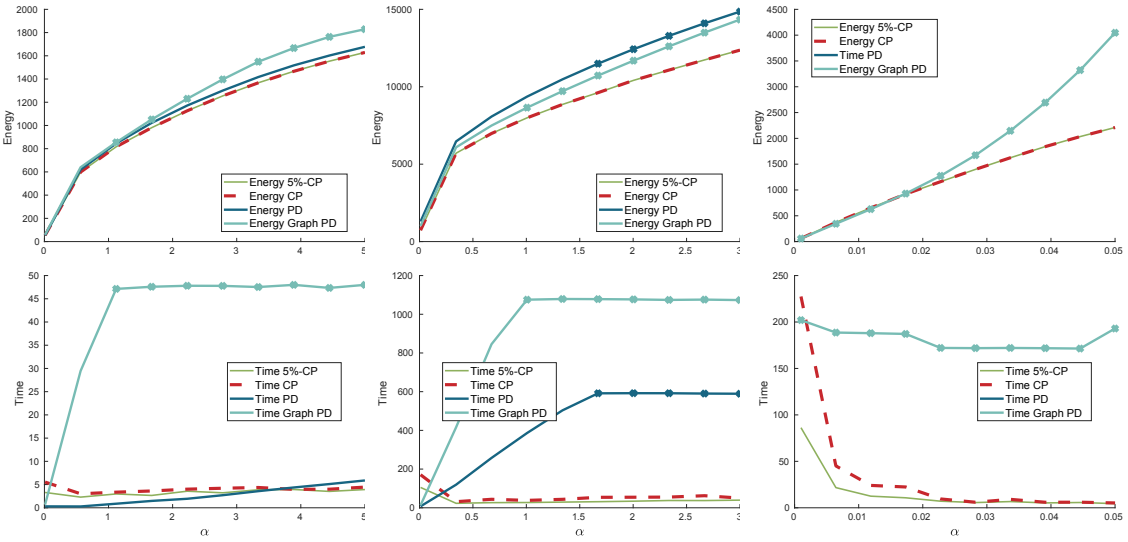
Figure 6.7.: Reduced graph (left) with 11,291 vertices corresponding to the TV full solution on 196,608 pixels (right).

6.2 Cut-Pursuit Convergence and Runtime

In this section, we study the convergence of the Cut-Pursuit algorithm on three different data sets: the grayscale image *cameraman*, the RGB image *parrot* and the 3D point cloud *bunny*. For the images we want to compare the direct primal-dual implementation using finite differences, the corresponding graph implementation and the Cut-Pursuit algorithm. For the Cut-Pursuit algorithm we use two different variants: The first variant stops when no new cut can be found in the partition problem. The second one stops if the number of segments did not increase by more than 5% between iterations. As we have seen in the former section, this stopping condition can be reasonable since the Cut-Pursuit algorithm first cuts out the coarser details and then becomes finer in every iterations. Hence, one can expect the algorithm to make many small cuts in the iterations before convergence where the energy does not decrease much anymore. Then it would be appropriate to stop the algorithm and use that solution. This method saves some iterations, and consequently, multiple graph cuts and computations of the reduced solution. Additionally, as the reduced graph is finer in later iteration steps solving the reduced problem is more costly than in the former iterations. This can slow down the algorithm in later iteration steps even though the gain of a better energy value is negligible and also visually the difference is insignificant.

In Figure 6.8 we illustrate energy and run time plots of the different algorithms for multiple regularization parameter settings. The first notable observation is that we see that the primal-dual algorithm on graphs to solve the ROF problem always has the worst run times. As we stated before the implementation of primal-dual algorithm on graphs is not efficient. The location of values in memory can be far away from each other. The bottlenecks are the computation of the gradient and the divergence on graphs.

For the primal-dual algorithm on images we use constant initial step sizes that get updated in every iteration as defined in [17]. The step size for primal-dual algorithm on graphs is given by the diagonal preconditioning in (3.21). Here we did not use any acceleration step size updates as we also use for the reduced problem of the Cut-Pursuit algorithm. The crosses mark the parameter where the primal-dual algorithms did not converge within 8,000 iterations, i.e., the current iterate did not have a primal-dual-gap (cf. [17]) smaller that



(a) Grayscale Image: Cameraman (b) RGB Image: Parrot (c) Point Cloud: Bunny

Figure 6.8.: Energy and runtime plots to show the exactness of the results and the runtime. We compare primal-dual on the images, on the graphs and a converged Cut-Pursuit algorithm and one that stops when it has not change in the number of segments over 5%. The crosses mark primal-dual algorithms that did not converge (gap $< 10^{-5}$) within $8,000$ iterations.

10^{-5}. Applying Cut-Pursuit on the grayscale image does not yield any sufficient speed up, but, however, has a lower energy than the result computed by primal-dual algorithm. The primal-dual algorithm on graphs did not even converge within the $8,000$ iterations. For the RGB images we can see a massive speed up of the Cut-Pursuit algorithm compared to the primal-dual algorithms. Additionally, it also converged to a much better energy value.

In the point cloud setting where we denoise the *bunny* point cloud we can only use the primal-dual algorithm on graphs. The graph was computed as a k-NN graph with $k = 7$ and the weights were set as above. We see that the primal-dual algorithm does not converge and always takes around 200 seconds for $8,000$ iterations. For smaller regularization parameters the energies are close, but as the regularization parameter increases the primal-dual algorithm on graphs cannot reach the energy in this number of iterations. Here the superior efficiency of the Cut-Pursuit algorithm can be seen. In Chapter 8 we investigate the sparsification properties of the Cut-Pursuit algorithm and apply it to multiple point cloud data sets and study different settings.

6.3 Isotropic Cut-Pursuit and Choosing Directions

As we have discussed in Section 4.4.3, it is crucial to have a good rationale for the decision of the directions assigned to each segment to solve the partition problem. The algorithm stops if the new computed partition is the same as the former one, which means that the partition problem has a trivial solution where $B = \emptyset$ or $B = V$. However, this does not mean that we reached the optimum since we would have to check if the solution for all directions yields no cut. To tackle this problem we show two choices of directions and also show the influence

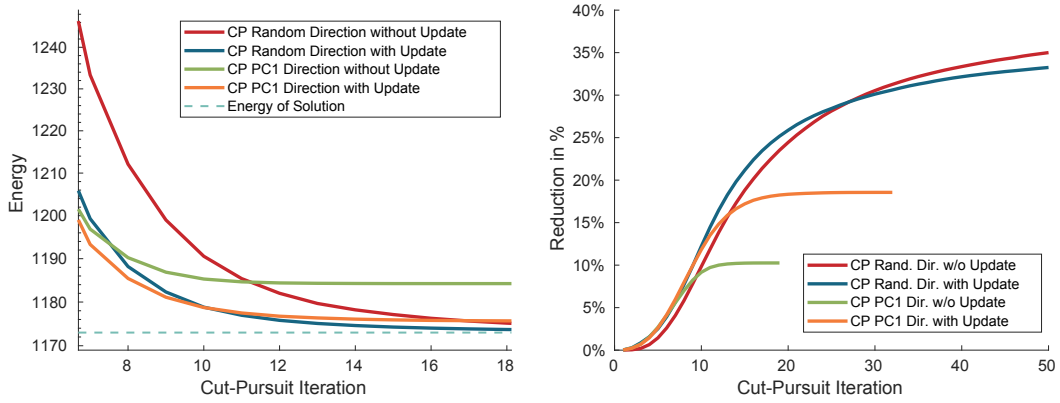


Figure 6.9.: Denoising of a noisy RGB image via isotropic Cut-Pursuit with $\alpha = 0.01$. Left: Energy plots for the different direction methods compared to the solution of the problem. Right: Corresponding reduction rate $\frac{\text{number of sets}}{\text{number of pixels}}$.

of updating the cut and the direction in the partition problem as we do in the alternating algorithm in (4.81).

The first most straightforward idea would be to just compute a random direction for every segment in the partition and compute a cut for that. This can be expected to converge very close to the exact solution of the ROF problem since the probability that all segments have selected a random direction that does not induce a new cut is rather low. Even when this happens this can be expected to be very close to the optimal solution. However, the randomness might also lead to not optimal directions that yield a long iteration and also many cuts that are not needed which provides a too fine final partition to represent the solution.

As a second idea, we propose to use the principal component analysis (PCA) [41] [79] to compute the first principal component. This direction can be understood as the best fitting line through the data that minimizes the squared distance of all points to the line. This line points into the direction with the most variance in the data which makes it a reasonable choice for a direction. Hence, we compute the first principal component for all the data points in every segment. This direction can be expected to be a good choice in the beginning but might yield a solution that is not optimal in terms of the energy. We denote this direction as the *PC1* direction in the following.

Let us take a look at the provided experiments where we apply Cut-Pursuit to the *parrot* RGB image and the *bunny* point cloud data. We use both methods to generate directions and also show results with and without updating the directions as in (4.81). The number of inner iterations is set to 5 since in most cases this is enough. We also state which direction type we use for the initial direction for the update routine. In Figure 6.9 and Figure 6.10 we illustrate the energy plots for the four different selections on the left side and the rational of number of sets to the total number of vertices in the original data. For the RGB image we can see that using PC1 without updates yields a solution that does not converge to the optimal energy level. The random and PC1 directions with update steps on the other hand have a fast decrease in energy, where the PC1 direction at first has a better energy value but then gets overtaken by the random direction. Additionally, we can see that the random direction with updates seem to converge to a better energy value that is very close to the

optimum. Moreover, the random direction without updates has by far the worst decrease in energy but even outperforms the PC1 with updates in the end. In the right plot in Figure 6.9 we see that the random selection leads to a much finer partition in the end. This can be interpreted as a reason for the energy decrease as the partition is much finer, and hence, the reduced problem can yield a solution with a low energy value. In Figure 6.11 we see the given results that correspond to the plots in Figure 6.9. Visually the results look very similar and well denoised. The difference of the converged (primal-dual-gap $< 10^{-8}$) primal-dual solution and the Cut-Pursuit results can be only seen in small details. As we see in the boxes in the right corner, the primal-dual solution has a smoother solution in smoother regions than Cut-Pursuit can provide for any method. However, if we evaluate the denoising visually it is not possible to tell which denoising is the best without direct and detailed comparison. Consequently, the choice of algorithm is then dependent on the application and whether the optimal solution of the ROF problem is the preferred one or if the solution only has to be visually satisfying.

For the second experiment we see in Figure 6.10 that the denoised solution of the input point cloud data seems to be independent of the choice of directions if we use them as initialization of the updates. Also PC1 without using the update algorithm yields the same decrease in energy and a comparable optimal energy value. On the other hand the random directions without updates seem to converge worse than the PC1 directions but catch up later and only need a few more iterations to converge to the same level of energy. As expected the PC1 direction yields a coarser solution, which, in this context, means less segments. In Figure 6.12 we see the visual results to the plots in Figure 6.10. As we see, they are visually not distinguishable and all seem to yield the same results. However, we see in Figure 6.10 that the PC1 direction without updates yields the coarsest solution and takes the least iterations. Hence, this will be the Cut-Pursuit algorithm of choice in Chapter 8.

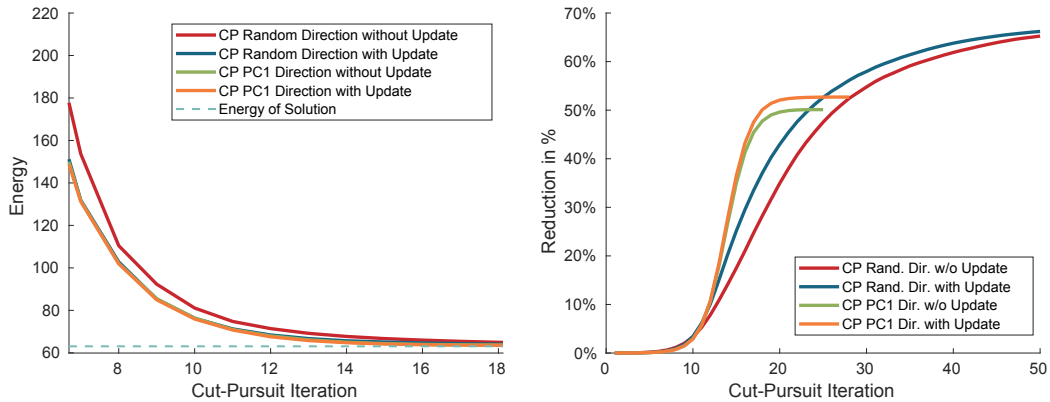
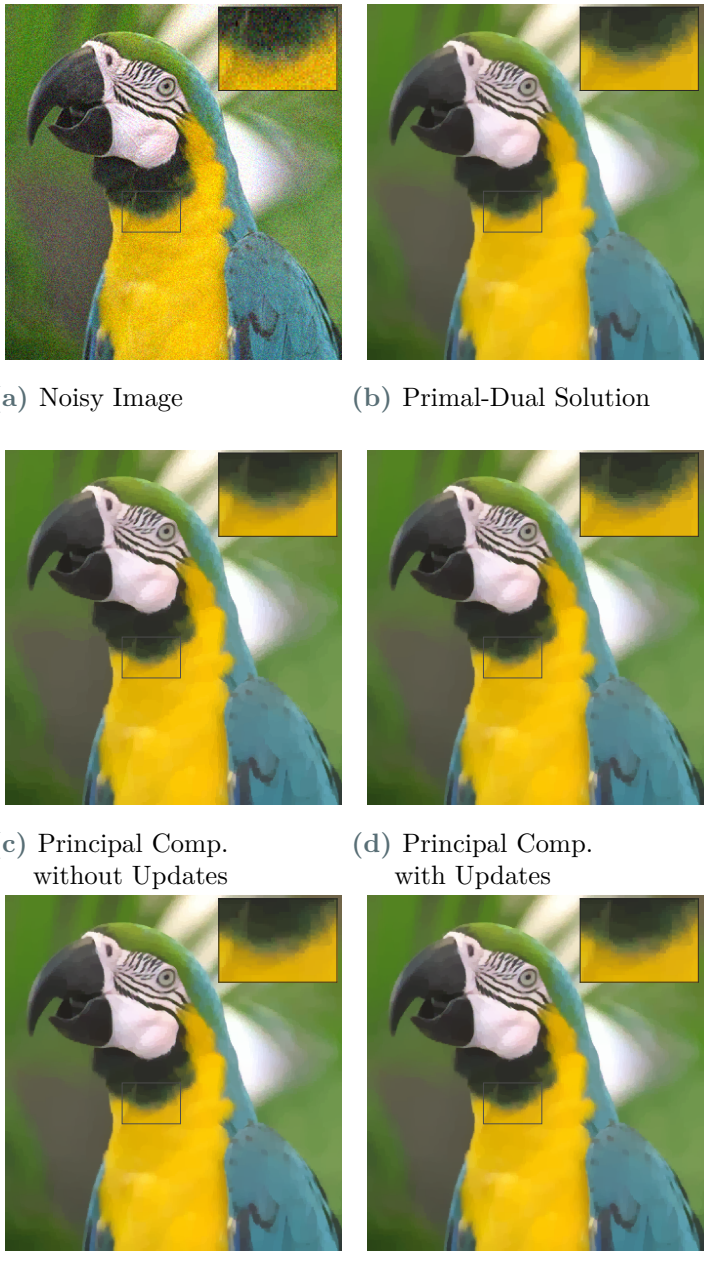


Figure 6.10.: Denoising of a noisy Bunny point cloud via isotropic Cut-Pursuit with $\alpha = 0.1$. Left: Energy plots for the different direction methods compared to the solution of the problem. Right: Corresponding reduction rate $\frac{\text{number of sets}}{\text{number of points}}$.



(a) Noisy Image

(b) Primal-Dual Solution

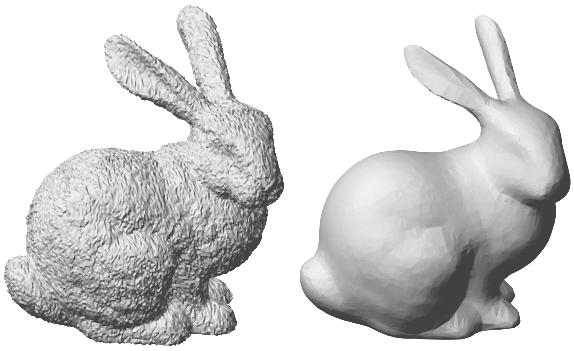
(c) Principal Comp. without Updates

(d) Principal Comp. with Updates

(e) Random Dirs. without Updates

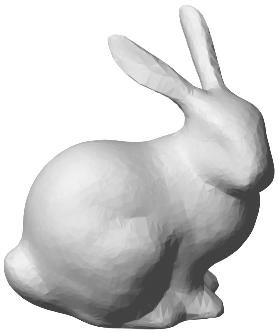
(f) Random Dirs. with Updates

Figure 6.11.: Denoising of a noisy RGB image via Cut-Pursuit. The graph was built as a grid graph.

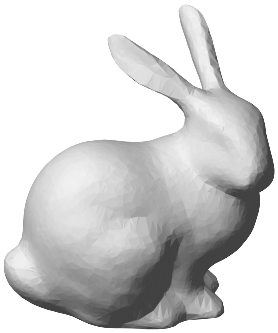


(a) Noisy Image

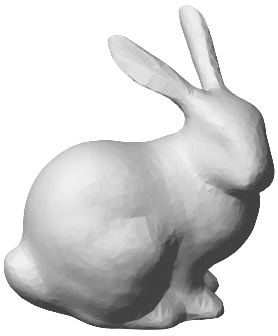
(b) Primal-Dual Solution



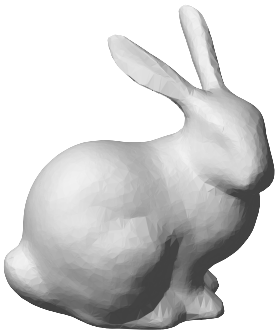
(c) Principal Comp. without updates



(d) Principal Comp. with updates



(e) Random Dirs. without updates



(f) Random Dirs. with updates

Figure 6.12.: Denoising of a noisy Bunny point cloud via isotropic Cut-Pursuit. The graph was built via kNN-graph generation with $k = 7$.

6.4 Debiasing

In this section, we want to take a look at how to apply a bias reduction to a TV solution computed with Cut-Pursuit like Brinkmann et. al. introduce in [12]. In their work they propose a two-step debasing method that is applicable to variational regularization problems. The first step is to compute the solution f_α^* of a variational problem with a regularization parameter α and then minimize only the data-term on the so-called *model manifold* which is the space of functions sharing the same regularity, i.e., jump set, as f_α^*. In case of a TV-regularized problem, the model manifold consists of all functions sharing the same jump set as f_α^*.

In fact, if we compute a solution f_α^* with Cut-Pursuit , it is piecewise constant on the segments in partition Π^*. Sometimes if neighboring segments have the same value - which can happen since the Cut-Pursuit is not optimized to yield the perfect partition for a solution - we have to apply a merge step that combines these segments. The resulting partition Π^* can be interpreted as the analog to the discrete *model manifold* presented in [12], which essentially captures the jump set between the piecewise constant segments in Π^*. With this model manifold at hand, we can minimize the data-term on each segment and get the debiased solution.

In the context of ROF problems the solution of the L_2-data-term is the mean value of the data in each segment in Π^*. Hence, we see that Cut-Pursuit yields the possibility to apply debiasing to the TV-regularized solution without any further optimization strategies. However, we keep in mind that the channel-isotropic Cut-Pursuit algorithm does not always yield a minimizer as we discussed in Section 4.2.3 and in the numerics in Section 6.3. Nevertheless, we can still perform debiasing procedure over the partition the algorithm yields.

Let us start with the application of the Cut-Pursuit algorithm on a one-dimensional signal in Figure 6.13 where we see the original noisy signal as the blue line and the corresponding TV solution as the red line. We see that the solution suffers from contrast loss. As we have seen in Figure 6.4, the piecewise constant parts of the solution are each represented by one vertex. Hence, we can compute the mean value on each of these segments and get the

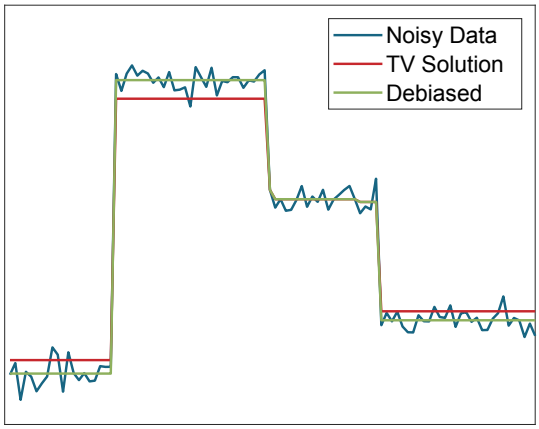


Figure 6.13.: Debiased solution of a noisy 1D signal by applying Cut-Pursuit and solving data-term on partition.

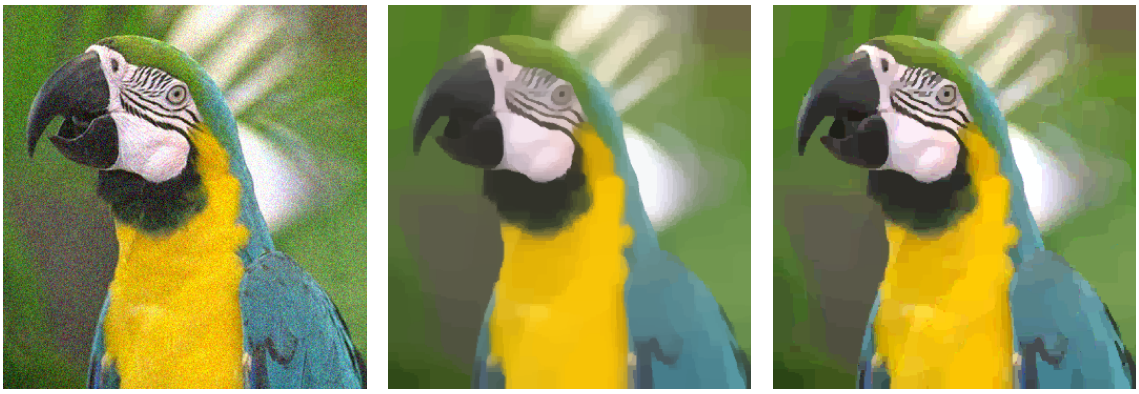


Figure 6.14.: Debiased solution of a noisy RGB image by applying Cut-Pursuit and solving data-term on partition.

debiased solution that is plotted as the green line. This recovers the results presented in [12] and [65].

Turning to channel-isotropic TV regularization problems we show in Figure 6.14 that the TV solution suffers from blurriness due to contrast loss. Computing the mean RGB value over each segment in the solution yields a nicely denoised and debiased result on the right side.

Finally, we apply the debiasing method to the *bunny* point cloud data. Again we solve the problem using the channel-isotropic Cut-Pursuit algorithm. Note that we have seen in the former section that in the setting of the point clouds it is reasonable to just use the PC1 direction as they all converge to the solution. As we see in Figure 6.15, the TV solution compresses the point cloud. Applying the debiasing reinflates the point cloud to the original size of the bunny, but in a more sparse variant. Note that we used a noise-free point cloud due to visualization purposes. This works also on noisy data.

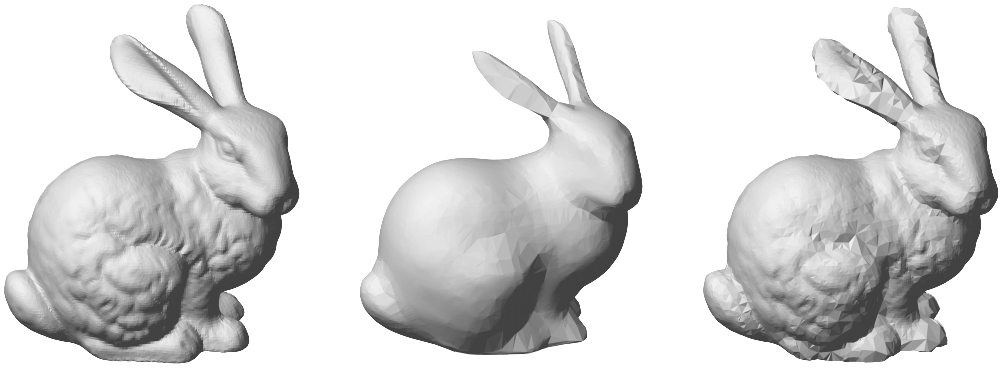


Figure 6.15.: Debiased solution of the bunny point cloud by applying Cut-Pursuit and solving data-term on partition.

To summarize this we can conclude that the Cut-Pursuit algorithm gives us the ability to reduce the contrast loss of any solution by minimizing the data-term over the reduced graph. This is also possible for more complicated data-terms, e.g. the Kullback-Leibler data-term from Definition 2.9. In the case of an ROF problem it is given by the computation of mean value of the given data over each segment.

7
Numerics: Minimal Partition Problem

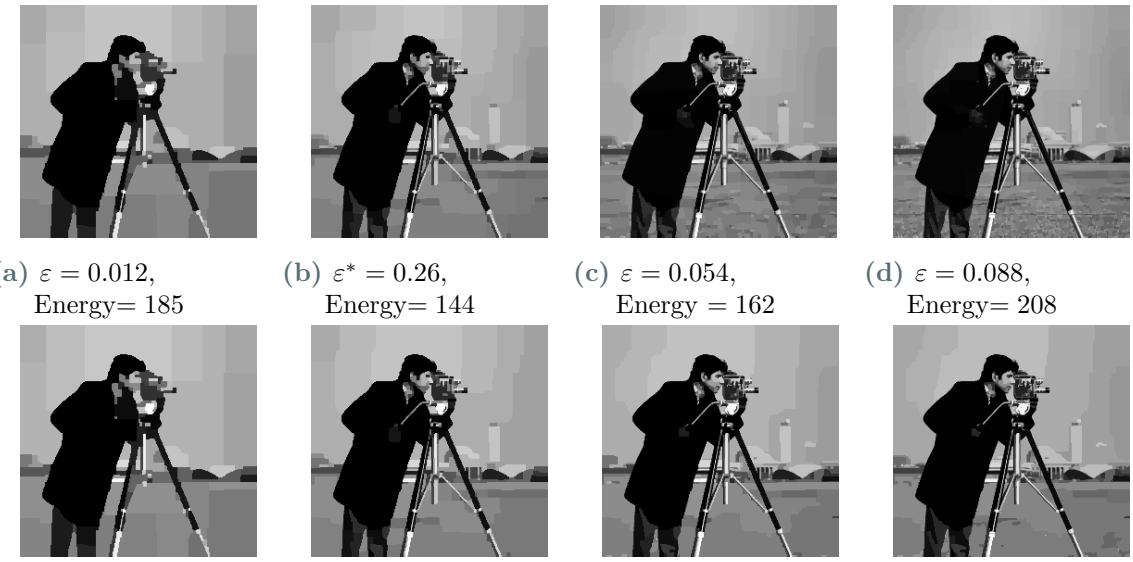
In this section, we apply the algorithms proposed in Chapter 5 to different data setups. Throughout this section we consider the L_0-ROF problem as stated in (5.37). We start off by looking at the problem of choosing a fixed $\varepsilon > 0$ without optimizing its value. Then we study the convergence of the different strategies numerically and also compare them with the provided implementations of Fast MS [83] and Cut-Pursuit [46].

7.1 Comparing Different Constant Step Sizes

In this section, we investigate the difference in the results by solving the partition problem in (5.17) with an L_2-data-term by the proposed strategies from Section 5.1.1 and Section 5.1.2. We denote this problem as the L_0-ROF problem in this section.

To show the importance of the choice of $\varepsilon > 0$ we first take a look at the energy of solutions provided by L_0-Cut-Pursuit from Algorithm 6 for different values of fixed ε. The reduced problem is solved by minimizing the reduced data-term which is just the mean value over each segment as derived in (5.55). The plot we depict in Figure 7.2a shows the energy of the solution of L_0-Cut-Pursuit applied to the grayscale *cameraman* image for different ε and a fixed regularizer $\alpha = 0.01$. As we see in Figure 7.2a, setting $\varepsilon = 1$ leads to a non-optimal resulting energy, while setting $\varepsilon \approx 0.026$ yields a near optimal energy value.

Looking at Figure 7.3a we see that for many choices of ε the energy does not monotonically decrease, but rather starts increasing after hitting a certain iteration. This might be interpreted as a too large step size for a descent step on the partitioning problem. Additionally, we see that for $\varepsilon^* \approx 0.026$ the energy decreases monotonically and has the lowest resulting energy. To explain this behavior let us take a look at the partition problem in (5.49). There we see that this problem equivalent to the problem divide by $\varepsilon > 0$, and thus, can deduce that the step size ε does influence the regularity of the problem directly antiproportional. In fact, we divide the regularization parameter α by ε which implies that the regularization changes for the partition problem with the choice of the step size. If we additionally drop the regularizer in the reduced problem, as we state in Section 5.1.2, this term is the only regularization in the algorithm. Thus, selecting a too large ε leads to a smaller regularization parameter which then yields a partition that is finer than a lower ε would have generated. Thus, we can expect that a in some sense too large selected ε follows a higher energy value



(a) $\varepsilon = 0.012$, Energy= 185 **(b)** $\varepsilon^* = 0.26$, Energy= 144 **(c)** $\varepsilon = 0.054$, Energy = 162 **(d)** $\varepsilon = 0.088$, Energy= 208

(e) $\varepsilon = 0.012$, Energy= 185 **(f)** $\varepsilon^* = 0.26$, Energy= 138 **(g)** $\varepsilon = 0.054$, Energy = 127 **(h)** $\varepsilon = 0.088$, Energy= 138

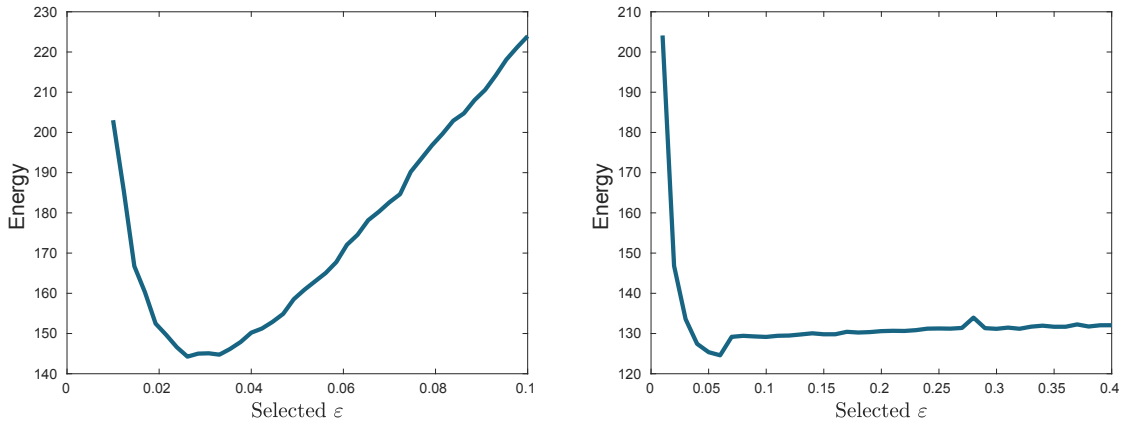
Figure 7.1.: Different results for $\alpha = 0.01$ with different fixed $\varepsilon > 0$. In (a)-(d) the mean value is computed on each segment in every iteration. In (f)-(h) use merging for reduced problem.

as for a smaller step size. This is what we can observe in the upper row of Figure 7.1 where we show different results for different selected step sizes. The fineness of the regularization of the solution is antiproportional to the selected ε. From Figure 7.2a and the upper row in Figure 7.1 we can deduce that for this data set and regularization parameter α the choice of $\varepsilon = 0.026$ would give a good result.

However, if we extend this strategy and solve the reduced problem using the merging strategy described in Lemma 5.9, we see in Figure 7.2b that choosing different ε seems to yield more consistent results according to the resulting energy. This consistency in results can be confirmed visually in the bottom row in Figure 7.1 where we show the resulting images for the same selected ε as in the upper row. Visually, these images are very similar and even the energy is comparable as we specify underneath the figures. Additionally, we see that the overall energy of the solutions with applied merging is much lower in Figure 7.2b as in Figure 7.2a.

Actually, this is what we expect since merging not only yields a solution with a better energy to the reduced problem, but also shrinks the number of segments again, and thus, can keep the partition from becoming too fine which would lead to higher regularity. This results in a lower overall energy and also a better total energy for every ε. Note that the selection of $\varepsilon > 0$ depends on many facts like regularization parameter, data sets and the corresponding structure of the graph and value intervals of the data. This makes the choice of ε hard, and hence, we show how to optimize the step sizes in the next subsection.

Before closing this subsection we want to emphasize an interesting property we take advantage of in Section 7.4. As state in a former part of this section, the partition problem of (5.49) can be divided by $\varepsilon > 0$ which yields a different regularization of the partition

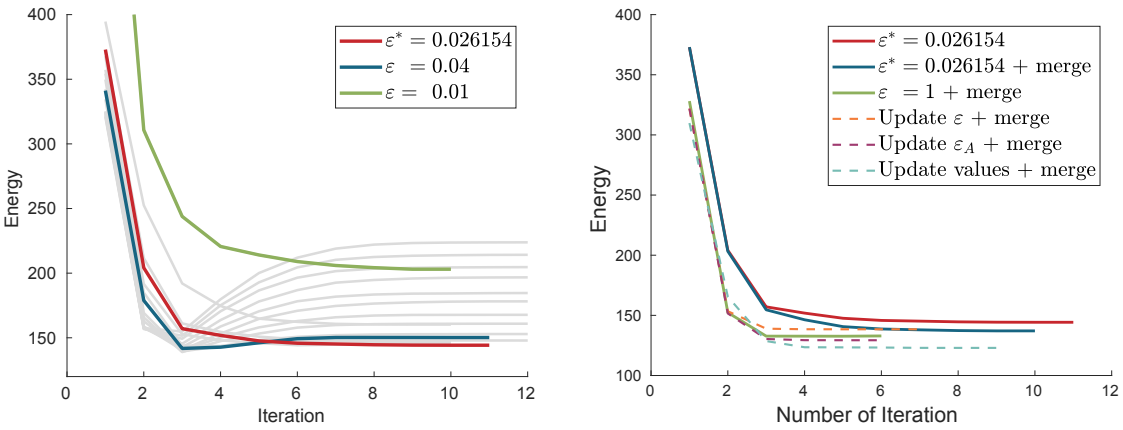


(a) Solving reduced problem by computing mean value.

(b) Solving reduced problem by mean value and merging.

Figure 7.2.: Energy values of solutions computed by L_0-Cut-Pursuit in Algorithm 7 with different fixed $\varepsilon > 0$ and $\alpha = 0.01$. In (a) we only compute the mean value over each segment $A \in \Pi$ and drop the regularization term. In (b) we see the same as in (a) where we just apply the merging step as in (5.54) after each mean value computation.

problem. We see in Figure 7.2a and Figure 7.3a for a regularization parameter $\alpha = 0.01$ that the optimal selected step size is $\varepsilon^* = 0.026154$. Hence, this is effectively the same result as what we get when applying the L_0-Cut-Pursuit algorithm to solve for $\tilde{\alpha} = \frac{\alpha}{\varepsilon^*} \approx 0.38$ and the step size $\tilde{\varepsilon} = 1$. The other way around if we compute the solution of a problem with α and a fixed $\varepsilon = 1$ we assume that there exists some regularization $\tilde{\alpha}$ which yields with its optimal selected $\tilde{\varepsilon}$ that $\alpha = \frac{\tilde{\alpha}}{\tilde{\varepsilon}}$. This it what we use in Section 7.4 to compute a feature dependent discretization of a given data set. We also show that the ideas from above can be applied and the intuition of decreasing the regularization parameter to get a finer discretization and increasing it to get a coarser one holds.



(a) Solving with fixed ε.

(b) L_0-Cut-Pursuit strategies.

Figure 7.3.: Energy plots. In (a) we see energy plots for $\alpha = 0.01$ for different choices of ε with three highlighted plots. The red line is the optimal selected ε^*. In (b) we see different energy plots for different methods.

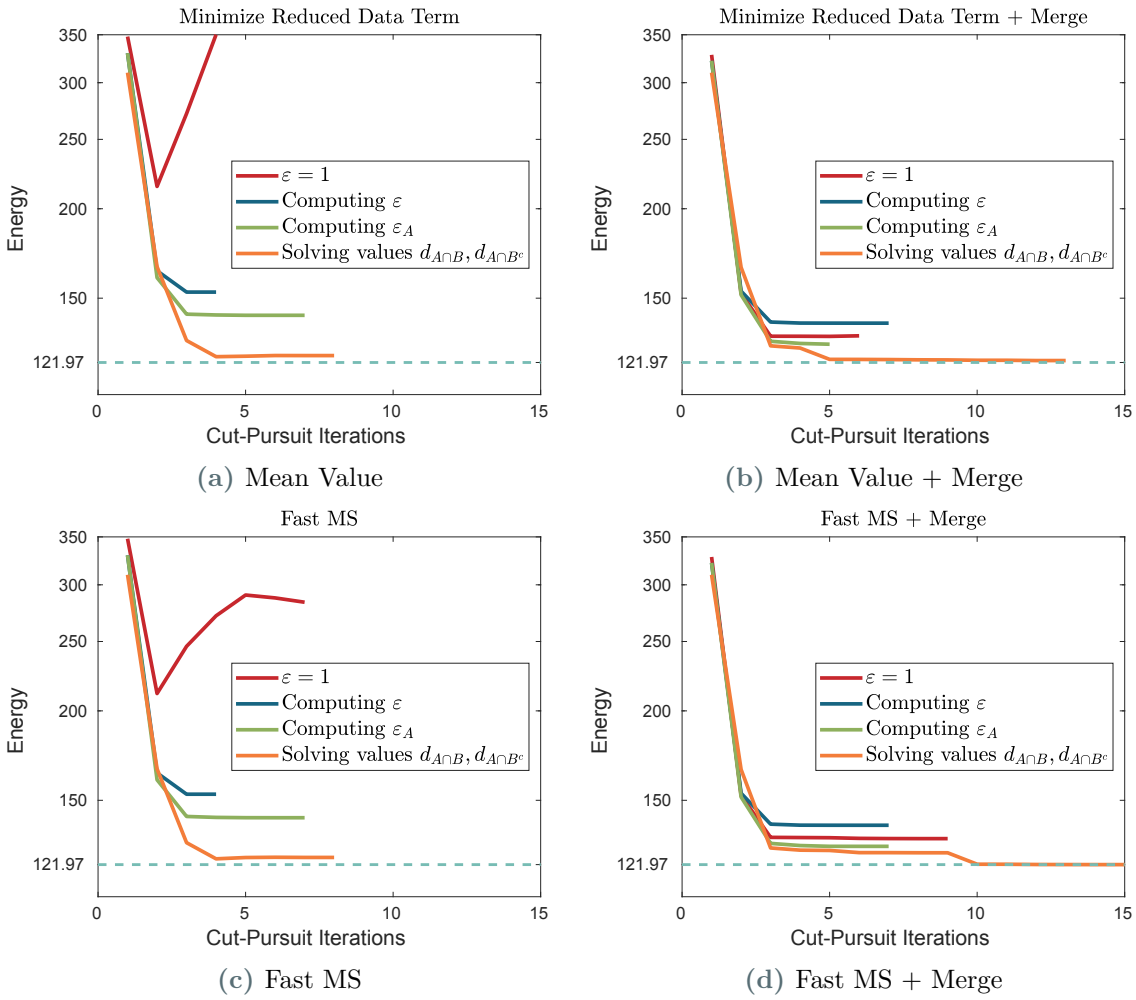
7.2 Different Partition Optimization Strategies

Selecting the value of ε by hand or heuristically might not yield satisfying results and would have to be selected once again for every data set and graph. It also is not very practical to test multiple such step sizes. To tackle this we introduce three different alternating optimization strategies to solve the problem of selecting the feasible step size ε in Section 5.1.1.

The first strategy is to optimize over the values $d_{A\cap B}, d_{A\cap B^c}$ directly for each segment $A \in \Pi$ as in (5.43), the second strategy is to compute a direction $\bar{\gamma}^A$ for every segment $A \in \Pi$ optimize for a corresponding step size ε_A as in (5.46) and the third one is to optimize one fixed $\varepsilon > 0$ for every segment A and the corresponding directions $\bar{\gamma}^A$ all together as in (5.49). Each of these strategies is an alternating algorithm to find optimal values for a corresponding cut and an optimal cut for the corresponding values. In Figure 7.3b we plot the energy for regularization parameter $\alpha = 0.01$ in each iteration for the different strategies. We compare them to the manual choices of $\varepsilon = 1$ with merge and $\varepsilon = 0.026$ as we have deduced from Figure 7.2 as a convincing manual value with and without merging. We, additionally, show the energy plots of the three described strategies with a merging strategy to solve the reduced problem. From this figure we can deduce that the near optimal constant ε^*, as we have derived before, yields a convincing, monotonic decreasing energy plot, but cannot compete against applying the merging strategies. Nevertheless, we see that even selecting $\varepsilon = 1$ and merging yields an energy even better than ε^* with merging. This is also what we see in Figure 7.2b where we plot results for fixed epsilons and merging on the reduced graph. The three strategies from Section 5.1.1 yield better solutions analogously to their degree of freedom. This means that the updating multiple step sizes is better than optimizing over one and computing a value for each new set is better than multiple step sizes. Another interesting observation is that, if we alternatingly try to find the best partition we need less steps in the outer iteration until convergence compared to the other methods. In practice computing a solution by optimizing the values as in (5.43) and merging afterwards yields the best solutions.

Even though these observations suggest applying the more elaborated strategies, we can see in Table 7.1 for the *cameraman* image and Table 7.2 for the RGB image *peppers* that the strategies with inner updates are in most cases significantly more costly. This comes from the multiple graph cuts that have to be computed for every inner iteration step. The computational time depends on the structure of the problem and of the graph. Still, these minimization strategies are convincing algorithms of choice to find a solution with low energy. If one is only interested in a good approximation of the underlying data by piecewise constant parts it might be sufficient to use one of the more simple strategies, due to their faster runtime. This is what we investigate in Section 7.4 where we show how to use the most simple strategy of the former ones to compute a coarse representation of the data set and work on the with this induced discretization.

Finally let us compare the energy curves of all of the different strategies we have proposed above. In Figure 7.4 we see the four plots of the three proposed strategies for solving the reduced problem (5.23) plus solving the reduced problem with our fast MS implementation on graphs from (5.6) applying a merging afterwards. Merging is applied as given in Algorithm 5.



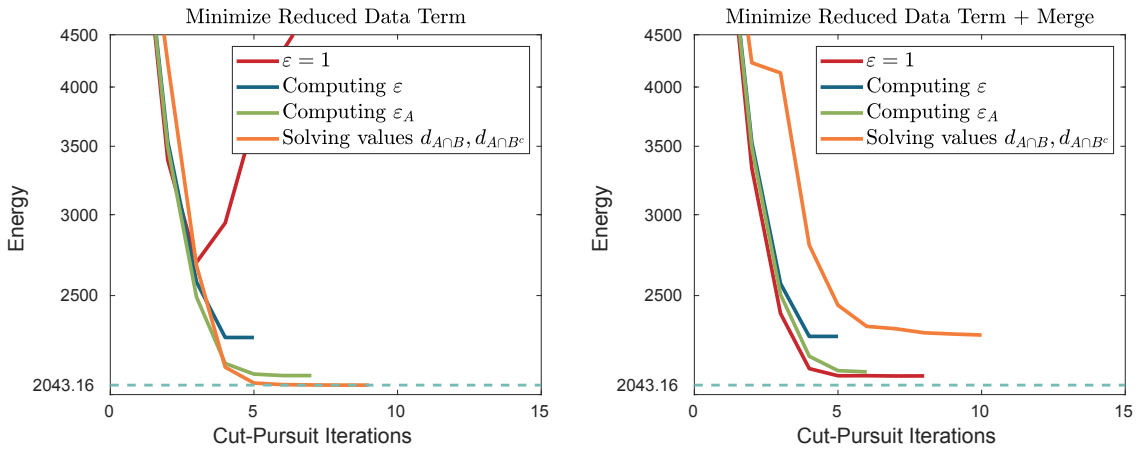
(a) Mean Value (b) Mean Value + Merge

(c) Fast MS (d) Fast MS + Merge

Figure 7.4.: Comparing energy curves of different reduced problem solving methods and the corresponding used partition problem method. The dashed line is the lowest energy found by these methods. These plots correspond to the *cameraman* grayscale image for a regularization parameter $\alpha = 0.01$.

From top-left to bottom-right it is solving the reduced problem by minimization of the reduced data-term as in (5.55), merging after minimization, computing the reduced problem more accurately by using the fast MS primal-dual algorithm proposed in [83] which graph version we state in (5.28), and last by merging after applying fast MS. Note that we do a simple merge step after computing the solution of fast MS that just merges segments that have the same value. This does not change the energy of the problem and just gives the partition problem the chance to find a cut that yields a better energy decrease.

For each of these methods we plot the energy curves for solving the partition problem. We see that without merging we have to solve the partition problem more accurately. If we merge after solving the reduced problem the energy always decreases and we have a convergence to a local optimum, since the algorithm stops if it cannot find a new cut. Nevertheless, minimizing over the values $d_{A\cap B}, d_{A\cap B^c}$ in the partition problem yields the best results for grayscale images. Combined with merging, this is the best algorithm depending on the energy value. In practice we see similar behaviors for different data sets and data sets types. Merging is always the best practice.



(a) Mean Value (b) Mean Value + Merge

Figure 7.5.: Comparing energy plots of computing the mean value as the solution of the reduced problem with and without merging and the corresponding used partition problem method. The dashed line is the energy of the best solution in terms of energy value provided by the different methods. These plots correspond to the *peppers* RGB image for a regularization parameter $\alpha = 0.1$. The direction used where given by the first principal component of the data in each segment.

	Mean Value	Mean Value + Merge	Fast MS	Fast MS + Merge
Fix $\varepsilon = 1$	2.37s	13.41	49.87s	107.70s
Optimizing ε	7.42s	7.74s	12.29s	13.15s
Optimizing ε_A	13.03s	28.24s	38.18s	31.35s
Optimizing $d_{A\cap B}, d_{A\cap B^c}$	5.47s	6.93s	21.76s	46.69s

Table 7.1.: Run time comparison of the different algorithms presented in Figure 7.4 for the same *cameraman* image.

However, for RGB data sets for example we can sometimes see that minimizing the values in the partition problem instead of using a pre-selected direction can lead to worse energies. Looking at Figure 7.5 we can see the same plots as before, but in this case for the RGB image *peppers*. For this we use the isotropic L_0-Cut-Pursuit where we always use the first principal component of a PCA as a direction for the partition problem as we did in Section 6.3. We can observe that in this case solving the values without predefined directions in the partition problem and merging afterwards does not lead to the best energy. This might come from the high non-convexity of the problem we are trying to solve here. Decreasing the energy by solving the partition problem as much as possible does not necessary lead to the best local optimum in the end. Hence, selecting a direction for the partition problem that it chosen with some rationale to fit to the data sometimes yields better results in terms of energy. This highly depends on application purposes. However, in this case wee see that still computing the values instead of searching step sizes for the PC1 directions wins when we are not using merging.

	Mean Value	Mean Value + Merge	Fast MS	Fast MS + Merge
Fix $\varepsilon = 1$	20.80s	23.56	18.06s	14.83s
Optimizing ε	7.42s	7.74s	12.29s	13.15s
Optimizing ε_A	84.77s	65.90s	111.82s	85.69s
Optimizing $d_{A\cap B}, d_{A\cap B^c}$	32.75s	21.49s	34.40s	30.98s

Table 7.2.: Run time comparison of the different algorithms presented in Figure 7.5 for the same *peppers* image.

7.3 Comparison to Other Algorithms

To have a better intuition of how well this algorithm competes against the Cut-Pursuit algorithm proposed by Landrieu et. al. in [47, 46] and the fast Mumford-Shah (MS) algorithm from [83] we aim to compare the energy values, total run time and sparseness for multiple regularization parameters. For these experiments we used the open source code provided from the authors of [47] and [83]. The algorithm from [47] that also performs a Cut-Pursuit algorithm is implemented in C++ and provided with a MATLAB wrapper. Hence, in general it might be faster than our implementation in MATLAB. Their implementation gives access to different variants of how to solve the minimal partition problem. We choose the slowest but most accurate strategy. For the algorithm [83] they also used a C++ implementation with a provided MATLAB wrapper. We set the number of iteration to a maximum of $20,000$ iterations. The algorithm stops if the total change in every pixel is smaller than 10^{-5}. We applied the algorithms on the noise-free *cameraman* grayscale image and on the noise-free *peppers* RGB image.

In Table 7.3 and Table 7.4 we compare the three algorithms for four different regularization parameters and present the final energy value of the results, the total amount of runtime, and the number of segments for both Cut-Pursuit algorithms. According to the plots in Figure 7.4b and Figure 7.5b, we use Algorithm 7 by solving the partition problem by alternatingly computing a cut and optimizing the values $d_{A\cap B}, d_{A\cap B^c}$ as they yield the

α	0.001	0.01	0.1	0.5	
L_0-Cut-Pursuit Algorithm 7	32.31	124.78	436.38	921.54	Energy
	7.5	5.55	4.93	1.79	Runtime (in s)
	5,158	777	34	5	Number Of Segments
Cut-Pursuit by Landrieu [46, 47]	34.02	135.54	487.19	1,070.01	Energy
	6.59	5.68	3.92	2.48	Runtime (in s)
	2,374	202	11	3	Number Of Segments
Fast MS [83]	42.84	170.28	523.88	1,093.1	Energy
	2.67	5.78	10.84	9.65	Runtime (in s)
	-	-	-	-	Number Of Segments

Table 7.3.: Energy and run time comparison of the proposed L_0-Cut-Pursuit algorithm in Algorithm 7, the Cut-Pursuit proposed by [47] and the fast Mumford-Shah primal-dual algorithm proposed by [83] for different regularization strengths. This was applied to the grayscale *cameraman* image.

α	0.001	0.01	0.1	0.5	
L_0-Cut-Pursuit Algorithm 7	331.20	1,450.00	4,340.5	8,800.7	Energy
	34.87	37.76	34.05	37.81	Runtime (in s)
	60,201	5,911	268	34	Number Of Segments
Cut-Pursuit by Landrieu [46, 47]	348.20	1,512.6	4,640.1	9,381.8	Energy
	45.17	35.78	42.07	38.35	Runtime (in s)
	26,712	1,571	101	10	Number Of Segments
Fast MS [83]	433.85	2,094.5	5,370.2	10,650.12	Energy
	13.47	57.48	84.52	166.48	Runtime (in s)
	-	-	-	-	Number Of Segments

Table 7.4.: Energy and run time comparison of the proposed L_0-Cut-Pursuit algorithm in Algorithm 7, the Cut-Pursuit proposed by [47] and the fast Mumford-Shah primal-dual algorithm proposed by [83] for different regularization strengths. This was applied to the RGB image *peppers*.

solution with the energy value. For the grayscale image we solve the reduced problem with merging while in the cases of the RGB image we solve by mean value without merging. As we can see in both tables our proposed method can keep up in run time with the Cut-Pursuit by Landrieu et. al.. The energy is better in most cases, but interestingly we produce more segments in the end. This might come from the multiple stages of merging and splitting in the Cut-Pursuit algorithm from [47] and that they try to keep the number of segments sufficient small. Both Cut-Pursuit algorithms surpass the Fast MS algorithm in terms of final energy values. This might come from the fact that the Cut-Pursuit algorithms start with coarse approximation and become finer with every iteration, and thus - also due to the merging steps - tend to stop in a very coarse local minimum which has a tendency for a lower energy. A direct method like Fast MS comes from the pixel space and might tend to converge into a local minimum with more piecewise constant segments, and thus, worse energy.

Obviously, comparing these algorithms by run time is not very meaningful, since they have completely different implementations and will run differently on different hardware setups. Nevertheless, we can get a better intuition on how well which algorithm performs and that Cut-Pursuit in general is a good idea to tackle such problems to find a solution with a good energy value.

In the next section we also show that the simplest L_0-Cut-Pursuit algorithm yields a satisfying oversegmentation, and hence, a discretization of a given data set.

7.4 Discretization via Minimal Partitions

In this section, we want to emphasize the capability of the L_0-Cut-Pursuit algorithm to find a coarse representation for any given d-dimensional data and the corresponding graph structure. This property of the L_0-Cut-Pursuit algorithm is already pointed out and used in the work of Landrieu et. al. in [48]. L_0-Cut-Pursuit does not only provide a solution to an optimization problem it also has the corresponding partition and reduced graph representing the solution as a by-product. In [48] they make usage of this property by applying L_0-Cut-Pursuit as a preprocessing step and then apply their semantic point cloud segmentation

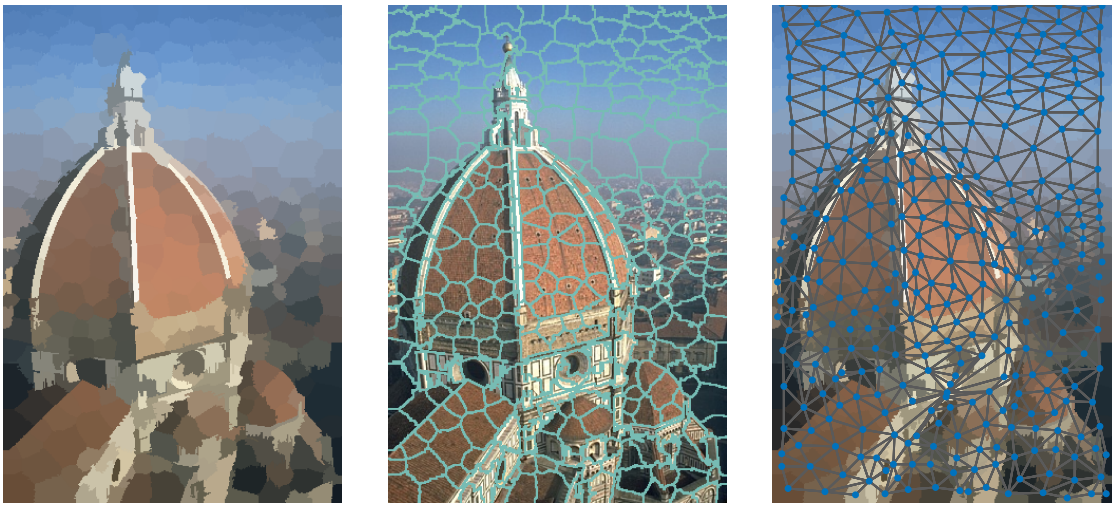


Figure 7.6.: SLIC result for 400 superpixels. This is only 0.26% of the pixels.

algorithms on the reduced so called *superpoint graph*. This is in fact a coarse discretization - or sampling - of the original vertex space and graph corresponding to the structural features of the given data. This discretization also is called *oversegmentation* and the segments are called *superpixels*.

In this section, we denote the L_0-Cut-Pursuit by Cut-Pursuit when not stated otherwise and use the term L_1-Cut-Pursuit for the Cut-Pursuit algorithm solving ROF problems. In the following, we want to investigate the usefulness of the proposed L_0-Cut-Pursuit algorithm in Algorithm 6 as a tool for discretization in its simplest variant where we set the step size constant as $\varepsilon = 1$. This is the most efficient strategy, since it does not need any inner iterations to optimize the produced partitions and does not solve the reduced problem with any complex algorithm. The only strategy we apply in some experiments - since it is very fast in most cases - is to merging strategy when solving the reduced problem. Note that this section is in fact an extensive numerical analysis of the ideas in [31]. Hence, from now on we only use the Cut-Pursuit as an oversegmentation method in its simplest variant. Note that most experiments and claims in the following subsections can be directly applied to other L_0-Cut-Pursuit strategies. But we only focus on the simplest one, as it is the fastest method, and, additionally, shows that this method is a sufficient choice for oversegmentation. In addition, note that we always use the first principal component (PC1) for the directions in the isotropic cases when not stated otherwise.

First we want to introduce two well-known sampling methods which we compare to the proposed approach namely the Simple Linear Iterative Clustering (SLIC) superpixel method [1] and a trivial sampling into squares. Then we analyse numerically by considering experiments if and how the Cut-Pursuit method in its simplest variant can be interpreted as a superpixel method like SLIC. Thereafter we compare the three methods amongst themselves for different properties. Then we show that SLIC does yield unsatisfying results on noisy images and that Cut-Pursuit still produces sufficient discretizations. And finally we investigate how we can suppress artifacts in the superpixel results of Cut-Pursuit using the provided reduced graph of the solution and an ROF problem.

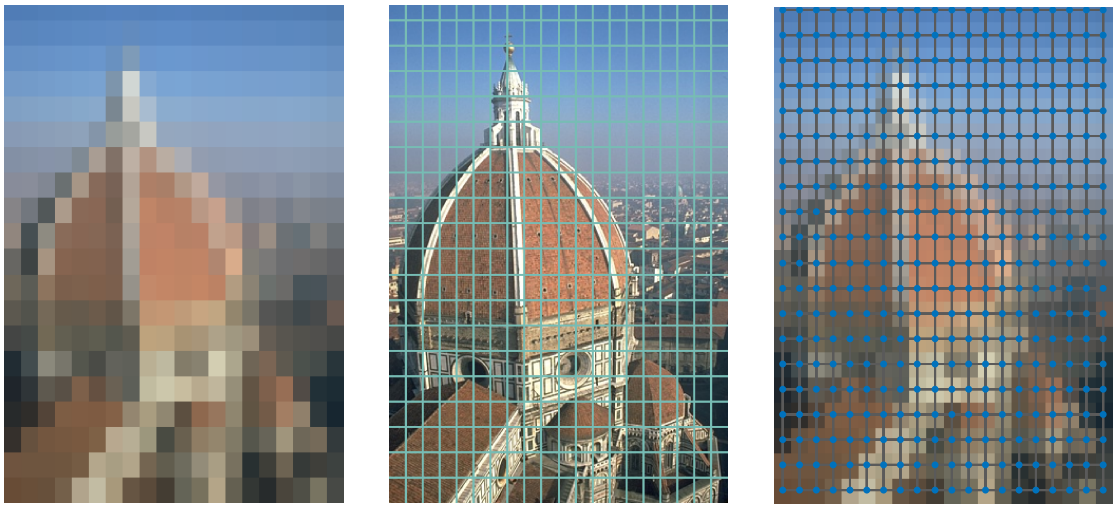


Figure 7.7.: Sampling result with 400 superpixels.

7.4.1 Related Superpixel Methods

In this subsection, we very shortly and superficially introduce the superpixel method SLIC [1] that is a well-known and widely used fast oversegmentation method. It is mostly used to reduce complexity of data to speed up analysis and methods while keeping as much information as needed. This method takes in a certain grayscale or RGB image and a desired number of clusters and outputs the partition of the image into these clusters. In Figure 7.6 we can see the SLIC method applied to the *florence cathedral* image where the outlines describe the superpixels boundaries. With this information we can directly build the reduced graph corresponding to this oversegmentation we also visualize in that figure. This is done by gathering the edges that are on the boundaries between the sets and sum up the weights there, which is in this case the boundary length between two segments. The reduced data is the mean value of the data over each superpixel. Then we can apply any reduced variational problem on graphs on this reduced setup. In this work we use the built-in MATLAB function *superpixels*. This function provides a local "compactness" variable that balances between the resulting superpixels being regular - which means very square - or the superpixels adhere to the boundaries in the image. We chose this variable as the default one in all our experiments which is just in the middle of the range of possible values. Looking at Figure 7.6, we can see that the superpixels can describe the overall structure of the image very well and we can still perceive the depicted cathedral. However, this method seems to fail at boundaries. In some cases the superpixels overlap boundaries in the image and in some cases the boundaries are not very exact and have a frayed structure. Additionally, SLIC samples the image very homogenously in some sense, and thus, samples independently on the underlying details. This means that where many fine details are it samples the superpixels just as in region with less detail. This can be seen in Figure 7.6 where the sky with less details is nearly as homogenously sampled as the finer structures on the cathedral. These problems are a huge drawback since boundaries and structural details in the image are an important information for all kinds of method, e.g. semantics, segmentation and denoising. Consequently, the user

always has to make a decision to have less superixels and therewith less fine information from the image or more finer structures, and hence, more superpixels.

Additionally to this method we want to use the most trivial discretization method that one can also understand as a superpixel method. This method divides the pixel space into square segments and is directly related to downsampling the data. In Figure 7.7 we show a downsampling of the cathedral of florence image and the corresponding reduced grid graph.

Both methods give us a coarse representation of the original image and graph which we interpret as a discretization of the original graph. In the next subsection we state why the L_0-Cut-Pursuit algorithm from Algorithm 6 in its simplest version is also a superpixel method and show the advantages of this method afterwards. Then we also compare all three methods.

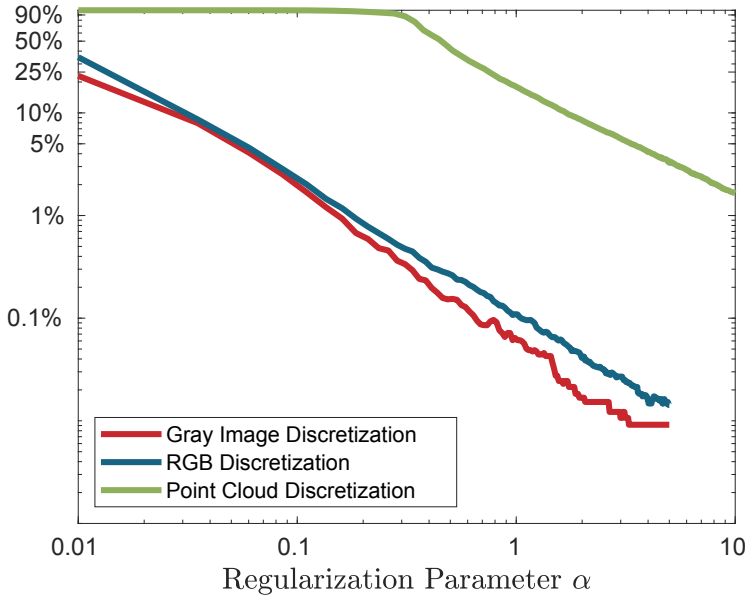


Figure 7.8.: L_0-Cut-Pursuit for three different data types as a grayscale image, a RGB image and a 3D point cloud. The regularization parameters are plotted against the discretization percentage. The y-axis is a logarithmic axis.

7.4.2 Minimal Partition as a Superpixel Method

In this subsection, we want to show that the L_0-Cut-Pursuit algorithm we state in Algorithm 6 in its easiest variant - where we do not update the partition problem, set $\varepsilon = 1$ and compute the mean value on the reduced graph - also yields a superpixel representation for a given graph G and some data g. In the beginning of Section 7.4 we argue that we actually can solve for a fixed $\varepsilon = 1$ and some regularization α, since the ε only scales the regularization in this easy case. We assume that we do not solve the problem for the given α, but rather for a different problem. This implies that we still have coarser results for higher regularization and finer results for lower regularization which is exactly the intuition as before. To prove this numerically we did the test for a grayscale image, an RGB image and a 3D point cloud for hundreds of different regularization parameters. Then we plot the parameter against the resulting discretization percentage which is the number of segments divided by the number of nodes in the vertex set. We present these results in Figure 7.8 where we see that the intuition of the regularization parameter holds. Hence, we have a heuristic for applying this simple Cut-Pursuit algorithm and use the provided reduced graphs as a discretization.

As we now have got an intuition for the regularization parameter, let us investigate the results on a certain example image. In Figure 7.9 we see the result for the regularization parameter $\alpha = 0.25$ which yields a reduced result with only $2,311$ superpixels which are only about 1.5% of the original pixels. We can see that the structural information of the image are preserved and the finer detailed regions like for example the walls of the cathedral have a detailed sampling and the sky for example is sampled with only a few segments describing the color gradient. This becomes even clearer when looking at the right image of Figure 7.9 where we see the reduced graph that has a accumulation of many nodes in the regions with

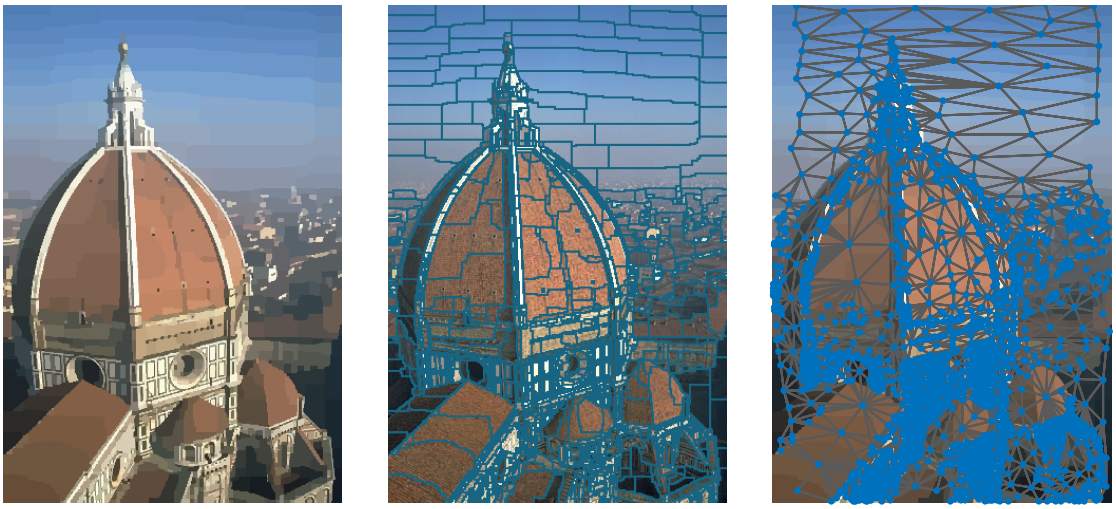
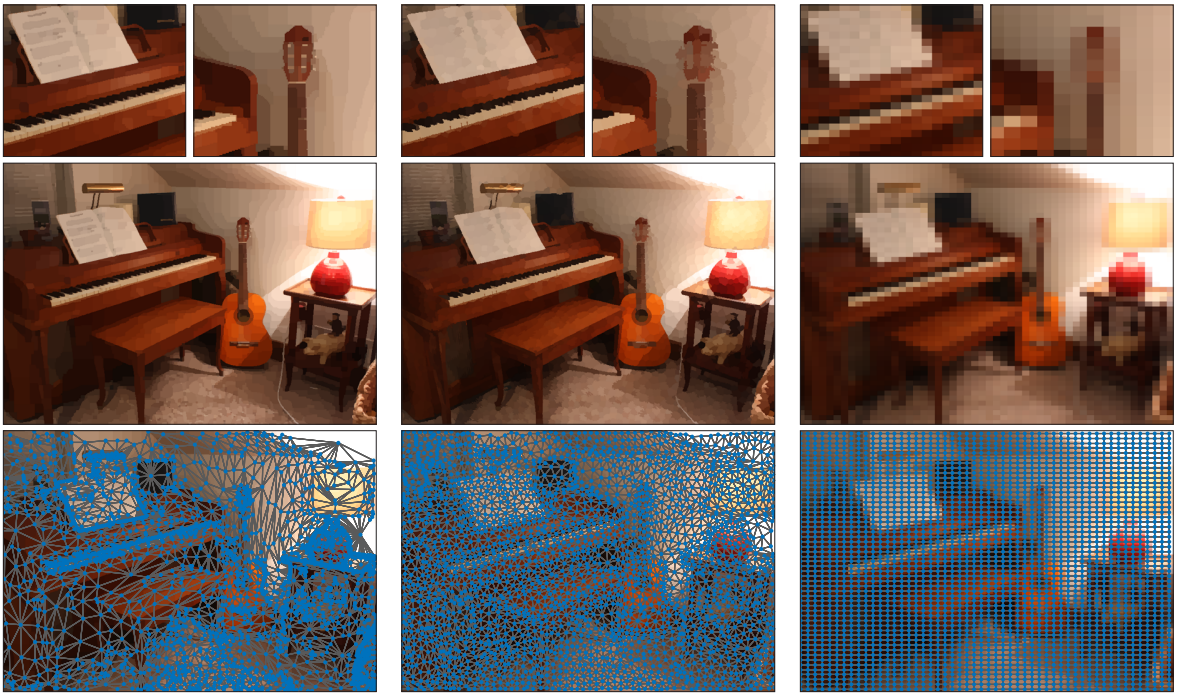


Figure 7.9.: L_0-Cut-Pursuit result with $\alpha = 0.25$. The image consists of $2,311$ superpixels. This is only 1.5% of the total pixels. Discretization on the left, outlined segments in the middle and reduced graph on the right.

more details and and is very sparse in less details regions. As we have said before, we can also apply the merging step from Algorithm 5 to solve the reduced problem. In Figure 7.10 we see the solution of the L_0-Cut-Pursuit algorithm with a merging step for solving the reduced problem. Due to the merging we have even less nodes in the reduced graph and the segments are even larger. Nevertheless, it still preserves the fine details in the cathedral wall and even the details on the roof. In summary, we can say that this method is a oversegmentation method that has the ability to preserve fine and coarse structural details in the image while reducing the amount of nodes in the corresponding graph significantly. The only downside of this algorithm is that it tend to have spatially-anisotropic solutions, since we can only cover spatially-anisotropic regularization. Still, the results seem to not suffer too much visually from

Figure 7.10.: L_0-Cut-Pursuit with merging result with $\alpha = 0.04$. The image consists of $1,045$ superpixels. This is only 0.67% of all pixels. Discretization on the left, outlined segments in the middle and reduced graph on the right.



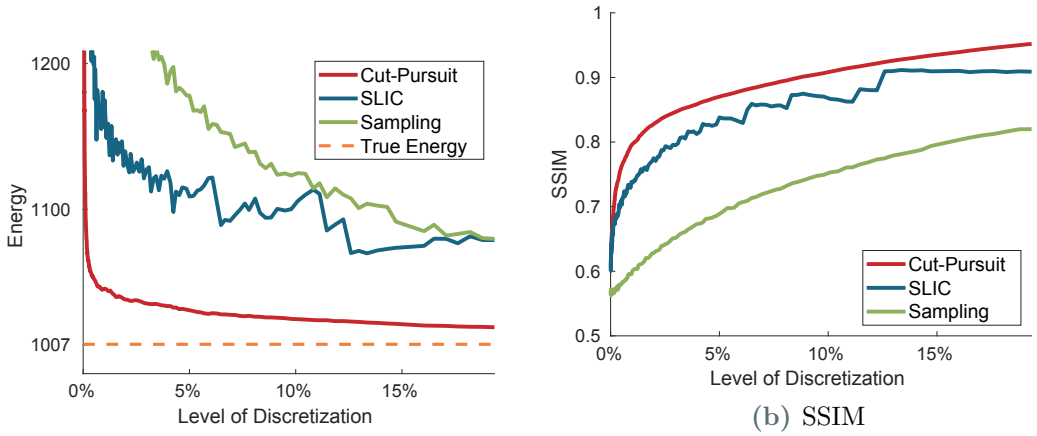
(a) Cut-Pursuit (b) SLIC (c) Sampling

Figure 7.11.: Comparing the reduced graphs. Cut-Pursuit solution was computed with $\alpha = 0.2$ which yields $2,724$. Then we used this number of segments to compute the oversegmentations by SLIC and a uniform sampling. In the top row are the oversegmented results and in the bottom row the corresponding reduced graphs.

this problem. In the next subsection we compare this method with SLIC and a downsampling method.

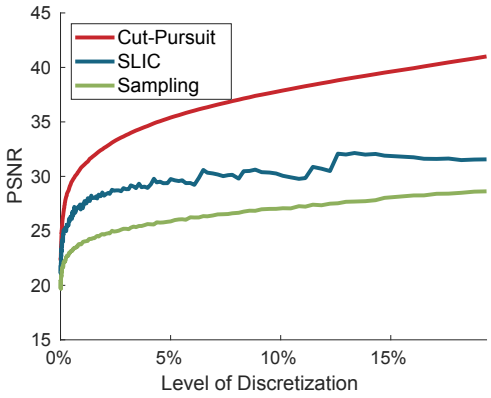
7.4.3 Compare Superpixel Methods

In this subsection, we compare discretization results using L_0-Cut-Pursuit , SLIC and downsampling. As we show in the two former subsections, using L_0-Cut-Pursuit does provide us with a very detail and structure sensitive oversegmentation while SLIC is more and can overlap over boundaries in the image. Hence, we would assume that the Cut-Pursuit algorithm does a better job in preserving the overall structure of the image, while keeping the number of segments low. To validate this assumptions, we show visual comparisons and also three different measures. The first one is the *Structural Similarity (SSIM)* index that tries to measure the difference between some image and a reference image as a perception based difference (cf. [88, 40]). We apply this to the RGB images using the built-in MATLAB *rgb2gray* function that generate grayscale images from an RGB image. The second one is the *Peak Signal To Noise Ratio (PSNR)* which is the ratio between the maximum pixel value in the reference image and the *mean squared error* of the reference image and the comparison image (cf. [40]). Additionally to these image quality indices, we also want to analyse the capability of the discretization to yield correct results when applying some algorithm to the reduced graph. To check this we want to solve a reduced ROF problem for some regularizer



(a) Energy

(b) SSIM



(c) PSNR

Figure 7.12.: Different discretization levels and corresponding measures for Cut-Pursuit, SLIC and sampling discretization. The discretization level is the ratio between the current number of sets and total amount of pixels in the image. Energy corresponds to an ROF problem applied to the corresponding reduced graph and data set with regularization parameter $\alpha = 0.3$. The dashed line in the upper left plot is the correct energy by solving it with primal-dual on the original graph.

α. The closer the energy of the solution on the reduced graph is to the solution on the original graph the better the discretization does preserve the important structural information.

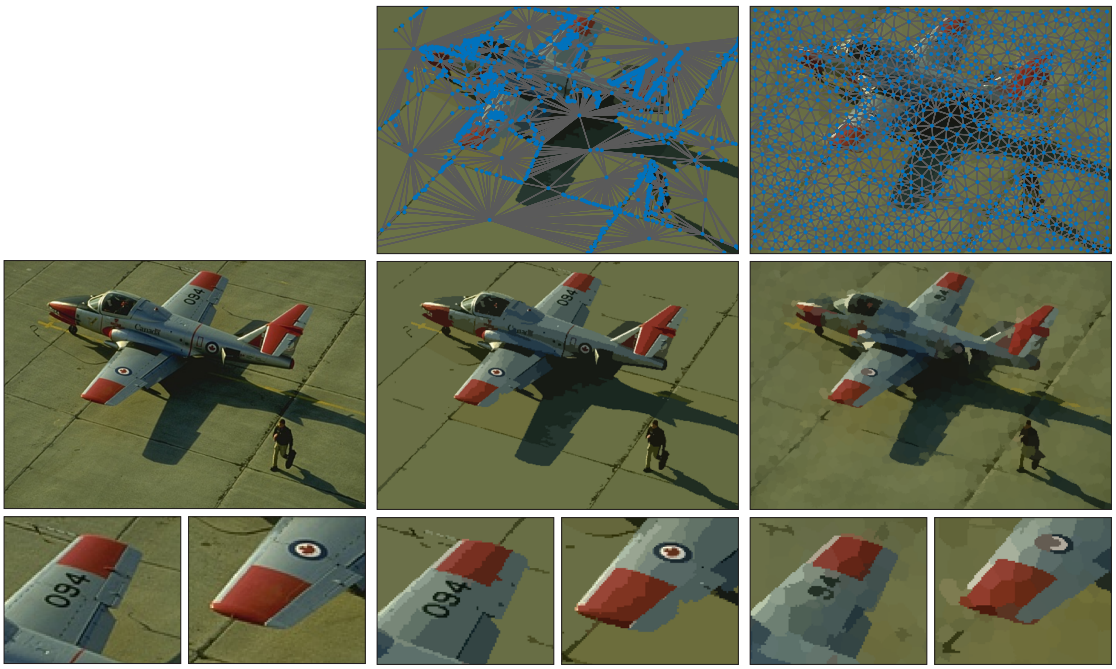
To compare the three methods we run an experiment where we apply Cut-Pursuit to the image of a plane in Figure 7.14a for a sequence of regularization parameters. For every regularization parameter we get a reduced graph and a certain oversegmentation of the image. Then we compute for every regularization parameter a SLIC oversegmentation and a downsampling with the total number of segments provided by the solution of the Cut-Pursuit algorithm. Furthermore, we compute the corresponding reduced graphs for SLIC and downsampling. Then for each regularization parameter we computed the SSIM and PSNR indices for each of the three solutions. This is what we plot in Figure 7.12b for the SSIM index and in Figure 7.12c for the PSNR index. Also we compute the solution of a reduced ROF problem (4.67) for a given regularization parameter $\alpha = 0.3$ for every reduced graph and plot

(a) Original (b) Cut-Pursuit (c) SLIC

Figure 7.13.: Compare Cut-Pursuit discretization with $\alpha = 0.2$ to SLIC oversegmentation with the same amount of superpixels. The number of superpixels is $3,726$ (2.41%). In the top row we visualize the corresponding reduced graphs.

the energies of the results in Figure 7.12a. As the x-axis we use the level of discretization, which is the ratio between number of segments and the total amount of pixels in the original image. We can deduce two important properties from these plots. First, we see that for every level of discretization the Cut-Pursuit algorithm dominates both methods. Unsurprisingly, the Cut-Pursuit especially is better in the resulting energy of the reduced ROF problem. This is somehow obvious, but important to notice, since L_0-TV does emphasize structural boundaries in the image and so does TV. But this also once again shows that SLIC superpixels tend to distort the boundaries. We can even deduce by the volatile plot of the SLIC algorithm in Figure 7.12a that one cannot expect to track more structural information with SLIC by just increasing the number of desired superpixels. This can be also seen in Figure 7.12b and Figure 7.12c. This brings us also to our second interesting observation that the number of superpixels provided by Cut-Pursuit increases monotonically and smooth along with the level of discretization. This makes handling the algorithm very intuitive and easy since one can expect the resulting discretization to become finer, and as well, increase the amount of structural information using a smaller parameter α. On the other hand, increasing the parameter α yields a coarser discretization which drops unimportant fine information first before dropping larger segments. This also dominates SLIC that does rather aimlessly decrease the number of superpixels.

After reviewing the results in terms of mathematical measures, let us take a look at visual results. In Figure 7.11 we visualize the results of the Cut-Pursuit algorithm, SLIC and the sampling with the same amount of superpixels applied to an RGB image. We also provide the



(a) Original (b) Cut-Pursuit (c) SLIC

Figure 7.14.: Compare Cut-Pursuit discretization with $\alpha = 0.2$ to SLIC oversegmentation with the same amount of superpixels. The number of superpixels is $1,394$ (0.9%). In the top row we visualize the corresponding reduced graphs.

reduced graphs in the bottom row. Obviously, we get the worst results for just downsampling the given image as one loses many structural details. On first sight, the Cut-Pursuit and SLIC oversegmentation yield good approximations of the overall structure of the image. Nevertheless, when we compare the reduced graphs we see as expected that the reduced graph of the SLIC oversegmentation is homogenously sampled all over the image compared to the Cut-Pursuit superpixels that have different sizes depending on the underlying level of detail. To see this, we provide two zooms in the upper row of the keyboard of the piano and the head of the guitar. As we see, the general structure is preserved, but in the Cut-Pursuit results we can see the keys of the piano in clearer detail and even see that there is text on the music sheet which nearly completely vanishes in the SLIC result. In the magnification image of the guitar head we see that the spaces in the head are more perceptible in the Cut-Pursuit then in the SLIC result where it is very blurry and the details are not distinguishable. Additionally, we can see that in the Cut-Pursuit result the color gradient generated by the light of the lamp is segmented in vertical, successive, elongated superpixels which depict the gradient in a sparse sense. The SLIC again samples this area with much more superpixels and produces this honeycomb-like structure. Last we also want to point to the carpet where the Cut-Pursuit again samples the irregular structure of the carpet very well.

In Figure 7.13 and Figure 7.14 we again depict the results for Cut-Pursuit compared to SLIC but this time on the RGB image of a plane. The interesting property of this data is that it has many large same-colored segments like the gras or the shadow and also important details like the number or the symbol on the wings of the plane. Hence, a good approximation

of this data would be to somehow sample the patches of grass and the shadows very coarse while preserving enough details on the symbols. We applied the Cut-Pursuit algorithm with a parameter $\alpha = 0.2$ which yields a solution with $3,726$ superpixels which is about 2.41% of the original data size. When we compare the magnified images in Figure 7.13b and Figure 7.13c we can see that the number on the wing is preserved nearly perfect in the Cut-Pursuit result where in the SLIC result the number is not distinguishable anymore. We also observe this when we examine the other symbols and logos. Additionally, we see that the boundaries between the grass patches are preserved very well while they are blurred out in the SLIC result. The last but very satisfying observation is that the illumination of the person in the right corner is preserved very well whereas on the SLIC result the person seems to blend with the grass.

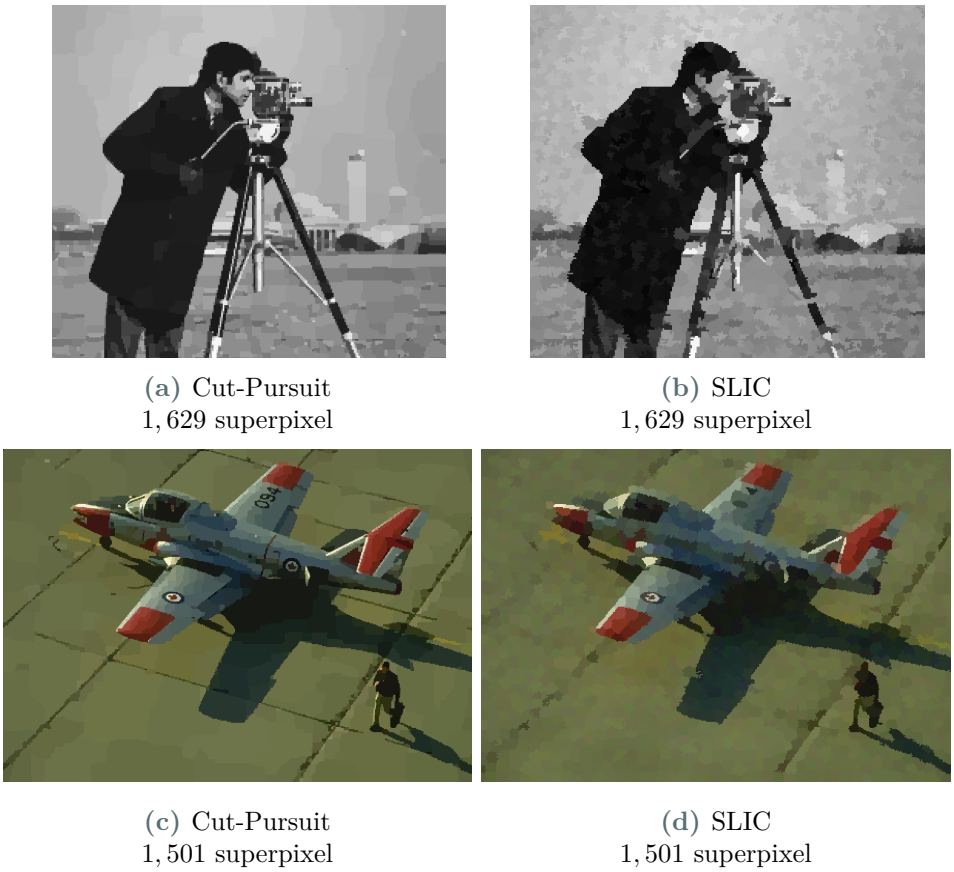
As a last experiment, we want to show the result of Cut-Pursuit if we also enable a merging to solve the reduced problem. This can be seen in Figure 7.14, where we computed the discretization for the parameter $\alpha = 0.2$ which yields $1,394$ superpixels which is a ratio of 0.9% of the original number of pixels. Hence, this is even a smaller amount of superpixels than in Figure 7.13. As we can see in Figure 7.14b, the logo and the number are still visible on the plane and the grass patches are preserved very well. In the reduced graph we can see that only smaller structures like the boundaries of the grass patches or parts of the person in the corner are sampled in detail. Nevertheless, the image has a rather unnatural look for humans, but using this as the reduced graph for any computer vision method, e.g., semantics or image recognition, should yield a sufficient discretization. Since this solution yields an smaller amount of superpixels, the SLIC algorithm cannot track the logos and numbers anymore and even loses the 0 of the number on the wing. Hence, the Cut-Pursuit algorithm is in terms of discretization superior.

As we have seen in this subsection Cut-Pursuit has satisfying discretization properties as it preservers fine details and accumulates areas with less detail. It also provides an intuitive parameter handling to dial in satisfying results, and thus, is a powerful superpixels method to generate a structural meaningful discretization.

7.4.4 Solving Problems on Discretization

In this subsection we want to investigate how noise does corrupt the discretization by the former proposed methods. We compare the SLIC and the Cut-Pursuit superpixel results for noisy images and their corresponding graph structures. Then we take these graphs and solve an ROF problem on it. To this end, we compare the energy of the solution computed on the discretized graphs with the energy of the solution computed on the original graph. Finally, we investigate if computing a discretization with Cut-Pursuit and then applying the reduced ROF is computational more efficient as applying the solving the ROF problem on the full graph.

Let us start with a noisy grayscale image of the cameraman Figure 7.17a and a noisy RGB image of the plane Figure 7.18a and show in Figure 7.15 the discretization by Cut-Pursuit and SLIC. As we see, the Cut-Pursuit can handle the noise pretty well and has a denoising effect. On the other hand, the SLIC oversegmentation really struggles with the noise. It does not only have the problems of overlapping boundaries in the image and representing



(a) Cut-Pursuit
1, 629 superpixel

(b) SLIC
1, 629 superpixel

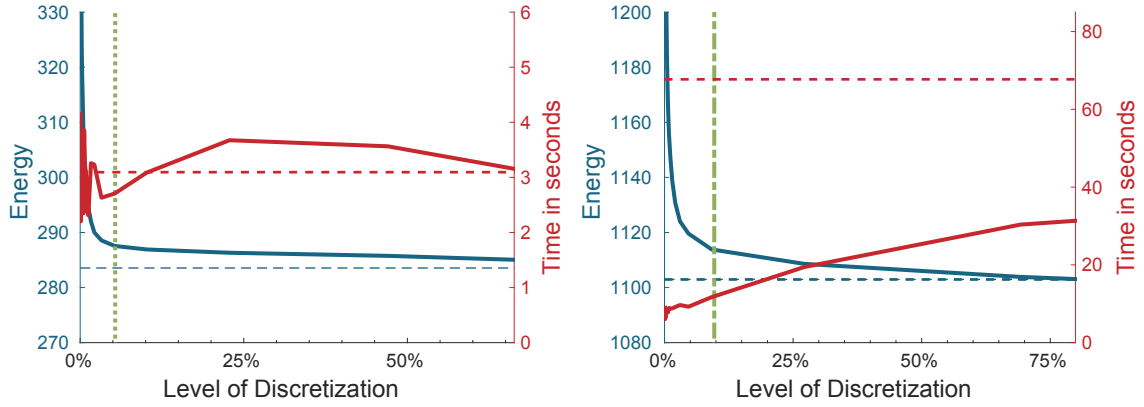
(c) Cut-Pursuit
1, 501 superpixel

(d) SLIC
1, 501 superpixel

Figure 7.15.: Discretization via Cut-Pursuit and SLIC with the same number of sets for a noisy input image.

the boundaries rather inexact, but also generates totally fuzzy and jagged boundaries of the superpixels. This makes the SLIC method very unreliable in this setup and not feasible for using the reduced graph provided by the oversegmentation in a noisy setting. Hence, we focus on Cut-Pursuit as our discretization method of choice.

From now on we use the Cut-Pursuit algorithm to produce discretizations for the images. As we point out in the former subsection, one can apply any image problem on the reduced graph and solve this on the very sparse discretization of the full graph. This yields a faster computational time on the reduced graph which can be very efficient as was shown in [31] where where they solve very inefficient lifting problems very efficiently on discretizations of this kind. We show this by solving an ROF problem on the reduced graph and compare it to the full graph solution. For that reason, we use the Cut-Pursuit algorithm from Section 4.3 to solve TV denoising problems to denoise the images in Figure 7.17a and Figure 7.18a for a given regularization parameter. Then we use Cut-Pursuit for a discretization of the image for some parameter and get a reduced graph and reduced data. We then apply the primal-dual algorithm from Algorithm 3.24 for reduced graphs to the discretized graph to solve a reduced ROF problem - as stated in (4.67) - for the same regularization parameter α. In Figure 7.16a and Figure 7.16b we visualize these experiments where we evaluated Cut-Pursuit for different parameters to get different levels of discretizations and plot the corresponding energy for a given regularization parameter $\alpha = 0.1$. The dashed blue line is the energy level of the true



(a) Gray-value Image Cameraman (b) RGB Image Plane

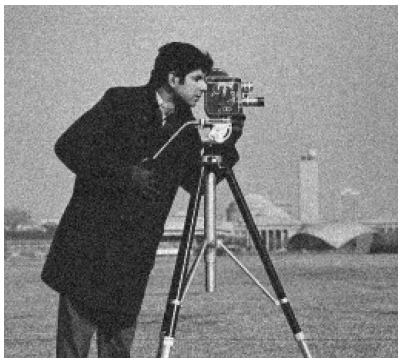
Figure 7.16.: The plots show the energy the results of the reduced ROF on the discretizations provided by Cut-Pursuit. The red lines show the time for running Cut-Pursuit plus solving the reduced problem afterwards The red dashed line is the run time of a L_1-Cut-Pursuit on the full graph and the blue dashed line the corresponding energy.

solution. We additionally took the time for computing the solution on the full graph with L_1-Cut-Pursuit which is visualized by the dashed red line and the time for discretization and solving on the reduced graph plotted as the red line. It also shows the discretization level of the L_1-Cut-Pursuit result by the green vertical line.

The energy plots show us what we already expected: as finer the discretization gets the better the energy of the reduced problem becomes and closer it gets to the energy for the solution of the full problem. For the grayscale image we see that the timing to compute the full problem and to compute discretization and reduced problem are nearly equivalent to some extent. For the RGB image the reduced approach surpasses the run time of the L_1-Cut-Pursuit algorithm. As we discuss in Chapter 4, the channel-isotropic setting is not too easy to solve for Cut-Pursuit , since we have to check for sufficient directions which costs multiple graph cuts for one outer iteration. This makes this algorithm on the discretization much more efficient.

We also provide the noisy images, the correct TV denoised solution and two approximations via discretization for the cameraman image in Figure 7.17 and for the RGB plane image in Figure 7.18. We see that even though the energies might not be exactly equivalent to the energy of the solution of the full problem, but these solutions are visually very close. It is hard to find an obvious visual difference and without comparing it directly to the solution. Hence, it ultimately does fulfill a denoising purpose.

To conclude this section, we want to talk about how good L_0-Cut-Pursuit can handle high noise levels, what kind of artifacts we expect and how we can handle these.

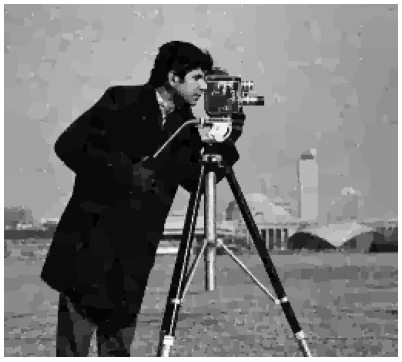


(a) Noisy image
65, 536 pixels

(b) TV solution on the full image
Energy: 283.55.

(c) Discretization:
649 superpixel (0.99%).

(d) TV solution reduced graph
Energy: 297.84



(e) Discretization: 3, 450
segments (5.26%).

(f) TV solution reduced graph
segments. Energy: 287.55

Figure 7.17.: The discretization properties of the L_0-Cut-Pursuit approximation. It is tested on the an grayscale image of the cameraman with $65,536$ pixels and denoised by TV regularization with $\alpha = 0.1$. Note the discretization in (e) and (f) corresponds to the discretization level of the optimal solution marked with the yellow line in Figure 7.16a.

(a) Noisy image
154, 401 pixels

(b) TV solution on the full image
Energy: 1, 102.9

(c) Discretization:
1, 751 superpixel (1.13%).

(d) TV solution reduced graph
Energy: 1147.29

(e) Discretization: 14, 166
segments (9.17%).

(f) TV solution reduced graph
segments. Energy: 1, 113

Figure 7.18.: The discretization properties of the L_0-Cut-Pursuit approximation. It is tested on the an RGB image of the plane with 154, 401 pixels and denoised by TV regularization with $\alpha = 0.1$. Note the discretization in (e) and (f) corresponds to the discretization level of the optimal solution marked with the yellow line in Figure 7.16b.

7.4.5 Denoising Minimal Partitions

In this subsection, we want to investigate how the L_0-Cut-Pursuit discretization reacts to high levels of noise. With respect to the former subsection we also give a nice and easy way to handle occurring artifacts.

Let us first talk about what we would expect from a L_0-Cut-Pursuit oversegmentation in the presence of high levels of noise. To this end, we give a visualization of two different levels of noise in Figure 7.19. In the first row we see the noisy RGB images of a parrot. The second row visualizes L_0-Cut-Pursuit results for some regularization parameter. As we can see, the solution in face of the high noise value suffers from single, pointy noise pixels that are somehow cut from the piecewise constant segments. For the low level noise the oversegmentation gives good results without these outliers. These artifacts come from the property of the regularizer that any jump in the image or graph is equally penalized and thus the boundary length of the segments is measured. Hence, it might happen that it is much more expensive for the data-term to have a pixel in a certain segment if its value differs too much from the other pixels in that segment. Then the conserved energy of the regularization by not cutting it from the segment is less than the emitted energy of the data-term. Then the partition problem would prefer to cut the pixel out of the segment. Since we only compute the mean values of the segments in the reduced problem, this pixel would just get its original value assigned which is some by noise distorted value. Hence, these pointy artifacts occur in the oversegmentation results.

As we discussed about the artifacts, let us talk about their alleviation. As we learn in the former subsections, we can apply any method to the reduced graph that works on the full graph. Hence, we can use a denoising method as in the former section where we solve an ROF problem on the reduced graph. Since the TV regularizer does penalize the height of jumps it should penalize these outliers very harshly since they are far off from the neighboring segments. These results can be seen in Figure 7.19e and Figure 7.19f where we applied a $L1$-Cut-Pursuit algorithm to solve the ROF problem for some sufficient regularization parameter. We see that the pointy artifacts are completely suppressed, and since, we used L_1-Cut-Pursuit the number of segments can only be smaller or equal as before in the reduced graph. Hence, the segments in the L_1-Cut-Pursuit solution gather multiple segments from the former L_0-Cut-Pursuit solution. Nevertheless, the new result suffers from a loss of contrast that might be visually undesirable or even a problem for later applied methods. As we show in Section 6.4 we can directly apply a debiasing step solving the data-term on each segment which is just the mean value in this case. Hence, we can see in Figure 7.19h a contrast enhanced image with suppressed pointy artifacts. Additionally, in Figure 7.19g we show the proof of concept where we see the debiased low noise solution that is exactly the same as in Figure 7.19c. Consequently, we have a sufficient method for discretization even for noisy images using a L_0-Cut-Pursuit algorithm.

(a) Parrot with low noise.

(b) Parrot with high noise.

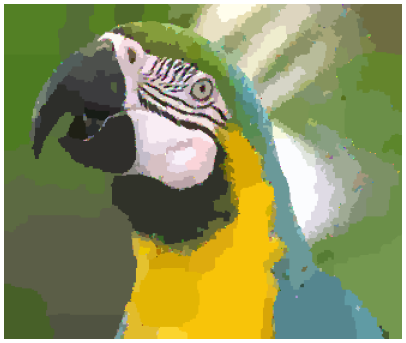
(c) L_0-Cut-Pursuit: Optimizing the values.

(d) L_0-Cut-Pursuit: Optimizing the values.

(e) Cut-Pursuit for ROF on the reduced graph.

(f) Cut-Pursuit for ROF on the reduced graph.

(g) Debiasing (e).



(h) Debiasing (f).

Figure 7.19.: Solutions of L_0-Cut-Pursuit on the *parrot* RGB image with a low noise level (left) and a high noise level (right).

8 Numerics: Point Cloud Sparsification via Cut-Pursuit

In this section, we want to investigate the sparsification properties of the proposed L_0- and L_1-Cut-Pursuit algorithms in Algorithm 7 and Algorithm 3. These results have been published in [85]. We again compute the graph for the point cloud data with a k-NN graph with $k = 7$ and set the weighting to

$$w(u, v) = \exp\big(-\frac{\|f(v) - f(u)\|_2^2}{2}\big).$$

In the following sections we compare the results of point cloud sparsification on three different 3D point clouds via the proposed Cut-Pursuit algorithm and a direct minimization of the energy functional (4.58) via the primal-dual algorithm on graphs we introduce in Algorithm 3.20. The point cloud data that is not corrupted with any additional geometric noise as we visualize them in Figure 6.3. We perform primal-dual algorithm for the full graph until a relative change ΔJ_{rel} of the energy functional between two subsequent iterations is below 10^{-5}. The resulting point clouds show many clusters of points that have been attracted to common coordinates. We apply a filtering step on these resulting clusters that removes all but one point in a neighborhood of radius $\epsilon = 10^{-3}$ relative to the size of the data domain. This approach can be seen as *fine-to-coarse* sparsification and has been used before, e.g., in [52, 51]. On the other hand the proposed Cut-Pursuit algorithm is clearly a *coarse-to-fine* sparsification strategy. For the L_0-Cut-Pursuit algorithm we again use the easiest strategy as in Section 7.4 where we only perform one graph cut for the partition problem and then compute the mean value of the given data over the finer segments. For the application of Cut-Pursuit to channel-isotropic regularizations we always use the first principal component (PC1) as the direction of choice without any additional updates.

8.1 Anisotropic L_1-TV Regularization

To analyze the run times of these approaches, we compare the three data sets, namely *Bunny*, *Happy* and *Dragon*, from the Stanford 3D Scanning Repository [82], we present in Figure 6.3, for anisotropic L_1 regularization and two different regularization parameters. Ad-

Data set / Regularization parameter	Direct optimiz. via PPD	Cut-Pursuit with PD	Cut-Pursuit with PPD
Bunny (35,947 points):			
$\alpha = 0.001$	50s	413s	23s
$\alpha = 0.01$	119s	145s	6s
Dragon (435,545 points):			
$\alpha = 0.005$	5,636s	1,097s	117,4s
$\alpha = 0.002$	6,314s	591s	62,5s
Happy (543,524 points):			
$\alpha = 0.0002$	1,568s	2,400s	222,2s
$\alpha = 0.001$	3,407s	1,186s	93.2s

Table 8.1.: Comparison of overall runtime in seconds between a direct optimization via primal-dual optimization (PD) and Cut-Pursuit where the reduced problem was solved with a primal-dual and with a diagonal preconditioned primal-dual (PPD) algorithm on different point cloud data for anisotropic L_1 regularization and two different regularization parameters α.

ditionally, we compare the run times of the L_1-Cut-Pursuit algorithm where the reduced problem is solved using a primal-dual (PD) algorithm with updated step sizes as in [17] and using a primal-dual algorithm with diagonal preconditioning (PPD) as we introduce in Definition 3.21. These compete against the primal-dual algorithm with diagonal preconditioning (PPD) on the full graph describing the point cloud. The diagonal preconditioning is essential here as otherwise the optimization would be slower by orders of magnitude. Additionally, one would have to compute the step size by using the norm of the gradient which is hard to compute for huge graphs. This problem also arises when using a step size update acceleration as described in [17].

In the following we compare the run time results of our numerical experiments gathered in Table 8.1 for two different regularization parameters for each data set. As the results of both optimization strategies are identical to some extend and cannot be seen visually on the resulting sparsified point clouds, we refrain from showing any point cloud visualization here. However, the results of point cloud sparsification using anisotropic L_1-TV regularization can be seen in Figure 8.1 below.

The first and most obvious observation is that the direct optimization approach, i.e., the primal-dual algorithm on the full graph, performs only well for small point clouds as in the *Bunny* data set. For the *Happy* and *Dragon* data set the measured run time is not reasonable for any real application. Additionally, one can see that the direct optimization approach takes increasingly longer for higher regularization parameters α. This means that for an increasingly sparse results one has to take a longer computation time into account.

While comparing the two variants of the Cut-Pursuit algorithms with only using primal-dual optimization (PD) and the diagonally preconditioned primal-dual algorithm (PDD) we observed that the latter one is always faster than the simple version. This is due to the reasons we pointed out earlier in Definition 3.21, i.e., bad conditioning due to different amount of vertices gathered in each subset of the partition and highly varying values of the accumulated weights between these subsets. Notably, in all tested experiments except the *Bunny* data set the L_1-Cut-Pursuit algorithm *without preconditioning* is significantly faster than the fine-to-

Data set	Direct optimization via PPD	Weighted L_0 Cut-Pursuit
Bunny: 35,947 points	126s 8,034 points left (22.35%)	2.3s 8,065 points left (22.42%)
Happy: 543,524 points	3,305s 29,168 points left (5.37%)	47s 28,484 points left (5.24%)
Dragon: 435,545 points	8,239s 16,438 points left (3.77%)	28.2s 16,405 points left (3.77%)

Table 8.2.: Comparison of overall runtime in seconds between a direct optimization via preconditioned primal-dual optimization (PPD) and the simplest L_0-Cut-Pursuit algorithm for point cloud sparsification tested on the three different data sets presented in Figure 8.1, Figure 8.3 and Figure 8.2.

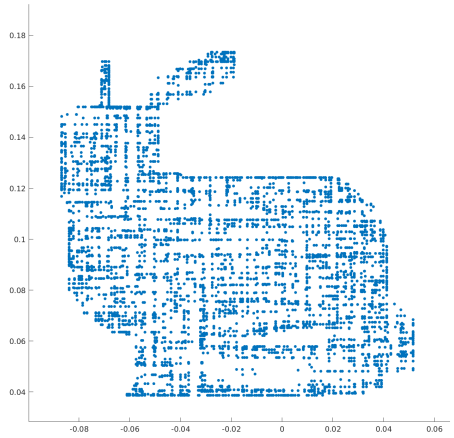
coarse strategy with preconditioned primal-dual minimization.

One interesting observation is that the Cut-Pursuit algorithm is not necessarily getting faster for an increasingly higher regularization as one might expect. This can be seen for the *Happy* data set. The reason for this is that there are two opposite effects competing. With increasingly higher regularization parameter α one can expect the total number of graph cuts to decrease, which leads to less iterations in Algorithm 2. However, at the same time the costs of computing the maximum flow within the finite weighted graph may increase due to the increased flow graph edge capacities. Thus, in some cases the computational costs of minimum graph cuts outweighs the benefit of computing less graph cuts for higher regularization parameters. In case of the preconditioned primal-dual algorithm the overall run times are less affected by the choice of the regularization parameters compared to the standard primal-dual variant.

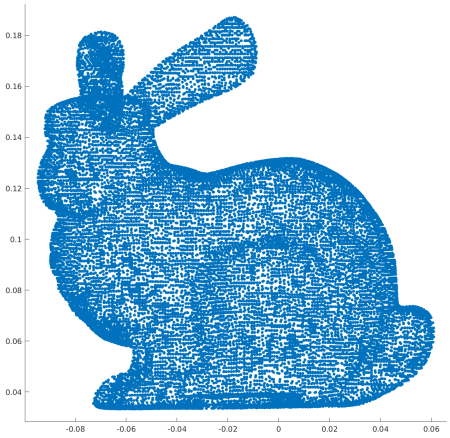
In summary we observe that for large point clouds a Cut-Pursuit approach with a diagonal preconditioned primal-dual optimization to solve the reduced problem is significantly faster than a diagonal preconditioned primal-dual algorithm on the full graph.

8.2 Comparison of Anisotropic L_1-TV and L_0-TV Regularization

In Figure 8.1, Figure 8.2 and Figure 8.3 we compare the sparsification results of the anisotropic L_1-TV and the weighted L_0-TV regularization (5.14) on the three data sets used before. We choose the regularization parameters for both methods in such a way that they yield roughly the same number of points in the resulting sparse point clouds. As one can see, the resulting point cloud of the L_1 regularization for different data sets always induces a very strong block structure to the data. This is clear as we have chosen an anisotropic TV regularization for this experiment. In addition to this structural bias one can also observe a volume shrinkage in the resulting point cloud. This is comparable to the typical contrast loss when using channel-anisotropic TV regularization for denoising on images, e.g., cf. [12] and Section 6.4. On the other hand we see that the proposed L_0 regularization yields a much more detailed and bias-free result.

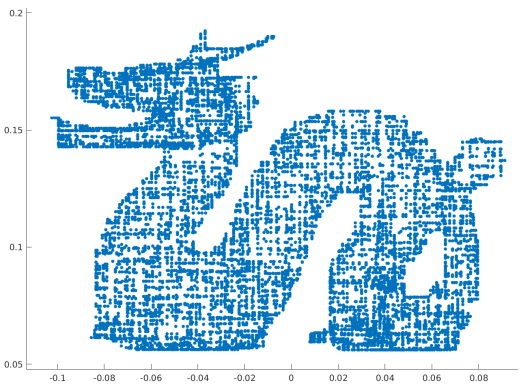


(a) Result of anisotropic L_1 regularization with $\alpha = 0.5$. Time needed: 126 seconds. Points left: $7,794$ (21.68%)

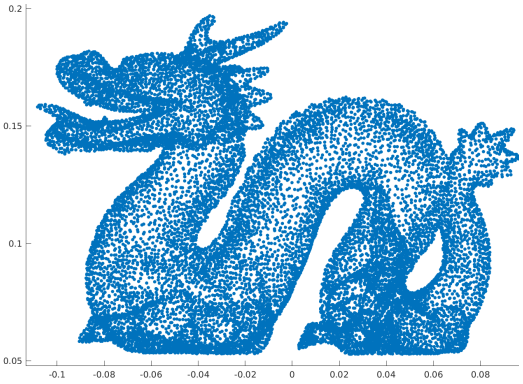


(b) Result of proposed L_0 regularization with $\alpha = 5$. Time needed: 3.7 seconds. Points left: $8,034$ (22.35%)

Figure 8.1.: Results on the Bunny data set.



(a) Result of anisotropic L_1 regularization with $\alpha = 0.5$. Time needed: $8,239$ seconds. Points left: $16,438$ (3.77%)

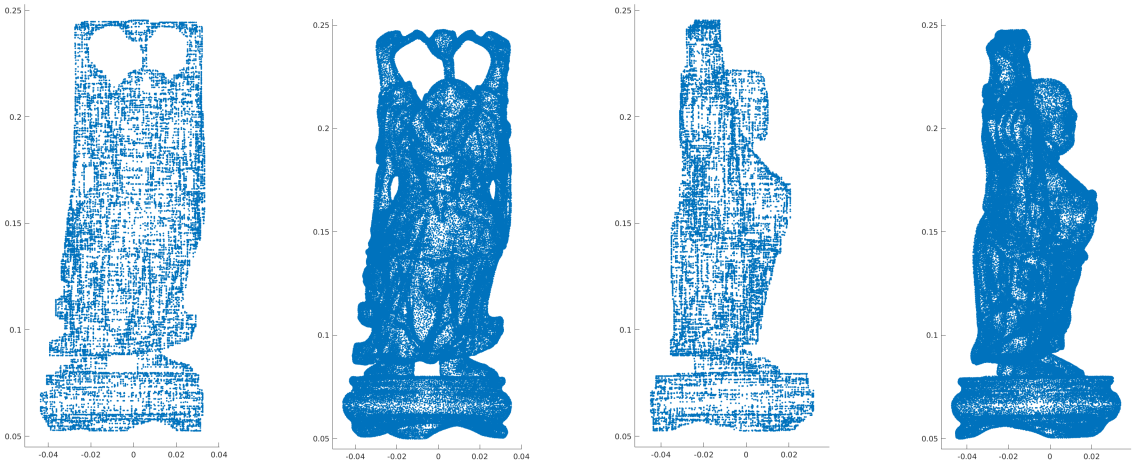


(b) Result of proposed L_0 regularization with $\alpha = 6.5$. Time needed: 70.6 seconds. Points left: $16,138$ (3.71%)

Figure 8.2.: Results on the Dragon data set.

As we would like to highlight by this experiment, the striking argument for using the simplest L_0-Cut-Pursuit algorithm is the significant efficiency gain for point cloud sparsification, which can be seen by comparing the computational times in Table 8.2. Comparing the fine-to-coarse strategy proposed in [52, 51] there is a speed-up by a factor of between 60 to 290 depending on the number of points in the original data set. This speed up of two orders of magnitude (without exploiting any parallelization techniques) renders the proposed method valuable for applications in which point cloud data has to be processed and analyzed in near-realtime conditions.

To summarize our observations above, we can state that when noise-free data is given, point cloud sparsification can best be performed using the L_0-Cut-Pursuit algorithm as in Algorithm 6 where we solve the partition problem by using the PC1 direction and no updates



(a) Result of anisotropic L_1 regularization with $\alpha = 0.5$. Time needed: $3,305$ seconds. Points left: $26,247$ (4.83%)

(b) Result of proposed L_0 regularization with $\alpha = 4$. Time needed: 94 seconds. Points left: 29168 (5.37%)

(c) Result of anisotropic L_1 regularization with $\alpha = 0.5$. Time needed: $3,305$ seconds. Points left: $26,247$ (4.83%)

(d) Result of proposed L_0 regularization with $\alpha = 4$. Time needed: 94 seconds. Points left: $29,168$ (5.37%)

Figure 8.3.: Results on the Happy data set.

and solving the reduced problem by computing the mean value of the data over the segments as described.

8.3 Visual Comparison of Different TV Regularizations

In the following we compare the qualitative difference of point cloud sparsification between the channel-anisotropic and the channel-isotropic L_1-TV-regularized reduced problem (4.67) for $p = q = 1$ and $p = 1$, $q = 2$, respectively. In the channel-isotropic case we again use the first principal component as a direction. Additionally, we visually compare the results of the L_1-TV-regularized point cloud sparsification with the results of the simple weighted isotropic L_0-Cut-Pursuit algorithm.

As point cloud data we chose the Fandisk model (cf. [30]), which consists of a combination of roundish and flat surfaces as well as sharp edges. This data set is often used to evaluate the effectiveness of point cloud denoising methods in the literature, e.g., see [30, 94, 84]. For this experiment we constructed a symmetrized k-nearest neighbor graph for $k = 7$ and set the regularization parameter α such that all resulting point clouds have roughly the same compression rate of 17%. The regularization parameters used for each regularization term are indicated in Figure 8.4.

In Figure 8.4 the results of point cloud sparsification with the three described regularization terms are displayed. The left column shows a mesh triangulation of the data, while we present a corresponding surface rendering with MATLAB built-in Phong lighting [61] in the right column. The first row in Figure 8.4a and Figure 8.4b shows the original Fandisk data set, which consists of $11,949$ 3D points. In the second and third row we present the results of anisotropic and isotropic L_1-TV regularization, respectively. As can be seen the anisotropic L_1-TV regularization induces flat surface regions that coincide with the planes that are spanned between the coordinate axes of the data set. This is not surprising as the anisotropic L_1-TV regularization decouples the 3D point coordinates and only enforces regularity within each dimension. This leads to typical staircase artifacts in Figure 8.4c and Figure 8.4d as it is well-known for total variation regularization in imaging applications. On the other hand, using the isotropic L_1 regularization term for $q = 2$ couples the coordinates of each 3D point and hence does not lead to any staircase artifacts as can be seen in Figure 8.4e and Figure 8.4f. The resulting surfaces appear much smoother compared to the previously discussed anisotropic L_1-TV regularization. While the round and flat parts of the data set are well-preserved by this regularization term the sharp edges are lost as can been observed. The last row shows the results of our proposed weighted L_0-TV regularization. As we demonstrate in Figure 8.4g and Figure 8.4h the mesh triangulation of the sparsified point cloud is much more regular compared to our experiments with the L_1-TV regularization terms. Additionally, the sharp edge features of the Fandisk data set are significantly better preserved.

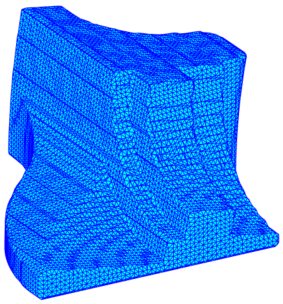
To wrap this section up we want to shortly investigate the discretization property of the isotropic L_0-Cut-Pursuit algorithm on point cloud data as we have investigated on images in Section 7.4. There we showed experimental that the regularization parameter α can be interpreted as a control parameter for the expected level-of-detail, and thus, of the resulting number of points as we demonstrate in Figure 8.5 and Table 8.3. Due to the fact that this approach leads to a sparsification result that is close to the original point cloud, there is no volume shrinkage effect and hence no need for an explicit debiasing step as discussed in

Data Set	$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 1$	$\alpha = 5$
Bunny	35947 (100%)	22651 (63%)	8034 (22.3%)	1473 (4%)
Happy	543524 (100%)	213901 (39.4%)	85719 (15.8%)	24555 (4.5%)
Dragon	435545 (100%)	177448 (40.7%)	71040 (16.3%)	19844 (4.5%)

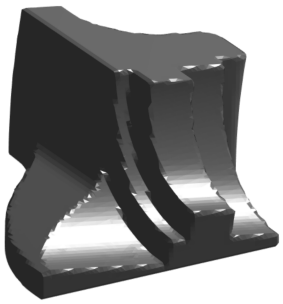
Table 8.3.: Comparison of sparsification rates of different regularization parameter selection for the L_0-Cut-Pursuit algorithm. It shows the number of leftover points and the overall percentage.

Section 6.4.

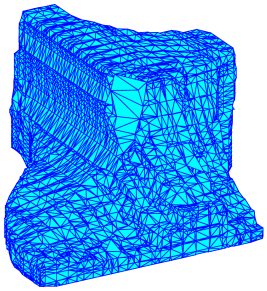
With these observations we wrap up this part and refer to Section 15.1 for a conclusion and a small outlook on future work.



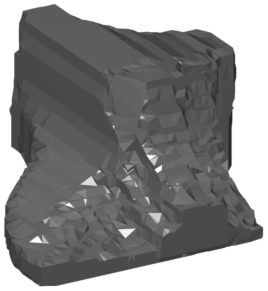
(a) Mesh triangulation of the full point cloud of the Fandisk data set



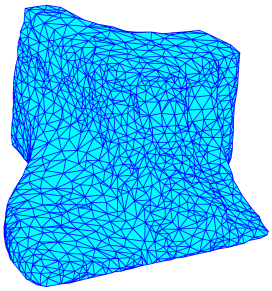
(b) Surface rendering of the full point cloud of the Fandisk data set



(c) Mesh triangulation of results for anisotropic L_1-TV with $\alpha = 0.15$

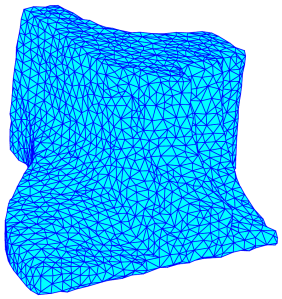


(d) Surface rendering of results for anisotropic L_1-TV with $\alpha = 0.15$

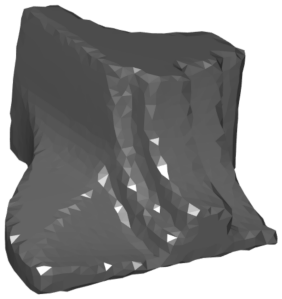


(e) Mesh triangulation of results for isotropic L_1-TV with $\alpha = 0.065$

(f) Surface rendering of results for isotropic L_1-TV with $\alpha = 0.065$

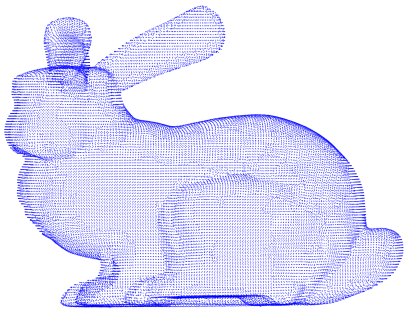


(g) Mesh triangulation of results for isotropic L_0-TV with $\alpha = 0.05$

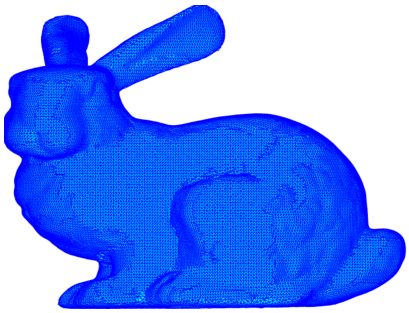


(h) Surface rendering of results for isotropic L_0-TV with $\alpha = 0.05$

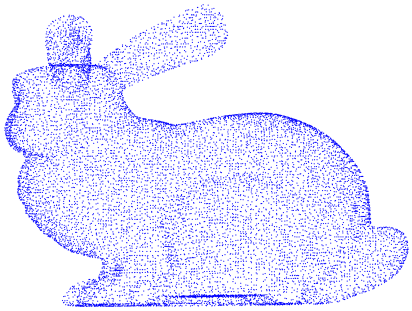
Figure 8.4.: Comparison of point cloud sparsification results of the Fandisk model for anisotropic/isotropic L_1-TV and the proposed isotropic L_0-TV regularization. The regularization parameter α is chosen such that all compressed point clouds consist only of 17% of the original point cloud.



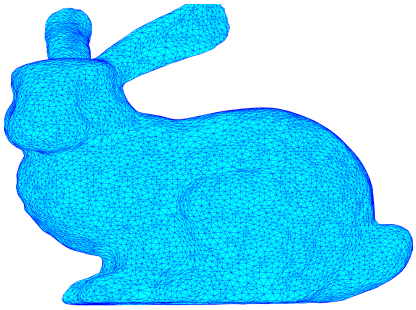
(a) Full point cloud data of the *Bunny* model (35,947 points)



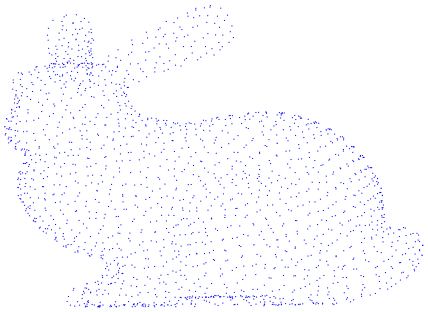
(b) Triangulation of full point cloud data of the *Bunny* model



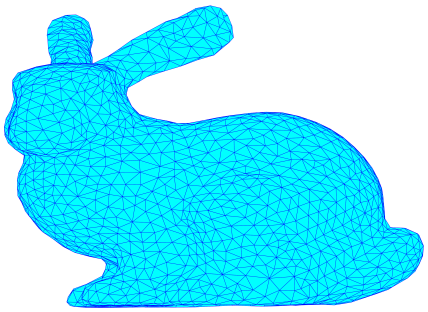
(c) Point cloud sparsification for $\alpha = 0.3$ (12,105 points)



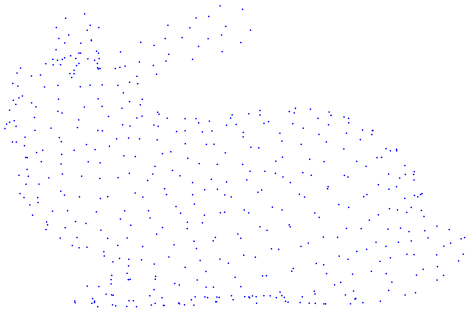
(d) Triangulation of sparsified point cloud data for $\alpha = 0.3$



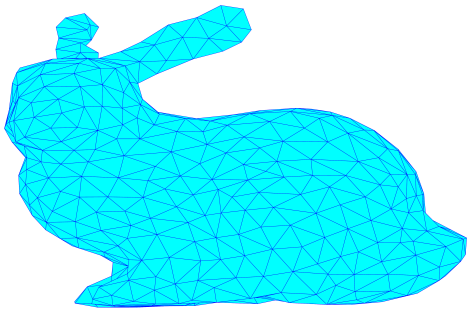
(e) Point cloud sparsification for $\alpha = 1$ (1,912 points)



(f) Triangulation of sparsified point cloud data for $\alpha = 1$



(g) Point cloud sparsification for $\alpha = 3.5$ (498 points)



(h) Triangulation of sparsified point cloud data for $\alpha = 3.5$

Figure 8.5.: Comparison of point cloud sparsification results using the proposed weighted L_0-TV regularization with different values of α

Part II
Dynamic PET Image Reconstruction

9 Introduction to PET Reconstruction

In this section, we introduce the theoretical foundations of Positron Emission Tomography (PET). To do so, we introduce how PET data is measured in practice, how we can build a mathematical model for it and how to reconstruct the spatial distribution of the tracer inside a given image domain. After developing the foundations we add regularization to the reconstruction method which, in particular, is the well-known Total Variation (TV) as we introduce in Section 3.1.4. To solve such problems we introduce two different methods to reconstruct regularized images. We start by describing PET and how data is measured and stored. We keep this short in the following and refer to [89, 59, 73] for deeper and more detailed insights.

9.1 Positron Emission Tomography

In short, Positron Emission Tomography (PET) is an imaging method where a certain amount of radioactive tracer is injected into a patient or subject. This tracer, e.g. 18F-FDG (Fludeoxyglucose F 18), tends to be retained by tissues and organs with a high metabolic activity and emits positrons over time. These positrons then hit free electrons that annihilate with each other and emit two γ-photons that are cast away from this position in an angle of about 180 degree. We call the process of annihilation an *event* in the following. These γ-photons then hit photo-sensitive detectors that pass the measured impact to the software that stores the ID of the detector that was hit and the time of the impact. The goal is now to find the detector pairs that where hit by the photons generated by the same event in the image domain, which is equivalent to finding a *coincidence line* that passes through the position of the event. Afterwards, one can gather all these lines and count the number of events that where tracked on them. Then we reconstruct the distribution of the tracer with well-known methods that use lines to reconstruct images like in CT or with the *EM* reconstruction method we will explain later on. To combine the measured impacts to the correct event is a difficult task due to scattering effects, attenuation by tissue and bones [42, 8], random measurements that generate a line for two independent events [3, 91] and dead crystals, to just name a few (cf. [89]). These problems were tackled successfully over the last decades and can be overcome by preprocessing methods. Note that a PET scanner is built of a finite number of detectors (e.g. [21]). Thus, there is only a finite number of possible

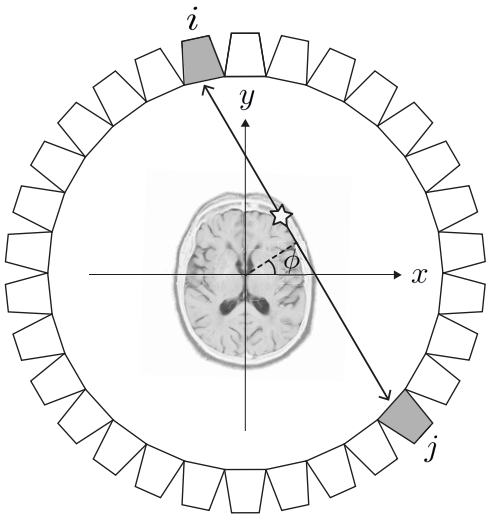


Figure 9.1.: Visualization of an event emitting two γ-photons which hit a detector pair (i, j).

lines that can technically be considered to store events. Additionally, in order to store less lines and make reconstruction faster multiple lines of neighboring detectors with the same directions are gathered in bins.

First, we aim to reconstruct a static, time-independent image or distribution map from the given PET data. Thus, every data point is a unique line coupled with the total amount of tracked events on this line. There are two possible ideas of how this data can be described. As lines can be represented by an angle of the line to one axis of the domain and an offset of the line to some origin of the axis, one can visualize this as a so called *sinogram* (cf. [59]).

Another method is just to list the possible lines and store the counts of events in a vector where the index corresponds to the number of the line. In this work we only consider the second type of data description since it fits the best to the models we will use throughout this work.

Since the measuring of events on coincidence lines is time dependent and only a very small number of events happen in a time period we assume Poisson noise in the data as mentioned in [59]. Thus, it is not rational to reconstruct the image by the same methods as used in Computed Tomography (CT) since these only model Gaussian noise. To challenge this problem the *Expectation Maximization* (EM) method is used for reconstruction [59]. In the following section we derive the EM algorithm for given static PET data.

9.2 Expectation Maximization Reconstruction

In this section, we introduce how to reconstruct a positive image $u \geq 0$ that represents the spatial tracer distribution in the image domain Ω from a given data set g. As we measure the number of emitted photons the distribution u of the tracer can only be positive. Let us assume that we are given a PET scanner that has L lines and a discrete 2D image domain Ω of the size $m \times n$. Note that this can be done analogously for 3D image domains. Let

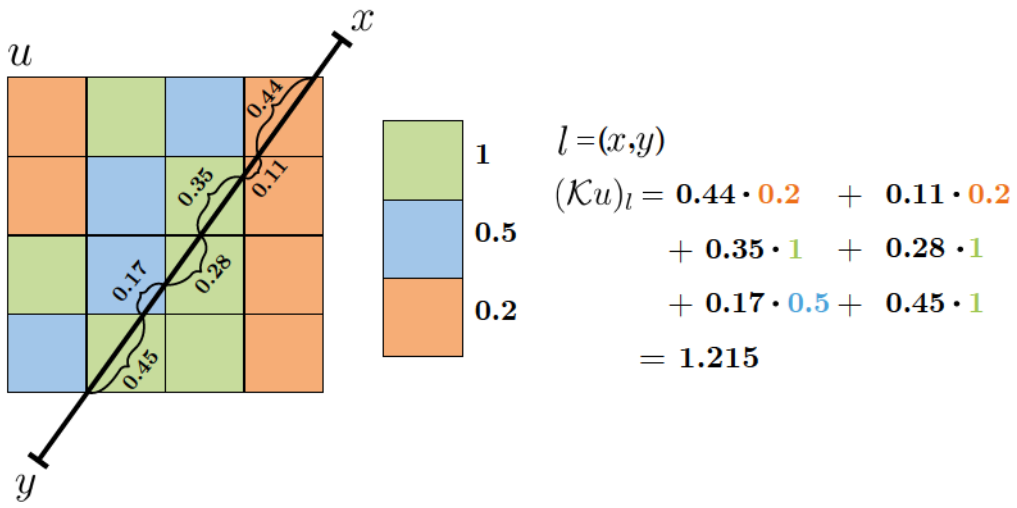


Figure 9.2.: Example of an 4×4 image u and a line l between two detectors x and y. The image consists of three different values. The value $(\mathcal{K}u)_l$ is the "line integral" over the line l over the image which sums up the length of intersections of the line with a pixel multiplied with its value.

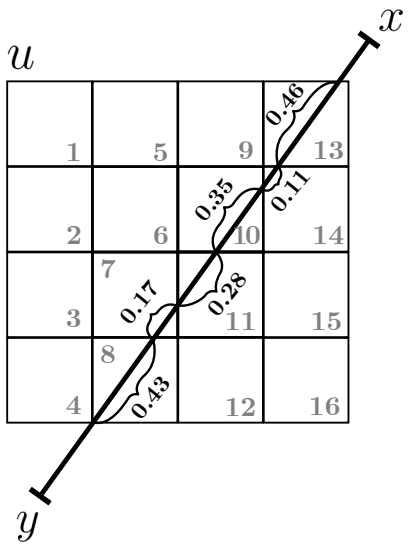
us denote N as the total number of pixels, i.e., $N = m \cdot n$. In this work we only consider discrete image domains Ω with N pixels. Then we can write each image in a vectorized form as $u \in \mathbb{R}_+^N$, where we stack the n columns of the image on top of each other. In this work we denote $\mathbb{R}_+ = \mathbb{R}_{\geq 0}$. Additionally, we introduce the *line projection operator* or *PET operator* $\mathcal{K} : \mathbb{R}_+^N \to \mathbb{N}^L$ for a given scanner with a set of L lines which maps an image from the image domain Ω to the data space. Thus, for given data $g \in \mathbb{N}^L$ and PET operator $\mathcal{K}$ the PET reconstruction problem is to find an image $u \in \mathbb{R}_+^N$ such that

$$\mathcal{K}u = g.$$

Note that the operator $\mathcal{K}$ does preserve positivity, i.e., $\mathcal{K}u \geq 0$ as $u \geq 0$. Additionally, we can assume background noise on practice such that the underlying distribution map $u > 0$, and since, $\mathcal{K}$ preserves the positivity also the data $g > 0$.

Before we take a deeper look at the data and the corresponding model we want to dismantle the PET operator $\mathcal{K}$. A very easy example describing the following can be seen in Figure 9.2. The operator $\mathcal{K}$ works by taking every line of the scanner and casting it through the image domain Ω, which is in fact a line integral operator [59]. For each line it measures the length of the intersection with every pixel in its path and multiplies it by the value of the respective pixel. Then it sums up these values for the line. Thus, it generates a vector of length L for the image u where each index corresponds to the same line as in the data g. Then we can compare the values generated by $\mathcal{K}u$ with the data g. Now we understand how we can map any image $u \in \mathbb{R}_+^N$ into the data space with $\mathcal{K}$. We want to point out two features we will use in this work and that might also reinforce the understanding of the PET operator $\mathcal{K}$.

First let us note that computing the line integrals is a process with huge computational costs. As we have seen above one has to compute the line, find the intersections of the line



$$K_{l,7} = \mathbf{0.17}$$
$$K_{l,8} = \mathbf{0.43}$$
$$K_{l,10} = \mathbf{0.35}$$
$$K_{l,11} = \mathbf{0.28}$$
$$K_{l,13} = \mathbf{0.46}$$
$$K_{l,14} = \mathbf{0.11}$$

Figure 9.3.: Example of an 4×4 image domain and a given line l. The image is provided with a linear indexing (or could be thought of as a vectorized image). On the right hand side we can see the lengths of intersection of the line l with the pixels i gathered in the entries $K_{l,i}$.

with the pixels in u, measure the length, multiply with the value and sum everything up. This is a redundant task since one would generate the line and the intersections every time the operator $\mathcal{K}$ is applied. Luckily, as long as the number of pixels and lines is not too big one can build a sparse matrix to represent the mapping functionality of the operator $\mathcal{K}$. We present an easy example of how to generate a matrix from a given set of lines and an discrete image domain in Figure 9.3. For the projection of a line l we always need the length of the intersection l_i of line l with a pixel i in the image. We could store these intersection lengths in a vector of length N with l_i as the i'th entry. Then, computing the scalar product of this vector with the (vectorized) image yields the value of $(\mathcal{K}u)_l$. This can be done for every line l in the PET scanner and stored in a (sparse) matrix $K \in \mathbb{R}_+^{L \times N}$ as the rows. Then we have a matrix representation $Ku = \mathcal{K}u$ for every $u \in \mathbb{R}_+^N$. The computation of the projection with K is much faster and also highly optimized in practice since it only uses matrix vector multiplication. Nevertheless, when we think about real applications with a high amount of pixels and millions of lines - and maybe even 3D - it would be hard to store K in memory since it would have - even though it is very sparse - a huge amount of non-zero entries. In this case one has to apply $\mathcal{K}$ again.

Now that we have investigated the operator we want to focus on the measured data $g \in \mathbb{R}_+^L$. As mentioned before, the data g is distorted by Poisson noise. Thus, to compare the closeness of the data g to the reconstruction u we take the *Kullback-Leibler divergence* [45] that measures the *directed divergence* between two discrete probability distributions P and Q on the same probability space $\mathcal{X}$ as

$$KL(P, Q) = \sum_{j=1}^{L} P_j \log\left(\frac{P_j}{Q_j}\right). \tag{9.1}$$

This divergence tends to zero for distributions that are very close. Hence, using this di-

vergence as $\mathrm{KL}(g, \mathcal{K}u)$ would give the closeness of the data to the projection of the current distribution u, which would imply that u is close to the distribution that generated g. As derived in detail in [73, Section 4.2] using a maximum a-posteriori estimator, the corresponding variational problem to find a positive spatial distribution $u \in \mathbb{R}_+^N$ for given Poisson distributed PET data g is given as

$$\operatorname*{argmin}_{u \in \mathbb{R}_+^N} \sum_{j=1}^{L} (\mathcal{K}u)_j - g_j + g_j \log\left(\frac{g_j}{(\mathcal{K}u)_j}\right) \tag{9.2}$$

which is equivalent to

$$\operatorname*{argmin}_{u \in \mathbb{R}^N} \sum_{j=1}^{L} (\mathcal{K}u)_j - g_j + g_j \log\left(\frac{g_j}{(\mathcal{K}u)_j}\right) + \delta_+(u_i) \tag{9.3}$$

where

$$\delta_+(x) = \begin{cases} 0, & \text{if } x_j \geq 0, \\ \infty, & \text{else.} \end{cases} \tag{9.4}$$

In Section 2.2.3 we introduce the *Kullback-Leibler data-term* in Definition 2.9 and we denote it by the functional

$$\begin{aligned} D_{KL}(\mathcal{K}u, g) &= \sum_{j=1}^{L} (\mathcal{K}u)_j - g_j + KL(g, \mathcal{K}u) \\ &= \sum_{j=1}^{L} (\mathcal{K}u)_j - g_j + g_j \log\left(\frac{g_j}{(\mathcal{K}u)_j}\right). \end{aligned} \tag{9.5}$$

Note that we assume here that $g > 0$ and that $\mathcal{K}u > 0$ as we always assume background noise and that every line has measured a photon at least once. Otherwise the logarithm cannot be evaluated. This is the same assumption as in [73].

In addition to the chosen data-term we want to make an assumption that the reconstructed image cannot have any negative values since a negative amount of tracer cannot exist. Hence, we will apply a positivity constraint to enforce the reconstruction u to be positive everywhere. Thus, to reconstruct the image from a given PET data set g we optimize the problem

There are many different approaches to derive the classical algorithm to optimize this problem. We will follow the ideas of [15]. Hence, we use the Karush-Kuhn-Tucker (KKT) conditions defined in Proposition 2.1 with the constraint $h(x) \leq 0$ and the first order optimality of the terms in (9.3). Looking at Proposition 2.1 and the positivity constraint as $u_i > 0$ we need to set $h(x) = -x$ to meet the condition that $h(u) \leq 0$. Now we easily see that $\partial h(u) = -1$. We already computed the derivative of the KL data-term ∇D_{KL} in (2.10). Thus, we can state the KKT that assures the existence of a Lagrange multiplier $\lambda \geq 0$ for

which the stationary points of (9.2) satisfy the conditions

$$0 = \mathcal{K}^*\mathbf{1} - \mathcal{K}^*\left(\frac{g}{\mathcal{K}u}\right) - \lambda \tag{9.6}$$

$$0 = \lambda u \tag{9.7}$$

where $\mathcal{K}^*$ is the adjoint operator of $\mathcal{K}$. This exists since we are operating in Hilbert spaces. Note that we use element-wise multiplication and division when handling vectors in the following. Scalar products of vectors will be denoted explicitly. Now we can multiply (9.6) by u which leads to

$$0 = \mathcal{K}^*\mathbf{1}u - u\mathcal{K}^*\left(\frac{g}{\mathcal{K}u}\right) - \lambda u \tag{9.8}$$

where we can drop λu since it is zero by (9.7). Then we can reformulate (9.8) into the following fixed-point equation

$$\begin{aligned} & \mathcal{K}^*\mathbf{1}u = u\mathcal{K}^*\left(\frac{g}{\mathcal{K}u}\right) \\ \Leftrightarrow \quad & u = \frac{u}{\mathcal{K}^*\mathbf{1}}\mathcal{K}^*\left(\frac{g}{\mathcal{K}u}\right). \end{aligned} \tag{9.9}$$

This then leads to the algorithm that is well-known in the literature as the *EM-algorithm*:

Algorithm 9.1. (EM-algorithm)
Let Ω be an image space with N pixels and assume a scanner with L lines. The operator $\mathcal{K}$ is a projection operator for projecting the lines through the N pixels. Let $g \in \mathbb{N}^L$ a measured data set. Then an underlying spatial distribution that generated g can be recovered by the following fixed-point iterations scheme

$$u^{n+1} = \frac{u^n}{\mathcal{K}^*\mathbf{1}}\mathcal{K}^*\left(\frac{g}{\mathcal{K}u^n}\right). \tag{9.10}$$

Another way to derive this via stochastic processes and methods can be found in [59].

Let us make a small remark on the adjoint operator $\mathcal{K}^*$. As the (forward) PET operator $\mathcal{K}$ computes the sum over intersection lengths and pixels values the adjoint operator takes a value v_l and passes it through the image u over the line l. It takes the intersection length l_i of the line l with a pixel i and adds the value $v \cdot l_i$ to the current value in the i'th pixel. Thus, we can understand $\mathcal{K}^*$ as an operator that distributes the values in a vector of length L over the lines onto the image domain.

The EM algorithm (9.10) enables us to reconstruct a given data set g for a predefined scanner and the corresponding projection operator $\mathcal{K}$. In practice, even when we assume that the lines and data are corrected for the problems like scattering, attenuation and random measures etc., we still face the problem of reconstructed distribution maps with a high amount of noise. For sure, one could reconstruct the image and afterwards use a denoising method like solving the ROF problem on the reconstructed image as shown in Algorithm 3.20. But this leads to the problem, that the process of reconstruction and denoising are completely

decoupled and could lead to undesirable results. Thus, it is advisable to couple these two steps of reconstruction and denoising during the reconstruction. This can be done by adding a TV regularization term to (9.2) as we will do in the following section.

10

Total Variation Regularization on Reconstruction

In this section, we introduce the well-known *total variation* (TV) regularization to the PET reconstruction to handle noise that is present in the reconstructed image. We already discussed the total variation in Section 3.1.4. Since we want to reconstruct and denoise the reconstruction at the same time we add a total variation regularization term to the minimization problem (9.2) that yields

$$\underset{u\in\mathcal{H}(\Omega)}{\operatorname{argmin}} \sum_{j=1}^{M} (\mathcal{K}u)_j - g_j + g_j \log\left(\frac{g_j}{(\mathcal{K}u)_j}\right) + \delta_+(u) + \alpha\|u\|_{\mathrm{TV}}. \tag{10.1}$$

Here $\alpha > 0$ is a regularization parameter that handles the balance between denoising and pure reconstruction. The term $\|u\|_{\mathrm{TV}}$ is called the *TV regularizer* and can be defined in an (spatial-)anisotropic or (spatial-)isotropic sense depending on the definition of the inner norm as we described in Section 3.1.4. In particular it depends on the norm used for the gradient $\nabla\colon \mathbb{R}_+^N \to \mathbb{R}^\eta$ where

$$\|\nabla u\|_{2,1}$$

is the *isotropic* TV regularization term and

$$\|\nabla u\|_{1,1}$$

is the *anisotropic* TV regularization term. For the sake of simplicity we do not go into detail here and denote the semi-norm used as

$$\|u\|_{TV} = \|\nabla u\|_1$$

from now on. For more information we refer to Section 3.1.4 and the references therein.

In this thesis we introduce two different methods to compute the solution for this minimization problem. The first one is a direct approach by deriving a first-order primal-dual scheme as we already presented in Algorithm 3.20 on graphs. We will learn that this is a

very inefficient and costly method to solve this problem. The second method is a splitting algorithm to tackle these inefficiencies by the name *Forward-Backward EM-TV* introduced in [74] which splits the evaluation of the operator $\mathcal{K}$ from a modified ROF problem.

10.1 First-Order Primal-Dual for PET-TV

We start by deriving the first-order primal-dual scheme for problem (10.1). The key difference to the ROF problem we studied before in Algorithm 3.22 is that we have a positivity constraint and a data-term that depends on a huge operator $\mathcal{K}$. Since in the ROF problem we dualized the TV term to get rid of the term with the gradient operator, we will dualize the Kullback-Leibler data-term *and* the TV regularization term at once to somehow substitute both operators. Then the positivity constraint is the only primal part. We first reformulate (10.1) into a form that was introduced in [17] and is given in this special case by

$$\operatorname*{argmin}_{u \in \mathcal{H}(\Omega)} \; F_1(\nabla u) + F_2(\mathcal{K}u) + G(u), \tag{10.2}$$

where

$$\begin{aligned} F_1(v) &= \alpha \|v\|_1, \\ F_2(w) &= \sum_{j=1}^{M} w_j - g_j + g_j \log\left(\frac{g_j}{w_j}\right), \\ G(u) &= \delta_+(u). \end{aligned}$$

When we now substitute

$$p = \nabla u \text{ and } q = \mathcal{K}u$$

we get the following minimization problem

$$\operatorname*{argmin}_{u,\, p,\, q} \; F_1(p) + F_2(q) + G(u), \quad \text{s.t. } p = \nabla u,\; q = \mathcal{K}u. \tag{10.3}$$

For this minimization problem we can formulate a saddle-point problem by computing the Lagrange multipliers $y \in \mathbb{R}_+^\eta$, $z \in \mathbb{R}_+^L$ and use the definition of a convex conjugate (c.f. (2.1)) as we did for deriving the first-order primal-dual in Algorithm 3.20. Then we have to solve the following saddle-point problem

$$\min_{u} \max_{y,\, z} \; \langle y, \nabla u \rangle - F_1^*(y) + \langle z, \mathcal{K}u \rangle - F_2^*(z) + G(u). \tag{10.4}$$

From Algorithm 3.22 and in particular from Appendix 3.C we can take the convex conjugate of the TV regularizer as

$$F_1^*(y) = (\|\cdot\|_1)^*(y) = \delta_{B^\infty(\alpha)}(y).$$

This is a characteristic function that vanishes inside the ∞-ball $B^\infty(\alpha)$ of size α. The convex conjugate of the Kullback-Leibler data-term can be taken from Proposition 2.11 and is given

as

$$F_2^*(z) = D_{KL}^*(z) = \sum_{j=1}^{M} g_j\big(\log(g_j) - \log(1 - z_j) - 1\big), \quad z_j < 1.$$

To build the primal-dual algorithm from (3.25) we have to compute the *proximity operators* of each term in (10.4). For the conjugate of TV and the KL data-term these can be found in Appendix 3.C and Proposition 2.12 respectively. To recapitulate they are given for D_{KL}^* as

$$\operatorname{prox}_{\sigma_z D_{KL}^*}(\phi) = \frac{1}{2}\left(\phi + 1 - \sqrt{(\phi - 1)^2 + 4\sigma_z g}\right) \tag{10.5}$$

with some step size $\sigma_z > 0$ and for $\|\cdot\|_{TV}^*$ as

$$\operatorname{prox}_{\sigma_y \|\cdot\|_1^*}(\phi) = \operatorname{proj}_{B^\infty(\alpha)}(\phi) \tag{10.6}$$

for a step size $\sigma_y > 0$. To finish up the derivation of the algorithmic scheme we have to compute the proximity operator of the primal positivity constraint δ_+. It can easily be seen that the proximity operator is just a projection given by

$$\operatorname{prox}_{\tau\delta_+}(\phi) = \operatorname{proj}_{\geq 0}(\phi) = \max(\phi, 0) \tag{10.7}$$

where max is evaluated pointwise. Finally, we can write done the whole iterative first-order primal-dual scheme to solve the problem (10.1) as follows:

Algorithm 10.1. (First-Order Primal-Dual PET-TV)
Let $\theta = 1$ and $\tau, \sigma_y, \sigma_z \geq 0$ adequately chosen step sizes and initialize with $u^0 > 0$. Then the first-order primal-dual algorithm to solve (10.1) is given by:

$$\begin{cases} y^{n+1} &= \operatorname{proj}_{B^\infty(\alpha)}\left(y^n + \sigma_y \nabla u^n\right) \\ z^{n+1} &= \frac{1}{2}\left(z^n + \sigma_z \mathcal{K} u^n + 1 - \sqrt{(z^n + \sigma_z \mathcal{K} u^n - 1)^2 + 4\sigma_z g}\right) \\ u^{n+1} &= \max\left(u^n - \tau\left(\mathcal{K}^* z^{n+1} + \nabla^* y^{n+1}\right), 0\right) \\ u^{n+1} &= u^{n+1} + \theta\left(u^{n+1} - u^n\right). \end{cases} \tag{10.8}$$

The mentioned step sizes τ, σ_y and σ_z have to be chosen wisely. There exist several methods to handle these by approximating them [17], updating them adaptively with some kind of rule [17] in each step or computing fixed [62] or adaptive diagonal preconditioning [34].

To understand the motivation for the *Forward-Backward EM-TV* algorithm proposed in [74] - and explained in the following section - we have to get an understanding of the complexity of the computation of the algorithm presented in Algorithm 10.1. We have two problems that make the usage of a primal-dual algorithm critical in this application. The first one is that in most cases primal-dual has to do many iteration steps to converge close enough to a solution. Thus, in practice, we can expect to iterate for several thousand of steps. The second problem is the operator $\mathcal{K}$ which is very expensive. Its complexity scales with the number of pixels N in the image domain and the number of lines L. When we assume for example a 3D image domain of the size $n_1 \times n_2 \times n_3$ we can deduce directly that a line can hit

a maximum of $l_{max} = n_1 + n_2 + n_3 - 1$ pixels in the domain since this would be the longest path through the domain. This implies that the matrix K can have a maximum of $l_{max} \cdot L$ non-zero elements, which is much less than the total number of entries $N \cdot L$, and typically about 2-5% of the number of elements in K. Thus, this matrix is always highly sparse. Still, in practice it is hard to store the matrix operator in memory since we have several millions of lines and a high resolution of the image domain. As a result, reconstructions have to be carried out by the usage of the continuous PET operator $\mathcal{K}$ instead of the matrix operator K. Even if the evaluation of the PET operator can be highly parallelized it is a huge effort to compute a full projection. In practice it is favorable to apply the operator as little as possible especially when reconstructing high resolution images. The combination of many iteration steps with a huge complex operator makes the application of the primal-dual in Algorithm 10.1 for solving the problem (10.1) very inefficient if not impractical. This is where the next method comes into play which is a rather straightforward idea.

The aim is to decouple the application of the operator $\mathcal{K}$ and the primal-dual algorithm by splitting both steps. This is the key for the *Forward-Backward EM-TV* method that we will cover in the next section.

10.2 Forward-Backward EM-TV

As we have motivated before it is desireable to split the application of the operator $\mathcal{K}$ and the primal-dual algorithm to exclude the computational heavy evaluation of the operator from the long iterations of the primal-dual algorithm. This is what will be done in the following by a forward-backward splitting method, introduced in [74]. We recapitulate the main steps to derive the algorithm as we will need some of the ideas later on. See [74] for deeper insights and details we do not cover here.

As we know, the TV regularizer is non-differentiable, but we can provide the subdifferential (cf. [11, 73, 65]). If we take (10.1) and compute the KKT as in (9.6) and (9.7) we get to the following conditions for the stationary points of (10.1):

$$0 \in \partial D_{KL}(u) + \alpha \partial \|\nabla u\|_1 - \lambda \tag{10.9}$$

$$0 = \lambda u. \tag{10.10}$$

Since D_{KL} is differentiable we can compute the gradient for the data-term as derived in Proposition 2.10. The TV regularizer is - as mentioned before - not differentiable. Nevertheless, we know that there exists a subgradient $p \in \partial \|\nabla u\|_1$ such that it satisfies (10.9). Then we can state

$$0 = \mathcal{K}^* \mathbf{1} - \mathcal{K}^* \left(\frac{g}{\mathcal{K}u} \right) + \alpha p - \lambda, \quad p \in \partial \|\nabla u\|_1. \tag{10.11}$$

By pointwise multiplying (10.11) with u and applying $\lambda u = 0$ from (10.10) we get the

formulation

$$0 = \mathcal{K}^*\mathbf{1}u - \mathcal{K}^*\left(\frac{g}{\mathcal{K}u}\right)u + \alpha p u - \lambda u, \qquad p \in \partial\|\nabla u\|_1 \tag{10.12}$$

$$\Leftrightarrow \quad u = \frac{u}{\mathcal{K}^*\mathbf{1}}\mathcal{K}^*\left(\frac{g}{\mathcal{K}u}\right) - \alpha\frac{u}{\mathcal{K}^*\mathbf{1}}p, \qquad p \in \partial\|\nabla u\|_1. \tag{10.13}$$

From this one can construct a semi-implicit iterative scheme that is given as

$$u^{n+1} = \frac{u^n}{\mathcal{K}^*\mathbf{1}}\mathcal{K}^*\left(\frac{g}{\mathcal{K}u^n}\right) - \alpha\frac{u^n}{\mathcal{K}^*\mathbf{1}}p^{n+1}, \quad p^{n+1} \in \partial\|\nabla u^{n+1}\|_1. \tag{10.14}$$

This iteration can be realized by introducing a intermediate step an first update the left term and then take the update and update with the right term. This gives a two step alternating algorithm

$$\begin{cases} u^{n+\frac{1}{2}} = \dfrac{u^n}{\mathcal{K}^*\mathbf{1}}\mathcal{K}^*\left(\dfrac{g}{\mathcal{K}u^n}\right) & \text{(EM-step)} \\ u^{n+1} = u^{n+\frac{1}{2}} - \alpha\dfrac{u^n}{\mathcal{K}^*\mathbf{1}}p^{n+1}, \quad p^{n+1} \in \partial\|\nabla u^{n+1}\|_1. & \text{(TV-step)} \end{cases} \tag{10.15}$$

Looking at (9.10) in Algorithm 9.1 we see that the first step is actually one EM iteration step. By reformulation the TV-step in (10.15) to

$$0 = \frac{\mathcal{K}^*\mathbf{1}(u^{n+1} - u^{n+\frac{1}{2}})}{u^n} + \alpha p^{n+1}, \quad p^{n+1} \in \partial\|\nabla u^{n+1}\|_1 \tag{10.16}$$

we can directly see that this is just the optimality condition of the following minimization problem

$$u^{n+1} = \operatorname*{argmin}_{u\in\mathcal{H}(\Omega)} \left\langle \frac{\mathcal{K}^*\mathbf{1}}{2u^n}(u - u^{n+\frac{1}{2}}), u - u^{n+\frac{1}{2}} \right\rangle + \alpha\|\nabla u\|_1 \tag{10.17}$$

for which u^{n+1} is a solution. The data-term is in fact a weighted L_2-data-term as defined in Definition 2.6 with the weighting $\omega^n = \operatorname{diag}(\frac{\mathcal{K}^*\mathbf{1}}{u^n})$. Then the minimization problem is given as a weighted ROF problem

$$u^{n+1} = \operatorname*{argmin}_{u\in\mathcal{H}(\Omega)} \frac{1}{2}\|u - u^{n+\frac{1}{2}}\|^2_{2,\omega^n} + \alpha\|\nabla u\|_1. \tag{10.18}$$

One can easily derive a similar first-order primal-dual algorithm to solve this problem as we have done in Algorithm 3.24 on graphs. As mentioned in [74] the two-step algorithm (10.15) is a modified version of a *forward-backward splitting* method, and thus, is named *Forward-Backward EM-TV* (FB-EM-TV).
Let us summarize the iterative scheme in the following:

Algorithm 10.2. (Forward-Backward EM-TV)
Let $\mathcal{K}$ be a PET operator and g the PET data as before. The forward-backward splitting algorithm to solve (10.1) is given by

$$\begin{cases} u^{n+\frac{1}{2}} = \dfrac{u^n}{\mathcal{K}^*\mathbf{1}}\mathcal{K}^*\left(\dfrac{g}{\mathcal{K}u^n}\right) & \text{(EM-step)} \\ u^{n+1} = \underset{u\in\mathcal{H}(\Omega)}{\operatorname{argmin}} \ \dfrac{1}{2}\|u - u^{n+\frac{1}{2}}\|^2_{2,\omega^n} + \alpha\|\nabla u\|_1. & \text{(TV-step)} \\ \omega^{n+1} = \dfrac{\mathcal{K}^*\mathbf{1}}{2u^{n+1}} \end{cases} \tag{10.19}$$

According to the authors of [74] the algorithm preserve the positivity when the initialization is positive and the operator is positive. For more insight into the convergence and how to compute stopping criteria we refer to [74].

Remark 10.3.
We want to note that the TV-step in Algorithm 10.2 can be solved by a standard primal-dual algorithm for weighted ROF problems as we state in Algorithm 3.24 or by applying the Cut-Pursuit algorithm that we derived in Section 4.4.

As we can see in Algorithm 10.2 the TV-step does not incorporate any evaluation of the PET operator $\mathcal{K}$ anymore. We already pointed out that the TV-step in fact is an ROF problem with a weighted L_2-data-term and that we can solve this problem with the primal-dual algorithm stated in Algorithm 3.24. This algorithm only works on the image, and consequently, is as efficient as a primal-dual algorithm to solve a weighted ROF problem on an image. On the other hand, the EM-step only evaluates the operator *once* for each outer iteration. Hence, we have split the operator evaluation from the most costly part of Algorithm 10.1. Nevertheless, EM-TV trades the operator evaluation in every primal-dual step against solving the primal-dual algorithm multiple times. Still, for huge operators with a very costly evaluation this method is favorable.

This concludes the chapter where we introduced PET data, described how to reconstruct distributions from measurements and gave insight in TV regularization methods that we need later on. In the next chapter we will introduce non-static i.e., time-dependent PET data and how to reconstruct time-dependent image sequences. We also will introduce regularization that is close to the ideas of the EM-TV method.

11
Dynamic PET Reconstruction

In practice a PET scan takes a certain amount of time, which can take up to 30 minutes in many clinical cases. For a *static* PET reconstruction - as we have investigated in the former chapters - one accumulates all tracked events for each detector pair over the whole time period of the scan and stores these counts for each line in the PET data. Then one can use any reconstruction method from the former chapters to reconstruct a static image which is time independent. The problem that arises in practice is that any living subject, e.g. a patient or a mouse, has vital functions as the heart beat or breathing. These functions change the spatial position of organs and tissue in the body, and consequently, the position of the induced tracer. This means, if we reconstruct a static image from a breathing person, we actually reconstruct the organs in different positions into one image, which yields blurry edges of the organs. This can be interpreted like the exposure time in photography where for longer exposure time any motion results in blurry images, e.g., Figure 12.5a. Thus, if one just takes the accumulated static data from the scanner and reconstructs the distribution of tracers, one will be provided with a blurry reconstruction due to the motion of the subject. This makes it especially very hard to distinguish sharp boundaries of organs in the reconstructions and to make well-founded diagnoses. In this work we denote by *motion* every actual movement in the subject, e.g. heart beat, respiration.

On top of the motion blur we also face the problem of *dynamics*. Different organs and tissues have a different amount of tracer activity at different time points. This origins from different effects, e.g. tracer distribution in the body after injection, uptake rates of different organs and decay of the tracer over time. This time dependent tracer activity is called *dynamics* and can be modeled by a function that outputs the total tracer activity at any given time point. We denote these functions as *Time Activity Curves (TACs)* or *basis functions*. These functions are linked to the spatial positions of the corresponding tissues and organs. We provided an artificial example of these functions in Figure 11.1.

To reconstruct motion in given data sets there exist many different variational methods for joint motion estimation during the reconstruction of the image (e.g. [22] and references therein). However, in this work we only consider methods to reconstruct dynamics and images. Additionally, we take a look at the influence of motion onto the reconstructed basis functions.

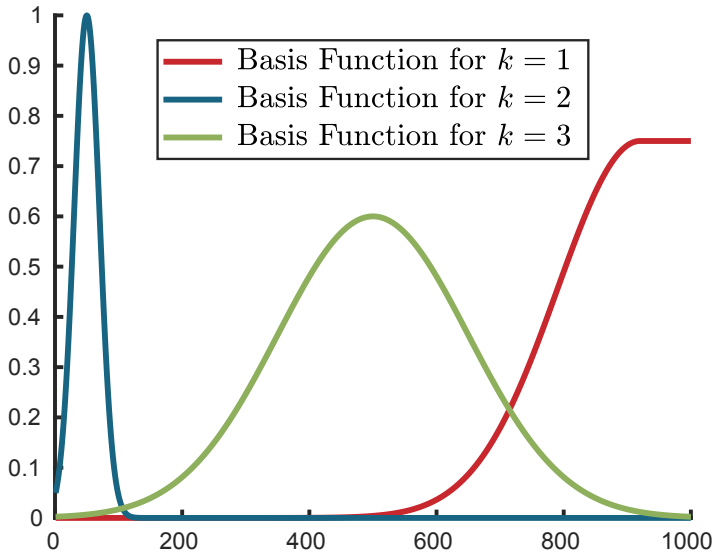


Figure 11.1.: Visualization of $N_b = 3$ artificial time activity curves (TACs) for $1,000$ time points representing three different dynamics.

In the following we will first introduce time dependent PET data that is called *listmode data* and show how to reconstruct these to a static image. Then we will introduce a method called *Fully4D* reconstruction first presented in [70]. Afterwards we will analyse the properties of these reconstructions on some artificial data and show real reconstructions and TACs of a human body PET scan.

11.1 Listmode PET Data and EM Reconstruction

As we have pointed out before we now focus on time-dependent PET data. To this end, one does not store how many events where measured by detector pairs over the whole experiment time, but gather the time points - also called *time tags* - when an event was tracked by a detector pair represented by a line. Assuming a continuous measurement, the PET data is not represented as a discrete vector with the length of the numbers of considered lines anymore. It is rather a continuous function over time that tracks if there was an event on a line l at time t or not. For now, we assume that it is possible in practice to have a infinite time resolution. Let L be the number of lines and $\mathbb{L} = \{1, \dots, L\}$ the set of lines of the scanner. Let T be the experiment time. Then this data function is defined as

$$\rho : \mathbb{L} \times [0, T] \to \{0, 1\}, \tag{11.1}$$

such that $\rho(l, t)$ equals 1 if there was an event measured on line l at time point t and 0 else. In addition, the time-dependent data we also reconstruct a time dependent "image"

$$u : \Omega \times [0, T] \to \mathbb{R}_+, \tag{11.2}$$

where $\Omega \subset \mathbb{R}^2$ a continuous image space for now. We also call these *spatio-temporal images* from now on. This implies that we also need to have a time-dependent continuous operator

$$\boldsymbol{\mathcal{K}} : \Omega \times [0,T] \to \mathbb{L} \times [0,T] \tag{11.3}$$

that maps any event at any time point to the corresponding line. Since the PET data is still distorted by Poisson noise we reconstruct the image by minimizing a continuous Kullback-Leibler data-term - analog to the discrete version in (2.9) - that is given as

$$\mathcal{D}(\boldsymbol{\mathcal{K}}u, \rho)_{KL} = \int_0^T \int_{\mathbb{L}} \left((\boldsymbol{\mathcal{K}}u)(l,t) - \rho(l,t) + \rho(l,t) \log \left(\frac{\rho(l,t)}{(\boldsymbol{\mathcal{K}}u)(l,t)} \right) \right) dl dt. \tag{11.4}$$

In practice, measuring continuously over time is not possible and one can only measure a finite number of events over a finite, sampled period of time. This comes from the technical limitations of the detectors and the hardware in a real scanner that can only have a discrete time resolution. Let T the number of possible time points that can be measured. Then we can denote every time point successive by an index in $\mathbb{T} = \{1, \dots, T\}$. Note that in practice most PET scanners have a temporal resolution of about 1 millisecond. As before the reconstruction image domain Ω is in fact a discrete space for $m \times n$ images with N equidistantly distributed pixels or voxel. We denote $\Omega_N = \{1, \dots, N\}$ as the index set of the pixel in Ω.

These considerations lead to a discrete reconstruction setting for PET data on a time-dependent (discrete) image domain $\Omega \times \mathbb{T}$ and the time-dependent data space $\mathbb{T} \times \mathbb{L}$. The so called *listmode data* is an array of time tags coupled with a line ID which implies that every data point is a single event that was tracked at a certain time point on a certain line. The listmode data thus consists of a finite amount M of events that where tracked over the measuring time. Any event m in $\mathbb{M} = \{1, \dots, M\}$ can be coupled with a certain line $l \in \mathbb{L}$ and a discrete time point $t \in \mathbb{T}$. For this triple we define the set of all events m that were measured on a line l at a time point t as

$$I = \{ \ (m,t,l) \in \mathbb{M} \times \mathbb{T} \times \mathbb{L} \ \}.$$

Further we also define $I_t = \{ \ (m,l) \mid (m,t,l) \in I \ \}$ as the set of events and line (m,l) combinations at time point t and $I_{tl} = \{m \mid (m,t,l) \in I\}$ as the events that were tracked on a certain line l at a certain time point t. Additionally, we see that the total amount of measured event is $M = |I|$, and hence define $M_t = |I_t|$ and $M_{tl} = |I_{tl}|$. The discrete corresponding data to ρ form above then is given as a vector

$$g \in \mathbb{N}^{TL}$$

which can be written element-wise as $g = (g_{tl})_{t \in \mathbb{T}, l \in \mathbb{L}}$ where each element is defined as

$$g_{tl} = \begin{cases} M_{tl}, & \text{if } I_{tl} \neq \emptyset \\ 0, & \text{else.} \end{cases} \tag{11.5}$$

This means that the number of measured events on a line l at t are tracked in g. Furthermore, one can easily see that this g is highly sparse in a case of a high temporal resolution and with

a huge set of lines. This will be important later on. The discrete time-dependent image is defined by

$$u \in \mathbb{R}_+^{NT}.$$

Following these definitions the continuous operator $\mathcal{K}$ becomes a discrete time-dependent matrix operator

$$\mathbf{K} \in \mathbb{R}_+^{TL \times NT}$$

that maps from a time-dependent image u to a given the (listmode) data domain.

Before we talk about how $\mathbf{K}$ works let us first introduce some notation. As we have everything in a vector-wise notation we will always arrange the vectors u and g as a stack of time-independent parts $u_t = (u_{it})_{i \in \Omega_N} \in \mathbb{R}_+^N$ and $g_t = (g_{tl})_{l \in \mathbb{L}} \in \mathbb{N}^L$. Then the vectors are defined as stacked vectors

$$u = (u_t)_{t \in \mathbb{T}}, \qquad g = (g_t)_{t \in \mathbb{T}}. \tag{11.6}$$

Thus, to map all the pixels in u_t for a certain t to the data g_t at the same time point t we can use the formerly introduced static matrix operator K from Section 9.2. With this in mind we would like to solve "$Ku_t = g_t$" for every $t \in \mathbb{T}$. To do this for any time point in $\mathbb{T}$ and write it as one huge system we consider

$$\mathbf{K} = \operatorname{diag}(K)_{t \in \mathbb{T}} = I_T \otimes K, \tag{11.7}$$

where I_T is a $T \times T$ unity matrix, and thus, want to solve "$\mathbf{K}u = g$". Note, that again $\mathbf{K}$ is not a diagonal matrix but the static operator matrices K are distributed as asymmetrically blocks on a diagonal.

Having derived the discrete setting with all of the notations and formulations from above for time-dependent listmode data, we are able to reformulate the continuous KL data-term (11.4) into a discrete spatio-temporal form:

$$D_{KL}(\mathbf{K}u, g) = \sum_{t \in \mathbb{T}} \sum_{l \in \mathbb{L}} (\mathbf{K}u)_{lt} - g_{tl} + g_{tl} \log\Big(\frac{g_{tl}}{(\mathbf{K}u)_{lt}}\Big). \tag{11.8}$$

To once again underline the simple construction of $\mathbf{K}$ we can rewrite (11.8) as

$$D_{KL}(\mathbf{K}u, g) = \sum_{t \in \mathbb{T}} \sum_{l \in \mathbb{L}} (Ku_t)_l - g_{tl} + g_{tl} \log\Big(\frac{g_{tl}}{(Ku_t)_l}\Big). \tag{11.9}$$

The *listmode PET reconstruction problem* then becomes

$$\operatorname*{argmin}_{u \in \mathbb{R}_+^{NT}} \sum_{t \in \mathbb{T}} \sum_{l \in \mathbb{L}} (Ku_t)_l - g_{tl} + g_{tl} \log\Big(\frac{g_{tl}}{(Ku_t)_l}\Big) \tag{11.10}$$

with a positivity constraint. Then the EM algorithm as in (9.10) can be stated as

$$u^{n+1} = \frac{u^n}{\mathbf{K}^*\mathbf{1}} \mathbf{K}^* \left(\frac{g}{\mathbf{K}u^n}\right) \tag{11.11}$$

where $\mathbf{1} = \mathbf{1}_{LT}$ is a vector of ones. This algorithm can be reformulated into a formulation

for every time point $t \in \mathbb{T}$ as

$$u_t^{n+1} = \frac{u_t^n}{K^*\mathbf{1}} K^* \left(\frac{g_t}{K u_t^n} \right) \tag{11.12}$$

with $\mathbf{1} = \mathbf{1}_L$. This is just a *static* EM reconstruction for every time point t. As mentioned before, in practice g is a very sparse vector since the time resolution is very high, and thus, only a few detector pairs have the chance to track events at a time point t. Hence, when implementing the algorithm from (11.11) we would compute the complete $\mathbf{K}u$ even though most of the entries would be multiplied by 0 from g which would just vanish from the computation. These are computation costs that one would like to avoid. To this end, we reformulate the part by changing the appearance of $\mathbf{K}$. To do so we use the notation of I, I_t and M_{tl} from above and additionally define $\bar{I}_t = \{\, l \in \mathbb{L} \mid M_{tl} \neq 0 \,\}$. Computing the right fraction of the EM algorithm in (11.12) for a fixed time point $t \in \mathbb{T}$ and fixed pixel $i \in \Omega_N$ leads to

$$\begin{aligned} \left(K^* \left(\frac{g_t}{K u_t} \right) \right)_i &= \sum_{l \in \mathbb{L}} K_{li} \frac{g_{tl}}{(K u_t)_l} \\ &= \sum_{l \in \mathbb{L}} K_{li} \frac{M_{tl}}{(K u_t)_l} \\ &= \sum_{l \in \bar{I}_t} K_{li} \frac{M_{tl}}{(K u_t)_l} \\ &= \left(K_t^* \left(\frac{\mathbf{1}_{M_t}}{K_t u_t} \right) \right)_i \end{aligned} \tag{11.13}$$

with a different operator matrix K_t that is defined as

$$K_t = (K_{[l,:]})_{l \in \bar{I}_t}. \tag{11.14}$$

This is a matrix storing the rows $l \in \bar{I}_t$ of the system matrix K where g_t respectively M_{tl} is non-zero. We can apply the same change to $\mathbf{K}$ by defining

$$\mathbf{K}_{\mathbb{T}} = \operatorname{diag}(K_t)_{t \in \mathbb{T}}. \tag{11.15}$$

Defining $\bar{g}_t = (M_{tl})_{l \in \bar{I}_t}$ and $\bar{g} = (\bar{g}_t)_t$ as corrected data with no zero entry we can state the following two equalities

$$K^* \left(\frac{g_t}{K u_t} \right) = K_t^* \left(\frac{\bar{g}_t}{K_t u_t} \right) \tag{11.16}$$

$$\mathbf{K}_{\mathbb{T}}^* \left(\frac{g}{\mathbf{K} u} \right) = \mathbf{K}_{\mathbb{T}}^* \left(\frac{\bar{g}}{\mathbf{K}_{\mathbb{T}} u} \right). \tag{11.17}$$

With these two different derivations we can state the EM algorithm to solve a time-dependent listmode PET problem with data-term (11.8) as the following scheme:

Algorithm 11.1. (Listmode-EM-Algorithm)
Let Ω be an image space with N pixels and a discrete time domain $\mathbb{T} = \{1, \dots, T\}$ with T timetags. Assume a measured data set given as $g \in \mathbb{N}^{LT}$ with M tracked events - which

implies M non-zero entries in g. Let $\mathbf{K}$ be as defined in (11.7). Then the corresponding listmode EM reconstruction algorithm is given as:

$$u^{n+1} = \frac{u^n}{\mathbf{K}^*\mathbf{1}}\mathbf{K}^*\left(\frac{g}{\mathbf{K}u^n}\right). \tag{11.18}$$

As derived above this can be reformulated to a more efficient *algorithm using the definition of $\mathbf{K}_{\mathbb{T}}$ as in (11.15) and $\bar{g}$ as above as follows:*

$$u^{n+1} = \frac{u^n}{\mathbf{K}^*\mathbf{1}}\mathbf{K}_{\mathbb{T}}^*\left(\frac{\bar{g}}{\mathbf{K}_{\mathbb{T}}u^n}\right). \tag{11.19}$$

Now that we have derived a listmode EM algorithm in Algorithm 11.1 we can start reconstructing listmode data by defining a discretization $\mathbb{T}$ of the time defined by the scanner, gathering the amount of measured events on every line l over a time interval represented by $t \in \mathbb{T}$ and reconstruct a sequence u of images that describe the tracer distribution over time. Then we would get a reconstruction of the distribution u for every pixel in every time point. One problem is that, the finer the temporal resolution gets, the less events occur in every time interval. Thus, the amount of information in every time interval is antiproportional to the level of temporal resolution. This leads to very noisy and distorted reconstructions. Consequently, this method would not be a good approach in practice especially if one would favour to inject as little tracers as possible into the patient.

In the following we look at a very straightforward approach to tackle these problems. The approach we will cover here is called *binning*. For this method one tries to gather time points with similar information in so called *bins* and reconstruct the data from each bin separately. Note that the time points can be taken from all over the temporal space $\mathbb{T}$ and do not have to be in successive temporal order, but should be taken in some rational fashion. There exist multiple methods to do so.

As we have pointed out before reconstructing a static image from data of a living subject that is breathing, whose heart is beating and might also move a little every now and then will result in blurry images by motion. When we are using binning to reconstruct multiple static images one has to select them such that the reconstructed images are as good, sharp and noiseless as possible. The position of organs in the body - and therefore the blurriness in the static reconstruction - is highly influenced by the volume state of the lung. Thus, in most cases it is important to take the information of lung volume into account when binning the data. The easiest way is to track the respiration signal during the acquisition. This can then be divided into reasonable parts that represent certain states of the lung and gather the time points accordingly into bins. Reconstructing the images for these bins will result in reconstructions with an overall good signal-to-noise ratio and less blurriness.

There exist many different methods to track the respirational signal with different hardware-driven [60, 38, 50, 29] and data-driven methods (e.g. [16, 37]). If one would like to have a good temporal reconstruction of the heart one can track the heart beat and apply the same ideas as above.

This method is a very straightforward and heuristic method that is very easy to apply to given temporal data and can be directly reconstructed by the EM algortihm (11.18). But it

still is hard to reconstruct one static image from this sequence of reconstructions without the motion blur we talked about before, but with the information of all time points. There exist multiple variational methods that work on this sequence of images or take in the temporal data and reconstruct the static image and a flow field (e.g. [22]). Also there were proposed methods that work on data taken from a PET-CT or PET-MR that takes PET and CT or MRI data simultaneously (e.g. [65, 69, 67, 53, 36]). These methods incorporate high resolution reconstructions of MRI and CT and try to generate better PET reconstructions by joining these data sets. Going more into detail here is beyond the scope of this work.

All of these mentioned methods now track *motion* in and of the body. But they all do not incorporate *dynamics* that we have introduced at the beginning of the chapter. Thus, the temporal reconstructions still are influenced by these dynamics. Furthermore, as we have selected bins that gather time points from all over the time period the image sequence is *not* successive and it is not possible to decouple events from e.g. the beginning of the measurement and some later point in time. In the following section we introduce a 4D reconstruction method called *Fully4D* that reconstructs static images and the corresponding temporal TACs accordingly. This is not only a good splitting of the dynamics from the data, it also enables the possibility to compute a time-dependent distribution u where we get a reconstruction for every time point t. This image sequence is then an actual successive reconstruction of the events in the listmode data. Afterwards we investigate some properties of this reconstruction methods especially facing the challenge of additional motion.

11.2 Fully4D Reconstruction

In this section, we will investigate the *Fully4D* PET reconstruction method formerly introduced in [70] by Andrew Reader et. al. This method attempts to split the spatio-temporal image u into N_b TACs or - as we call them in this context - (time-dependent) *basis functions* $b_k \in \mathbb{R}_+^T$ and corresponding spatial weight matrices $w_k \in \mathbb{R}_+^N$ that are defined as the static images from above. Then one can write the spatio-temporal image u pixel-wise as

$$u_{it} = \sum_{k=1}^{N_b} (w_k)_i (b_k)_t, \quad \forall i \in \Omega, t \in \mathbb{T}. \tag{11.20}$$

For simplification we denote $w_{ki} = (w_k)_i$ and $(b_k)_t = b_{kt}$. Note that we always denote a time step $t \in \mathbb{T}$, a pixel $i \in \Omega$, a basis function $k \in \{1, \dots, N_b\}$ for simplification. The image sequence u is once again arranged as a stack of static images u_t such that $u = (u_t)_t$. The static images are defined as

$$u_t = \sum_k w_k b_{kt} \in \mathbb{R}_+^N, \quad \forall t \in \mathbb{T}. \tag{11.21}$$

Note that we denote $(\cdot_t)_\mathbb{T} = (\cdot_t)_{t \in \mathbb{T}}$ from now on. With these properties and the definition of the Kronecker product in (2.21) we can rewrite u as follows

$$u = (u_t)_\mathbb{T} = (\sum_k w_k b_{kt})_\mathbb{T} = \sum_k (w_k b_{kt})_\mathbb{T} = \sum_k b_k \otimes w_k. \tag{11.22}$$

Applying the (time-dependent) operator matrix $\mathbf{K}$ defined in (11.7) onto u yields

$$\mathbf{K}u = \mathbf{K}\sum_k b_k \otimes w_k = \sum_k \mathbf{K}\,(b_k \otimes w_k) = \sum_k \mathbf{K}(w_k b_{kt})_{\mathbb{T}}. \tag{11.23}$$

This can be written even more simple. As we know that $\mathbf{K}$ is a diagonal block matrix where the static operator matrix K is distributed as a block for every time point t we can write

$$\mathbf{K}u = \sum_k \mathbf{K}(w_k b_{kt})_{\mathbb{T}} = \sum_k (K w_k b_{kt})_{\mathbb{T}}. \tag{11.24}$$

We use the former simplifications and observations to find an algorithm to reconstruct not only an image sequence u but rather the dynamics b_k and the weights w_k simultaneously from listmode PET data. To do so we minimize the following problem that is derived by plugging $\mathbf{K}u$ from (11.24) into the problem (11.10) and minimizing over w_k and b_k for all k:

$$\underset{w_k \in \mathbb{R}_+^N, b_k \in \mathbb{R}_+^T}{\operatorname{argmin}} \sum_t \sum_l \sum_k (K w_k b_{kt})_l - g_{tl} + g_{tl} \log\Big(\frac{g_{tl}}{\sum_k (K w_k b_{kt})_l}\Big). \tag{11.25}$$

The most straightforward approach is to construct an alternating algorithm that fixes b_k in one step and updates w_k and fixes w_k and updates b_k in the other step. We do so by computing an EM step for both updates depending on w_k and b_k using the KKT conditions for both updates as when deriving (9.10).

11.2.1 Update for the Spatial Weights

Let us start by deriving an update for w_k for any k. To this end, we fix the b_k's and compute an EM step for w by the use of the KKT conditions. Let us recall the formulation from above that

$$\mathbf{K}\sum_k (b_k \otimes w_k) = (K b_{kt} w_k)_{\mathbb{T}}. \tag{11.26}$$

Note that we do this in detail now and it will lead to a simple EM update step for every w_k. As in the derivation of the EM algorithm in 9.10 we will use the KKT condition again. To this end, we will compute the derivative of the terms in (11.25) with respect to w_k by computing it for the left and right term separately. let us denote $\sigma_{b_k} = \sum_t b_{kt}$. The derivative of the left part can be calculated as follows

$$\begin{aligned}
\partial_{w_k}\left(\sum_t \sum_l \sum_k (K w_k b_{kt})_l\right) &= \partial_{w_k}\left(\sum_t \sum_l \sum_i K_{li} w_{ki} b_{kt}\right) \\
&= \sum_t b_{kt}\,\big(\sum_l K_{li}\,\partial_{w_{ki}} w_{ki}\big)_{i\in\Omega_N} \\
&= \sigma_{b_k}\big(\sum_l K_{li}\big)_{i\in\Omega_N} \\
&= \sigma_{b_k} K^* \mathbf{1}.
\end{aligned} \tag{11.27}$$

The right part of the functional can be differentiated as follows

$$\begin{aligned}
\partial_{w_k}\left(\sum_l\sum_t g_{tl}\log\Big(\frac{g_{tl}}{\sum_k (Kw_kb_{kt})_l}\Big)\right) &= \sum_l\sum_t g_{tl}\partial_{w_k}\left(\log\Big(\frac{g_{tl}}{\sum_{\hat{k}} (Kw_{\hat{k}}b_{\hat{k}t})_l}\Big)\right)\\
&= \sum_l\sum_t g_{tl}\Big(-\frac{1}{\sum_{\hat{k}} (Kw_{\hat{k}}b_{\hat{k}t})_l}\Big)(K_{li}b_{kt})_{i\in\Omega_N}\\
&= -\sum_l\sum_t \frac{g_{tl}}{(Ku_t)_l}b_{kt}(K_{li})_{i\in\Omega_N}\\
&= -\sum_l (K_{li})_{i\in\Omega_N}\sum_t \frac{b_{kt}g_{tl}}{(Ku_t)_l}\\
&= -K^*\sum_t \frac{b_{kt}g_t}{(\mathbf{K}u)_t}.
\end{aligned} \tag{11.28}$$

With these derivatives we have the full derivative of the functional in (11.25) and can state the following KKT condition with a Lagrange multiplier $\lambda > 0$:

$$\begin{aligned}
0 &= \sigma_{b_k}K^*\mathbf{1} - K^*\sum_t \frac{b_{kt}g_t}{(\mathbf{K}u)_t} - \lambda\\
0 &= \lambda w_k
\end{aligned} \tag{11.29}$$

Again we multiply the upper equation pointwise by w_k and can deduce with $\lambda w_k = 0$ that

$$\begin{aligned}
&0 = \sigma_{b_k}K^*\mathbf{1}w_k - K^*\sum_t \frac{b_{kt}g_t}{(\mathbf{K}u)_t}w_k\\
\Leftrightarrow\quad &w_k = \frac{w_k}{\sigma_{b_k}K^*\mathbf{1}}K^*\sum_t \frac{b_{kt}g_t}{(\mathbf{K}u)_t}.
\end{aligned} \tag{11.30}$$

Let $u^n = \sum_k b_k^n \otimes w_k^n$, then this yields the fixed-point iteration update step

$$w_k^{n+1} = \frac{w_k^n}{\sigma_{b_k^n}K^*\mathbf{1}}K^*\left(\sum_t \frac{b_{kt}g_t}{(\mathbf{K}u^n)_t}\right). \tag{11.31}$$

This update is directly related to the EM update of a static image as we stated in (9.10).

Now as we have derived the update for w_k^{n+1} for a fixed b_k^n we want to derive the update for the basis functions in the following.

11.2.2 Update for the Basis Functions

As in the former subsection we will see that the update step for a basis function b_k is also a modified EM step. Again we will derive the derivative of the functional of (11.25) for every b_k, state the KKT condition and finally derive the update steps. As we have seen before we can reformulate $\mathbf{K}u$ as follows

$$\mathbf{K}u = \sum_k \mathbf{K}b_k \otimes w_k = \sum_k (Kw_kb_{kt})_{\mathbb{T}} = \sum_k b_k \otimes Kw_k. \tag{11.32}$$

Thus, we compute the derivative of the left and right part of (11.25) separately again starting with the left one. With (11.32) in mind we can compute the following

$$\begin{aligned}\partial_{b_k}\left(\sum_l\sum_t(\mathbf{K}u)_{tl}\right) &= \sum_l (Kw_k)_l \partial_{b_k}\big(\sum_t b_{kt}\big)\\ &= \sum_l\sum_i K_{li}w_{ki}(1)_{t\in\mathbb{T}}\\ &= \langle w_k, K^*\mathbf{1}\rangle \mathbf{1}_T.\end{aligned} \tag{11.33}$$

Note that we used the following identity

$$\sum_l\sum_i K_{li}w_{ki} = \sum_i w_{ki}\sum_l K_{li} = \sum_i w_{ki}(K^*\mathbf{1})_i = \langle w_k, K^*\mathbf{1}\rangle.$$

The next step is to compute the derivative of the right term in (11.25) which we will do pointwise for every t separately:

$$\begin{aligned}\partial_{b_{kt}}\left(\sum_l\sum_t g_{tl}\log\Big(\frac{g_{tl}}{\sum_{\hat{k}}(Kw_{\hat{k}})_l b_{\hat{k}t}}\Big)\right) &= -\sum_l (Kw_k)_l\Big(\frac{g_{tl}}{\sum_{\hat{k}}(Kw_{\hat{k}})_l b_{\hat{k}t}}\Big)\\ &= -\sum_l\sum_i K_{li}w_{ki}\Big(\frac{g_{tl}}{(\mathbf{K}u)_{tl}}\Big)\\ &= -\sum_i w_{ki}\sum_l K_{li}\Big(\frac{g_{tl}}{(\mathbf{K}u)_{tl}}\Big)\\ &= -\langle w_k, K^*\Big(\frac{g_t}{(\mathbf{K}u)_t}\Big)\rangle.\end{aligned} \tag{11.34}$$

By defining a matrix $W_k = (w_k)_{\mathbb{T}} \in \mathbb{R}_+^{N\times T}$ we can write the derivative from above as

$$\partial_{b_k}\left(\sum_l\sum_t g_{tl}\log\Big(\frac{g_{tl}}{(\mathbf{K}u)_{tl}}\Big)\right) = -W_k^*\mathbf{K}^*\frac{g}{\mathbf{K}u}. \tag{11.35}$$

Let us introduce the following notation

$$Kw_k = K_{w_k} \in \mathbb{R}_+^L \tag{11.36}$$

$$(K_{w_k})_{\mathbb{T}} = \mathbf{K}_{w_k} \in \mathbb{R}_+^{TL} \tag{11.37}$$

for a more compact way to write the algorithms and to distinguish between applying the operator K on w_k and using the precomputed vector K_{w_k}. This is essential for an efficient implementation with as little as possible operator evaluations as we will discuss later on and also state in Algorithm 11.2. Then we can deduce for (11.33) that

$$\langle w_k, K^*\mathbf{1}\rangle = \langle K_{w_k}, \mathbf{1}\rangle \in \mathbb{R}_+ \tag{11.38}$$

which is a scalar and additionally we can write (11.35) as

$$-W_k^*\frac{g}{\mathbf{K}u} = -\mathbf{K}_{w_k}^*\frac{g}{\mathbf{K}u}. \tag{11.39}$$

Combining this with (11.33) and (11.34) we get the following KKT condition

$$0 = \langle K_{w_k}, \mathbf{1} \rangle - \mathbf{K}^*_{w_k} \frac{g}{\mathbf{K}u} - \lambda \tag{11.40}$$

$$0 = \lambda b_k. \tag{11.41}$$

By multiplying again pointwise with b_k we can deduce the following update for any b_k

$$b_k^{n+1} = \frac{b_k^n}{\langle K_{w_k^{n+1}}, \mathbf{1} \rangle} \mathbf{K}^*_{w_k^{n+1}} \frac{g}{\mathbf{K}u^{n+\frac{1}{2}}} \tag{11.42}$$

where $u^{n+\frac{1}{2}} = \sum_k b_k^n \otimes w_k^{n+1}$. Note that we assumed here that w_k was updated before the update of b_k as w_k^{n+1}.

This concludes the derivation of the updates for b_k and w_k. In the following subsection we will state the complete iterative Fully4D scheme and provide an efficient numerical implementation.

11.2.3 Fully4D Algorithm and Implementation Details

In the former subsections we have derived the update steps for the basis functions b_k in (11.42) and spatial weightings w_k in (11.31). These updates can directly be applied to multiple b_k and w_k to reconstruct an spatio-temporal image $u = \sum_k b_k \otimes w_k$. Combining these we can state the following alternating Full4D update scheme:

Algorithm 11.2. *(Fully4D algorithm)*
Let $g \in \mathbb{R}_+^{LT}$ be time-dependent listmode data and $K \in \mathbb{R}_+^{L \times N}$ the static *matrix operator defined for a given PET scanner definition with L lines and an image domain with N pixels. Let $\mathbf{K} = diag(K)_t$ and $\mathbf{K}_w = (K_w)_{\mathbb{T}} \in \mathbb{R}^{NT}$ as defined before. Let us consider N_b basis functions and the scalar $\sigma_{b_k} = \sum_t b_{kt}$. Also we denote $u^n = \sum_k b_k^n \otimes w_k^n$ and $u^{n+\frac{1}{2}} = \sum_k b_k^n \otimes w_k^{n+1}$. Then an algorithm to solve the problem stated in (11.10) with images $u = \sum_{k=1}^{N_b} b_k \otimes w_k$ is given as the following fixed-point iteration:*

$$\begin{cases} w_k^{n+1} = \dfrac{w_k^n}{\sigma_{b_k^n} K^* \mathbf{1}} K^* \left(\displaystyle\sum_t \frac{g_t}{K u_t^n} \right) \\ b_k^{n+1} = \dfrac{b_k^n}{\langle K_{w_k^{n+1}}, \mathbf{1} \rangle} \mathbf{K}^*_{w_k^{n+1}} \dfrac{g}{\mathbf{K}u^{n+\frac{1}{2}}} \end{cases} \tag{11.43}$$

So we have derived a fixed-point algorithm to calculate a dynamic image over time that depends on the dynamics provided by the basis functions. The most interesting part about this is that we can reconstruct basis functions from the data without any prior knowledge about them. Still, when we have prior knowledge or we just assume to know some of the basis functions we could just fix the b_k and just update the w_k. In the following we will talk about how to construct a numerical algorithm that efficiently computes these updates. In the next section we will provide different numerical results and discuss the influence of motion onto

this reconstruction.

As we have pointed out multiple times before, we cannot assume that we are able to compute the operator matrix K in practice due to memory limitations. This leads to the problem that we have to compute a line integral over each line in $\mathbb{L}$ in every iteration of (11.43) and also for every basis function. Further, in the update for b_k we use $\mathbf{K}$ which would imply that we have to apply the operator T times. A direct naive implementation of the updates in (11.43) would then be computational very heavy if not impossible. To tackle this problem we first want to point out that in fact we only have to compute the line operator once every time we update w_k. Let us assume we just updated w_k^{n+1} for every k. Then we apply the operator for every k only *once* and get the vectors as defined in (11.36) as

$$Kw_k^{n+1} = K_{w_k^{n+1}} \in \mathbb{R}_+^L$$

which are much smaller, and thus, easily to store. As we can see in the update step for b_k^n in (11.43) we always use this vector in a scalar product with the fraction that has the size of the data. Thus, we do not need to apply the operator again for this update and exchange it with a very cheap scalar product for every time point. Then the operator only has to be applied N_b times in each iteration.

Another obvious problem is that we cannot - or at least should not, even if we can - store the resulting $u \in \mathbb{R}_+^{NT}$ since this is not sparse at all and it can be expected to have only a few zero entries. So we once again use simple tricks to deal with this. When computing $\mathbf{K}u^{n+\frac{1}{2}}$ for the update of b_k^{n+1} we can again use the fact that

$$\mathbf{K}u^{n+\frac{1}{2}} = \sum_k (K_{w_k^{n+1}} b_{kt}^n)_t$$

and that we already computed $K_{w_k^{n+1}}$ before. The same applies for the update of w_k^{n+1}. The divisor $\mathbf{K}u \in \mathbb{R}_+^{LT}$ can be computed without computing u explicitly.

Still, if we work with a scanner with millions of lines L and have a high temporal resolution T, even $\mathbf{K}u$ would be hard to store and process. That is where the next observation comes in that $\mathbf{K}u$ is in both updates only present in the divisor of the fractian and g is the dividend. Thus, if g is zero at (t, l) it is irrelevant which value we compute for $\mathbf{K}u$ at that point. This leads to the possibility to compute $\mathbf{K}u$ only at the points where g is non-zero. Then $\mathbf{K}u$ is a sparse array with only M entries at most and we can compute it as

$$(\mathbf{K}u)_{tl} = \begin{cases} \sum_k (K_{w_k})_l b_{kt}, & \text{if } g_{tl} \neq 0 \\ 0, & \text{else.} \end{cases} \tag{11.44}$$

Thus, we also only have to compute as many divisions as non-zero entries in g. With these considerations it is now possible to compute the updates with as less evaluations of the line operator and without storing huge matrices. These observations are gathered and summarized in Algorithm 8.

We will see in Chapter 12 the application of Algorithm 8 on artificial data and discuss advantages and problems of this approach. Additionally, we will provide examples where we

Algorithm 8: Fully4D algorithm for Algorithm 11.2.

Input:
$g \leftarrow$ Time-dependent listmode PET data stored in sparse array
$N_b \leftarrow$ Number of basis functions to reconstruct from data
$N \leftarrow$ Number of pixels
$L \leftarrow$ Number of lines
$T \leftarrow$ Number of time steps
$K \leftarrow$ Operator matrix for given scanner with L lines
$nIter \leftarrow$ number if iteration
Initialization:
$w_k^0 \leftarrow \mathbf{1}_N$
$b_k^0 \leftarrow \mathbf{1}_T$
$n \leftarrow 0$
Method:
while $n < nIter$ **do**
  for $k \in \{1, \dots, N_b\}$ **do**
    $w_k^{n+1} \leftarrow$ update as in (11.31) with b_k^n.
    $K_{w_k^{n+1}} \leftarrow$ apply K on w_k^{n+1}.
  $\mathbf{K}u^{n+\frac{1}{2}} \leftarrow$ as a sparse vector as described in (11.44) with w_k^{n+1} and b_k^n.
  for $k \in \{1, \dots, N_b\}$ **do**
    $b_k^{n+1} \leftarrow$ update as in (11.42) with w_k^{n+1} and $K_{w_k^{n+1}}$.
  $\mathbf{K}u^{n+1} \leftarrow$ as a sparse vector as described in (11.44) with w_k^{n+1} and b_k^{n+1}.
  $n \leftarrow n + 1$
Result: The N_b weighting matrices w_k and corresponding basis functions b_k.

combine motion and dynamics and show how the motion influences the reconstruction of the dynamics.

In the next section we will talk about how to regularize the method by applying a TV regularization onto the weighting matrices and a Tikhonov regularization onto the basis functions.

11.3 Regularized Fully4D

In this section, we want to show how one can regularize the spatial weightings and the basis functions. We know from results in Chapter 12 that reconstructions of static images tend to be very noisy. To tackle this, we introduced the TV regularization in Chapter 10 to the minimization problem (10.1) and derived different methods to solve this problem. As we have introduced in Section 10.2 the EM-TV algorithm is a relatively simple and straightforward method to construct an algorithm to solve this problem. In this section, we will provide *every* weighting function w_k with a TV regularization. Furthermore, the basis functions tend to be noisy as well, but since these functions are assumed to be smooth we want to regularize them with the L_2 norm of the gradient. This is called Tikhonov regularization [86, 65]. With these considerations, the minimization problem with the regularization parameters $\alpha_w, \alpha_b > 0$ is given by

$$\begin{aligned}\underset{\substack{w_k\in\mathbb{R}_+^N,\\ b_k\in\mathbb{R}_+^T}}{\operatorname{argmin}} \quad \sum_t\sum_l\sum_k (Kw_kb_{kt})_l - g_{tl} + g_{tl}\log\Big(\frac{g_{tl}}{\sum_k (Kw_kb_{kt})_l}\Big) \\ + \alpha_w\sum_k \|\nabla w_k\|_1 + \frac{\alpha_b}{2}\sum_k \|\nabla b_k\|_2^2.\end{aligned} \tag{11.45}$$

First take a look at the regularization of w_k. When we derive the KKT conditions for a weighting w_k, by combining the ideas of the KKT conditions from the Fully4D derivation (11.29) and the one from the EM-TV derivation (10.11), we get the following conditions

$$\begin{aligned}0 &= \sigma_{b_k}K^*\mathbf{1} - K^*\sum_t \frac{b_{kt}g_t}{(\mathbf{K}u)_t} + \alpha_w p - \lambda, \quad p\in\partial\|\nabla w_k\|_1,\\ 0 &= \lambda w_k.\end{aligned} \tag{11.46}$$

Analogously to the derivation for the TV step in (10.15) we derive a weighted ROF problem as an update given as

$$w_k^{n+1} \in \underset{w\in\mathbb{R}_+^N}{\operatorname{argmin}} \left\langle \frac{\sigma_{b_k^n}K^*\mathbf{1}}{2w_k^n}(w - w_k^{n+\frac{1}{2}}), w - w_k^{n+\frac{1}{2}}\right\rangle + \alpha_w \left\|\nabla w_k^{n+\frac{1}{2}}\right\|_1. \tag{11.47}$$

This step can be added to Algorithm 11.2 by denoting the update from the EM-step by $w_k^{n+\frac{1}{2}}$ and adding the TV-step (11.47) to it. This problem is exactly the same update rule as for the static EM-TV from (10.19) except for the scalar $\sigma_{b_k^n}$. This can also be moved from the left term to the right term without changing the solution by dividing by $\sigma_{b_k^n}$. This leads to exactly the same update rule as in (10.19) and just the regularization parameter α_w is scaled differently. These new two steps update can be seen in Algorithm 11.3.

The regularization for the basis functions b_k is differentiable since it is a squared L_2 norm. Nevertheless, we want to use the KKT conditions again to be consistent with the former derivations in this work. The derivative of the regularizer for any basis function b_k is

given as

$$\partial_{b_k} \frac{1}{2} \|\nabla b_k\|_2^2 = \nabla^* \nabla b_k = \Delta b_k \tag{11.48}$$

where Δ is the well-known Laplace operator that we also introduced on graphs in Appendix 3.B. The KKT condition for any b_k is then given as a combination of (11.40) and (11.48):

$$0 = \langle K_{w_k^{n+1}}, \mathbf{1} \rangle - \mathbf{K}^*_{w_k^{n+1}} \frac{g}{\mathbf{K}u} + \alpha_b \Delta b_k - \lambda \tag{11.49}$$

$$0 = \lambda b_k. \tag{11.50}$$

As before we multiply the upper term pointwise by b_k and then split it up into two different steps where the first one is the EM step as in (11.42) updating $b_k^{n+\frac{1}{2}}$ and a second step that is given as

$$b_k^{n+1} = b_k^{n+\frac{1}{2}} - \frac{\alpha_b}{\langle K_{w_k^{n+1}}, \mathbf{1} \rangle} \, b_k^n \, \Delta b_k^{n+1} \tag{11.51}$$

$$\Leftrightarrow \quad 0 \quad = \frac{\langle K_{w_k^{n+1}}, \mathbf{1} \rangle (b_k^{n+1} - b_k^{n+\frac{1}{2}})}{b_k^n} + \alpha_b \, \Delta b_k^{n+1}. \tag{11.52}$$

Note that this is the optimality condition of the minimization problem

$$b_k^{n+1} = \underset{b \in \mathbb{R}_+^T}{\operatorname{argmin}} \sum_t \frac{\langle K_{w_k^{n+1}}, \mathbf{1} \rangle (b_t - b_{kt}^{n+\frac{1}{2}})^2}{2 b_{kt}^n} + \frac{\alpha_b}{2} \|\nabla b\|_2^2. \tag{11.53}$$

The update of b_k^{n+1} can be computed by reformulating (11.52) by multiplying with $\langle K w_k, \mathbf{1} \rangle b_k^n$ and rewriting it as a linear system with b_k^{n+1} on the left side as follows:

$$(I + \frac{\alpha_b}{\langle K_{w_k^{n+1}}, \mathbf{1} \rangle} I_{b_k^n} \Delta) \; b_k^{n+1} = b_k^{n+\frac{1}{2}}. \tag{11.54}$$

The diagonal matrix $I_{b_k^n} = \operatorname{diag}(b_k^n)$ is defined with the values of b_k^n distributed over the diagonal. The Laplace operator Δ can be written as a matrix, and consequently this is a linear system that can be solved with standard methods. Hence, the update b_k^{n+1} is the solution of this linear system that we will denote as

$$b_k^{n+1} = \left(I + \frac{\alpha_b}{K^*_{w_k^{n+1}} \mathbf{1}} I_{b_k^n} \Delta \right)^{-1} b_k^{n+\frac{1}{2}}. \tag{11.55}$$

In practice it is not that easy to choose a good α_b since its regularization influence highly depends on the scalar $K^*_{w_k} \mathbf{1}$ and the values in b_k. The magnitude of these are not easy to anticipate before running the algorithm, because it depends on many aspects. Hence, it is also possible to use a more heuristic approach that leads to similar results by just applying a gaussian filter to the $b_k^{n+\frac{1}{2}}$. This methods works pretty well in practice even though it does not incorporate the former result b_k^n.

With these derivations we can define the overall regularized Fully4D alternating algorithm with the following 4 steps alternating iterative scheme:

Algorithm 11.3. *(Regularized Fully4D algorithm)*
Let every condition be as in Algorithm 11.2. Let $\alpha_w, \alpha_b \geq 0$ be regularization parameters for the TV regularization of the spatial weightings w_k and of the basis functions b_k respectively. Then an algorithm to approximate a solution for problem stated in (11.45) is given as follows:

$$\begin{cases} w_k^{n+\frac{1}{2}} = \dfrac{w_k^n}{\sigma_{b_k^n} K^*\mathbf{1}} K^* \left(\displaystyle\sum_t \frac{g_t}{K u_t^n} \right) \\ w_k^{n+1} \in \underset{w \in \mathbb{R}_+^N}{\operatorname{argmin}} \left\{ \left\langle \dfrac{\sigma_{b_k^n} K^*\mathbf{1}}{2w_k^n} (w - w_k^{n+\frac{1}{2}}), w - w_k^{n+\frac{1}{2}} \right\rangle + \alpha_w \left\| \nabla w_k^{n+\frac{1}{2}} \right\|_1 \right\} \\ b_k^{n+\frac{1}{2}} = \dfrac{b_k^n}{K^*_{w_k^{n+1}}\mathbf{1}} \mathbf{K}^*_{w_k^{n+1}} \dfrac{g}{\mathbf{K} u^{n+\frac{1}{2}}} \\ b_k^{n+1} = \left(I + \dfrac{\alpha_b}{K^*_{w_k^{n+1}}\mathbf{1}} I_{b_k^n} \Delta \right)^{-1} b_k^{n+\frac{1}{2}} \end{cases} \tag{11.56}$$

Now we are not only abele to reconstruct the dynamics from the listmode PET data and get a real sequence of images over time, but furthermore also to denoise the spatial weighting with a TV and the basis functions with a Tikhonov regularization. We will conclude this chapter by showing numerical results and point out some interesting observations.

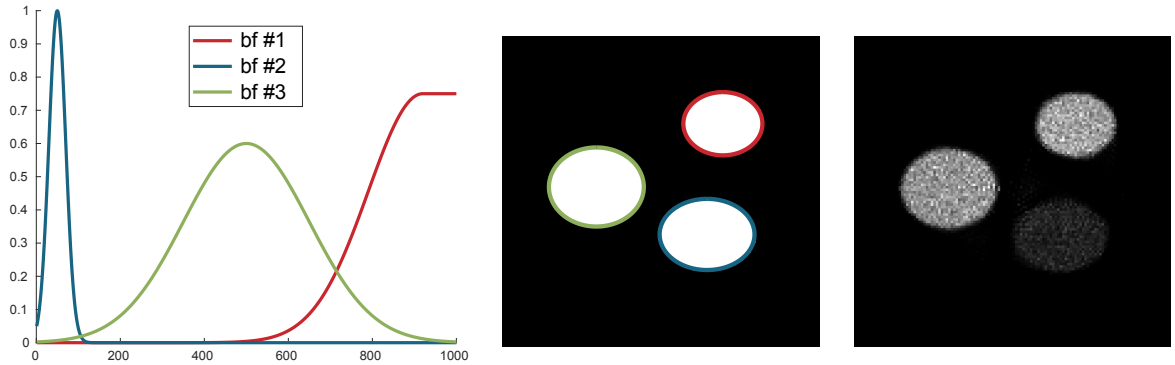
12

Numerics: Dynamic PET Reconstruction

In this chapter we apply the formerly derived *Fully4D* in Algorithm 8 and its regularized variant in (11.56) to artificial listmode data corresponding to $2D$ temporal PET data. We also investigate the provided spatial weights and basis functions for moving objects to get an intuition of how the dynamics are influenced by motion.

The artificial listmode data is generated with a custom implemented listmode data simulation tool. We can predefine an artificial scanner with a certain number of detectors and possible detector pairs. We use various types of ellipses as phantoms, with predefined basis functions that are linked to the phantoms. Then we generate randomly distributed time tags over a certain time period and produce an event for every time tag. The position of the event is computed by randomly selecting an object with a probability that is proportional to the size of the phantom, some predefined intensity value and the current value of the linked basis function at that time point. After selecting the object a random position inside of it is selected; this is where the event should occur. Then we compute a random angle, cast a line with that angle through the position and compute the two target detectors and store the detector pair ID. With this process we can store artificial listmode data and use this for the algorithms. Note that these computations are done in double-precision, and hence, the computed position is not dependent on any preselected discretization of the image space. This gives us the possibility to reconstruct in any image resolution as long as the detectors can cover it. We see an example of three different objects in Figure 12.1b and the three linked basis functions in Figure 12.1a. The colors in both figures indicate the link between an object and its corresponding basis function.

Note that we do not provide any run time comparison since the goal of this chapter is to present a proof of concept of the proposed algorithms. Nevertheless, it is easy to see that the run time scales with the number of expected basis functions. For example, let us assume that we expect N_b basis functions and we want to iterate for 100 iterations. For each spatial weight we have to compute one EM-step in every iteration, and therefore, we have in fact a full EM reconstruction with 100 iterations for every spatial weight matrix. Consequently, this algorithm is as costly as N_b times 100 EM iterations. In addition we have the rather cheap computations of the basis functions. The application of the TV-regularized Fully4D is even worse as we also have to solve the weighted ROF problem in each iteration for every spatial



(a) 3 Basisfunctions (b) Groundtruth (c) Static EM

Figure 12.1.: Artificial data set of three basis functions and three objects. The colors indicate the influence of basis function to object. On the right we see an EM reconstruction from the static data provided by the simulation.

weighting using a primal-dual algorithm. In combination this algorithm is very expensive so that we only provide examples on rather small image discretizations as a proof of concept.

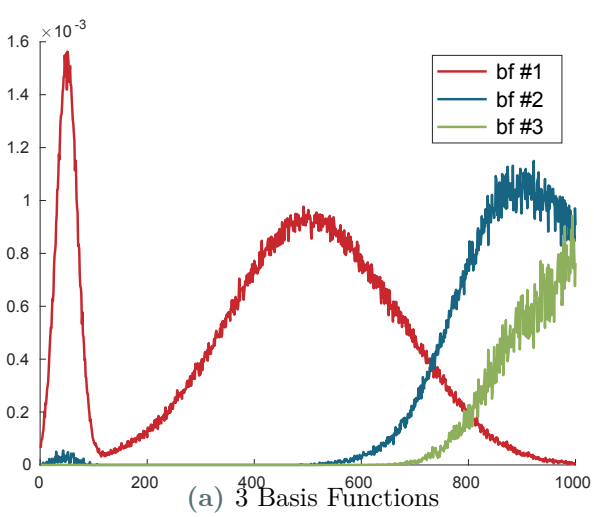
We start by a simple reconstruction of the three objects and basis functions from above.

12.1 Dynamic Data without Motion

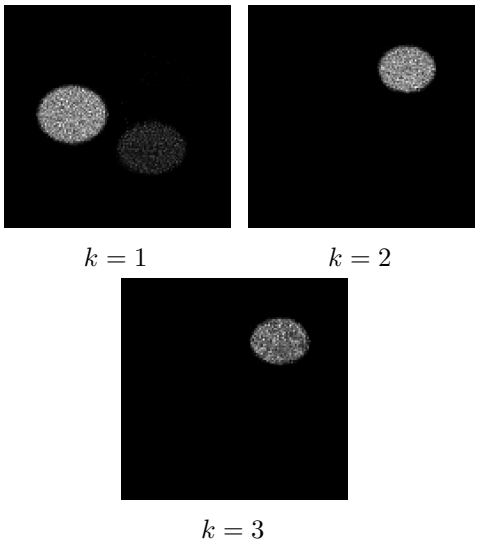
In this section, we investigate data that was generated for $1,000$ time tags with approximately one million generated events for the objects in Figure 12.1b linked with the three basis functions in Figure 12.1a. Every object is only influenced by one basis function. We reconstruct this problem at a resolution of 128×128 pixels. After the generation of the listmode data we can build a *static* data set from the listmode data - as we have described in the former chapter - by counting the events that were tracked by every detector pair. This static data then can be reconstructed by the EM algorithm we presented in (9.10). The resulting reconstruction can be seen in Figure 12.1c where we observe that the lower object is barely visible. This comes from the pointy basis function that is linked to that object and is indicated by the blue line in Figure 12.1a. The object is only active for a very short amount of time and inactive afterwards. Hence, when reconstructing the static data set this object has barely visible intensities.

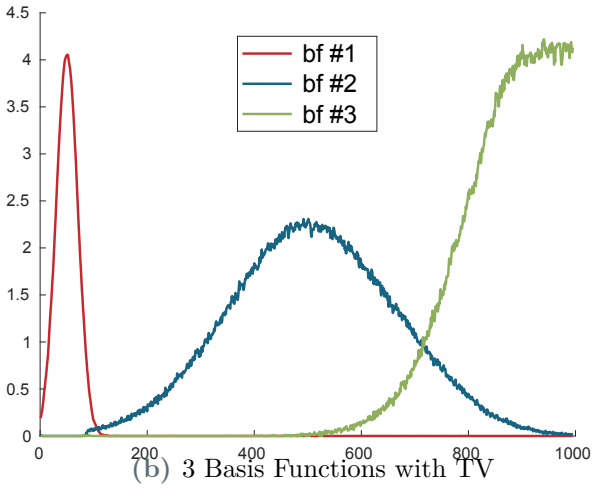
As we are given three basis functions as a ground truth, let us investigate the results for applying the algorithm with an expected number of three basis functions, i.e., $N_b = 3$. In an optimal case, it would reconstruct the three objects and three basis functions perfectly. The result of the algorithm for 100 iterations can be seen in Figure 12.2. We normalize every spatial weight matrix by dividing it by its maximum value and multiplying the corresponding basis function with this value. With this method we have a more consistent appearance of the reconstructions. Note, that in the end one would compute the spatio-temporal solution u at a time point t by summing up the weight matrices multiplied by the current value in the basis function, i.e., $u(x,t) = \sum_{k=1}^{N_b} w_k(x) b_k(t)$. Hence, when we have basis functions that are both active in one time point we have to combine these for our analysis.

We see in Figure 12.2a that the algorithm reconstructed three basis functions that describe the overall dynamics in the data well, but which are not capable of separating the objects



(a) 3 Basis Functions





(b) 3 Basis Functions with TV

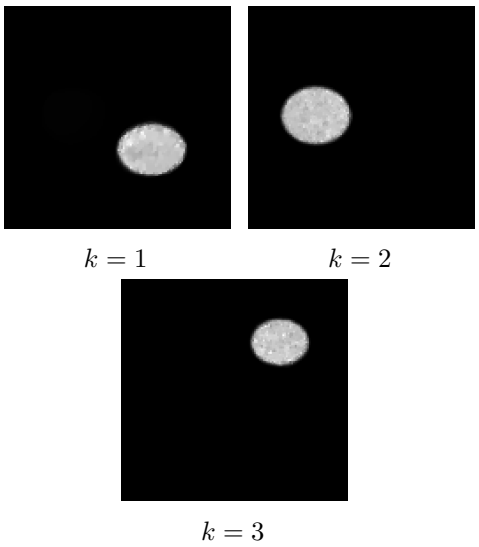
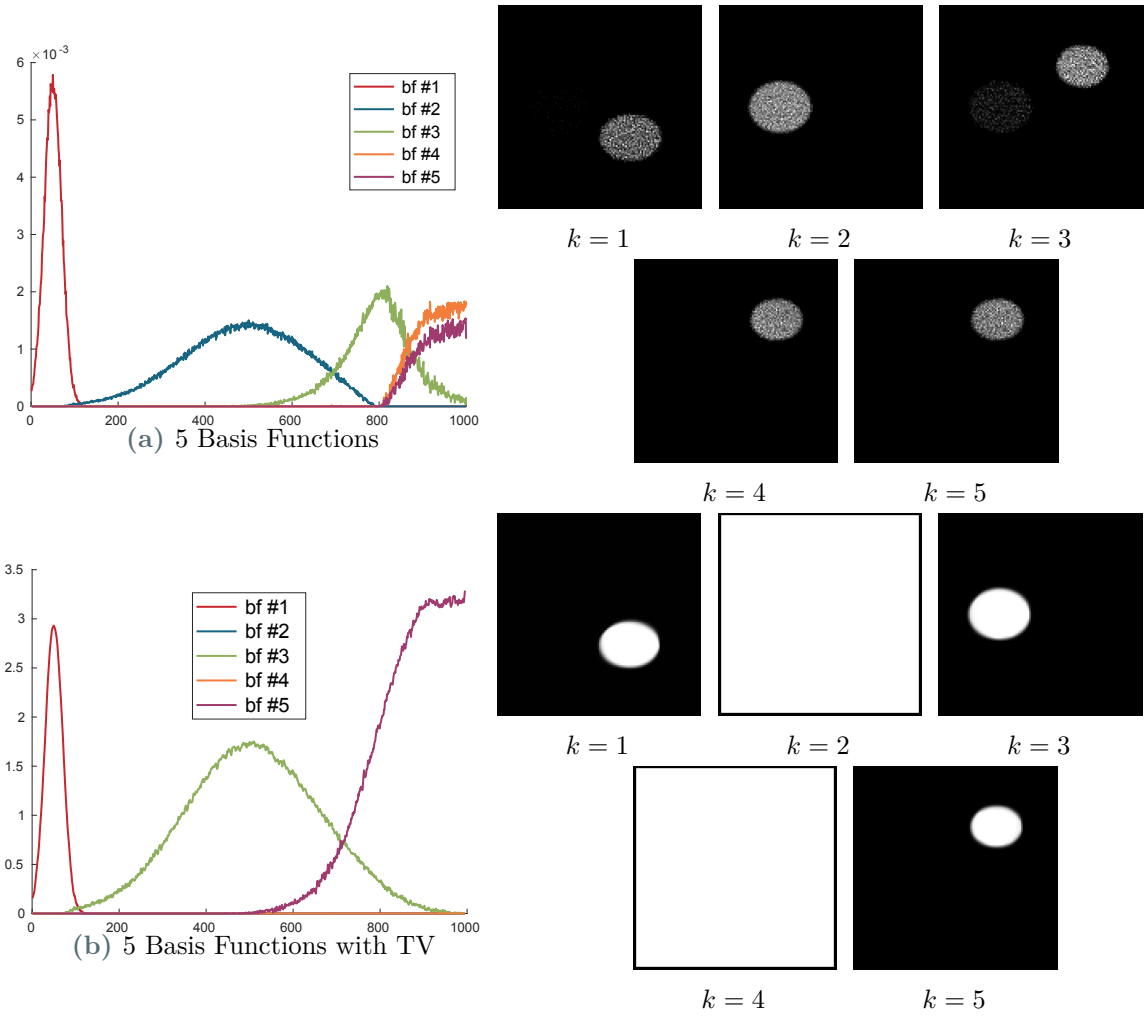


Figure 12.2.: Comparison of Fully4D reconstruction to reconstruct $N_b = 3$ basis functions and weights with and without TV regularization.

correctly. As we see, the red line for $k = 1$ actually describes the lower right object *and* the upper left object as it combines the ground truth basis functions for $k = 1$ and $k = 3$ from Figure 12.1a. On the other hand combining the reconstructed basis functions $k = 2$ and $k = 3$ results in the original basis function $k = 2$. Hence, the upper right object is described twice in the reconstruction. Additionally, we see that the basis functions are pretty noisy.

Now let us apply the TV regularization to the spatial weights and the Tikhonov regularization to the basis functions as we introduce in Section 11.3 and state in (11.3) on this data set. For this problem we set the regularization parameter for the TV regularization as $\alpha_w = 0.225$ and the parameter for the Tikhonov regularization to $\alpha_b = 10^9$. Note that the α_b always has to be set to a high value as it is scaled with $(K^*_{w_k^{n+1}}\mathbf{1})^{-1}$ as we see in (11.3). We ran this algorithm for 30 iteration and solved the TV-step with a primal-dual algorithm for spatially-isotropic TV regularization - we defined it in Definition 3.15 - until convergence



(a) 5 Basis Functions

(b) 5 Basis Functions with TV

Figure 12.3.: Comparison of Fully4D reconstruction to reconstruct $N_b = 5$ basis functions and weights with and without TV regularization.

(primal-dual-gap lower than 10^{-5} cf. [17]). The result can be seen in Figure 12.2 in the bottom figure where we see perfectly separated basis functions and corresponding objects in the weighting matrices. Hence, the Fully4D algorithm incorporated with TV regularization of the spatial weighting and Tikhonov regularized basis functions yields a major improvement in the separation of objects with different dynamics and in the tracking of the underlying dynamics.

Let us investigate the results of another experiment where we do not change the input data, but increase the number of basis functions that we expect to $N_b = 5$. We hope for a better separation of the basis functions and the spatial weights such that in no weighting matrix appears more than one object. Again we run the Fully4D algorithm with 100 iteration and the regularized Fully4D algorithm with 30 iterations and a converged TV-step. The regularization parameters have been set to $\alpha_w = 20$ and $\alpha_b = 10^9$ again. As we can see in Figure 12.3, it actually yields a better reconstruction as the red line and the blue line are now separated and not combined in a single line as in Figure 12.2a. The basis functions for $k = 4$ and $k = 5$ represent the upper right object as in the former experiment. The problematic

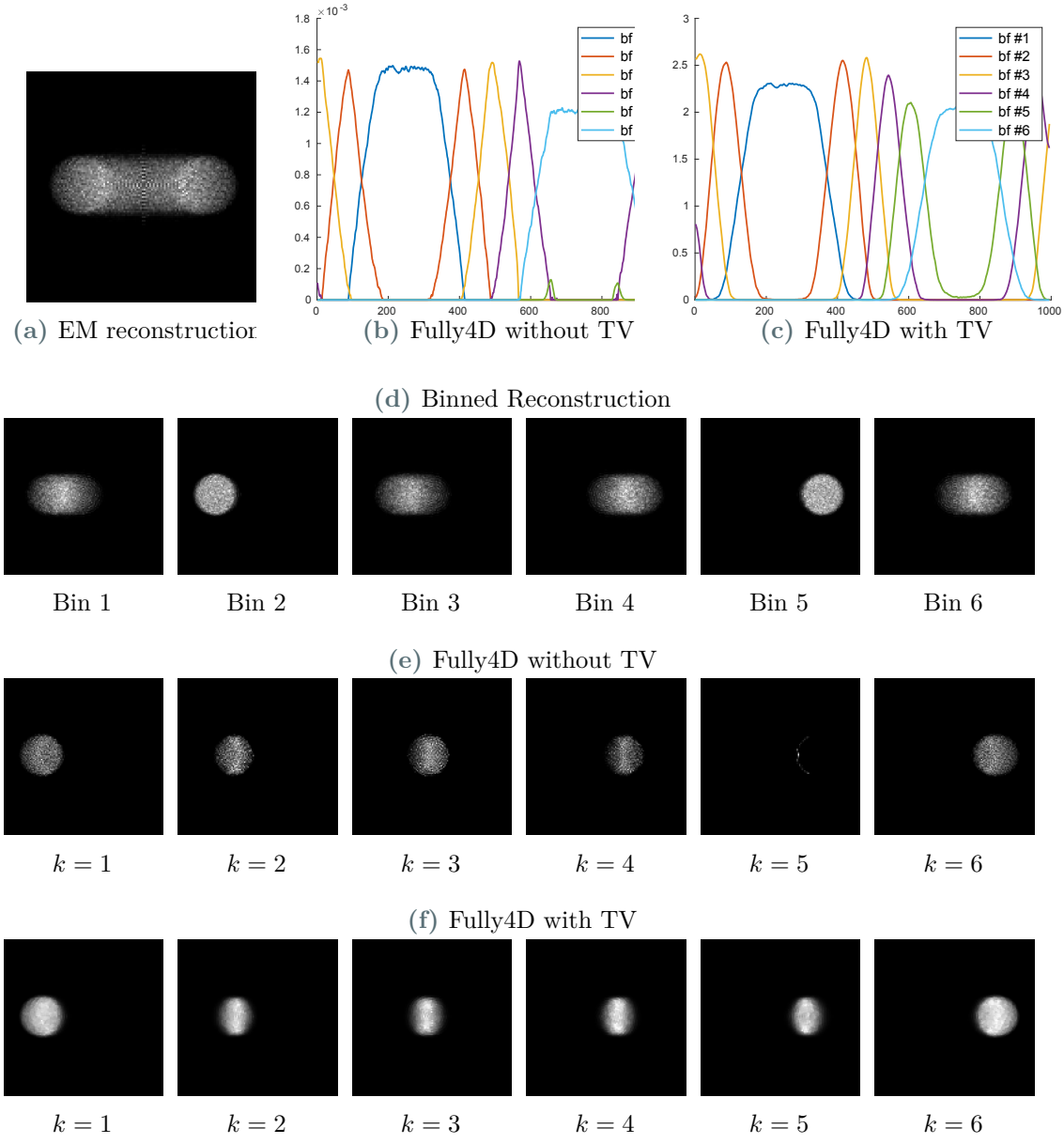
result is the green basis function for $k = 3$ as it seems to be somewhere between the upper right and the left object as we also see in the weighting matrix for $k = 3$. Moving over to the regularized Fully4D solutions we can see that the result does not only perfectly separate the different basis functions and reconstruct their actual shape, but also only finds three active basis functions and two inactive zero basis functions. Looking at the ground truth basis functions in Figure 12.1 we see that this is an almost perfect reconstruction. Consequently, the regularized version of the Fully4D algorithm does not only provide a better separation of the basis functions - if we know the exact number of them - it also does suppresses additional basis functions which do not provide any new information. Hence, we do not need to somehow recombine basis functions in a post-processing step to get only *one* basis function for each object.

In fact, these results can be expected from the regularization with TV. If one sets a high regularization parameter, it is very costly for the TV term to have the same object in two weighting matrices without providing any further dynamic information. Consequently, the TV regularizer tries to suppress any additional object in the results, and thus, also any additional basis function.

Let us now introduce additional motion into the data and investigate the basis functions provided then.

12.2 Dynamic Data with Motion

In this section, we want to investigate how the Fully4D algorithm can handle dynamic data with additional motion of the objects. To this end, we start with a very simple example where we have one object that moves from the middle to the left, then to the right and to the middle again with no dynamics at all. As we see in Figure 12.4a, the EM reconstruction of a moving objects results in a blurry moving object. As we have discussed in the previous chapter, an easy way to reconstruct a sequence of images describing the movement of the object is by gathering data from certain time points into multiple *bins* and reconstruct the data in these bins separately. For this example we divided the time line of the experiment into six equidistant parts and reconstructed the data therein. The resulting sequence of images can be seen in Figure 12.4d which describes the movement of the object as expected. Now the question arises what happens when we apply the Fully4D algorithm and expect $N_b = 6$ basis functions. As we know, we have not added any dynamics to the data, and hence, as we see in Figure 12.4b the basis functions have similar maximum height and seem to take turns in being active and inactive. Looking at the weight functions in Figure 12.4e we can observe that each weight corresponds to one position of the object in its movement. Note that these images are not ordered as the bins are. In fact, the basis functions tell in which time interval which spatial weight matrix describes the current state of the object. Hence, one can interpret each basis function as one disjoint bin as for example the orange, yellow and purple lines have multiple peaks, and hence, activate their corresponding spatial weight multiple times. This makes sense as the object passes some positions multiple times in its movement. In addition, we see that the basis functions overlap at many points which implies that the spatio-temporal solution u blends the corresponding scaled spatial weight matrices, and therefore, provides a more smooth sequence as the binning does. In this case the regularized Fully4D algorithm



(a) EM reconstructior (b) Fully4D without TV (c) Fully4D with TV

(d) Binned Reconstruction

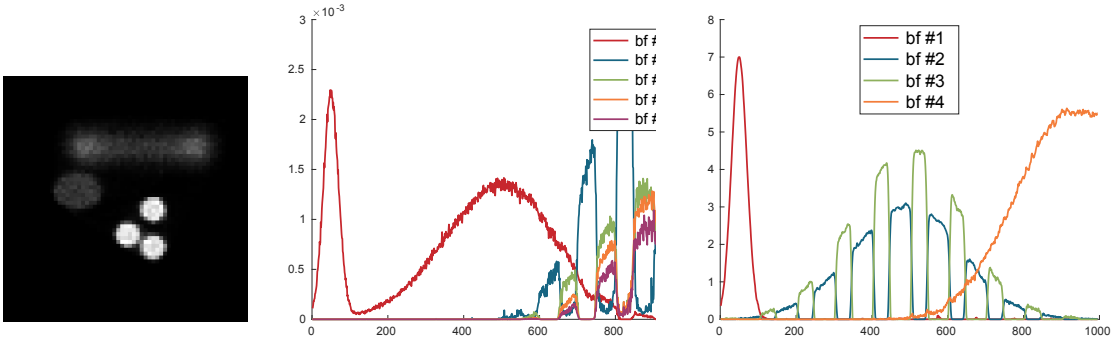
(e) Fully4D without TV

(f) Fully4D with TV

Figure 12.4.: Object moving from the left to the right. Binned reconstruction with six equidistantly selected bins and Fully4D reconstruction in without and with TV regularization for $N_b = 6$ expected basis functions.

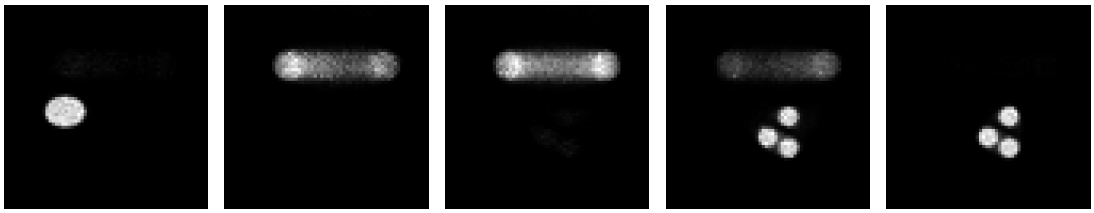
outputs a very similar solution, but does not have the strange solution for $k = 5$ as in the not regularized algorithm.

So as we have seen, the Fully4D solution is influenced by motion in the data. This problem arises from the fact that dynamics are a change in activity over time in a certain region. Hence, a moving object that passes a certain region multiple times also induces trackable dynamics for this region. Hence, moving objects do induce multiple basis functions even though they could be represented by one basis function and some motion field. This implies that applying Fully4D to not motion corrected data might have the side effect that one has to tell the algorithm to look for more basis functions than actually needed which makes the algorithm much more expensive.



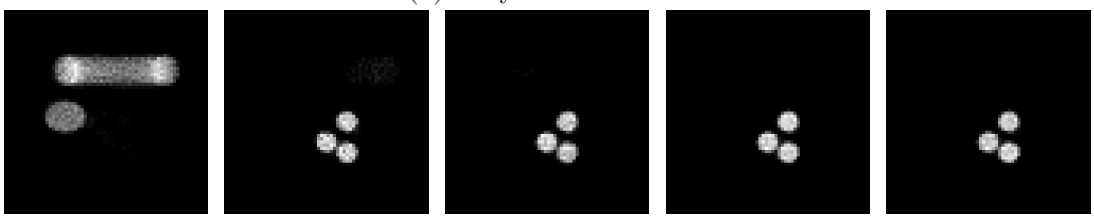
(a) EM reconstruction (b) Fully4D without TV (c) Fully4D with TV

(d) Binned Reconstruction



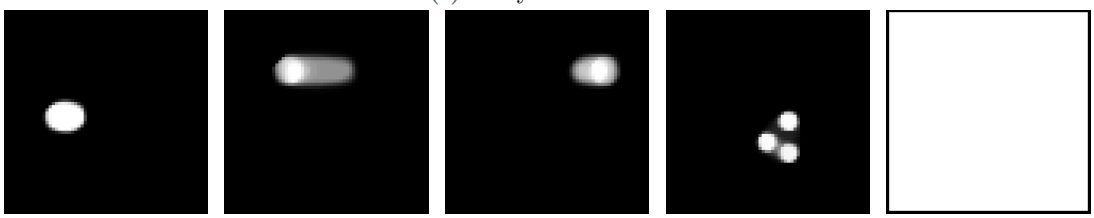
Bin 1 Bin 2 Bin 3 Bin 4 Bin 5

(e) Fully4D without TV



$k = 1$ $k = 2$ $k = 3$ $k = 4$ $k = 5$

(f) Fully4D with TV



$k = 1$ $k = 2$ $k = 3$ $k = 4$ $k = 5$

Figure 12.5.: Object moving from the left to the right and additional objects. Data is provided with dynamics from Figure 12.1a. Binned reconstruction with six equidistantly selected bins and Fully4D reconstruction in without and with TV regularization for $N_b = 6$ basis functions.

In the following example, we want to look on multiple still objects and one moving object all provided with a basis function. We want to investigate if the algorithms can reconstruct the actual dynamics in the data and the motion induced dynamics. The example consists of a total of five objects which are linked with the basis functions in Figure 12.1a. One object moves from the left to the right multiple times during measurement and is linked with the green line in Figure 12.1a. A second object on the left is linked with the pointy blue function.

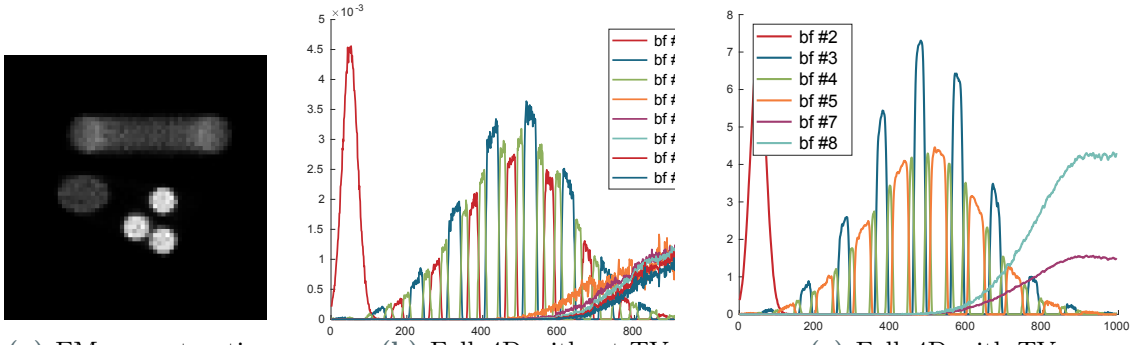
In addition, we also provide three small objects in the lower right with the red function in Figure 12.1a.

Hence, we face the problem of a moving object distorting the basis functions and actual underlying dynamics. We want to investigate in Figure 12.5 how this influences the results. In these examples we reconstruct 64×64 images. We can see a static EM reconstruction in Figure 12.5a where the upper object is distorted by motion blur and the other objects have different intensities due to the dynamics. In Figure 12.5d we can see that the left object is in fact only active at the beginning and the triplet comes in at a later time point. We again apply the Fully4D algorithm and run it for 100 iterations with $N_b = 5$ basis functions. We can see in Figure 12.5b, that the first basis function for $k = 1$ seems to be a combination of the pointy function and the dynamics of the moving object as we can see in the corresponding weighting matrix in Figure 12.5e for $k = 1$. Hence, the motion is not represented in the basis functions which yields to a blurry reconstruction in the spatial weight matrix for $k = 1$. The rest of the basis functions and weights describe the triplet in the lower right. Note that the high values in the blue line for $k = 2$ only look so out of place because the green, orange and purple describe the same time interval, and hence, their heights have to be added up.

Now let us look at the results for the application of the regularized Fully4D. We again ran the algorithm for 30 iterations. For the TV regularization of the weight matrices we selected $\alpha_w = 7$ and for the Tikhonov regularization of the basis functions again $\alpha_b = 10^9$. We can see in Figure 12.5c that the dynamics are separated well and also the dynamics are represented nicely. The moving object is described by two basis functions and weighting matrices. In fact, the basis functions again represent different spatial positions of the object and alternate between each other. In addition the dynamics of the object is tracked very well when we combine the reconstructed basis functions for $k = 2$ and $k = 3$ compared to the ground truth in Figure 12.1a. Nevertheless, for some reason the total variation seems to force one weighting matrix to be constant 1 and the basis functions to be constant zero. Still, the results are much better than the results of the unregularized Fully4D algorithm.

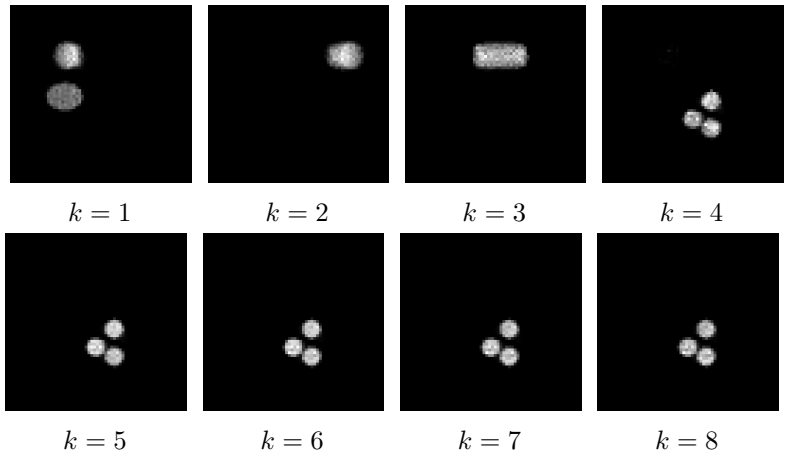
Finally, let us investigate the same problem as above, but now try to achieve that the Fully4D actually separates the moving object from the left object and also depicts the motion with basis functions better. To this end, we have set the expected number of basis functions to $N_b = 8$. We present these results in Figure 12.6 where we see that this time the basis functions actually separate the movement of the object into three parts. Also the dynamics are tracked very well when combining these basis functions. However, it still cannot separate the moving object from the left object. In addition to that it also describes the triplet in the lower right by *five* basis functions and weighting matrices. When comparing this result to the TV-regularized result in Figure 12.5c for expected 5 basis functions we can deduce that the TV-regularized Fully4D algorithm has a better representation of the dynamic and movement for less expected basis functions.

Let us now take a look at the TV-regularized results for $N_b = 8$ expected basis functions and regularization parameter $\alpha_w = 8$. First, we see that for $k = 1$ and $k = 6$ the basis functions are actually zero everywhere, and hence, we can drop the weight matrices. Again this solution does separate the dynamic perfectly, even though in this case it uses two functions and weighting matrices to describe the triple. In the weights for $k = 3$, $k = 4$ and $k = 5$



(a) EM reconstruction (b) Fully4D without TV (c) Fully4D with TV

(d) Fully4D without TV



(e) Fully4D with TV

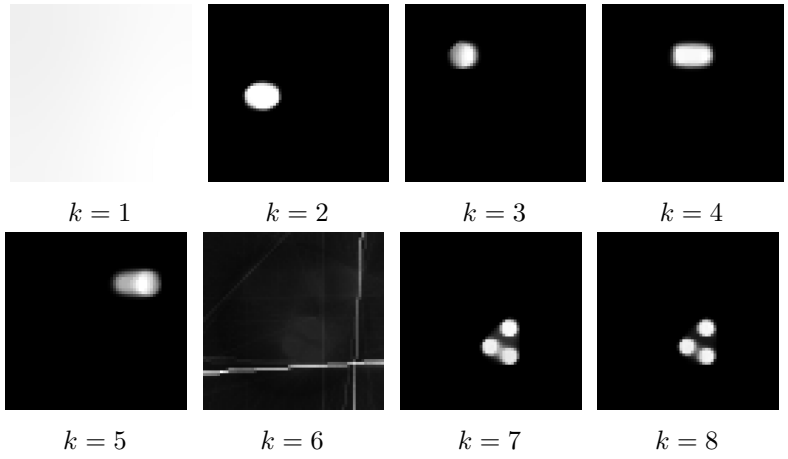


Figure 12.6.: Object moving from the left to the right and additional objects. Data is provided with dynamics from Figure 12.1a. Reconstruction for $N_b = 8$ basis functions with Fully4D reconstruction in without and with TV regularization.

we see that the motion is depicted by three different positions and that in this case they are completely separated from other objects in the data. Hence, again the regularized Fully4D does yield more satisfying results.

The conclusion of this part can be found in Section 15.2.

Part III
Cut-Pursuit Based Reconstruction

13
Cut-Pursuit on PET Reconstruction

In this chapter we introduce a Cut-Pursuit algorithm for PET reconstruction with added TV regularization as we already introduced in (10.1) in Chapter 10. As we know from Chapter 4, the Cut-Pursuit algorithm can be applied to regularization problems with differentiable data-terms and non-differentiable but directional differentiable regularizers. In Section 9.2 we already discussed how the PET operator $\mathcal{K}$ works and that its evaluation can be problematic in high resolution settings. In Chapter 4 we have seen that Cut-Pursuit is not only a very efficient denoising method, but also splits the image space into segments such that iterative algorithms work much faster on the resulting coarser discretizations. The aim of this chapter is to derive a Cut-Pursuit algorithm that solves problem (10.1) more efficiently than direct iterative approaches and that uses reduced PET operators that only live on coarse partitions of the image space. We start by translating the former reconstruction setting into a graph setting and derive the corresponding minimization problem on graphs.

13.1 Graph Setting for Reconstructions

Before constructing a corresponding graph setting to deal with reconstruction problems let us start by recalling the ideas and definitions of Chapter 9. In Chapter 9 we introduced a discrete reconstruction setting with a scanner definition that is defined by L lines in $\mathbb{L} = \{1, \dots, L\}$ and a reconstruction image space Ω of $m \times n$, or in 3D of $m \times n \times d$, with a total of N pixels. We denote by $\Omega_N = \{1, \dots, N\}$ as the index space for the pixels in Ω. We defined the PET operator $\mathcal{K} : \Omega \to \mathbb{R}_+^L$ and also stated its matrix representation $K \in \mathbb{R}_+^{L \times N}$. For all of the following assumptions and definitions it does not matter if we investigate a 2D or 3D setting since the math is the same, but only the construction of the graph is different. For simplification we will from now on always say *pixel*. To translate this setting into a graph setting we first point out that every pixel in the image domain Ω is tagged with an unique number $i \in \Omega_N$. Thus, can directly state that $V = \Omega_N$.

The only difference between the image space Ω and V is that the image space Ω knows the location of every pixel i in the $m \times n$ image or $m \times n \times d$ image. To represent this information one defines an edge set E that connects neighboring pixels as we already described in detail Chapter 6 and show a visualization of such a graph in Figure 6.1. In 2D we connect every vertex with the vertices representing the four direct neighbors of the corresponding pixel.

On the other hand to represent the 3D volume image we connect the six neighboring voxel of every voxel. Equivalently to the definition of finite differences as the derivatives in the common image setting we will set the weights $w : E \to \mathbb{R}_+$ to 1 for every edge in E. Note, that in practice it might be interesting to modify the weight with some prior knowledge about the structure of the reconstruction or even change the construction of the edge set. For example, one could think of taking an MRI reconstruction from the same object and setting the weights according to properties of edges in the MRI image. This might be related to some kind of joint reconstruction as introduced in e.g. [65, 67, 25].

Finally, we want to mention that extending the connection of the graph by adding edges to diagonal neighbors, and weight them correctly, might give more *isotropic* results in an spatial image sense. This happens even when we solve with a *spatially-anisotropic* TV regularization which we talk about in Section 3.1.4. For more insight we refer to Chapter 6 where we discuss these results in detail.

As a result, we are now able to define an undirected finite weighted graph $G = (V, E, w)$ to represent the images in the image domain Ω. The PET operator $\mathcal{K}$ is directly translated to the graph setting as $\mathcal{K} : V \to \mathbb{R}_+^L$ since it only operates on the vertex set V. Since we use the PET operator $\mathcal{K}$ only in a discrete setting from now on we represent it by the corresponding - very sparse - matrix $K \in \mathbb{R}_+^{L \times N}$ as we discuss in Chapter 9. As we have pointed out before in Section 9.2 using the matrix might be easier for the understanding of derivation, but one can still interchange both interpretations. With theses definitions any method that we introduce in Chapter 10 to solve (10.1) can be also used to solve the problem in the graph setting represented by G - as we will define in (13.1) - and to compute the same solution. This lays the foundation for solving regularized reconstruction problems on graphs by applying Cut-Pursuit algorithm.

13.2 Cut-Pursuit for Regularized PET Reconstruction

In this section, we translate the problem (10.1) into a graph problem and apply the Cut-Pursuit algorithm to solve it. The undirected finite weighted graph is defined in the former section, i.e., a image grid or volume graph, and given as $G = (V, E, w)$ and $g \in \mathbb{R}_+^L$ is the (static) PET data. In this section, we denote the functions in the image space as f instead of u to be consistent with the formulations in Chapter 4 where we derived the Cut-Pursuit algorithm. The optimization problem (10.1) in on a graph G is given as

$$\operatorname*{argmin}_{f \in \mathbb{R}_+^N} \sum_{l=1}^{L} (Kf)_l - g_l \log(Kf)_l + \frac{\alpha}{2} \|\nabla_w f\|_1 + \delta_+(f). \tag{13.1}$$

Remark 13.1. *The edges in E are always directed in both directions with the same weight as we already pointed out in Chapter 3. Thus, computing the TV seminorm of the gradient $\nabla_w f$ is always two times the result when using finite differences where only one direction is incorporated. Consequently, we have to divide the regularization parameter α by 2 to get the same results as we also discuss in Section 3.3.1.*

Let us start by analyzing the terms of (13.1) separately and see how they can be translated to the partition problem of (4.10). As we know, this problem is 1D on the graph, i.e., the

vertex function f maps to $\mathbb{R}_+$. Hence, we are in the channel-anisotropic setting and do not have to compute any directions. Additionally, we know that the Cut-Pursuit algorithm converges to the optimum of (13.1) (cf. [46]).

The (discrete) Kullback-Leibler data-term defined on G is the same as on the image since it lives on the pixels and is given as

$$D_{KL}(f) = \sum_l (Kf)_l - g_l + g_l \log\Big(\frac{g_l}{(Kf)_l}\Big). \tag{13.2}$$

This term is differentiable with respect to f as we have shown in (13.3) where we already computed the analytic derivative as

$$\nabla D_{KL}(f) = K^*\mathbf{1} - K^*\frac{g}{Kf}. \tag{13.3}$$

The second term is the total variation on graph G which is the same regularization as in the ROF problem for which we already introduced the Cut-Pursuit algorithm explicitly in Chapter 4 and which is given in Algorithm 2. So the remaining question is how to handle the positivity constraint. As we know from Proposition 4.10 we plug the former solution of the reduced problem f_{Π^n} into the partition problem to compute the $(n+1)$'th partition Π^{n+1}. Since the reduced problem is provided with a positivity constraint the solution of the n'th reduced problem f_{Π^n} has to be positive everywhere. Thus, the term $\delta_+(f_{\Pi^n})$ is only evaluated at positive points, and consequently 0.

We have gathered all the information that is necessary to construct the Cut-Pursuit algorithm to solve (10.1). Let us assume that we are in the $(n+1)$'th iteration of the Cut-Pursuit algorithm. Then we are given the former partition Π^n and the solution of the n'th reduced problem $f_{\Pi^n} \in \mathbb{R}_+^N$ - that is piece-wise constant on Π^n. Let $G_r^{n+1} = (\Pi^{n+1}, E_r^{n+1}, w_r^{n+1})$ be the reduced graph corresponding to Π^{n+1}. Recalling the definition of P_n in (4.10) we can find a vector $c^n \in \mathbb{R}_+^{N_{\Pi^n}}$ with N_{Π^n} such that

$$f_{\Pi^n} = \sum_{A\in\Pi^n} \mathbf{1}_A c_A^n = P_n c^n \in \mathbb{R}_+^N \tag{13.4}$$

with $c^n = (c_A^n)_{A\in\Pi}$. The reduced problem defined on the reduced graph G_r^{n+1} is then given as

$$c^{n+1} = \operatorname*{argmin}_{c\in\mathbb{R}_+^{|\Pi^{n+1}|}} \sum_{l=1}^{L} (KP_{n+1}c)_l - g_l \log(KP_{n+1}c)_l + \frac{\alpha}{2}\|\nabla_{w_r^{n+1}} c\|_1. \tag{13.5}$$

Given the solution of the n'th reduced problem c^n and $f_{\Pi^n} = P_n c^n$ the partition problem to compute Π^{n+1} would be then given as

$$B^{n+1} \in \operatorname*{argmin}_{B\in\mathcal{P}(V)} \langle K^*\mathbf{1}_V - K^*\frac{g}{Kf_{\Pi^n}} + \alpha\nabla R_S(f_{\Pi^n}), \mathbf{1}_B\rangle + \frac{\alpha}{2} R'_{S^c}(f_{\Pi^n}; \mathbf{1}_B). \tag{13.6}$$

This can again be computed by a graph cut, as we describe in Section 4.2.2. The partition Π^{n+1} is constructed as in (4.55) by taking the connected components of all intersections $A \cap B^{n+1}$, $A \cap (B^{n+1})^c$ for all $A \in \Pi^n$. In both updates (13.5) and (13.6) we apply the

operator K to a function f_c that is constant on the segments of the corresponding partition. For this reason, let us take a closer look into applying K to a function $f_c = P_\Pi c = \sum_{A\in\Pi} \mathbf{1}_A c_A$ for some partition Π. We assume $K \in \mathbb{R}_+^{L\times N}$ to be a matrix, and let us define

$$K_A := K\mathbf{1}_A = \sum_{i\in A} K_{[:,i]} \in \mathbb{R}_+^L, \quad A \in \Pi \tag{13.7}$$

as the sum of the columns in K corresponding to a segment A. In addition, we define

$$K_\Pi := (K_A)_{A\in\Pi} \in \mathbb{R}_+^{L\times|\Pi|} \tag{13.8}$$

as the matrix concatenating the summed up columns K_A. We call K_Π the *reduced PET operator* on Π. Note that the following holds

$$K_{\Pi^n} = KP_n \tag{13.9}$$

since one can write $P_n = (\mathbf{1}_A)_{A\in\Pi^n}$ as concatenated vectors $\mathbf{1}_A$ for every segment in Π^n which yields (13.9) by matrix multiplication. With these definitions and $c = (c_A)_{A\in\Pi}$ we can compute that

$$Kf_c = K\sum_{A\in\Pi} \mathbf{1}_A c_A = \sum_{A\in\Pi} K\mathbf{1}_A c_A = \sum_{A\in\Pi} K_A c_A = K_\Pi c.$$

Plugging this into the reduced problem (13.5) defined on the reduced graph G_r^{n+1} and reformulating yields

$$c^{n+1} = \operatorname*{argmin}_{c\in\mathbb{R}_+^{|\Pi^{n+1}|}} \sum_l (K_{\Pi^{n+1}}c)_l - g_l \log(K_{\Pi^{n+1}}c)_l + \frac{\alpha}{2}\|\nabla_{w_r^{n+1}} c\|_1 \tag{13.10}$$

with $f_{\Pi^{n+1}} = P_{n+1}c^{n+1}$. Then the update given in (13.10) is a reduced optimization problem that only has to be solved on the reduced graph G_r^{n+1}, and thus, is efficient. Additionally, we see that the reduced operator matrix K_Π for some partition Π has only $|\Pi|$ columns which makes it much smaller than the original one. In practice it also preserves the sparsity of the original operator and can then be stored in memory even for huge systems. We analyse these result with numerical experiments in Chapter 14. In the next section we also show how to compute the reduced operator K_Π for some given partition Π efficiently from the PET operator $\mathcal{K}$ without the need of a matrix representation.

With the ideas from above we can formulate the Cut-Pursuit algorithm to solve the problem (10.1):

Algorithm 13.2. (Cut-Pursuit for Regularized PET)
Let $g \in \mathbb{R}^L$ be some measured data and $K \in \mathbb{R}^{L\times N}$ a forward operator that projects some distribution image $f \in \mathbb{R}^N$ into the data space as described before. Additionally, let us define K_A as in (13.7) for a segment $A \subset \Omega$ and the reduced operator matrix $K_\Pi \in \mathbb{R}^{L\times|\Pi|}$ as in (13.8) for some partition Π. With these definitions the Cut-Pursuit algorithm that solves the

minimization problem (10.1) *is given as follows:*

$$\begin{cases} B^{n+1} \in \underset{B\in\mathcal{P}(V)}{\operatorname{argmin}} \langle K^*\mathbf{1}_V - K^*\dfrac{g}{K_{\Pi^n}c^n} + \nabla R_S(f^n), \mathbf{1}_B\rangle + \dfrac{\alpha}{2}R'_{S^c}(f^n;\mathbf{1}_B) \\ \Pi^{n+1} = \{\operatorname{connComp}(A\cap B), \operatorname{connComp}(A\cap B) \mid A\in\Pi^n\} \\ c^{n+1} = \underset{c\in\mathbb{R}_+^{|\Pi^{n+1}|}}{\operatorname{argmin}} \displaystyle\sum_{l=1}^{L} (K_{\Pi^{n+1}}c)_l - g_l\log(K_{\Pi^{n+1}}c)_l + \frac{\alpha}{2}\|\nabla_{w_r^{n+1}}c\|_1 \\ f^{n+1} = \displaystyle\sum_{A\in\Pi^{n+1}} \mathbf{1}_A c_A^{n+1}. \end{cases} \tag{13.11}$$

Since, the data-term is differentiable we can directly state that Algorithm 13.2 converges to the optimum of (10.1) in a finite number of steps as was proved in [47]. We refer to Section 4.3 for more details.

The bottleneck of this approach is that we still have to solve the reduced minimization problem for which we can use a graph version of the primal-dual algorithm given in Section 10.1, the FB-EM-TV as in Section 10.2 or any other preferred method. But since the reduced operator K_Π on a partition Π has only $|\Pi|$ many columns in each iteration it reduces to a much easier task, as we see in Chapter 14. Nevertheless, we still have to compute the reduced forward operator matrix K_Π from K and Π somehow efficiently. This is what we are doing in the next section.

13.3 Efficient Reduced Operator Computation

In this section, we assume that we have a given PET operator $\mathcal{K}:\Omega\to\mathbb{R}_+^L$ for L lines in $\mathbb{L}$ and N pixels in the image domain Ω, and its discrete sparse matrix representation $K\in\mathbb{R}^{L\times N}$. Note again, that for high resolution image spaces, especially 3D images, the computation of matrix operator K from $\mathcal{K}$ is a computational expensive task. Additionally, for a huge amount of measurements and a high resolution - maybe even 3-dimensional - image this matrix has a high memory cost. We tackle this challenge in the second part of this section. First we want to recall how to compute a discrete operator matrix K from the continuous operator $\mathcal{K}$.

As already explained in Section 9.2 and in Figure 9.3, the matrix operator K to a given operator $\mathcal{K}$ can be computed row-wise. To do so, one casts a line l through the given image domain Ω and fills up the array $v_l\in\mathbb{R}_+^N$ by assigning to every entry i the lengths of intersection of the line l with the pixel i. If the pixel was not hit a 0 is stored. Then we stack the arrays v_l^T for every $l\in\mathbb{L}$ and store them in K as follows:

$$K = (v_l^T)_{l\in\mathbb{L}}. \tag{13.12}$$

Hence, when we apply this matrix to $f\in\mathbb{R}_+^N$ we see by the definition of $\mathcal{K}$ that $Kf=\mathcal{K}f$. In the former section, we have defined the reduced operator K_Π for some partition Π of Ω in (13.8) by concatenating the vectors K_A as defined in (13.7) for every segment $A\in\Pi$. The column K_A in K_Π is defined as the sum of the columns $K_{[:,u]}$ that corresponds to the pixel

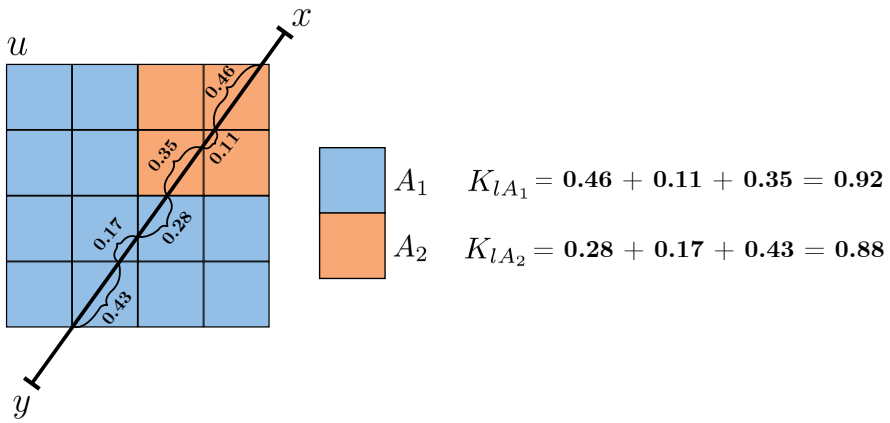


Figure 13.1.: Visualization of how to compute the entries in the reduced operator K_Π for a 4×4 example image and a partition Π of Ω with two sets A_1 and A_2. The entries in K_{lA_1} and K_{lA_2} are given as the total intersection length of the line l from x to y with the segments A_1 and A_2.

$u \in A$. Thus, for every line $l \in \mathbb{L}$ we can see that

$$(K_A)_l = \langle v_l, \mathbf{1}_A \rangle. \tag{13.13}$$

This is exactly equal to the sum over all the intersection lengths of the line l with the pixels in A, which is the total length of the intersection of line l with segment A. Consequently, when computing the reduced operator K_Π one has to compute the intersection length of every line l with every segment $A \in \Pi$ and store it in $(K_\Pi)_{lA} = \langle v_l, \mathbf{1}_A \rangle$. In Figure 13.1 we visualized a simple example one line, a 4×4 image and a partition with two segments.

With this method it is possible to compute the reduced matrix operator with applying $\mathcal{K}$ *just once*. Hence, computing the reduced operator matrix K_Π is as costly as computing the full operator matrix K. Yet, the number of sets is much smaller than the number of pixels and - if the number of segments is not too small - the matrix K_Π is still as sparse as K. Nevertheless, the reduced operator matrix K_Π has less non-zero entries in total, which makes it possible to store it in memory even in a high-resolution setting. In Chapter 14 we show these theoretical considerations and properties for artificial examples.

Now we know how to construct a Cut-Pursuit algorithm as formulated in Algorithm 13.2 to solve (10.1) by only using reduced matrix operators K_{Π^n} for every iteration. Furthermore, we have shown how to construct these reduced matrix operator with a single evaluation of the PET operator $\mathcal{K}$ in every Cut-Pursuit iteration. The last open question is how to solve the reduced problem in (13.11). This is covered in the next section.

13.4 Solving the Reduced Problem

In this section, we introduce two variants of formerly introduced algorithms to solve the reduced problem in (13.11) namely with primal-dual presented in Section 10.1 and FB-EM-TV as in Section 10.2. Deriving the algorithms is straightforward, since the only thing one has to change to state the algorithms is the used matrix operator K_Π. Still, we want to

state the algorithms explicitly for clarification and talk about how the algorithms get more efficient. We start by observing this for the primal-dual algorithm.

13.4.1 Primal-Dual for Reduced Problem

The primal-dual algorithm to solve (10.1) was derived in Algorithm 10.1. Let us assume we are in some iteration of the Cut-Pursuit algorithm in (13.11) and have computed some particular partition Π. Let K_Π be the corresponding reduced operator matrix and $c \in \mathbb{R}_+^{|\Pi|}$ a function living on the reduced graph $G_r = (\Pi, E_r, w_r)$. Then the primal-dual updates are given by

$$\begin{cases} y^{n+1} &= \operatorname{proj}_{B^\infty(\alpha)}\left(y^n + \sigma_y \nabla_{w_r} c^n\right) \\ z^{n+1} &= \frac{1}{2}\left(z^n + \sigma_z K_\Pi c^n + \mathbf{1} - \sqrt{\left(z^n + \sigma_z K_\Pi c^n - \mathbf{1}\right)^2 + 4\sigma_z g}\right) \\ c^{n+1} &= \max\left(c^n - \tau\left(K_\Pi^* z^{n+1} + \nabla^* y^{n+1}\right), 0\right) \\ c^{n+1} &= c^{n+1} + \theta\left(c^{n+1} - c^n\right). \end{cases} \tag{13.14}$$

As discussed in Section 3.3 and Section 4.4.1, the diagonal preconditioning is crucial for graph applications and especially on reduced graphs increases the convergence significantly. Hence, we want to derive the diagonal preconditioning as defined in Definition 3.21. Let us define the weighted differential operator $\mathcal{D}$ such that $\mathcal{D}f = \nabla_w f$ as in (3.28). As we have two operators in this cases namely ∇_w and the PET operator K we have to compute the column and row sums of the following operator

$$A = \begin{pmatrix} \nabla_w \\ K \end{pmatrix} = \begin{pmatrix} \mathcal{D} \\ K \end{pmatrix}.$$

Let M be the number of edges and assign to each edge $e_j = (u_j, v_j) \in E$ a index $j \in \{1, \dots, M\}$. Then we can deduce the following component-wise preconditioner for A as

$$\begin{aligned} \tau_u &= \frac{1}{\sum_{l=1}^L |K_{l,u}|^{2-\alpha} + \sum_{v\sim u} w(u,v)^{\frac{2-\alpha}{2}}}, && \forall u \in V \\ \sigma_{y,j} &= \frac{1}{\sum_{u\in V} |\mathcal{D}_{i,u}|^\alpha} = \frac{1}{2w(u,v)^{\frac{\alpha}{2}}}, && \forall j \in \{1, \dots, M\} \\ \sigma_{z,l} &= \frac{1}{\sum_{u\in V} |K_{l,u}|^\alpha}, && \forall l \in \{1, \dots, L\} \end{aligned} \tag{13.15}$$

for any $\alpha \in [0, 2]$. This leads to the diagonal preconditioners

$$\begin{aligned} T &= \operatorname{diag}\left(\tau_1, \dots, \tau_N\right) \\ \Sigma_y &= \operatorname{diag}\left(\sigma_{y,1}, \dots, \sigma_{y,M}\right) \\ \Sigma_z &= \operatorname{diag}\left(\sigma_{z,1}, \dots, \sigma_{z,L}\right). \end{aligned} \tag{13.16}$$

Then we can replace τ, σ_y and σ_z in (13.14) with these diagonal preconditioners.

As we see, the dual variable y is of the size of the reduced gradient $\nabla_{w_r} c$ and the primal variable c is reduced to the size of the partition Π. These updates are both much more

efficient compared to Algorithm 10.1, especially, due to the usage of the reduced operator and the use of a matrix multiplication instead of computing millions of line integrals all the time. Nevertheless, the reduced operator still maps into the full L-dimensional data space. Thus, the dual variable z is still of the size L and does not reduce at all. This is a bottleneck in later numerical computations. We analyse the numerical realization of this method in Chapter 14. This concludes this subsection and we go on by deriving a reduced EM-TV algorithm in the next subsection.

13.4.2 FB-EM-TV for Reduced Problem

We have already derived the Forward-Backward EM-TV (EM-TV) algorithm in Algorithm 10.2 to solve problem (10.1). The translation into a graph setting is again straightforward. The EM-step in (10.19) is pointwise on every vertex, and hence equivalent. The graph variant of the TV-step in (10.19) is actually a weighted ROF problem on the reduced graph G_r and can be directly solved with the primal-dual algorithm in Algorithm 4.14. The EM-TV algorithm on a graph then is given as

$$\begin{cases} f^{n+\frac{1}{2}} = \dfrac{f^n}{\mathcal{K}^*\mathbf{1}}\mathcal{K}^*\left(\dfrac{g}{\mathcal{K}f^n}\right) & \text{(EM-step)} \\ f^{n+1} \in \underset{f\in\mathbb{R}_+^N}{\operatorname{argmin}} \left\langle \dfrac{\mathcal{K}^*\mathbf{1}}{2f^n}(f-f^{n+\frac{1}{2}}), f-f^{n+\frac{1}{2}} \right\rangle + \dfrac{\alpha}{2}\|\nabla_w f\|_1. & \text{(TV-step)} \end{cases} \tag{13.17}$$

As we want to derive the reduced version of this algorithm to solve the reduced problem in (13.11), we exchange $\mathcal{K}$ by K_Π and f by $P_\Pi c$ for some partition Π. Since computing the KKT condition and drawing the subsequent deductions for the reduced problem in (13.11) is analog to the deductions in Section 10.2 we can directly write EM-TV on a reduced graph G_r as

$$\begin{cases} c^{n+\frac{1}{2}} = \dfrac{c^n}{K_\Pi^*\mathbf{1}}K_\Pi^*\left(\dfrac{g}{K_\Pi c^n}\right) & \text{(EM-step)} \\ c^{n+1} \in \underset{c\in\mathbb{R}_+^{|\Pi|}}{\operatorname{argmin}} \left\langle \dfrac{K_\Pi^*\mathbf{1}}{2c^n}(c-c^{n+\frac{1}{2}}), c-c^{n+\frac{1}{2}} \right\rangle + \dfrac{\alpha}{2}\|\nabla_{w_r} c\|_1. & \text{(TV-step)} \end{cases} \tag{13.18}$$

Note that $\mathbf{1} = \mathbf{1}_L$. The reduced sensitivity map $\mathbf{K}_\Pi^*\mathbf{1} \in \mathbb{R}_+^{|\Pi|}$ can be computed easily by

$$(K_\Pi^*\mathbf{1})_A = \sum_{i\in A}\sum_l K_{li} = \sum_{i\in A}(\mathcal{K}^*\mathbf{1})_i. \tag{13.19}$$

Thus, when applying Cut-Pursuit the computation of the sensitivity map $\mathcal{K}^*\mathbf{1} \in \mathbb{R}_+^N$ has only to be done *once* at the beginning. To compute the reduced sensitivity map $K_\Pi^*\mathbf{1}$ afterwards it is sufficient to sum up the entries in $K_\Pi^*\mathbf{1}$ corresponding to a segment $A \in \Pi$.

Remark 13.3. *Looking at the reduced EM-TV algorithm in* (13.18) *we see that by dropping the TV-step we already have a working EM algorithm in the reduced graph setting. This will compute a reconstruction that can be interpreted as a reconstruction in non-uniformly discretization of the image space by* Π*. Thus, when going back into the full image space the*

result will be a piece-wise constant reconstruction.

As we have translated the primal-dual algorithm from Section 10.1 and the EM-TV algorithm from Section 10.2 into the reduced graph setting we head to the numerical experiments in the next chapter. There we evaluate if and in which situations the use of Cut-Pursuit is a good method to solve the problem. We also face some challenges that have to be tackled in the future.

14 Numerics

In this chapter, we present the application of the Cut-Pursuit algorithm for solving a PET reconstruction problem with an incorporated TV regularization as stated in (10.1). In the previous chapter, we derived the Cut-Pursuit algorithm for TV-regularized PET reconstruction and stated it in Algorithm 13.2. In addition, we presented two algorithms to solve the reduced problem in Algorithm 13.2, via a primal-dual algorithm in (13.14) or an EM-TV algorithm on graphs in (10.2). We start by discussing the implementation details of the algorithms. Then we analyse how the Cut-Pursuit algorithm works on PET data and what challenges it faces. Afterwards, we compare the results of the different reduced solvers to investigate, which of these algorithms is preferable, both visually and in terms of energy values.

14.1 Implementation Details

The Cut-Pursuit and primal-dual algorithms were implemented in MATLAB in a modular, object-oriented framework that we already discussed in Chapter 6. The graph cut is computed by the built-in MATLAB function *maxflow*. For the TV-step of the EM-TV algorithm that we stated in (10.2) we again use the implementation for the algorithm in Algorithm 3.24 to solve weighted ROF problems.

To simulate the PET line operator $\mathcal{K}$, which we introduced in Chapter 9, we use the implementation of the *Siddon-algorithm* [78] of the EM reconstruction framework *EMRECON* developed in [44]. This is a parallelized implementation in `C` that computes the line integrals through an image by a predefined given scanner. For more information on the framework we refer to [44]. In addition, this framework also provides the implementation of the adjoint PET operator $\mathcal{K}^*$. For our applications in this work we only need to know that we can define a scanner with a certain set of possible lines and then apply it to some defined image space. By using this implementation we have the possibility to use real scanner definitions, e.g., [21], or even build an artificial scanner with artificial line definitions. We are therefore able to reconstruct real and artificial data sets. In this numerics section we only use artificial data that we generated by applying the operator $\mathcal{K}$ to an image and adding poisson noise to it. Note that we use the MEX interface of MATLAB to call the `C` code.

In addition to the full line operator $\mathcal{K}$ we also use the reduced operator matrices K_Π that

we have defined in (13.8) for a given partition Π. As we state in Section 13.3, we can compute the reduced operator matrix by only one application of the full operator $\mathcal{K}$. This is done by computing the lengths of the intersections of each line with every segment in the partition Π. To do so, we use a modified implementation of the operator from EMRECON where we only measure the lengths of intersections of the line with the segments instead of computing the line integral. On top of this we have implemented a pipeline in the MEX framework to directly provide us with a sparse representation of the reduced matrix. In addition, the application of the sparse matrix in MATLAB is very efficient.

To guarantee convergence of the Cut-Pursuit algorithm it is crucial to find an optimal solution of the reduced problem. To achieve this, we have to establish solvers that can accurately optimize the reduced problem and converge closely to its optimum. For example, in order to solve ROF problems on graphs using the Cut-Pursuit algorithm stated in Algorithm 3, we have discussed in Section 4.4.1 that it is crucial to use a diagonal preconditioning for solving the reduced problem (4.63) with a primal-dual algorithm, since otherwise the algorithm converges very slowly due to the bad condition of the gradient operator. If we do not apply this step size method, we cannot guarantee the algorithm for the reduced problem to converge in any reasonable number of iteration steps in practice.

This is a special case of the more general class of *nested algorithms*, where one has to use inexact solvers to update the iterates of an optimization algorithm. However, these procedures are known to diverge when the subproblems are not optimized to the necessary precision. This problem is discussed, e.g., in [2, 68, 76, 87] where they analyse how an error in the computation of an inner optimization problem, e.g. in the computation of the proximal point, induces erroneous iterates to the overall optimization, and therefore, yields incorrect results. This is similar to what happens here since we also have a nested algorithm with two optimization problems, namely the partition problem and the reduced problem. Both problems have to be solved well enough to yield an optimal result in the end. For more details on these inexact optimizations see [2, 68, 76, 87] and the references therein.

In this work we have stated two optimization strategies to solve TV-regularized PET reconstruction problems. One is a first-order primal-dual algorithm we state in (13.14) and the other one is an EM-TV algorithm we state in (13.18). As a third algorithm we will apply a stochastic variant of the primal-dual algorithm that we discuss later on. Before we show any experiments and plots let us discuss the advantages and drawbacks of each algorithm.

14.2 Solving the Reduced Problem

In this section, we discuss three different solvers that we use for solving the reduced problem of the Cut-Pursuit algorithm in Algorithm 13.2. Let us start with the primal-dual algorithm. In (13.16) we have already stated the diagonal preconditioning for this problem that can be directly applied to the reduced graph with w_r as the weighting and K_Π as the reduced PET operator matrix. This alleviates the bad condition of the gradient operator as before.

However, there remain two drawbacks. The first drawback is that the reduced problem only reduces the number of pixels and not the number of lines. As we know, the reduced operator matrix K_Π also only reduces in number of columns and not in number of rows.

Hence, we can see in Algorithm 13.2 that the update of the dual of the Kullback-Leibler data-term in (13.14) is still of the same size as in the original problem, corresponding to the number of considered lines. This makes this dual update as expensive as in the primal-dual algorithm for the full problem. Additionally, there has to be computed a square-root for every entry in the array of length of the number of lines, which is very costly as it is computed and leads to a bottleneck in run time.

Furthermore, in practice it is not trivial to find a sufficient stopping rule that indicates the convergence of the algorithm, e.g., a *primal-dual gap* [17] or a *primal-dual residual* [34]. This comes from the fact that one has to scale the involved quantities properly for each level of partitioning in the Cut-Pursuit algorithm to set a certain threshold for every level that stops the algorithm. Hence, we decided to use a combination of both primal-dual gap and primal-dual residual as well as the maximal relative change in the solution. As we have to ensure that the algorithm optimizes the reduced problem well enough, we also set a minimum number of iterations that the algorithm has to run before it stops. In this numerical section we always set the minimal number of iterations to $1,000$.

To alleviate the drawbacks of the primal-dual algorithm in (13.14), in particular the expensive full amount of lines, we also implemented the *stochastic primal-dual hybrid gradient* algorithm introduced in [18] and applied to PET reconstruction in [26] with a diagonal preconditioning on graphs. Using this algorithm one has to divide the set of lines into different subsets. Then, when running the dual updates, the algorithm randomly selects a subset of lines to update the dual iterate only for this selection. The convergence of this algorithm is much faster while reducing the number of applications of the full PET operator. For the reduced problem this means that we divide the reduced matrix K_Π by stacking the corresponding rows for every subset and compute a new reduced matrix for every subset. Note that we now get the best of both worlds: reduced pixels (columns) via partitioning in image space, and reduced lines (rows) by random sampling in data space. As this algorithm is not part of this work and the graph version is straight forward, we do not go into detail and refer to [26].

The third algorithm is the forward-backward EM-TV algorithm we presented in Algorithm 10.2 and its corresponding graph variant in (13.18). Applied on the reduced graph we use the reduced operator matrix K_Π for the EM-step and solve the TV-step - which is in this case defined on the reduced graph - with a primal-dual algorithm. We again use the diagonal preconditioning for reduced primal-dual algorithms as in (3.21).

Note that we have to distinguish between the different reduced solvers for the Cut-Pursuit algorithm. To this end, we use the abbreviations *CP-PD*, *CP-ST-PD* and *CP-EM-TV* for the Cut-Pursuit algorithm with the primal-dual algorithm, the stochastic primal-dual algorithm and the EM-TV algorithm as a reduced solver, respectively.

14.3 Full Operator Evaluations

In this section, we compare the Cut-Pursuit algorithm with different reduced solvers to the solutions of applying the same algorithms to the full problem. To do so, we investigate the number of applied full PET operator projections for the different algorithms, since this is the main drawback of the TV-regularized PET reconstruction in high resolution 2D and 3D applications. In these cases the full PET operator matrix cannot be stored in memory and the computation of the full PET operator is very costly for each application.

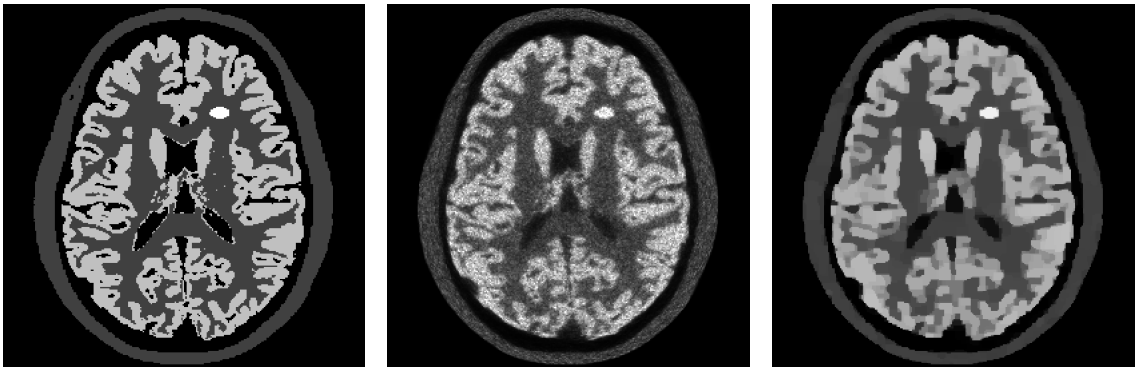
Taking a look at the reduced primal-dual algorithm in (13.14), we see that the operator matrix is used once in the update of the dual variable corresponding to the Kullback-Leibler data-term and once in the update of the primal variable as a backward projection. The differences in cost of applying the forward or backward projection is negligible, and therefore, we can tell that one iteration of the primal-dual algorithm requires two full operator applications.

For the stochastic primal-dual algorithm this is a little bit more complicated, since the lines are divided into subsets and selected randomly. Here, one iteration requires just a fraction of the whole operator. Hence, in [26] they use the term *epoch* "to denote the number of iterations of a randomized algorithm which are in expectation computationally equivalent to one iteration of the deterministic algorithm that uses all data for each iteration", which in fact means in this case approximately one operator evaluation. For example, we always divide the set of lines into ten subsets and give each a probability of $p = 0.1$ to be chosen. Then we can assume that ten iterations of the primal-dual algorithm require the full operator to be evaluated two times in total, one time forward and one time backward. Hence, we also refer to the term *epoch* when we talk about stochastic primal-dual which costs two operator applications.

The EM-TV algorithm in (13.17) uses one forward and one backward projection of the PET operator. The $\mathcal{K}^*\mathbf{1}$ in fact has to be pre-computed once before running the algorithm. Hence, in every iteration we also have to apply two full PET projections.

Finally, let us discuss the Cut-Pursuit algorithm we state in Algorithm 13.2. As we see for the partition problem to compute a new cut B^{n+1} we have to compute the full PET operator once in its adjoint variant. For the forward operator we can again use the reduced operator. Additionally, we have to apply the full operator once, as we have discussed at the beginning of this chapter, to compute the reduced matrix K_Π for a corresponding partition Π. Hence, in each Cut-Pursuit iteration we have to evaluate two full PET projections.

Consequently, we can see that for every algorithm the number of applied full PET operators in every iteration is two, and therefore, we can directly compare the number of iterations to reach a certain energy value as a measure for complexity in the following section.



(a) Ground Truth 272×272 **(b)** Noisy EM Reconstruction **(c)** Denoised Reconstruction ($\alpha = 4$)

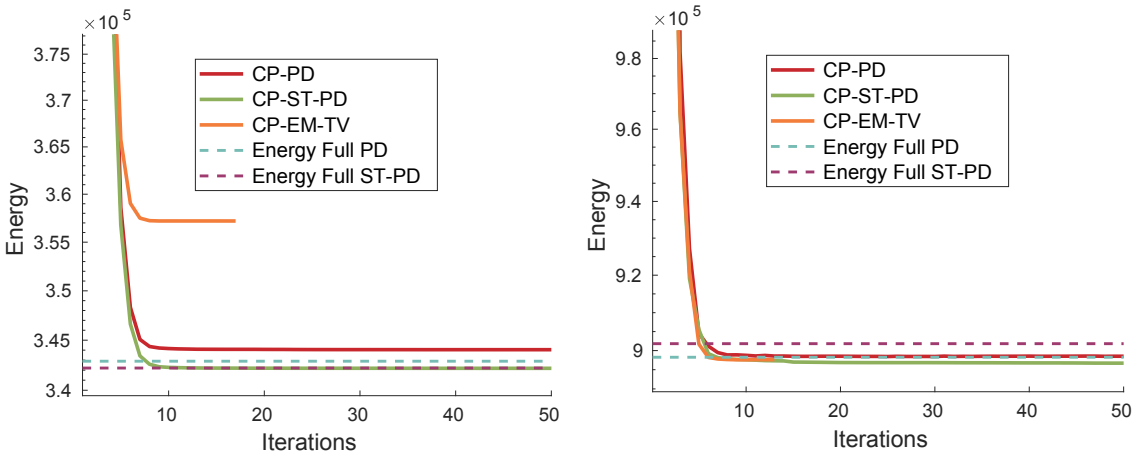
Figure 14.1.: Ground truth image taken from BrainWeb[20] and reconstruction via the EM algorithm and via a primal-dual algorithm for a TV-regularized PET problem with $\alpha = 4$.

14.4 Numerical Comparison

In this section, we want to compare the application of the different variants of the Cut-Pursuit algorithms proposed before on a noisy data generated by an artificial PET scanner definition with $L = 83,950$ lines and an image resolution of 272×272. In Figure 14.1a we can see the 272×272 ground truth image taken from [20] and in Figure 14.1b the EM reconstruction of the noisy data that we generated with the implementation of the forward PET operator. We also provide a denoised reconstruction that was computed by using the full version of the primal-dual algorithm from (13.14) for a regularization parameter of $\alpha = 4$.

For the primal-dual algorithm we set the minimal number of iterations to $1,000$ and the maximal number of iterations to $10,000$. As discussed before, we use a combination of primal-dual gap, primal-dual residual and maximum relative change in the solution as a stopping criteria. As the stochastic primal-dual algorithm converges faster it only needs less total iterations. Hence, we set the maximum of iteration to only $1,000$ and apply the same stopping rules as for the primal-dual algorithm. For the EM-TV algorithm there does not exist any sufficient stopping rule. Hence, we use a maximum of 600 iterations and the maximum relative change in the solution as a stopping rule, to ensure the convergence of the reduced problem. The Cut-Pursuit algorithm has a maximum of 50 iterations and stops if no new cut can be computed.

Let us start by investigating the evolution of their energy in Figure 14.2 for two regularization parameters, i.e., $\alpha = 4$ and $\alpha = 50$. We also plot the energy levels of using the primal-dual algorithm for $10,000$ and stochastic primal-dual for $1,000$ iterations with the equivalent stopping rules as for the reduced problems. As we can directly observe, the Cut-Pursuit algorithm with EM-TV as the reduced solver stops very early, and not at an optimal energy level. The visual results can be seen in Figure 14.4 where we also plot the difference images compared to the solution of applying the stochastic primal-dual algorithm. We assume that the problem here is that the EM-TV algorithm on its own is a forward-backward splitting method where we also have to solve an inner problem with primal-dual. If for some



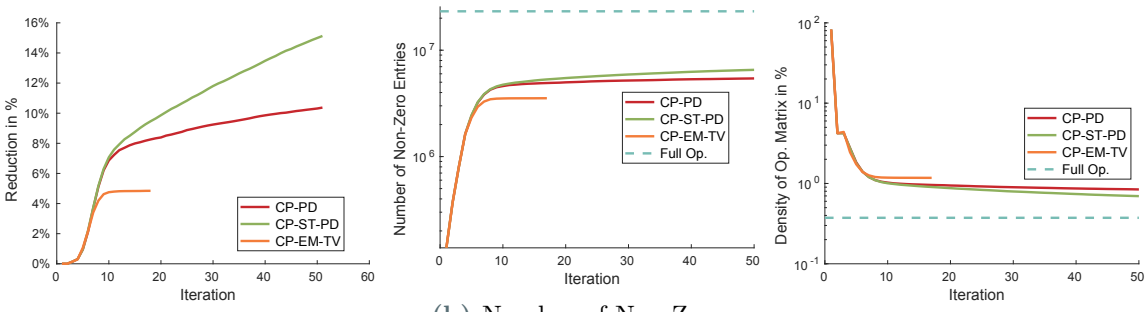
(a) Energy for $\alpha = 4$

(b) Energy for $\alpha = 50$

Figure 14.2.: Energy plots of different reduced solvers. The dashed lines indicate the energy levels of the converged primal-dual and stochastic primal-dual algorithms.

reason we produce errors in that inner optimization, it might lead to erroneous solutions in the outer problem (cf. [68]). Hence, EM-TV does not seem to be a good reduced solver for this kind of problem. Using the primal-dual (PD) algorithm as a reduced solver leads to a good energy minimization, but does not eventually reach the energy level achieved by the full optimization. Nevertheless, it converges very close to the optimum. On the other hand, using the stochastic primal-dual (ST-PD) algorithm even yields a slightly better optimum than applying the stochastic primal-dual algorithm on the full problem for 1,000 iteration. For higher regularization, e.g., $\alpha = 50$, we see in Figure 14.2b that all algorithms dominate the algorithms on the full problems. Still, in terms of energy using the stochastic primal-dual on the reduced problem yields the best results.

When working with the Cut-Pursuit algorithm it is not only important to compute the correct solution for a minimization problem, but also to provide a good discretization of the underlying image. In Figure 14.3 we can see a comparison of the ratio of the number of segments in the partition and the original number of pixels, the density and the number of



(a) Reduction

(b) Number of Non-Zero Entries

(c) Density

Figure 14.3.: Comparison of the number of non-zero entries and the density of the reduced operator matrices. The reduction is the ratio of number of segments to number of pixels given in %.

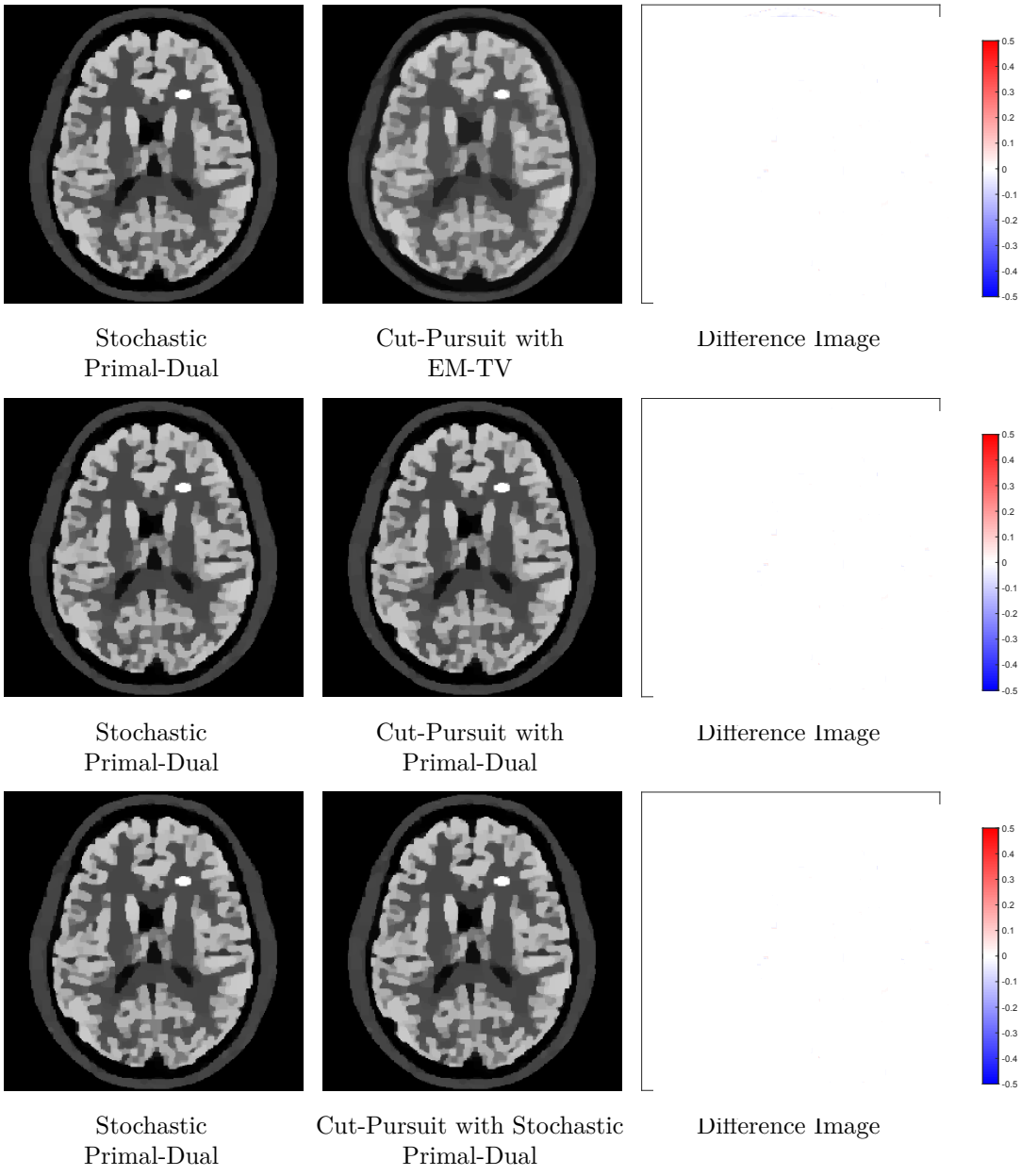


Figure 14.4.: Visual comparison of solutions for $\alpha = 4$ computed with stochastic primal-dual on the full problem and Cut-Pursuit algorithms with different reduced solvers. On the right we see difference images. The CP-PD algorithm yields visually very close result with only a few incorrect segments in the difference image. The difference for the direct method and the CP-ST-PD algorithm is not visible.

non-zero entries in the sparse, reduced operator for the three variants in every iteration. As we can see in Figure 14.3b, in each iteration the number of non-zero elements in the reduced operator matrix is always very small and also smaller than for the full matrix operator that is shown by the dashed line. Note that in this small problem the number of non-zero entries in the full operator is still pretty small. In a more complex 3D setting this value would not even be any close to the number of non-zeros in the reduced matrix. In Figure 14.3c we can also see that the density of the reduced matrix is always under 10% in iterations after the

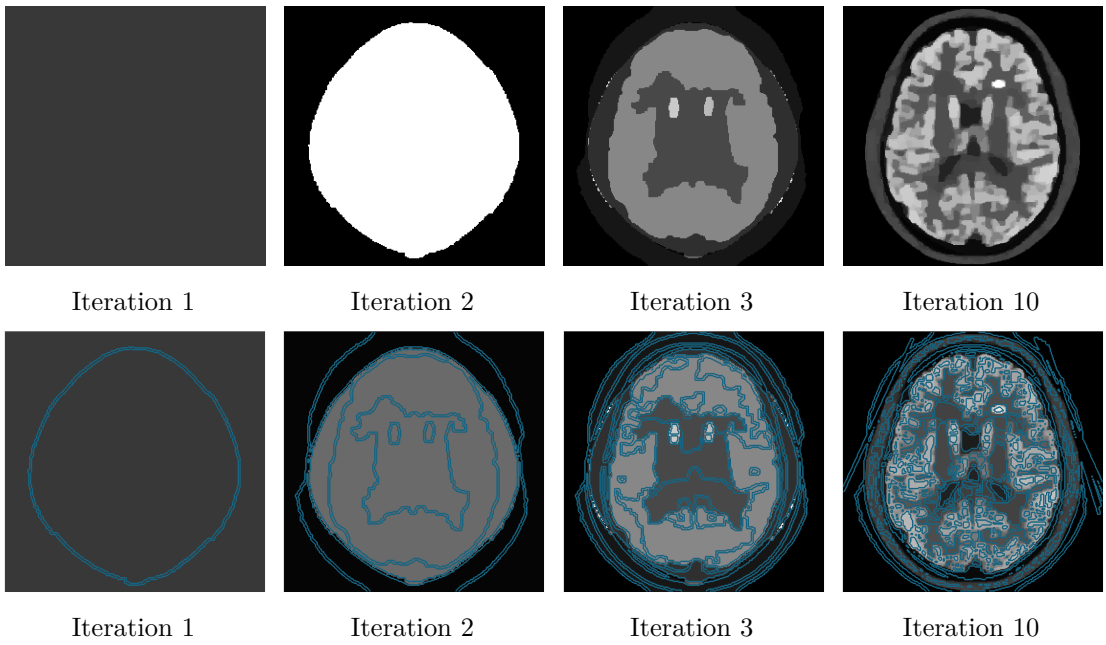
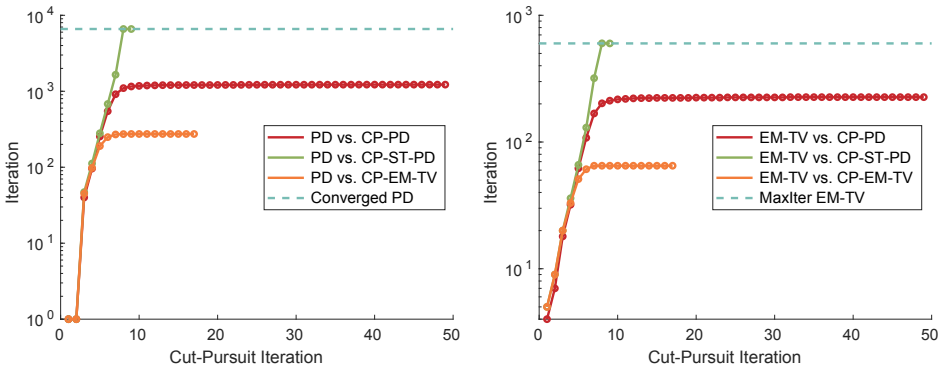


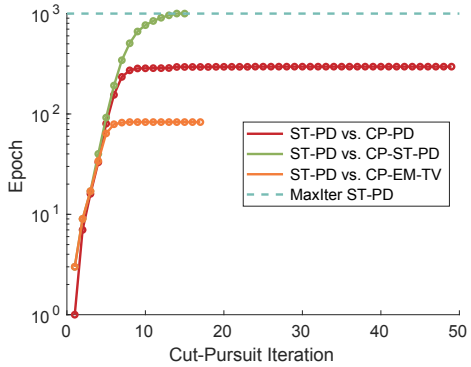
Figure 14.5.: Cut-Pursuit iteration. Top row shows the solution in a certain iteration and the bottom row the cut computed in that iteration.

first iteration. The first iteration has such a high density as it only consists of two segments, and therefore, the reduced matrix only consists of two columns storing all of the information of the full operator. However, for this small operator it does not matter if it is sparse or not. In Figure 14.3a we show a plot of the reduction of the current Cut-Pursuit iteration. We see that the CP-EM-TV algorithm stops after 18 iteration since it does not incorporate any new segments into the partition. However, the CP-PD algorithm seems to cut further. Especially for the CP-ST-PD algorithm this is surprising, since it already is at the optimal energy after only 9 to 10 iterations as we see in Figure 14.2. Still, the algorithm computes a cut and splits the partition in every iteration. This can be interpreted as the problem we face for channel-isotropic Cut-Pursuit algorithms in Algorithm 1 as we discuss in the numerics in Section 6.3 where we see that the Cut-Pursuit struggles to find cuts to describe smooth parts of the solution. In this case this can come from the fact that the partition problem never knows the exact position of the edges in the image as it always compares the reconstruction that was generated with cuts with the data. Hence, it seems to have problems to find the exact pixel where to cut at the boundaries in the image. This leads to further cutting even though the energy is nearly optimal. We can see this problem in Figure 14.5 where we show the solutions in some iterations and the cuts that the partition problem yields. We see that it struggles to find sufficient edges in the solutions as it does not know the underlying ground truth image. We postpone this problem for the outlook where we also discuss a strategy that can alleviate this problem.

In summary, we see that the CP-ST-PD algorithm as the reduced solver yields the most satisfying results. In the following we want to also compare the Cut-Pursuit methods to the application of the primal-dual, stochastic primal-dual and the EM-TV algorithm on the full problem where we also use the full PET operator.



(a) Primal-Dual (b) EM-TV



(c) Stochastic Primal-Dual

Figure 14.6.: Compares the energy in different iterations of the Cut-Pursuit algorithm for different reduced solvers with the energy of the full algorithms. It illustrates for every Cut-Pursuit iteration in which iteration of the compared full algorithm it outperforms the energy.

14.5 Comparison of Cut-Pursuit with Direct Algorithms

In this section, we want to compare the different Cut-Pursuit strategies to the application of the three algorithms applied to the full problem. We first compare the number of full PET operator applications and then also have a visual comparison of the solutions.

As we have discussed in Section 14.3, in each iteration of the algorithms the full PET operator is applied twice. Hence, we can compare the energy value in each iteration of the algorithms and deduce how many full PET operator evaluations were required to get to this point. To this end, we take the energy value of the Cut-Pursuit algorithm in every iteration and search for the number of the first iteration where the direct algorithm achieves a lower energy value. Then we plot the Cut-Pursuit iteration against this iteration number. The comparison of all three Cut-Pursuit strategies can be found in Figure 14.6. Let us start with the comparison of the primal-dual algorithm on the full problem with the Cut-Pursuit algorithms in Figure 14.6a. The dashed line in this plot marks the iteration where the primal-dual algorithm converges. This is somewhere between $6,000$ to $7,000$ iterations, and therefore, we know that it required $12,000$ to $14,000$ full PET operators evaluations. To understand this plot properly, let us interpret the orange line and its behavior which corresponds to the CP-EM-TV algorithm. We see by the orange line that this algorithm has

a better energy value after three iterations as the primal-dual algorithm after 50 iterations. But then it flattens at around eight Cut-Pursuit iterations corresponding to around 250 primal-dual iterations. This means that applying CP-EM-TV is not able to yield a better energy value than applying the primal-dual algorithm for 250 iterations. Nevertheless, in this case we can get a energy value with only 16 full operator compared to around 500 and more in the primal-dual algorithm. The same behavior of the curve can be seen for the CP-PD algorithm, where the energy value of the primal-dual algorithm after over $1,000$ iterations is better. Again, the Cut-Pursuit only needs around ten iterations to get to this energy value. Now let us take a look at the CP-ST-PD algorithm. As we know from Figure 14.2a, this algorithm yields a nearly optimal energy value. This behavior can also be seen in this plot, where it dominates all other strategies and even the primal-dual algorithm after only *eight* Cut-Pursuit iterations. In fact, the Cut-Pursuit algorithm only needs around 20 full operator evaluations to get to the energy level of the converged primal-dual algorithm that had to apply around $12,000$ to $14,000$ full operators.

Moving on to the next figure where we compare the Cut-Pursuit algorithms to the EM-TV algorithm on the full problem we see the exact same behavior of the algorithms. The EM-TV algorithm was run for 600 iterations, and therefore, had to evaluate $1,200$ full operator projections. The CP-ST-PD algorithm again dominates these algorithms and overcomes the EM-TV after 9 iterations. Consequently, even though we use a method that was designed to decouple the operator evaluation, it cannot cope with the efficiency of the Cut-Pursuit algorithm.

Last we want to compare the Cut-Pursuit algorithms to the direct optimization of the problem using the stochastic primal-dual algorithm. This algorithm was run for $1,000$ epochs which is approximately $2,000$ full operator evaluations. The CP-ST-PD algorithm again wins over this algorithm, but this time it takes with 15 iterations longer than in the other cases. Nevertheless, the Cut-Pursuit algorithm is superior in terms of full operator evaluations.

With these observations and analysis we want to close this section and refer to the conclusion in Section 15.3 where we also discuss future work.

15 Summary and Conclusion

In this work we have introduced a wider spectrum of the application of the Cut-Pursuit algorithm presented in [46]. We used Cut-Pursuit as a denoising method to solve ROF problems on grayscale and RGB images and large point cloud data, and have shown numerically its superior efficiency over classical methods on graphs, e.g., a first-order primal-dual algorithm. We have seen that the Cut-Pursuit algorithm can also provide solutions and approximations of minimal partition problems. As a by-product, the Cut-Pursuit algorithm yields a reduced graph structure for the corresponding computed result that can be used as a coarse representation of the underlying graph. On this reduced graph one can solve complex problems efficiently due to the comparatively small number of segments. With these outcomes in mind, we applied the Cut-Pursuit algorithm also to a TV-regularized PET reconstruction; a problem of high complexity due to the PET operator. We demonstrated that the Cut-Pursuit algorithm requires far fewer full PET operator evaluations than classical methods, while computing the correct solution using only reduced sparse matrix representations of the full PET operator.

In this chapter, we review the results of this thesis and give an outlook for future work.

15.1 Efficient Cut-Pursuit Algorithm

In the first part of this work, we introduced the Cut-Pursuit algorithm for variational problems with a channel-isotropic regularization in general and for a (weighted) channel-isotropic ROF problem in particular. We discussed the importance of a suitable selection of directions to guarantee convergence of the optimization. We have seen that the Cut-Pursuit algorithm adept at optimizing ROF-type problems on images and is superior to the direct primal-dual algorithm optimizing point cloud data. We have discussed different aspects of the solution as it does not only yield the denoised solution, but also the underlying discretization. One can use this discretization to easily apply a debiasing step to the solution on the given discretization (cf. [12]), but also to simply represent the solution by the corresponding reduced graph with only a small fraction of the original number of data points.

Furthermore, we introduced the L_0-Cut-Pursuit algorithm to tackle minimal partition problems and have shown that we can solve the problem with different strategies. We compared runtime, visual appearance and energy of the these with well-known established meth-

ods and have seen that our proposed strategies perform equally well or even outperform them. We have also discussed that the L_0-Cut-Pursuit algorithm has superpixel properties superior to the popular SLIC method [1], and can even handle noisy data very well.

We still face challenges and questions that we did not answer in this work. The main question about the channel-isotropic Cut-Pursuit algorithm in Algorithm 1 is if one can solve the partition problem without selecting a predefined direction or trying to find an optimal one by alternatingly computing a direction for a given cut and computing a cut for the computed direction as we did in (4.54). As we have seen, the Cut-Pursuit algorithm can suffer from bad convergence at a later stage of the algorithm and does not converge in a reasonable number of iterations in practice. Additionally, we have to compute multiple cuts for the alternating algorithm to solve one partition problem, which is very costly. Hence, a more sophisticated optimization strategy to optimize this problem has to be found.

Talking about the partition problem, the computational efficiency of computing a graph cut for complex graph problems suffers from the implementation of the *maxflow* algorithm which only works on a single CPU-kernel and cannot be parallelized. Hence, on large problems the graph cut can become a problematic bottleneck. Nevertheless, one can again use a first-order primal-dual algorithm to solve the partition problem, which can be parallelized on CPU and GPU. This was shown in, e.g., [62] where they also state that their GPU version is faster than a *maxflow* implementation. Replacing the maxflow implementation with the primal-dual approach can potentially alleviate this bottleneck.

The most interesting part about the L_0-Cut-Pursuit algorithm is the property that it does provide a reduced graph representation of the computed solution. This can be interpreted as a coarse discretization of a given data set and its corresponding graph. We have seen that this discretization is adaptive on the level of detail of the given data set, i.e. it is very coarse where the data has many redundant information and fine where the data has high density of local information. This makes the discretization very meaningful, and paves the way for applying complex problems on this reduced discretization, which can then be solved efficiently and yield similar results to the results on the original discretization. This was already done in [31] where the L_0-Cut-Pursuit algorithm was used to discretize RGB images and compute the corresponding reduced graphs. Then costly lifting methods for segmentation and stereo images were applied to this reduced graph which then could be solved efficiently due to the low complexity of the reduced graph. Therefore, this method can potentially be used in many applications as a preprocessing step to simplify complex problems, while preserving the important, non-redundant information of the underlying problem.

As we have introduced the Cut-Pursuit algorithm on point cloud data and have seen the efficient optimization on point clouds, we still face the problem that the sparsification of the point clouds is not depending on the structural information of the point cloud. The reason is the application of an L_2-data-term that penalizes the euclidean distance between the points of the solution and their original position in the given data. Hence, it does not incorporate any further information about the structure of the surface of the point cloud. To do so, one can incorporate the normals of the point cloud as they describe the curvature of the surface of the object. Normals point in the same direction on flat surfaces and in different directions on parts with more details and edges. Hence, applying Cut-Pursuit while incorporating these

information yields to a sparsification of the normal field, which means that the normals on plane surfaces are represented by a few normals, while the normals on more detailed parts are represented by a bigger set of normals. Similar ideas of using normals for denoising point clouds were also presented in the literature, e.g., in [84, 90, 93].

15.2 Dynamic PET and Regularized Fully4D

In the second part of this work, we have introduced the basics of PET imaging and how to reconstruct given PET data. We also added a TV regularization to the problem to apply denoising in the reconstruction process and stated a primal-dual and the forward-backward EM-TV [74] algorithm to solve this problem. Then we discussed that the measured PET data is actually time-dependent, and thus, suffers from motion and dynamics in the data. To face the problem of dynamics in the data, we modified the Fully4D method from [70] and added a TV regularization to the spatial weights and a Tikhonov regularization to the basis functions. We also deduced an algorithm to solve this problem with incorporated regularization.

In the numerical experiments, we have gained an intuition of what kind of results one can expect from applying Fully4D to listmode data and what the advantage of using TV regularization is. We have also seen that motion has a high influence on the reconstructed basis functions as it is interpreted as some combinations of dynamics. On the one hand, this enables the algorithm to gather time points where the object has a certain spatial position, and therefore, reconstruct the path of the object. But on the other hand, one has to allow the Fully4D algorithm to look for many more basis functions as one would expect from the dynamics in the data to depict the motion, which makes the algorithm very costly. The TV regularization of the problem is much more costly since we need to solve a weighted ROF problem in each iteration. Nevertheless, in most cases the results are better after fewer outer iterations as for the unregularized Fully4D algorithm and also seem to depict motion better with less basis functions.

For applications in practice with real data we suggest to correct the data for motion beforehand, such that the basis functions are not influenced by moving organs or tissues. On the other hand, one can try to build a joint model that combines spatial reconstruction with motion and dynamics reconstruction, see, e.g., [22] and the references therein.

15.3 Cut-Pursuit Based Reconstruction

In the last part, we have deduced how we can apply the Cut-Pursuit algorithm to TV-regularized PET reconstructions via reduced matrix representations of the full PET operator $\mathcal{K}$ corresponding to the reduced graphs generated by the algorithm. We deduced that the Cut-Pursuit algorithm is superior in terms of the number of evaluations of the full operator as Cut-Pursuit only evaluates it twice in each iteration, while the direct methods apply it several thousand times. Furthermore, we can represent the reduced operator by a sparse matrix, which makes computation fast and storage efficient. We have investigated that the stochastic primal-dual algorithm proposed in [18] and applied to PET reconstructions in [26] is a sufficiently fast and accurate solver for the reduced problem. We have seen that the Cut-Pursuit algorithm yields a solution that is as good as the direct method with just a

small fraction of full operator evaluations.

The only downside of this algorithm that we face is that it does not necessarily stop when it is close to the optimal energy. This comes from the fact that this problem is a truly ill-posed problem, as we do not know anything about the underlying ground truth image. Hence, the partition problem has no boundaries to orientate on when computing the cuts, especially when computing the first few cuts. Hence, the boundaries are very inexact mid-iteration. To tackle this problem one can use, e.g., structural MRI priors (cf. [25, 69, 67]) and use the edge information of the MRI image to weight the graph. One can for example assign edges that correspond to spatial boundaries in the MRI image a smaller weight than edges in constant regions. Then the partition problem would produce a solution that cuts these edges with a higher probability and the first iterations of the Cut-Pursuit algorithm would have better results.

The next step in terms of applications would be to apply this algorithm to more complex data and find sufficient stopping rules for the reduced problem solver. This method has the potential to outperform the stochastic primal-dual algorithm for faster PET reconstruction.

List of Figures

Bibliography

[1] Radhakrishna Achanta et al. “SLIC Superpixels Compared to State-of-the-Art Superpixel Methods”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34.11 (Nov. 2012), pp. 2274–2282 (cit. on pp. 2, 133, 134, 224).

[2] J.-F. Aujol and Ch. Dossal. “Stability of over-relaxations for the forward-backward algorithm, application to FISTA”. In: *SIAM Journal on Optimization* 25.4 (2015), pp. 2408–2433. eprint: http://dx.doi.org/10.1137/140994964. URL: http://dx.doi.org/10.1137/140994964 (cit. on p. 214).

[3] Ramsey D Badawi et al. “Randoms variance reduction in 3D PET”. In: *Physics in Medicine & Biology* 44.4 (1999), p. 941 (cit. on p. 161).

[4] Egil Bae and Ekaterina Merkurjev. “Convex variational methods for multiclass data segmentation on graphs”. In: *arXiv Preprint # 1605.01443* (2016) (cit. on p. 17).

[5] Egil Bae and Xue-Cheng Tai. “Graph Cut Optimization for the Piecewise Constant Level Set Method Applied to Multiphase Image Segmentation”. In: *Scale Space and Variational Methods in Computer Vision*. Ed. by Xue-Cheng Tai et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1–13 (cit. on p. 84).

[6] Heinz H Bauschke, Patrick L Combettes, et al. *Convex analysis and monotone operator theory in Hilbert spaces*. Vol. 408. Springer, 2011 (cit. on pp. 5, 56).

[7] Dimitri P Bertsekas. *Constrained optimization and Lagrange multiplier methods*. Academic press, 2014 (cit. on p. 57).

[8] Ilja Bezrukov et al. “MR-based PET attenuation correction for PET/MR imaging”. In: *Seminars in nuclear medicine*. Vol. 43. 1. Elsevier. 2013, pp. 45–59 (cit. on p. 161).

[9] Oren Boiman, Eli Shechtman, and Michal Irani. “In defense of nearest-neighbor based image classification”. In: *2008 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2008, pp. 1–8 (cit. on pp. 1, 28, 110).

[10] Yuri Boykov and Vladimir Kolmogorov. “An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision”. In: *IEEE transactions on pattern analysis and machine intelligence* 26.9 (2004), pp. 1124–1137 (cit. on pp. 1, 2, 61, 69, 90, 96, 103).

[11] Eva-Maria Brinkmann. “Novel Aspects of Total Variation-Type Regularization in Imaging”. PhD thesis. WWU Münster, 2019 (cit. on pp. 5, 25, 26, 172).

[12] Eva-Maria Brinkmann et al. “Bias reduction in variational regularization”. In: *Journal of Mathematical Imaging and Vision* 59.3 (2017), pp. 534–566 (cit. on pp. 2, 121, 122, 151, 223).

[13] Thomas Bühler and Matthias Hein. “Spectral clustering based on the graph p-Laplacian”. In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM. 2009, pp. 81–88 (cit. on p. 17).

[14] Martin Burger and Stanley Osher. “A guide to the TV zoo”. In: *Level Set and PDE-based Reconstruction Methods*. Ed. by Osher Stanley Burger Martin. Lecture Notes in Mathematics. WWU::96089. 2013, pp. 1–70 (cit. on pp. 1, 5, 22).

[15] Martin Burger et al. *Level Set and PDE Based Reconstruction Methods in Imaging: Cetraro, Italy 2008, Editors: Martin Burger, Stanley Osher*. Vol. 2090. Springer, 2013 (cit. on pp. 6, 165).

[16] Florian Büther et al. “Impact of data-driven respiratory gating in clinical PET”. In: *Radiology* 281.1 (2016), pp. 229–238 (cit. on p. 180).

[17] Antonin Chambolle and Thomas Pock. “A first-order primal-dual algorithm for convex problems with applications to imaging”. In: *Journal of mathematical imaging and vision* 40.1 (2011), pp. 120–145 (cit. on pp. 1, 2, 27, 115, 150, 170, 171, 194, 215).

[18] Antonin Chambolle et al. “Stochastic primal-dual hybrid gradient algorithm with arbitrary sampling and imaging applications”. In: *SIAM Journal on Optimization* 28.4 (2018), pp. 2783–2808 (cit. on pp. 215, 225).

[19] Tony F Chan and Luminita A Vese. “Image segmentation using level sets and the piecewise-constant Mumford-Shah model”. In: *Tech. Rep. 0014, Computational Applied Math Group*. Citeseer. 2000 (cit. on pp. 2, 81).

[20] Chris A Cocosco et al. “Brainweb: Online interface to a 3D MRI simulated brain database”. In: *NeuroImage*. Citeseer. 1997 (cit. on p. 217).

[21] Gaspar Delso et al. “Performance measurements of the Siemens mMR integrated whole-body PET/MR scanner”. In: *Journal of nuclear medicine* 52.12 (2011), pp. 1914–1922 (cit. on pp. 161, 213).

[22] Hendrik Meinert Dirks. “Variational Methods for Joint Motion Estimation and Image Reconstruction”. PhD thesis. WWU Münster, 2015 (cit. on pp. 175, 181, 225).

[23] Wei Dong, Charikar Moses, and Kai Li. “Efficient k-nearest neighbor graph construction for generic similarity measures”. In: *Proceedings of the 20th international conference on World wide web*. 2011, pp. 577–586 (cit. on pp. 1, 28, 110).

[24] John Duchi et al. “Efficient projections onto the l 1-ball for learning in high dimensions”. In: *Proceedings of the 25th international conference on Machine learning*. ACM. 2008, pp. 272–279 (cit. on p. 39).

[25] Matthias J Ehrhardt and Marta M Betcke. “Multicontrast MRI reconstruction with structure-guided total variation”. In: *SIAM Journal on Imaging Sciences* 9.3 (2016), pp. 1084–1106 (cit. on pp. 204, 226).

[26] Matthias J Ehrhardt et al. “Faster PET reconstruction with a stochastic primal-dual hybrid gradient method”. In: *Wavelets and Sparsity XVII*. Vol. 10394. International Society for Optics and Photonics. 2017, 103941O (cit. on pp. 215, 216, 225).

[27] Abderrahim Elmoataz, Olivier Lezoray, and Sebastien Bougleux. “Nonlocal discrete regularization on weighted graphs: a framework for image and manifold processing”. In: *IEEE Trans. Image Processing* 17.7 (2008), pp. 1047–1060 (cit. on pp. 1, 17, 21, 22, 25).

[28] Abderrahim Elmoataz, Matthieu Toutain, and Daniel Tenbrinck. “On the p-Laplacian and ∞-Laplacian on Graphs with Applications in Image and Data Processing”. In: *SIAM Journal on Imaging Sciences* 8 (4 2015), pp. 2412–2451 (cit. on pp. 1, 17, 25, 26).

[29] Thomas Ersepke et al. “A contactless approach for respiratory gating in PET using continuous-wave radar”. In: *Medical physics* 42.8 (2015), pp. 4911–4919 (cit. on p. 180).

[30] Shachar Fleishman, Iddo Drori, and Daniel Cohen-Or. “Bilateral mesh denoising”. In: 22.3 (2003), pp. 950–953 (cit. on p. 154).

[31] Jonas Geiping et al. “Fast Convex Relaxations using Graph Discretizations”. In: (2020). arXiv: 2004.11075 [cs.CV] (cit. on pp. 133, 143, 224).

[32] Yves van Gennip et al. “Mean curvature, threshold dynamics, and phase field theory on finite graphs”. In: *Milan Journal of Mathematics* 82 (2014), pp. 3–65 (cit. on p. 17).

[33] Guy Gilboa and Stanley Osher. “Nonlocal operators with applications to image processing”. In: *Multiscale Modeling & Simulation* 7.3 (2008), pp. 1005–1028 (cit. on pp. 17, 21, 25).

[34] Tom Goldstein et al. “Adaptive primal-dual hybrid gradient methods for saddle-point problems”. In: *arXiv preprint arXiv:1305.0546* (2013) (cit. on pp. 64, 171, 215).

[35] L. Grady and C. V. Alvino. “The Piecewise Smooth Mumford–Shah Functional on an Arbitrary Graph”. In: *IEEE Transactions on Image Processing* 18.11 (2009), pp. 2547–2561 (cit. on p. 84).

[36] Eldad Haber and Michal Holtzman Gazit. “Model fusion and joint inversion”. In: *Surveys in Geophysics* 34.5 (2013), pp. 675–695 (cit. on p. 181).

[37] Mirco Heß, Florian Büther, and Klaus P Schäfers. “Data-driven methods for the determination of anterior-posterior motion in pet”. In: *IEEE transactions on medical imaging* 36.2 (2016), pp. 422–432 (cit. on p. 180).

[38] Mirco Heß et al. “A dual-Kinect approach to determine torso surface motion for respiratory motion correction in PET”. In: *Medical physics* 42.5 (2015), pp. 2276–2286 (cit. on p. 180).

[39] Jean-Baptiste Hiriart-Urruty and Claude Lemaréchal. *Convex analysis and minimization algorithms I: Fundamentals*. Vol. 305. Springer science & business media, 2013 (cit. on p. 6).

[40] A. Horé and D. Ziou. “Image Quality Metrics: PSNR vs. SSIM”. In: *2010 20th International Conference on Pattern Recognition*. 2010, pp. 2366–2369 (cit. on p. 138).

[41] IT Jolliffe. “Principal component analysis”. In: *Technometrics* 45.3 (2003), p. 276 (cit. on p. 117).

[42] Paul E Kinahan et al. “Attenuation correction for a combined 3D PET/CT scanner”. In: *Medical physics* 25.10 (1998), pp. 2046–2053 (cit. on p. 161).

[43] Vladimir Kolmogorov and Ramin Zabih. “What energy functions can be minimized via graph cuts?” In: *IEEE Transactions on Pattern Analysis & Machine Intelligence* 2 (2004), pp. 147–159 (cit. on pp. 31, 49).

[44] Thomas Kösters, Klaus P Schäfers, and Frank Wübbeling. “EMRECON: An expectation maximization based image reconstruction framework for emission tomography data”. In: *2011 IEEE Nuclear Science Symposium Conference Record*. IEEE. 2011, pp. 4365–4368 (cit. on p. 213).

[45] Solomon Kullback. *Information theory and statistics*. Courier Corporation, 1997 (cit. on pp. 11, 164).

[46] L. Landrieu and G. Obozinski. “Cut-Pursuit: Fast Algorithms to Learn Piecewise Constant Functions on General Weighted Graphs”. In: *SIAM Journal on Imaging Sciences* 10.4 (2017), pp. 1724–1766. URL: https://doi.org/10.1137/17M1113436 (cit. on pp. 1, 3, 41, 42, 44, 56, 84, 88, 90, 96, 125, 131, 132, 205, 223).

[47] Loic Landrieu. “Learning structured models on weighted graphs, with applications to spatial data analysis”. PhD thesis. PSL Research University, 2016 (cit. on pp. 1, 41, 131, 132, 207).

[48] Loic Landrieu and Martin Simonovsky. “Large-scale point cloud semantic segmentation with superpoint graphs”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4558–4567 (cit. on p. 132).

[49] Young-Su Lee and Soon-Yeong Chung. “Extinction and positivity of solutions of the p-Laplacian evolution equation on networks”. In: *Journal of Mathematical Analysis and Applications* 386 (2012), pp. 581–592 (cit. on p. 17).

[50] Lefteris Livieratos et al. “Respiratory gating of cardiac PET data in list-mode acquisition”. In: *European Journal of Nuclear Medicine and Molecular Imaging* 33.5 (2006), pp. 584–588 (cit. on p. 180).

[51] F. Lozes, A. Elmoataz, and O. Lezoray. “Partial Difference Operators on Weighted Graphs for Image Processing on Surfaces and Point Clouds”. In: *Image Processing, IEEE Transactions on* 23.9 (Sept. 2014), pp. 3896–3909 (cit. on pp. 17, 25, 149, 152).

[52] Francois Lozes. “Traitments d’images sur surfaces et variétés avec mise en application au patrimoine culturel 3D”. PhD thesis. Université Caen Normandie, 2006 (cit. on pp. 25, 149, 152).

[53] Richard Manber et al. “Joint PET-MR respiratory motion models for clinical PET motion correction”. In: *Physics in Medicine & Biology* 61.17 (2016), p. 6515 (cit. on p. 181).

[54] D. Martin et al. “A Database of Human Segmented Natural Images and its Application to Evaluating Segmentation Algorithms and Measuring Ecological Statistics”. In: *Proc. 8th Int’l Conf. Computer Vision*. Vol. 2. July 2001, pp. 416–423 (cit. on p. 111).

[55] Cyril Meurie et al. “Morphological hierarchical segmentation and color spaces”. In: *International Journal of Imaging Systems and Technology* 20.2 (2010), pp. 167–178 (cit. on p. 17).

[56] Michael Moeller et al. “Color Bregman TV”. In: *SIAM Journal on Imaging Sciences* 7.4 (2014), pp. 2771–2806 (cit. on p. 25).

[57] Jean Michel Morel and Sergio Solimini. “The Piecewise Constant Mumford and Shah Model: Mathematical Analysis”. In: *Variational Methods in Image Segmentation: with seven image processing experiments*. Boston, MA: Birkhäuser Boston, 1995, pp. 46–62. URL: https://doi.org/10.1007/978-1-4684-0567-5_5 (cit. on pp. 2, 81).

[58] Delio Mugnolo. *Semigroup Methods for Evolution Equations on Networks*. Springer New York, 2014 (cit. on p. 17).

[59] Frank Natterer and Frank Wübbeling. *Mathematical methods in image reconstruction*. Vol. 5. Siam, 2001 (cit. on pp. 161–163, 166).

[60] Philip J Noonan et al. “Accurate markerless respiratory tracking for gated whole body PET using the Microsoft Kinect”. In: *2012 IEEE Nuclear Science Symposium and Medical Imaging Conference Record (NSS/MIC)*. IEEE. 2012, pp. 3973–3974 (cit. on p. 180).

[61] Bui Tuong Phong. “Illumination for computer generated pictures”. In: *Communications of the ACM* 18.6 (1975), pp. 311–317 (cit. on p. 154).

[62] Thomas Pock and Antonin Chambolle. “Diagonal preconditioning for first order primal-dual algorithms in convex optimization”. In: (2011), pp. 1762–1769 (cit. on pp. 28, 171, 224).

[63] Thomas Pock et al. “An algorithm for minimizing the Mumford-Shah functional”. In: *2009 IEEE 12th International Conference on Computer Vision*. IEEE. 2009, pp. 1133–1140 (cit. on p. 27).

[64] Thomas Pock et al. “An algorithm for minimizing the Mumford-Shah functional”. In: *2009 IEEE 12th International Conference on Computer Vision*. IEEE. 2009, pp. 1133–1140 (cit. on p. 84).

[65] Julian Rasch. “Advanced Convex Analysis for Improved Variational Image Reconstruction”. PhD thesis. WWU Münster, 2018 (cit. on pp. 122, 172, 181, 188, 204).

[66] Julian Rasch, Eva-Maria Brinkmann, and Martin Burger. “Joint reconstruction via coupled Bregman iterations with applications to PET-MR imaging”. In: *Inverse Problems* 34.1 (Dec. 2017), p. 014001. URL: https://doi.org/10.1088%2F1361-6420%2Faa9425 (cit. on p. 13).

[67] Julian Rasch, Eva-Maria Brinkmann, and Martin Burger. “Joint reconstruction via coupled Bregman iterations with applications to PET-MR imaging”. In: *Inverse Problems* 34.1 (2017), p. 014001 (cit. on pp. 181, 204, 226).

[68] Julian Rasch and Antonin Chambolle. “Inexact first-order primal-dual algorithms”. In: *Computational Optimization and Applications* (Mar. 2020). URL: https://doi.org/10.1007/s10589-020-00186-y (cit. on pp. 214, 218).

[69] Julian Rasch et al. “Dynamic MRI reconstruction from undersampled data with an anatomical prescan”. In: *Inverse problems* 34.7 (2018), p. 074001 (cit. on pp. 181, 226).

[70] Andrew J Reader et al. “Joint estimation of dynamic PET images and temporal basis functions using fully 4D ML-EM”. In: *Physics in Medicine and Biology* 51.21 (Oct. 2006), pp. 5455–5474. URL: https://doi.org/10.1088%2F0031-9155%2F51%2F21%2F005 (cit. on pp. 2, 176, 181, 225).

[71] R Tyrrell Rockafellar. *Convex analysis*. 28. Princeton university press, 1970 (cit. on pp. 5, 84).

[72] Leonid I Rudin, Stanley Osher, and Emad Fatemi. “Nonlinear total variation based noise removal algorithms”. In: *Physica D: nonlinear phenomena* 60.1-4 (1992), pp. 259–268 (cit. on pp. 1, 8, 29).

[73] Alex Sawatzky. “(Nonlocal) Total Variation in Medical Imaging”. PhD thesis. WWU Münster, 2011 (cit. on pp. 161, 165, 172).

[74] Alex Sawatzky et al. “EM-TV methods for inverse problems with Poisson noise”. In: *Level Set and PDE Based Reconstruction Methods in Imaging*. Springer, 2013, pp. 71–142 (cit. on pp. 2, 3, 170–174, 225).

[75] Daniel Scharstein et al. “High-resolution stereo datasets with subpixel-accurate ground truth”. In: *German conference on pattern recognition*. Springer. 2014, pp. 31–42 (cit. on p. 111).

[76] Mark Schmidt, Nicolas L. Roux, and Francis R. Bach. “Convergence Rates of Inexact Proximal-Gradient Methods for Convex Optimization”. In: *Advances in Neural Information Processing Systems 24*. Ed. by J. Shawe-Taylor et al. Curran Associates, Inc., 2011, pp. 1458–1466. URL: http://papers.nips.cc/paper/4452-convergence-rates-of-inexact-proximal-gradient-methods-for-convex-optimization.pdf (cit. on p. 214).

[77] David I. Shuman et al. “The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains”. In: *IEEE Signal Processing Magazine* 30 (2013), pp. 83–98 (cit. on p. 17).

[78] Robert L Siddon. “Fast calculation of the exact radiological path for a three-dimensional CT array”. In: *Medical physics* 12.2 (1985), pp. 252–255 (cit. on p. 213).

[79] Lindsay I Smith. *A tutorial on principal components analysis*. Tech. rep. 2002 (cit. on p. 117).

[80] Suvrit Sra. “Fast projections onto l 1, q-norm balls for grouped feature selection”. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2011, pp. 305–317 (cit. on p. 39).

[81] Suvrit Sra. “Fast projections onto mixed-norm balls with applications”. In: *Data Mining and Knowledge Discovery* 25.2 (2012), pp. 358–377 (cit. on p. 38).

[82] Stanford University. *The Stanford 3D Scanning Repository*. URL: https://graphics.stanford.edu/data/3Dscanrep/ (cit. on pp. 111, 149).

[83] Evgeny Strekalovskiy and Daniel Cremers. "Real-time minimization of the piecewise smooth Mumford-Shah functional". In: *European conference on computer vision*. Springer. 2014, pp. 127–141 (cit. on pp. 84, 90, 92, 93, 96, 125, 129, 131, 132).

[84] Yujing Sun, Scott Schaefer, and Wenping Wang. "Denoising point sets via L0 minimization". In: *Computer Aided Geometric Design* 35 (2015), pp. 2–15 (cit. on pp. 154, 225).

[85] Daniel Tenbrinck, Fjedor Gaede, and Martin Burger. "Variational Graph Methods for Efficient Point Cloud Sparsification". In: (Mar. 7, 2019). arXiv: 1903.02858 [cs, math]. URL: http://arxiv.org/abs/1903.02858 (visited on 04/17/2019) (cit. on p. 149).

[86] A.N. Tikhonov and V.I.A. Arsenin. "Solutions of ill-posed problems". In: *Scripta series in mathematics* (1977) (cit. on pp. 3, 188).

[87] Silvia Villa et al. "Accelerated and inexact forward-backward algorithms". In: *SIAM Journal on Optimization* 23.3 (2013), pp. 1607–1633 (cit. on p. 214).

[88] Zhou Wang et al. "Image quality assessment: from error visibility to structural similarity". In: *IEEE transactions on image processing* 13.4 (2004), pp. 600–612 (cit. on p. 138).

[89] Miles N Wernick and John N Aarsvold. *Emission tomography: the fundamentals of PET and SPECT*. Elsevier, 2004 (cit. on p. 161).

[90] Sunil Kumar Yadav, Ulrich Reitebuch, and Konrad Polthier. "Mesh denoising based on normal voting tensor and binary optimization". In: *IEEE transactions on visualization and computer graphics* 24.8 (2018), pp. 2366–2379 (cit. on p. 225).

[91] Mehmet Yavuz and Jeffrey A Fessler. "Statistical image reconstruction methods for randoms-precorrected PET scans". In: *Medical Image Analysis* 2.4 (1998), pp. 369–378 (cit. on p. 161).

[92] Xiang Zeng, Wei Chen, and Qunsheng Peng. "Efficiently solving the piecewise constant mumford-shah model using graph cuts". In: *Technical report, Technical report, Dept. of Computer Science*. Citeseer, 2006 (cit. on p. 84).

[93] Yong Zhao et al. "Robust and effective mesh denoising using L0 sparse regularization". In: *Computer-Aided Design* 101 (2018), pp. 82–97 (cit. on p. 225).

[94] Youyi Zheng et al. "Bilateral normal filtering for mesh denoising". In: *IEEE Transactions on Visualization and Computer Graphics* 17.10 (2011), pp. 1521–1530 (cit. on p. 154).

[95] Xueyuan Zhou and Mikhail Belkin. "Semi-supervised learning by higher order regularization". In: *AISTATS*. 2011, pp. 892–900 (cit. on p. 17).