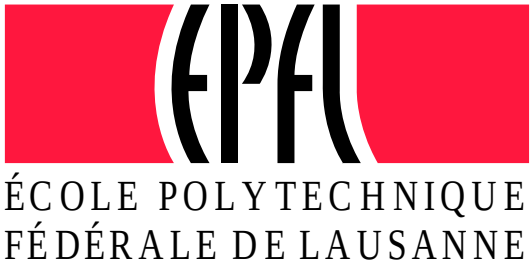

Master Thesis

Large scale stereo reconstruction by a minimum $s-t$ cut formulation

Zenklusen Rico

4th March 2005

Supervisor:
Prof. de Werra Dominique

Assistants:
Ekim Tınaz (ROSE)
Lagger Pascal (CVLAB)

ROSE

IMA, EPFL

Contents

Abstract **7**

Introduction **8**

1 A minimum $s-t$ cut formulation **10**
1.1 Introduction . . . 10
1.2 Preliminaries . . . 10
1.2.1 The Matching Volume . . . 10
1.2.2 Depth-representation and disparity-representation . . . 11
1.2.3 Valid surfaces . . . 12
1.3 Discretization . . . 13
1.3.1 Valid discretized surfaces . . . 14
1.3.2 Energy functions for valid discretized surfaces . . . 15
1.4 Network construction . . . 18
1.4.1 Introduction . . . 18
1.4.2 Preliminaries for directed networks . . . 19
1.4.3 The directed network $G = (V, E, k)$ corresponding to the stereo problem . . . 20
1.4.4 The general form of the network G we consider . . . 29

2 Preliminaries concerning flows and $s-t$ cuts **31**
2.1 Flows . . . 31
2.2 $s-t$ cuts . . . 35
2.3 Relationships between flows from s to t and $s-t$ cuts . . . 36

3 Algorithms for the resolution of the minimum $s-t$ cut problem **38**
3.1 Introduction . . . 38
3.2 Preliminaries concerning the complexity analysis . . . 39
3.2.1 Basic notations . . . 39
3.3 Upper bounds of the maximum flow value and preprocessing in a reconstruction network . . . 40
3.3.1 Preprocessing algorithm 1 . . . 40
3.3.2 Preprocessing algorithm 2 . . . 41
3.4 Augmenting path algorithms . . . 48
3.4.1 Introduction . . . 48

3.4.2 General labelling algorithm and Edmonds-Karp algorithm 49
3.4.3 Algorithm of Dinic 50
3.4.4 Capacity scaling algorithm 52
3.4.5 MA-ordering algorithm 53
3.4.6 Growing-trees algorithm 54
3.4.7 Algorithm of Goldberg and Rao 55
3.5 Preflow-push algorithms 56
3.5.1 Introduction and generic preflow-push algorithm 56
3.5.2 FIFO algorithm 62
3.5.3 Highest label algorithm 62
3.5.4 Wave algorithm 63
3.6 Performances of the different flow algorithms on reconstruction networks . 64
3.6.1 Sorting out algorithms unlikely to be efficient in practice 64
3.6.2 Empirical tests and analyzes 65
3.6.3 Choice of an appropriate algorithm 67

4 Disparity reduction **68**
4.1 Introduction 68
4.2 Graph formulations for disparity reduction 69
4.3 Internal and border conditions 75
4.4 Some important properties concerning internal and border conditions . . . 76
4.4.1 Completion property 76
4.4.2 Independence property 77
4.4.3 Monotonicity property 78
4.4.4 Application: Error characterization of subproblems 83
4.4.5 Application: Constructing locally upper and lower bounds for the nearest optimal solution 85
4.5 Application: Pyramidal approach 87
4.5.1 Some details on the implementation 87

5 Parallelization **91**
5.1 Introduction 91
5.2 Error analysis for determining the sets A and B 93
5.2.1 Diminishing the overlapping width through intelligent image-cutting 96
5.3 Some completion methods 97
5.4 Completion method allowing to give an error bound on the energy 102
5.5 Generalization of the division into two subproblems 105
5.6 A parallelizable method for finding the optimal solution 105

6 Code **108**

Conclusion **110**

List of Figures

1.1 Typical choice of the matching volume as a truncated pyramid corresponding to the viewing volume of the reference camera. 12
1.2 Matching space with a uniform discretization of the disparities. 14
1.3 Influence of the smoothness term. 17
1.4 The vertex set V' as another representation of the matching space. 22
1.5 The reconstruction graph G without capacities. 25
1.6 Smoothness energy formulated as capacities of smoothness arcs. 28

3.1 Network corresponding to the second preprocessing method. 43
3.2 The graph G^* that corresponds to $\widehat{G}$. 44
3.3 Finding a minimum $s-t$ cut $\widehat{C} = [\widehat{V}_s, \overline{\widehat{V}_s}]$ in $\widehat{G}$ by a shortest path $\widehat{p}$ from s^* to t^* in G^*. 45
3.4 Example illustrating the idea behind the highest label selection rule. . . . 62

4.1 A representation of the graph for the resolution of a problem with disparity reduction on intervals. 71
4.2 Illustration of the problem when we do not change capacities in the formulation for disparity reduction. 71
4.3 Formulation of a simple problem in 2D with reduced disparities. 72
4.4 A step by step reasoning of the network modification for problems with reduced disparities on intervals. 73
4.5 A special case that can arrive in disparity reduction by intervals. 74
4.6 A 2D example of our formulation for disparity reduction on intervals. . . 75
4.7 Illustration of the independence property. 77
4.8 Possible constellation for the monotonicity property. 79
4.9 Theoretical characterization of errors in subproblems. 84
4.10 Impossible error constellation in a subproblem. 85
4.11 An empirical test for determining the quality of an upper bound. 86
4.12 Illustration of the proposed pyramidal approach. 88

5.1 Simple method for dividing the original problem into two subproblems. . . 92
5.2 Example of a reference image and the associated nearest optimal solution. 93
5.3 A testserie for errors on subproblems. 94
5.4 Diagram representing the maximal reconstruction errors we still have when cutting columns on the right side. 95

5.5 Diagram representing the sum of the remaining disparity errors we still have when cutting columns on the right side. 96
5.6 Simple completion through straight division 98
5.7 Completion through merging path. 100
5.8 Choosing the merging path as a shortest path on a dual-like graph. 101
5.9 Dividing the original problem in more than two subproblems. 106
5.10 Breaking the non fixed region into independent problems. 107

Abstract

An important problem in computer vision is the reconstruction problem which consists in reconstructing a three dimensional surface on the base of different images of the surface. The classical formulations for solving this problem work with the so called ≪epipolar lines≫ that reduce the problem to one dimensional problems. Unfortunately they are an oversimplification of the reality and often yield solutions with limited quality. In 1998 a minimum $s-t$ cut formulation of the reconstruction problem was proposed, that achieves in general more accurate results than the classical methods but is much more expensive in computational time and memory needs. Therefore, this new formulation is only of limited usability for large problems.
In this master thesis we propose methods that allow to speed up this resolution technique and to find accurate approximations by parallelizing the problem. Furthermore we introduce a formulation allowing to reduce locally the reconstruction region and to profit from preliminary knowledge of the surface by imposing border and internal conditions. Based on this formulation we introduce a pyramidal approach that allows to reduce the complexity of the problem and that can be combined with parallelization techniques. Most of the proposed methods and algorithms were implemented in C++ such that they can be applied to real problems.

Introduction

The reconstruction problem in stereo is often described as follows. Given two or more images of a surface in 3D (with eventually some additional parameters of the cameras that have taken the images), how can we reconstruct the surface? The domain handling this problem is also known as ≪stereo≫ and the reconstruction problem as ≪stereo problem≫ or ≪stereo matching problem≫. The reconstruction problem can also be formulated with more general 3D objects than only surfaces. But often we can reduce this kind of problems, to the problem of reconstructing a 3D surface.
From a theoretical point of view, it is in general impossible to reconstruct exactly a surface on the basis of a certain number of images and parameters of the cameras. Therefore to solve the reconstruction problem means to approximate the real 3D surface ≪as good as possible≫.

In the last decades several formulations and methods were proposed to distinguish good from bad approximations and to solve the reconstruction problem. Most of them are based on the so called ≪epipolar lines≫. These methods allow to reduce the reconstruction problem to one dimensional problems (along the epipolar lines) that can be solved efficiently by dynamic programming (see for example [6] for more information). The drawback of these methods is that they often create artifacts [(1)] (typically perpendicular to the epipolar lines) because they use an oversimplified formulation of the reconstruction problem that can only guarantee smoothness of the reconstructed object in one dimension (along the epipolar lines). Another problem of these methods is that they often allow only two images for the reconstruction of a surface.

In 1998 a new formulation of the reconstruction problem, that allowed to find solutions that are more accurate and contain less artifacts, was proposed by Sébastien Roy and Ingemar Cox in [2]. Additionally, their formulation allowed to use an arbitrary number of images (≥ 2) for the reconstruction. In this formulation the resolution of the problem reduces to find a minimum $s-t$ cut [(2)] in a network. Since then, numerous other propositions were made to solve the stereo problem by finding a minimum $s-t$ cut.
Unfortunately, even for relatively little reconstruction problems, we get huge corresponding networks with several millions of vertices and edges. Therefore, large problems become quickly intractable because of memory needs and executional time.

[(1)] Artifacts are unrealistic discontinuities in the reconstruction.
[(2)] An $s-t$ cut is a partition of the vertex set of a network into two parts, and its value is the sum of the capacities on the arcs that are connecting these two parts.

The main objectives of this master thesis is to develop further this formulation of the reconstruction problem in the following points:

- Theoretical and practical comparison of different algorithms for the resolution of the minimum $s-t$ cut problem.
- Development of methods for treating internal and border conditions.
- Development of fast methods for the resolution of the reconstruction problem.
- Development of methods that allow to find approximative solutions by parallelizing the problem and therefore make it possible to solve large reconstruction problems.
- Implementation of an efficient program for the resolution of the reconstruction problem as well as the implementation of different proposed methods.

This work is divided into six chapters. In a first chapter we explain how the reconstruction problem can be formulated as a minimum $s-t$ cut problem. In a second chapter some important background about flows and $s-t$ cuts are introduced. Chapter three discusses different algorithms for the resolution of the minimum $s-t$ cut problem. In chapter four, we develop formulations for treating internal and border conditions as well as more general restrictions. The fifth chapter handles parallelization methods. Finally, in chapter six we shortly discuss the code we have written for this work.

1 A minimum $s-t$ cut formulation

1.1 Introduction

This chapter describes how we can formulate the stereo problem as a problem of finding a minimum $s-t$ cut in a network. The formulation that will be described in the following pages is based on article [1].
This method imposes some minor restrictions on the surface to reconstruct. Also some preliminary knowledge of the position of the surface to reconstruct is required. In a first section, we will precise these two points and introduce some basic definitions. A surface that we can try to reconstruct with this formulation will be called a *valid surface*. The basic idea in the following introduced formulation is to find the «optimal» valid surface by energy minimization. Unfortunately, an energy minimization on the set of valid surfaces would be a difficult variational problem that in general has to be solved by numerical methods. That is why we will discretize our model in a following section. In a third section, an energy function will be introduced on the discretized model that will allow us to give a formal description of our formulation as an optimization problem. Finally this optimization problem will be identified with a minimum $s-t$ cut problem in a network.

1.2 Preliminaries

This section will precise the traits of the surfaces we can try to reconstruct with the minimum $s-t$ cut formulation and introduce some basic definitions.

1.2.1 The Matching Volume

At first, a volume $\mathcal{MV}$ of the 3D world is selected to constrain where the stereo matching should occur. This volume will be referred as the *matching volume* and has to satisfy some properties. It has to be possible to define a *front region* $\mathcal{FR}$ and a *back region* $\mathcal{BR}$ of the matching volume that are disjoint, such that every point in the matching volume is between this two regions. Additionally the surface we are trying to reconstruct has to cut the matching volume in two areas such that the front region lies entirely in one area and the back region entirely in the other.
It is common to choose the matching volume as a truncated pyramid corresponding to the viewing volume of a fixed camera (see figure 1.1). This camera is called the «reference camera» and the corresponding image «reference image». To simplify the

explanations, we will concentrate on this case.

1.2.2 Depth-representation and disparity-representation

We now introduce a coordinate system for the matching volume. We fix a point p in the real world. Let q_{ref} be the (euclidian) distance between the projection center of the reference camera and the reference image and let w_p be the distance between p and the reference image. So we can represent p as $p = (p_x, p_y, p_z)_{depth}$ where (p_x, p_y) are the coordinates of the projection of p on the reference image and $p_z = q_{ref} + w_p$. The z-component p_z is called the *depth* of point p and we call the representation $p = (p_x, p_y, p_z)_{depth}$ the *depth-representation* of p. With this coordinate system, we can describe the matching volume $\mathcal{MV}$, the front $\mathcal{FR}$ and the back $\mathcal{BR}$ in the following way:

$$\begin{aligned}
\mathcal{MV} &= \{(x, y, z)_{depth} \mid 0 \leq x \leq x_{max},\ 0 \leq y \leq y_{max},\ z_{min} \leq z \leq z_{max}\} \\
\mathcal{FR} &= \{(x, y, z_{min})_{depth} \mid 0 \leq x \leq x_{max},\ 0 \leq y \leq y_{max}\} \\
\mathcal{BR} &= \{(x, y, z_{max})_{depth} \mid 0 \leq x \leq x_{max},\ 0 \leq y \leq y_{max}\}
\end{aligned}$$

We can replace the depth component of a point by another term that we call disparity. We define the disparity p_d of a point p as

$$p_d = \frac{1}{p_z} \quad (1)$$

So we can represent a point p also as $p = (p_x, p_y, p_d)_{disp}$. This is the *disparity-representation* of point p. In general we will use the disparity-representation because it has an important advantage compared to the depth-representation that we will see later. We have the following equalities:

$$\begin{aligned}
\mathcal{MV} &= \{(x, y, d)_{disp} \mid 0 \leq x \leq x_{max},\ 0 \leq y \leq y_{max},\ d_{min} \leq d \leq d_{max}\} \\
\mathcal{FR} &= \{(x, y, d_{max})_{disp} \mid 0 \leq x \leq x_{max},\ 0 \leq y \leq y_{max}\} \\
\mathcal{BR} &= \{(x, y, d_{min})_{disp} \mid 0 \leq x \leq x_{max},\ 0 \leq y \leq y_{max}\} \\
&\text{where} \quad d_{min} = \frac{1}{z_{max}} \\
&\qquad\quad\ \ d_{max} = \frac{1}{z_{min}}
\end{aligned}$$

(1) In fact the traditional definition of disparity in stereo is a little bit different (for further information see [6]). But there too, the disparity p_d is proportional to $\frac{1}{p_z}$. For our purposes, the factor of proportionality is not of great interest.

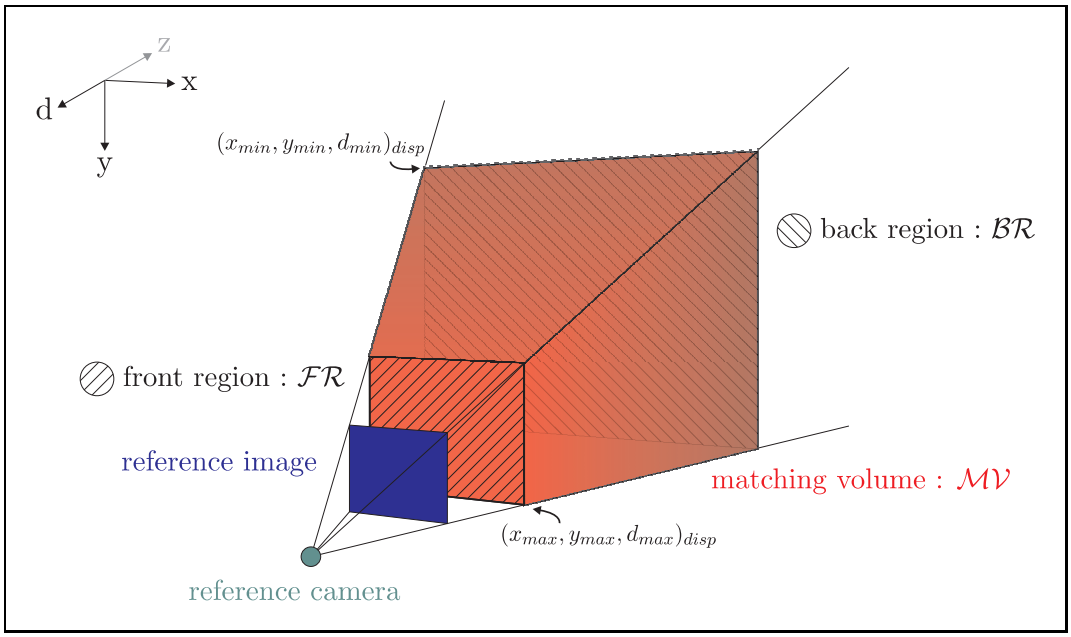

Figure 1.1: Typical choice of the matching volume as a truncated pyramid corresponding to the viewing volume of the reference camera.

1.2.3 Valid surfaces

With the previous notations we can now give a definition of a valid surface. A *valid surface* (with respect to the matching volume $\mathcal{MV}$) is a surface that cuts each straight line of the type $D_{(x,y)} = \{(x, y, d)_{disp} \mid d_{min} \leq d \leq d_{max}\}$ exactly once for all $(x, y) \in [0, x_{max}] \times [0, y_{max}]$. In other words, a valid surface can be parameterized by its front (as well as back) region, i.e. if we define the *domain* $\mathcal{D}$ as

$$\mathcal{D} = [0, x_{max}] \times [0, y_{max}]$$

so a surface S is valid, if and only if, we can parameterize it as follows:

$$\begin{aligned} S: \ \mathcal{D} &\longrightarrow \mathcal{MV} \\ (x, y) &\longmapsto (x, y, f_S(x, y))_{disp} \\ \text{with} \quad f_S: \ \mathcal{D} &\longrightarrow [d_{min}, d_{max}] \end{aligned}$$

So a valid surface assigns to every point in $\mathcal{D}$ exactly one disparity. Because of this parameterization property, a valid surface is often called *disparity surface* or *depth surface*. We denote with $\mathcal{S}$ the set of all valid surfaces with respect to $\mathcal{MV}$. Additionally, we introduce the notation $\mathcal{F} = \{f_S \mid S \in \mathcal{S}\}$ which is the set of all *disparity mappings*. Because of the natural one-to-one correspondence between elements in $\mathcal{S}$ and $\mathcal{F}$, we denote

by S_f the valid surface corresponding to the disparity mapping $f \in \mathcal{F}$.

The surface $\widetilde{S}$ that we are trying to reconstruct, has to be a valid surface. This is an important additional restriction.

1.3 Discretization

The next step in the formulation is the discretization of the previously defined model.

The Matching Space

The discretized matching volume will be called *matching space* and will be denoted by $\mathcal{MS}$. There have been several propositions how to discretize the matching volume. The simplest and most intuitive way of discretization (that is also used in [1]) uses a 3D-grid as in Fig. 1.2. Here we have a uniform discretization in terms of the disparity-representation with n_x, n_y, n_d points in the corresponding axes. So we have the following meshs for directions x,y and d:

$$\begin{aligned}\Delta x &= \frac{x_{max}}{n_x - 1} \\ \Delta y &= \frac{y_{max}}{n_y - 1} \\ \Delta d &= \frac{d_{max} - d_{min}}{n_d - 1}\end{aligned}$$

The matching space can now be written as follows:

$$\begin{aligned}\mathcal{MS} = \{(x_i, y_j, d_k)_{disp} \mid\ & i \in \{0, 1, \ldots, n_x - 1\}, \qquad \text{where} \quad x_i = i \cdot \Delta x \\ & j \in \{0, 1, \ldots, n_y - 1\}, \qquad\qquad\ \ y_j = j \cdot \Delta y \\ & k \in \{0, 1, \ldots, n_d - 1\}\} \qquad\qquad\ \ d_k = d_{min} + k \cdot \Delta d\end{aligned}$$

It is important to note that the grid density of the matching space (i.e. the number of points in $\mathcal{MS}$ per unit volume) decreases linearly with respect to the depth (see Fig. 1.2). This property is particularly interesting because in the following we will try to approximate the surface $\widetilde{S}$ with a discretized surface that is defined on the matching space. So for big depth values, we will have a less detailed description of the surfaces as the 3D-grid is less dense. This corresponds well to the information that we have on the reference image because a region of a surface, that is near to the reference camera, will be described in more details on the reference image than a farer region. So with a uniform discretization in terms of the disparity-representation, we have that the mesh density in the matching space corresponds to a uniform «information density» on the reference image. This is an important advantage of the disparity-representation compared to the depth-representation.

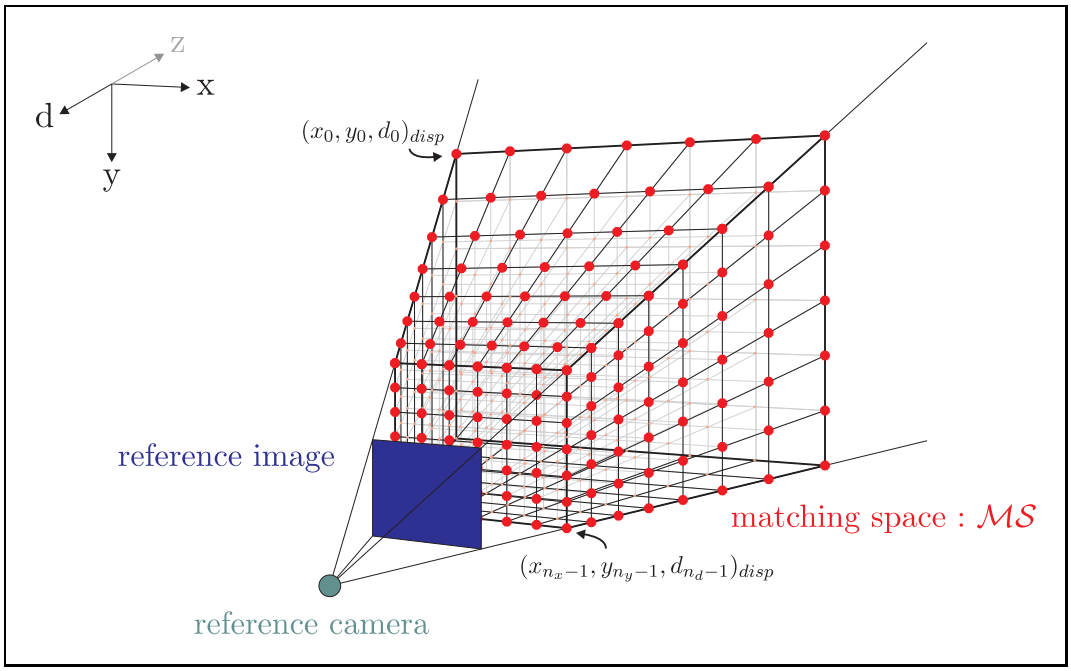

Figure 1.2: Matching space with a uniform discretization of the disparities.

1.3.1 Valid discretized surfaces

In a analogue manner as by the valid surfaces, we can now define a *valid discretized surface* (with respect to the matching space $\mathcal{MS}$). We begin with the definition of the discretized domain $\mathcal{D}^d$.

$$\mathcal{D}^d = \{x_0, x_1, \ldots, x_{n_x-1}\} \times \{y_0, y_1, \ldots, y_{n_y-1}\}$$

So S^d is a *valid discretized surface* if it is a function that can be written in the following way:

$$\begin{aligned} S^d : \ \mathcal{D}^d &\longrightarrow \mathcal{MS} \\ (x, y) &\longmapsto (x, y, f^d_{S^d}(x, y))_{disp} \\ \text{with} \quad f^d_{S^d} : \ \mathcal{D}^d &\longrightarrow \{d_0, d_1, \ldots, d_{n_d-1}\} \end{aligned}$$

Analogue to the continuous formulation, a valid surface assigns to every point in the discretized domain $\mathcal{D}^d$ exactly one disparity. The set of all valid discretized surfaces with respect to $\mathcal{MS}$ will be denoted by $\mathcal{S}^d$ and additionally, we introduce the notation $\mathcal{F}^d = \{f^d_{S^d} \mid S^d \in \mathcal{S}^d\}$ which represents the set of all *discretized disparity mappings*. Because of the natural one-to-one correspondence between elements in $\mathcal{S}^d$ and $\mathcal{F}^d$, we

write $S^d_{f^d}$ for the valid discretized surface corresponding to the discretized disparity mapping $f^d \in \mathcal{F}^d$.

1.3.2 Energy functions for valid discretized surfaces

In this subsection we will introduce energy functions for the set of all discretized surfaces $\mathcal{S}^d$. Such an energy function $\mathbf{E}$, is a function of the type

$$\mathbf{E}: \quad \mathcal{S}^d \longrightarrow \mathbb{R}_+$$

and should associate low energies for elements in $\mathcal{S}^d$ that represent good approximations of the surface $\widetilde{S}$ to reconstruct and high energies for bad approximations.

Before explaining the way of characterizing good approximations in [1], we introduce some notations. Let us suppose that we have N images (in black and white) of the surface to reconstruct. So each point $p = (x, y, d)_{disp} \in \mathcal{MV}$ can be projected on every of the N images and there we can associate an intensity to this projected point. Let $v_l(x, y, d) = v_l(p)$ be this intensity of the projection of p onto image l (where $l \in \{1, 2, \ldots, N\}$).
So we could say that a surface S^d is a good approximation of the real surface $\widetilde{S}$, if we have that for each point $p \in S^d$ the intensities $v_1(p), v_2(p), \ldots, v_n(p)$ are similar. This similarity could be measured by an empirical variance of the type

$$\begin{aligned} \kappa_{\mathrm{v}} : \mathcal{MS} &\longrightarrow \mathbb{R}_+ \\ p &\longmapsto \frac{1}{N}\sum_{l=1}^{N}(v_l(p) - \overline{v}(p))^2 \end{aligned}$$

$$\text{where} \qquad \overline{v}(p) = \frac{1}{N}\sum_{l=1}^{N} v_l(p)$$

and a corresponding energy function $\mathbf{EC}_{\kappa_{\mathrm{v}}} : \mathcal{S}^d \longrightarrow \mathbb{R}_+$ could be

$$\mathbf{EC}_{\kappa_{\mathrm{v}}}(S^d) = \sum_{i=0}^{n_x-1}\sum_{j=0}^{n_y-1} \kappa_{\mathrm{v}}\left((x_i, y_j, f^d_{S^d}(x_i, y_j))_{disp}\right)$$

This idea is based on a hypothesis that is often made in stereo, the so called lambertien hypothesis on $\widetilde{S}$. The surface $\widetilde{S}$ is called lambertian, if the intensity of a point $p \in \widetilde{S}$ does not depend on the point of view. In particular, if $\widetilde{S}$ is lambertian and $p \in \widetilde{S}$ is a point that is seen from all n images, so we have

$$v_1(p) = v_2(p) = \cdots = v_n(p)$$

This would be the ideal case for our first criteria for good approximations. But unfortunately, real surfaces are not perfectly lambertian but only approximately and a point

$p \in \widetilde{S}$ is often not visible on all n images. As a result, an optimal surface obtained by minimizing an energy function of the type $\mathbf{EC}_{\kappa_v}$, is in general not very smooth and shows a lot of unrealistic discontinuities. Fig 1.3 b) shows a result of such an optimization.
That's why typical energies for the stereo problem contain at least two terms. One term is an energy function of the type $\mathbf{EC}_{\kappa_v}$, that is called the *consistency term*, to which we add another term that tries to favor smooth surfaces, called *smoothing term*. The smoothing term **ES** used in [1] has the following form:

$$\begin{aligned} \mathbf{ES} : \mathcal{S}^d &\longrightarrow \mathbb{R}_+ \\ S^d &\longmapsto \sum_{\substack{q_1,q_2 \in \mathcal{D}^d, \\ \text{neighbors}}} K|f^d_{S^d}(q_1) - f^d_{S^d}(q_2)| \\ \text{where} \quad K &\in \mathbb{R}_+ \end{aligned}$$

Where two points $q_1 = (x_{i_1}, y_{j_1}), q_2 = (x_{i_2}, y_{j_2}) \in \mathcal{D}^d$ are considered as *neighbors*, if they are vertically or horizontally adjacent to each other in $\mathcal{D}^d$, i.e. if

$$\begin{aligned} i_2 = i_1 \text{ and } j_2 \in \{j_1 - 1, j_1 + 1\} \quad \text{or} \\ j_2 = j_1 \text{ and } i_2 \in \{i_1 - 1, i_1 + 1\} \end{aligned}$$

and K is a constant that gives a weight to the smoothing term and is therefore called *smoothing factor*.
Finally, the energy function **E** that is used in [1], has the following form

$$\begin{aligned} \mathbf{E} : \mathcal{S}^d &\longrightarrow \mathbb{R}_+ \\ S^d &\longmapsto \mathbf{EC}_{\kappa_v}(S^d) + \mathbf{ES}(S^d) \end{aligned}$$

and the corresponding optimization problem is

$$\underset{S^d \in \mathcal{S}^d}{argmin} \left(\mathbf{E}(S^d) \right) \tag{1.1}$$

This is the kind of problem, we will solve afterwards by graph cut methods. Figure 1.3 c) shows an example of an optimal solution using this formulation.

Other consistency terms

The choice of an appropriate energy function is not a trivial task. Various energy functions have been proposed for the resolution of the stereo problem.
By the definition of our matching space, it is for example interesting to give a greater influence to the reference image in the consistency term, because we know that every reconstructed point is visible from the reference camera. If we denote by $l_{ref} \in \{1, 2, \ldots, N\}$

a) b) c)

Figure 1.3: **a)** A colored version of the reference image. **b)** Minimization result corresponding to the energy function $\mathbf{EC}_{\kappa_\mathrm{v}}$ (which has no smoothness term). **c)** Solution of the problem 1.1 (that corresponds to an energy function containing a smoothness term).

the index of the reference camera, so we can define such a consistency term $\mathbf{EC}_{\kappa'_\mathrm{v}}$ in the following way.

$$\begin{aligned} \mathbf{EC}_{\kappa'_\mathrm{v}}(S^d) &= \sum_{i=0}^{n_x-1}\sum_{j=0}^{n_y-1} \kappa'_\mathrm{v}\left((x_i, y_j, f^d_{S^d}(x_i, y_j))_{disp}\right) \\ \text{where} \qquad \kappa'_\mathrm{v} &= \frac{1}{N}\sum_{l=1}^{N}(v_l(p) - v_{l_{ref}}(p))^2 \end{aligned} \tag{1.2}$$

The consistency terms $\mathbf{EC}_{\kappa_\mathrm{v}}$ and $\mathbf{EC}_{\kappa'_\mathrm{v}}$ are both based on the lambertian assumption and try to compare intensities of a points p that is projected on the different images. A different approach is to compare the intensity structures obtained by the projection of a neighborhood of p, on the different images. This approach is less dependent on the lambertian assumption and gives very interesting results. On the other hand, the calculation of such an energy function is much more expensive in computational time. Many well known energy functions of this type use a technique called ≪normalized cross correlation≫ (NCC) to compare the projected regions.

A lot of interesting consistency terms need preliminary optimization techniques for their calculation and are therefore very expensive in computational time.

Projections

Consistency terms typically use projections of a point on the different images. The theory of *projective geometry* gives powerful tools for implementing efficiently projections of this type. A good reference for the use of projective geometry in vision is [6].

Energy functions considered in this document

Unfortunately, there are energy functions that give optimization problems that cannot be solved efficiently by graph cut methods. For this reason, we will focus on some special energy functions in this document. The consistency terms we consider in this document, are terms of the type

$$\begin{aligned} \mathbf{EC}_\kappa(S^d) &= \sum_{i=0}^{n_x-1}\sum_{j=0}^{n_y-1} \kappa\left((x_i, y_j, f^d_{S^d}(x_i, y_j))_{disp}\right) \\ \text{where} \qquad \kappa &: \mathcal{MS} \longrightarrow \mathbb{R}_+ \end{aligned} \tag{1.3}$$

and for the smoothing term, we typically use **ES**. The function κ is called a *matching cost function*.
This leads to an energy function of the type

$$\mathbf{E}_\kappa = \mathbf{EC}_\kappa + \mathbf{ES} \tag{1.4}$$

and to the following optimization problem

$$\underset{S^d \in \mathcal{S}^d}{argmin}\left(\mathbf{E}_\kappa(S^d)\right) = \underset{S^d \in \mathcal{S}^d}{argmin}\left(\mathbf{EC}_\kappa(S^d) + \mathbf{ES}(S^d)\right) \tag{1.5}$$

In the next section, we will see how such an optimization problem can be formulated as a graph cut problem on a graph with a cubic topology.

The interested reader can find an analysis on the type of energy functions that lead to optimization problems which can be solved efficiently via graph cuts methods in [8].

Energy functions induced by variational formulations

Numerous energy functions can be seen as discretizations of energy functionals on a continuous problem that works on the matching volume $\mathcal{MS}$ and on the valid surface space $\mathcal{S}$. In particular, any energy of the type 1.4 can be seen as discretization of an energy functional on $\mathcal{S}$. In [5] can be found a short explanation of this passage. More general energy functionals that can be solved by discretization, can be found in [3] or [4].

1.4 Network construction

1.4.1 Introduction

In this section, we will begin by introducing some notations concerning directed graphs. Then we will construct, on the base of an energy function of the type 1.4, a directed

network $G = (V, E, k)$ [(2)], such that there is a natural correspondence between valid discretized surfaces $S^d \in \mathcal{S}^d$ and $s-t$ cuts in G. Additionally, we will see that with an appropriate choice of the capacity function k, we can achieve that the energy value $\mathbf{E}(S^d)$ is equal to the value of the $s-t$ cut in G that corresponds to S^d.

It is interesting to note that it is also possible to construct an undirected network, very similar to G, which allows to solve the problem 1.5 by graph cut techniques as well. A description of the network construction can be found in [1]. For our purposes, we only work with the directed network.

1.4.2 Preliminaries for directed networks

This subsection introduces some additional notations for directed networks. Let $G = (V, E, k)$ be a directed network. We begin by notations concerning vertices and edges and then introduce some notations that simplify the work with capacity functions.

Notation 1.4.1 (Complementary of a vertex set)**.** *The complementary $\overline{V_1}$ of a vertex set $V_1 \subset V$ is defined as*

$$\overline{V_1} = V \setminus V_1$$

Notation 1.4.2 (Cocircuit and cocycles)**.** *Let $V_1 \subset V$ be a subset of V. The positive cocircuit $\omega^+(V_1)$ of V_1 is the set of all edges in E exiting from V_1.*

$$\omega^+(V_1) = \{(v, w) \in E \mid v \in V_1, w \notin V_1\}$$

Analogously we define the negative cocircuit $\omega^-(V_1)$ of V_1 to be the set of all edges entering in V_1.

$$\omega^-(V_1) = \{(v, w) \in E \mid v \notin V_1, w \in V_1\} = \omega^+(\overline{V_1})$$

The cocycle $\omega(V_1)$ is the union of the positive and negative cocircuits of V_1.

$$\omega(V_1) = \omega^+(V_1) \cup \omega^-(V_1)$$

Notation 1.4.3 (Edges from V_1 to V_2)**.** *Let $V_1, V_2 \subset V$ be two subsets of V, so we introduce the following notation for the set of all edges from V_1 to V_2*

$$\omega^+(V_1, V_2) = \omega^+(V_1) \cap \omega^-(V_2)$$

Additionally we define

$$\omega^-(V_1, V_2) = \omega^+(V_2, V_1)$$

[(2)] V is the vertex set, E the set of edges, and $k : E \to \mathbb{R}_+ \cup \{\infty\}$ a capacity function. We use the term *network* to emphasize that we are considering a graph with capacities on its edges. Sometimes we use simply the term *graph* for a network if the capacities are not of great importance.

Notation 1.4.4 (Capacity of a set of edges)**.** *We use the following notation for capacity functions. If $U \subset E$, so*

$$k(U) = \sum_{u \in U} k(u)$$

Notation 1.4.5 (Capacities of the edges from V_1 to V_2)**.** *We now introduce a simple notation for the capacities of the edges from V_1 to V_2*

$$k(V_1, V_2) = k(\omega^+(V_1, V_2))$$

Property 1.4.6. *If $U_1, U_2 \subset E$ with $U_1 \cap U_2 = \emptyset$ so we have*

$$k(U_1 \cup U_2) = k(U_1) \cup k(U_2)$$

And as a consequence of this we have, if $V_{11}, V_{12}, V_{21}, V_{22} \subset V$ with $V_{11} \cap V_{12} = \emptyset$ and $V_{21} \cap V_{22} = \emptyset$, so

$$k(V_{11} \cup V_{12}, V_{21} \cup V_{22}) = k(V_{11}, V_{21}) + k(V_{11}, V_{22}) + k(V_{12}, V_{21}) + k(V_{12}, V_{22})$$

1.4.3 The directed network $G = (V, E, k)$ corresponding to the stereo problem

This subsection describes the directed network $G = (V, E, k)$ for the resolution of the problem 1.5. In a first step, we introduce the non terminal vertices V' of G and present one-to-one correspondences between V' and discretized terms introduced before. In a second step, we will complete the network construction by adding the terminals s and t, arcs and defining the capacity function k. Finally, we will treat the relationship between $s-t$ cuts in G and valid discretized surfaces and show how to solve the problem 1.5 by a minimum $s-t$ cut on G. Because the network G allows to solve the reconstruction problem, we will call it *reconstruction network*.

The vertex set $V' \subset V$ and his relations with the matching space

For the construction of the directed network G, we begin by partitioning the set of vertices V into the set of terminals $\{s, t\}$ and a set V'

$$V = V' \cup \{s, t\}$$

where the set V' represents a 3D-grid, that can be embedded in $\mathbb{N}^3$ or $\mathbb{R}^3$ and is defined as follows:

$$V' = \{0, 1, \ldots, n_a - 1\} \times \{0, 1, \ldots, n_b - 1\} \times \{0, 1, \ldots, n_c - 1\} \subset \mathbb{N}^3 \subset \mathbb{R}^3$$
$$\text{where} \quad n_a = n_x$$
$$n_b = n_y$$
$$n_c = n_d + 1$$

The set of vertices V' can be viewed as another representation of the elements in $\mathcal{MS}$ where we simply have added an additional slice as illustrated in Figure 1.4. To avoid ambiguities with the previous section, we nominate the three cartesian axis of the $\mathbb{N}^3$ respectively $\mathbb{R}^3$ space, where V' is embedded, with a-axis, b-axis and c-axis (they correspond to the x-, y- and d-axis of the matching space). So a vertex $v \in E'$ can be written as $v = (v_a, v_b, v_c)$.

Additionally, we define the *graph front* V_F and the *graph back* V_B of G as follows:

$$\begin{aligned} V_F &= \{(a,b,0) \in V' \quad\quad | \; a \in \{0,1,\ldots,n_a-1\}, b \in \{0,1,\ldots,n_b-1\}\} \\ V_B &= \{(a,b,n_c-1) \in V' | \; a \in \{0,1,\ldots,n_a-1\}, b \in \{0,1,\ldots,n_b-1\}\} \end{aligned}$$

The very direct relationship between the vertex set $V' \setminus V_B$ highlighted before, allows us to define one-to-one correspondences between discretized terms as $\mathcal{MS}$, $\mathcal{D}^d$, $\mathcal{S}^d$ and $\mathcal{F}^d$ and terms defined on the base of V'. This will help us to understand better the graph topology and gives us the possibility to concentrate on working with the network without permanent switching to other representations. Additionally, it simplifies the notation for further work.
In a first step, we define a simple one-to-one correspondence between the set $V' \setminus V_B$ and the points in the matching space $\mathcal{MS}$ as already described, by the following function:

$$\begin{aligned} \eta: \; V' \setminus V_B &\longrightarrow \mathcal{MS} \\ (a,b,c) &\longmapsto (x_a, y_b, d_c)_{disp} \end{aligned}$$

In a second step the discretized domain $\mathcal{D}^d$ can trivially be associated to the *graph domain* $\mathcal{D}^G = \{0,1,\ldots,n_a-1\} \times \{0,1,\ldots,n_b-1\}$ by the following function:

$$\begin{aligned} \chi: \; \mathcal{D}^G &\longrightarrow \mathcal{D}^d \\ (a,b) &\longmapsto (x_a, y_b) \end{aligned}$$

For the representation of $\mathcal{F}^d$ by the network G, we define the set $\mathcal{F}^G$ of all *graph disparity mappings* to be the set of all functions f^G of the type

$$f^G : \mathcal{D}^G \longrightarrow \{0,1,\ldots,n_c-2\};$$

The function ψ below gives a one-to-one correspondence between $\mathcal{F}^d$ and $\mathcal{F}^G$.

$$\begin{gathered} \psi : \mathcal{F}^G \longrightarrow \mathcal{F}^d \\ \text{with} \quad \psi(f^G)(x_i, y_j) = d_{f^G(i,j)} \quad \forall\, (x_i, y_j) \in \mathcal{D}^d \end{gathered}$$

In a next step, we define the set of all *graph surfaces* $\mathcal{S}^G$ to be the set containing all functions S^G that can be written in the following way

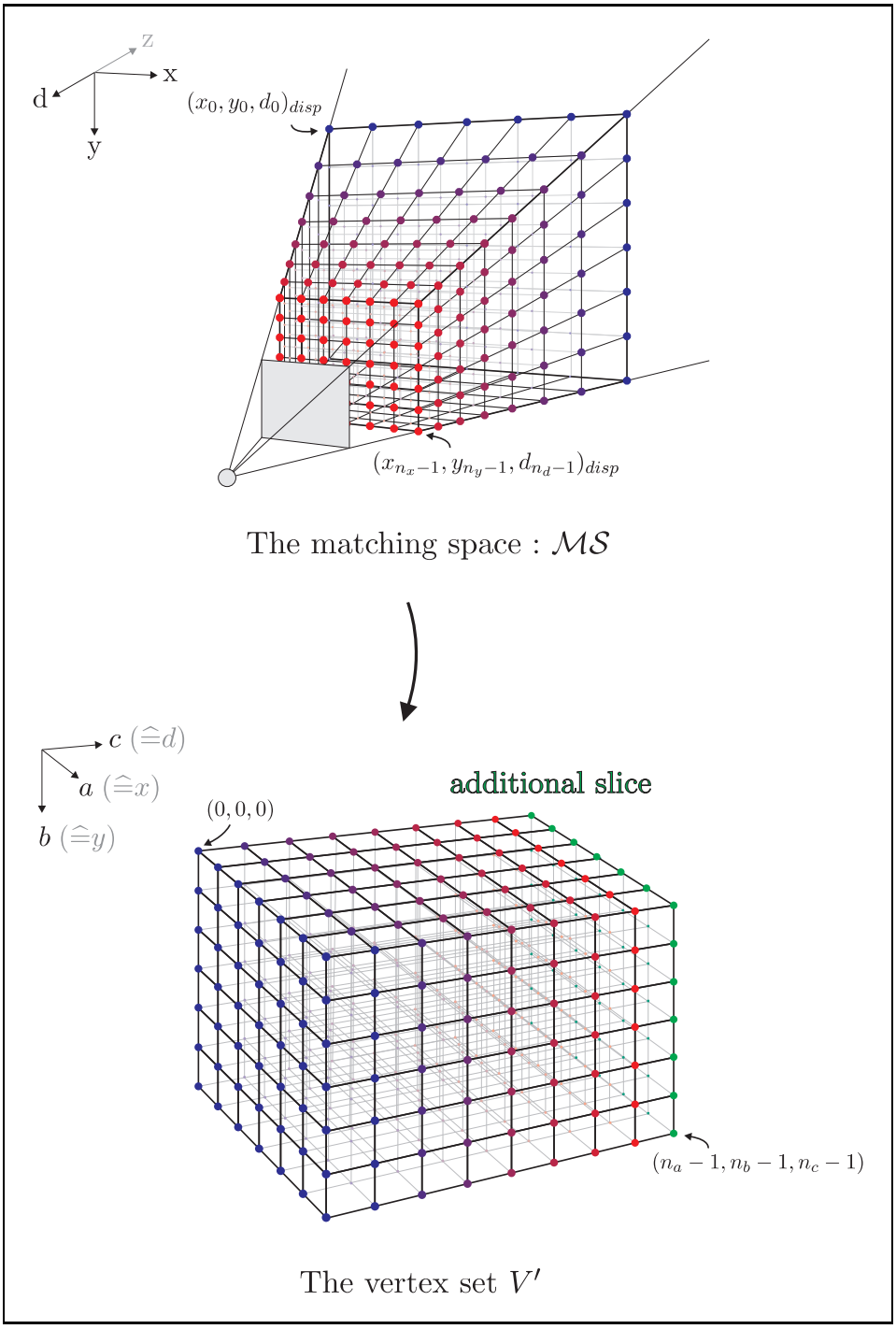

Figure 1.4: The vertex set V' as another respresentation of the matching space with an additional slice of vertices. Note that the orientation is changed in the new coordinate-system with the axes a, b and c.

$$\begin{aligned} S^G : \ \mathcal{D}^G &\longrightarrow V' \setminus V_B \\ (a,b) &\longmapsto (a,b,f_{S^G}(a,b))_{disp} \\ \text{with} \quad f_{S^G} &\in \mathcal{F}^G \end{aligned}$$

Analogous to our previous notations, we denote by $S^G_{f^G}$ the graph surface corresponding to the graph disparity mapping f^G.

We have also a bijection ϕ between $\mathcal{S}^G$ and $\mathcal{S}^d$ induced by the correspondence between $\mathcal{F}^G$ and $\mathcal{F}^d$.

$$\begin{aligned} \phi : \ \mathcal{S}^G &\longrightarrow \mathcal{S}^d \\ S^G_{f^G} &\longmapsto S^d_{\psi(f^G)} \end{aligned}$$

Thanks to this one-to-one mapping between valid discretized surfaces and graph surfaces, we can define for each energy function on $\mathcal{S}^d$, an energy function on $\mathcal{S}^G$. We simply write the superscript G on an energy function on $\mathcal{S}^d$ to denote the corresponding energy function on $\mathcal{S}^G$. So we have

$$\begin{aligned} \mathbf{EC}^G_{\kappa^G}(S^G) &= \sum_{a=0}^{n_a-1} \sum_{b=0}^{n_b-1} \kappa^G(a,b,f^G_{S^G}(a,b)) \\ \text{where} \quad \kappa^G : V' \setminus V_B &\longrightarrow \mathbb{R}_+ \\ v &\longmapsto \kappa(\eta(v)) \end{aligned} \tag{1.6}$$

$$\mathbf{ES}^G(S^G) = \sum_{\substack{q_1,q_2 \in \mathcal{D}^G, \\ \text{neighbors}}} K\Delta d |f^G_{S^G}(q_1) - f^G_{S^G}(q_2)| \tag{1.7}$$

and

$$\mathbf{E}^G_{\kappa^G} = \mathbf{EC}^G_{\kappa^G} + \mathbf{ES}^G \tag{1.8}$$

The optimization problem 1.5 can now be reformulated as

$$\underset{S^G \in \mathcal{S}^G}{argmin} \left(\mathbf{E}^G_{\kappa^G}(S^G)\right) = \underset{S^G \in \mathcal{S}^G}{argmin} \left(\mathbf{EC}^G_{\kappa^G}(S^G) + \mathbf{ES}^G(S^G)\right) \tag{1.9}$$

The function κ^G will be called *graph matching cost function.*

Terminals, arcs and capacities

Now we introduce the arcs on the vertices V' i.e. all the arcs in the set $E' = E \cap V' \times V'$. The vertices V' are linked with arcs parallel to a, b and c-axis to their neighbors in the way that we obtain a 3D-grid as illustrated in figure 1.5 a). As for every arc in E', his reversal arc[3] is also element of E', we represented an arc and his reversal by a simple line without arrows in order not to overcharge the figure.

Formally, we can describe E' in the following way:

$$E' = \{(v, w) \in V' \times V' \mid (w - v) \in \{(\pm 1, 0, 0), (0, \pm 1, 0), (0, 0, \pm 1)\}\}$$

We add the source s in front of the graph front V_F and we link s with arcs to all vertices of V_F. Similarly we add the sink vertex t behind the graph back V_B and we link all vertices of V_B with arcs to t. Figure 1.5 b) shows the graph obtained which is the network G without capacities. If we write E_{source} for the arcs adjacent to s and E_{sink} for the arcs adjacent to t, so we have

$$E = E' \cup E_{source} \cup E_{sink}$$

$$\begin{aligned} \text{where} \qquad E_{source} &= \{(s, v) \mid v \in V_F\} \\ E_{sink} &= \{(v, t) \mid v \in V_B\} \end{aligned}$$

We further partition the set E' in the following way:

$$E' = E_{consistency} \cup E_{backward} \cup E_{smoothness}$$

$$\begin{aligned} \text{where} \qquad E_{consistency} &= \{(v, w) \in E' \mid w_c = v_c + 1\} \\ E_{backward} &= \{(v, w) \in E' \mid w_c = v_c - 1\} \\ E_{smoothness} &= \{(v, w) \in E' \mid w_c = v_c\} \end{aligned}$$

For $q_1, q_2 \in \mathcal{D}^G$, we define

$$E_{smoothness}(q_1, q_2) = \{(v, w) \in E_{smoothness} \mid (v_a, v_b) = q_1, (w_a, w_b) = q_2\}$$

Clearly $E_{smoothness}(q_1, q_2) = \emptyset$, if q_1 and q_2 are not neighbors in $\mathcal{D}^G$. In a similar way, we define for $q \in \mathcal{D}^G$

$$\begin{aligned} E_{consistency}(q) &= \{(v, w) \in E_{consistency} \mid (v_a, v_b) = q\} \\ E_{backward}(q) &= \{(v, w) \in E_{backward} \mid (v_a, v_b) = q\} \end{aligned}$$

Finally we introduce the capacity function k. For $(v, w) \in E$ we define

[3] The reversal of an arc $e = (v, w)$ is defined as $e^R = (w, v)$.

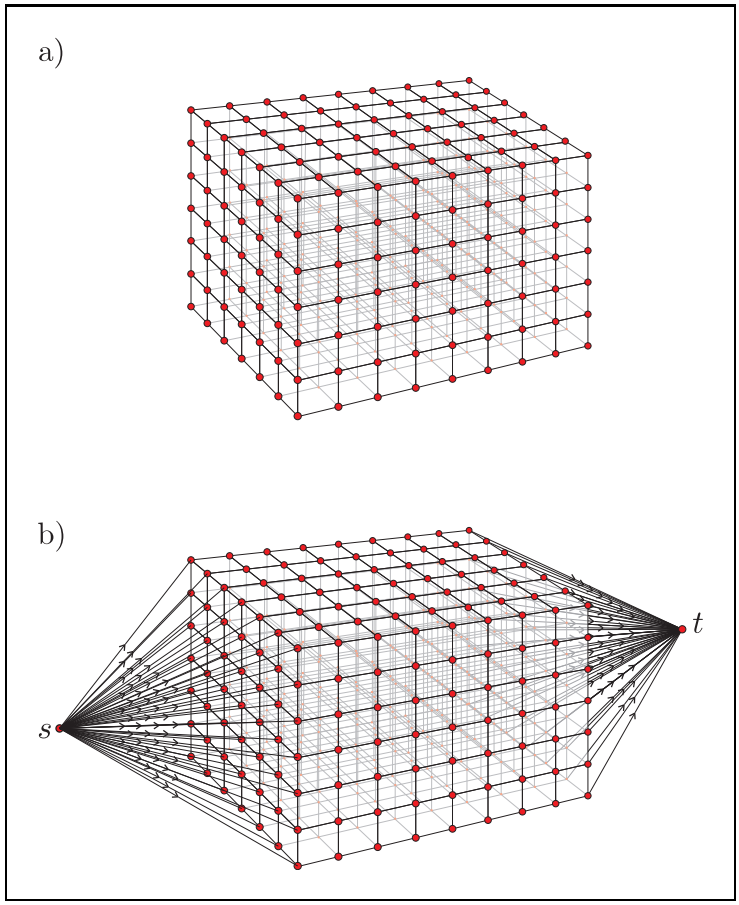

Figure 1.5: **a)** The graph (V', E') which is the network G without capacities and terminals. **b)** The graph (V, E) which is the network G without capacities.

$$k(v,w) = \begin{cases} \infty & \text{if } (v,w) \in E_{source} \cup E_{sink} \cup E_{backward} \\ \kappa^G(\eta(v)) & \text{if } (v,w) \in E_{consistency} \\ K^G = K\Delta d & \text{if } (v,w) \in E_{smoothness} \end{cases}$$

K^G will be called *graph smoothness factor*.

Relationship between valid discretized surfaces and $s-t$ cuts in G

Each graph surface $S^G \in \mathcal{S}^G$ can be associated to a partition of V into two subsets, one before the graph surface S^G, that we denote by A_{S^G} and one behind S^G, that we denote by $\overline{A_{S^G}} = V \setminus A_{S^G}$ i.e.

$$A_{S^G} = \{s\} \cup \{v \in V' \mid v_c \leq f^G_{S^G}(v_a, v_b)\}$$
$$\overline{A_{S^G}} = \{t\} \cup \{v \in V' \mid v_c > f^G_{S^G}(v_a, v_b)\}$$

So we can associate to a graph surface S^G the following $s-t$ cut [(4)]:

$$C_{S^G} = [A_{S^G}, \overline{A_{S^G}}]$$

It is important to note that conversely it is not always possible to associate to each $s-t$ cut $C = [V_s, V_t]$ a graph surface $S^G \in \mathcal{S}^G$ such that $C = C_{S^G}$. The problem is, that cuts $C = [V_s, V_t]$ that can be associated to a graph surface S^G (i.e. $C = C_{S^G}$) always satisfy that for each $q = (a, b) \in \mathcal{D}^G$ there is exactly one arc in $\omega^+(V_s) \cap E_{consistency}(q)$, namely the arc $((a, b, f^G_{S^G}(a, b)), (a, b, f^G_{S^G}(a, b) + 1))$.
Such an $s-t$ cut $C = [V_s, V_t]$ that corresponds to graph surface is called *valid $s-t$ cut* and can be characterized as follows.

Property 1.4.7 (Valid $s-t$ cut)**.**

$$\forall\ (a, b) \in \mathcal{D}^G \quad \exists\ c_{(a,b)} \in \{0, 1, \ldots, n_c - 2\} \text{ such that}$$
$$\{(a, b, 0), (a, b, 1), \ldots, (a, b, c_{(a,b)})\} \subset V_s \text{ and}$$
$$\{(a, b, c_{(a,b)} + 1), (a, b, c_{(a,b)} + 2), \ldots, (a, b, n_c - 1)\} \subset V_t$$

Where $c_{(a,b)}$ corresponds to $f^G_{S^G}(a, b)$. Conversely, it is easy to see that for every $s-t$ cut C satisfying 1.4.7, the graph surface $S^G \in \mathcal{S}^G$ defined by $f^G_{S^G}(a, b) = c_{(a,b)} \quad \forall\ (a, b) \in \mathcal{D}^G$ satisfies $C = C_{S^G}$. This shows that property 1.4.7 is effectively a characterization of valid $s-t$ cuts. We denote the graph surface associated to a valid $s-t$ cut C by S^G_C.
This establishes a one-to-one correspondence between valid $s-t$ cuts on G and graph surfaces.

Definition 1.4.8 (Value of an $s-t$ cut)**.** *The value of an $s-t$ cut $C = [V_s, V_t]$ on the network $G = (V, E, k)$ is defined by*

$$k(C) = k(\omega^+(V_s))$$

The following property gives a first justification of the construction of the network G and is a key ingredient for the passage from problem 1.9 to a minimum $s-t$ cut problem.

Property 1.4.9.

$$\mathbf{EC}^G_{\kappa^G}(S^G) + \mathbf{ES}^G(S^G) = k(C_{S^G}) \quad \forall\ S^G \in \mathcal{S}^G \tag{1}$$

more precisely we have:

$$\mathbf{EC}^G_{\kappa^G}(S^G) = k(\omega^+(A_{S^G}) \cap E_{consistency}) \tag{2}$$
$$\mathbf{ES}^G(S^G) = k(\omega^+(A_{S^G}) \cap E_{smoothness}) \tag{3}$$

[(4)] An $s-t$ cut C of G is a partition of the vertices V into two subsets V_s and V_t, such that $s \in V_s$ and $t \in V_t$ and we write $C = [V_s, V_t]$.

Proof. By the definition of a graph surface, we have that $\omega^+(A_{SG}) \subset E_{consistency} \cup E_{smoothness}$. So equation 1 of property 1.4.9 is a consequence of the other ones and it suffices to prove the two last equations.
We begin by the proof of equation 2. We fix $q = (a, b) \in \mathcal{D}^G$. Because $C_{SG} = [A_{SG}, \overline{A_{SG}}]$ is a valid $s - t$ cut, we know that the only arc in $\omega^+(A_{SG}) \cap E_{consistency}(q)$ is $e_{(a,b)} = ((a, b, f^G_{SG}(a, b)), (a, b, f^G_{SG}(a, b) + 1))$, and we have $k(e_{(a,b)}) = \kappa^G\left((a, b, f^G_{SG}(a, b)\right)$. By summing these capacities over $\mathcal{D}^G$, we establish the equality under consideration.
For equation 3, we will show that the part of the smoothness energy term $\mathbf{ES}^G$ associated to two neighboring points $q_1, q_2 \in \mathcal{D}^G$ (i.e. the term of the sum in 1.7 corresponding to q_1, q_2), corresponds to the capacities of the arcs $(v, w) \in \omega^+(A_{SG}) \cap (E_{smoothness}(q_1, q_2) \cup E_{smoothness}(q_2, q_1))$. We suppose without loss of generality that $f^G_{SG}(q_1) \geq f^G_{SG}(q_2)$. Therefore, the part of the term $\mathbf{ES}^G$ associated to the two neighboring points q_1, q_2 is $K\Delta d(f^G_{SG}(q_1) - f^G_{SG}(q_2))$. Additionally we have that $\omega^+(A_{SG}) \cap E_{smoothness}(q_2, q_1) = \emptyset$ and

$$\begin{aligned}\omega^+(A_{SG}) \cap E_{smoothness}(q_1, q_2) = \{&((q_1, f^G_{SG}(q_1)), (q_2, f^G_{SG}(q_1))),\\ &((q_1, f^G_{SG}(q_1) - 1), (q_2, f^G_{SG}(q_1)) - 1),\\ &\dots,\\ &((q_1, f^G_{SG}(q_2) + 1), (q_2, f^G_{SG}(q_2) + 1))\}\end{aligned}$$

Figure 1.6 illustrates the set $\omega^+(A_{SG}) \cap E_{smoothness}(q_1, q_2)$. For simplicity only a 2D slice of the network G that contains the two neighbors $q1$ and $q2$ is represented in this figure.

By the previous observations we have

$$|\omega^+(A_{SG}) \cap E_{smoothness}(q_1, q_2)| = f^G_{SG}(q_1) - f^G_{SG}(q_2)$$

and finally

$$k(\omega^+(A_{SG}) \cap E_{smoothness}(q_1, q_2)) = K\Delta d(f^G_{SG}(q_1) - f^G_{SG}(q_2))$$

what finishes the proof. □

Further we have the following two properties which allows us finally to solve problem 1.9 by the way of a minimum $s - t$ cut on G.

Property 1.4.10. *If $C = [V_s, V_t]$ is an $s - t$ cut on the network G, so we have*

$$C \text{ is valid } \Leftrightarrow k(C) < \infty$$

Proof.
$\Rightarrow$) By property 1.4.7 it is easy to see that if $C = [V_s, V_t]$ is a valid $s - t$ cut, so

$$\omega^+(V_s) \cap \{E_{source} \cup E_{sink} \cup E_{backward}\} = \emptyset$$

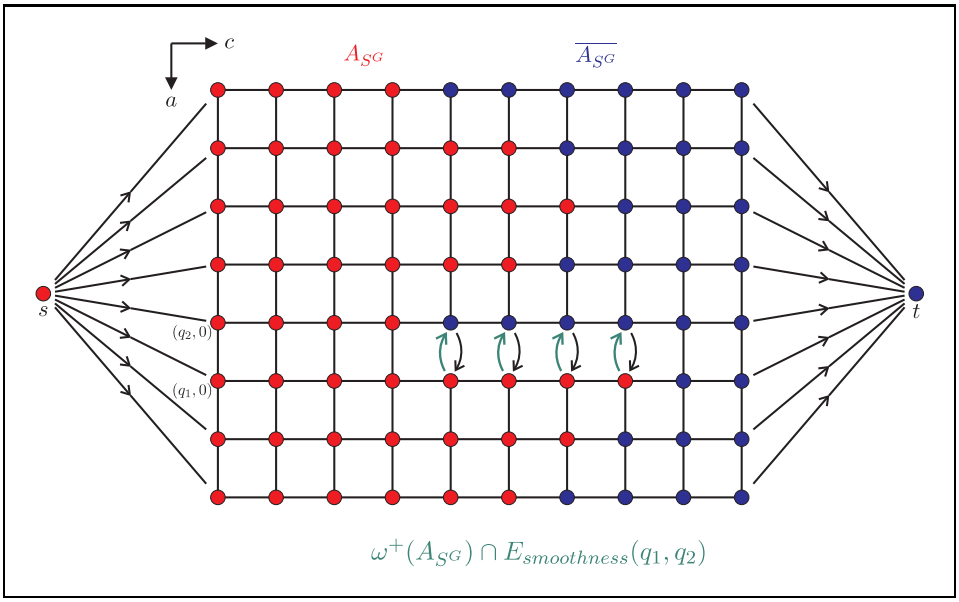

Figure 1.6: Illustration of the set $\omega^+(A_{SG}) \cap E_{smoothness}(q_1, q_2)$. For simplicity, we only represent a 2D slice of the network G containing the two neighbors $q_1, q_2 \in \mathcal{D}^G$. A line without arrows means that we have arcs in both directions (but for reasons of illustration, we sometimes draw explicitly both arrows).

Therefore $\omega^+(V_s)$ contains no arc of infinite capacity and we have $k(C) < \infty$.

$\Leftarrow$) We will prove this implication by contraposition. If C is not valid, so C does not satisfy property 1.4.7, i.e. $\exists\ (a, b) \in \mathcal{D}^G$ such that (a, b) does not satisfy the conditions in the mentioned property. There are only three possibilities for (a, b) to violate this conditions.

case 1: $\{(a, b, 0), (a, b, 1), \ldots, (a, b, n_c - 1)\} \subset V_s$
But in this case the arc $e = ((a, b, n_c - 1), t) \in \omega^+(V_s)$ and $k(e) = \infty$

case 2: $\{(a, b, 0), (a, b, 1), \ldots, (a, b, n_c - 1)\} \subset V_t$
In this case the arc $e = (s, (a, b, 0)) \in \omega^+(V_s)$ and $k(e) = \infty$

case 3: $\exists\ c \in \{1, 2, \ldots, n_c - 1\}$ with $(a, b, c) \in V_s$ and $(a, b, c - 1) \in V_t$
Here $e = ((a, b, c), (a, b, c - 1)) \in \omega^+(V_s)$ and $k(e) = \infty$

In all cases we have $k(C) = \infty$. □

Property 1.4.11. *If $C = [V_s, V_t]$ is a minimum $s - t$ cut on G, so C is valid.*

Proof. Because the cut $C' = [\{s\} \cup V_F, \overline{\{s\} \cup V_F}]$ satisfies

$$k(C') = \sum_{q \in \mathcal{D}^G} k((q, 0), (q, 1)) < \infty$$

and C is minimum, we have

$$k(C) \leq k(C') < \infty$$

By property 1.4.10 we have that C is valid. □

Finally we can state the following theorem that reduces problem 1.9 on a minimum $s-t$ cut problem on G.

Theorem 1.4.12.

$$C \text{ is a minimum } s-t \text{ cut on } G \Rightarrow S_C^G \in \underset{S^G \in \mathcal{S}^G}{argmin} \left(\mathbf{E}_{\kappa^G}^G(S^G)\right)$$

Conversely, we have

$$S^G \in \underset{S^G \in \mathcal{S}^G}{argmin} \left(\mathbf{E}_{\kappa^G}^G(S^G)\right) \Rightarrow C_{S^G} \text{ is a minimum } s-t \text{ cut on } G$$

Proof. Because of property 1.4.9 and the one-to-one correspondence between valid $s-t$ cuts and graph surfaces, we know that the minimization

$$\underset{S^G \in \mathcal{S}^G}{argmin} \left(\mathbf{E}_{\kappa^G}^G(S^G)\right)$$

is equivalent to the minimization of

$$\underset{C \text{ valid } s-t \text{ cut}}{argmin} \quad (k(C)) \tag{$*$}$$

where we associate to a graph surface S^G the valid $s-t$ cut C_{S^G} and inversely we associate to a valid $s-t$ cut C the graph surface S_C^G. By property 1.4.11, we know that a minimum $s-t$ cut is valid and therefore solution of $(*)$, which finishes the proof. □

Thus we have reduced the problem 1.9 to the problem of finding a minimum $s-t$ cut in the network G that can be described as follows

$$\underset{C,\ s-t \text{ cut}}{argmin} \ (k(C)) \tag{1.10}$$

1.4.4 The general form of the network G we consider

The network G described in the previous section is determined up to the following terms

- The dimensions $n_a = n_x, n_b = n_y, n_c = n_d + 1 \in \mathbb{N}$
- The graph smoothness factor K^G
- The graph matching cost function κ^G

For the dimensions we impose naturally

$$\begin{aligned} n_a, n_b \geq 1 \qquad & \text{because } n_x \text{ and } n_y \text{ have to be} \geq 1 \\ n_c \geq 2 \qquad & \text{because } n_d \text{ has to be} \geq 1 \end{aligned}$$

The remaining terms to discuss are K^G and κ^G which are the ones, that determine the capacity k. For numerous technical reasons, it is very comfortable to work with integer capacities. On the one hand many efficient algorithms for the resolution of the minimum $s-t$ cut problem need integer capacities and on the other hand it is a big advantage for the implementation. We do not have to use floating point algebra, which is in general expensive in memory and in execution time. Beside this, it is not a serious restriction because if we have capacities in $\mathbb{R}_+$, so we can multiply them by an arbitrary big constant $\alpha \in \mathbb{R}_+$ and the resulting network G' has the same nature as G what concerns the values of the $s-t$ cuts, because they were all multiplied by α. Now we can round the capacities in $G' = (V, E, k')$ to the nearest number in $\mathbb{N}$ to obtain a network $G'' = (V, E, k'')$ with integer capacities. One can easily show that for every $\epsilon > 0$, we can rapidly find a constant α, such that the real value $k(C)$ of a minimum $s-t$ cut respective to G'' is less than $(1+\epsilon)$ times the value of the minimum $s-t$ cut in G. It is even possible to find a constant α such that a minimum $s-t$ cut in G'' is also a minimum $s-t$ cut in G, but this is more difficult.
Anyway, in reality the integrality assumption is not restrictive because all modern computers store capacities as rational numbers and we can always transform them to integer numbers by multiplying them by a suitable large constant.

So this integrality assumption induces the following restriction on K^G and κ^G

$$\begin{aligned} & K^G \in \mathbb{N} \\ & \kappa^G \text{ has to be a function of type: } \kappa^G : V' \setminus V_B \longrightarrow \mathbb{N} \end{aligned}$$

This are all assumptions we make on the network G. So we do not force the graph matching cost function κ^G to have a certain structure. Thanks to this, our further discussion will be valid for a wide variety of graph matching cost functions.

Recapitulating, a general network G in this document that corresponds to a stereo problem, has constant positive integer capacities on smoothness arcs and arbitrary positive integer capacities on consistency arcs. For further analysis it is interesting to note that the graph smoothness factor is usually relatively small, to say ≤ 10.

2 Preliminaries concerning flows and $s-t$ cuts

Currently, the most efficient algorithms for solving an $s-t$ cut problem first solve its dual problem which is the maximum flow problem (from s to t) [1]. This chapter introduces basic definitions, notations and properties concerning flow and $s-t$ cut problems and shows important relationships between these two problems.

2.1 Flows

Let $G=(V,E,k)$ be a directed network without loops or multiple arcs [2] and with an integer capacity function k. Further, we assume that the set of edges E is completed by the reversal edges, i.e. if $(v,w)\in E$ so also $(w,v)\in E$. This assumption is not restrictive because we can for every arc $(v,w)\in E$ with $(w,v)\notin E$, add the reversal arc (w,v) with zero capacity to the set E. This completion of the set E evidently does not change the nature of the network with respect to cuts. As usual, we have two terminals $s,t\in V$ (with $s\neq t$), where s is called *source* and t *sink*.

The maximum flow problem can be stated in the following way: In a network $G=(V,E,k)$, we wish to send as much flow as possible from one special node s to another special node t, without exceeding the capacity $k(e)$ of any arc $e\in E$. More formally, we define such a flow φ in G in the following way.

Definition 2.1.1 (Flow)**.** *A flow φ from s to t in a network $G=(V,E,k)$ is a function*

$$\varphi : E \longrightarrow \mathbb{R} \quad \textit{that satisfies}$$

i)	*Skew symmetry:*	$\varphi(v,w)=-\varphi(w,v)$	$\forall\,(v,w)\in E$
ii)	*Capacity constraint:*	$\varphi(v,w)\leq k(v,w)$	$\forall\,(v,w)\in E$
iii)	*Flow conservation:*	$\sum_{e\in\omega^+(v)}\varphi(e)=0$	$\forall\, v\in V\setminus\{s,t\}$

When there is no danger of ambiguity we often call φ simply «flow in G».

[1] For further information between the primal-dual relation of flow and $s-t$ cut problems see [16]. In this work, we have not adopted a linear programming perspective and it should be possible to understand this document without preliminary knowledge in dualization of linear programs.

[2] Anyway, loops would make no difference in the value of any cut and multiple arcs can be replaced by a single arc whose capacity is the sum of the capacities of the corresponding multiple arcs. By removing the loops and handling the multiple arcs as described, the obtained network has the same nature with respect to cuts as the original network.

We will interpret the function φ in the following way. For $e \in E$, if $\varphi(e) \geq 0$ so we say that a flow of value $\varphi(e)$ goes through the arc e. And if $\varphi(e) < 0$, we say that no flow is passing through e. So f trivially satisfies the following property.

Property 2.1.2. *For every arc $e \in E$, either no flow passes through e or no flow passes through e^R.*

We further make the following definitions.

Definition 2.1.3. *For $v \in V$, we define the flow entering in v as*

$$\varphi^-(v) = \sum_{e \in \omega^-(v)} \max\{\varphi(e), 0\}$$

In an analogue manner we define the flow exiting from v as

$$\varphi^+(v) = \sum_{e \in \omega^+(v)} \max\{\varphi(e), 0\}$$

With this definition the flow conservation property *iii*) can be written as

$$iii) \quad \varphi^-(v) = \varphi^+(v) \quad \forall\, v \in V \setminus \{s, t\}$$

With this new formulation of property *iii*), it is easy to understand why we call property *iii*) the ≪flow conservation property≫, because the flow entering in v has to be equal to the flow exiting from v.

In network flow literature (for example [9]) it is also common not to impose skew symmetry on a flow (note that in this case, the flow function only returns positive values and the flow conservation property has to be reformulated). The flow formulation without skew symmetry is somewhat more intuitive because we do not have to give a special interpretation to the case $\varphi(e) < 0$ because flows are always positive in this formulation, but here we would have the disadvantage that it is possible to have a strictly positive flow on an arc $e \in E$ and on the same time to have a strictly positive flow on its reversal arc e^R. In this case, it is always possible to reduce the flows $\varphi(e)$ and $\varphi(e^R)$ by $\min\{\varphi(e), \varphi(e^R)\}$ to get a new flow with a value of 0 on e or e^R. So by this technique every flow can be transformed into a flow satisfying property 2.1.2. This justifies that the skew symmetry (which forces the flow to satisfy this property) is not restrictive.
We have chosen the formulation with the skew symmetry because it is often much easier to handle flows satisfying property 2.1.2.

Definition 2.1.4 (Saturation of an arc by a flow)**.** *We say that a flow φ in G saturates $e \in E$ if*

$$\varphi(e) = k(e)$$

Notation 2.1.5. *Let $V_1, V_2 \subset V$ be two subsets of V. So we define the flow from V_1 to V_2 to be the following term:*

$$\varphi^+(V_1, V_2) := \sum_{e \in \omega^+(V_1, V_2)} \varphi^+(e) \quad (3)$$

Additionally, we use the notation $\varphi^-(V_1, V_2) = \varphi^+(V_2, V_1)$.

Notation 2.1.6. *Let $V_1, V_2 \subset V$ be two subsets of V. So we define the flow between V_1 and V_2 to be the following term:*

$$\varphi(V_1, V_2) := \sum_{e \in \omega^+(V_1, V_2)} \varphi(e) = \varphi^+(V_1, V_2) - \varphi^-(V_1, V_2)$$

Note that the flow between V_1 and V_2 is in general not the same value as the flow between V_2 and V_1. More precisely, we have $\varphi(V_1, V_2) = -\varphi(V_2, V_1)$. With this notation, the point *iii*) of the definition of a flow can be written as

$$iii) \quad \varphi(V \setminus \{v\}, v) = 0 \qquad \forall\, v \in V \setminus \{s, t\}$$

We now introduce the *value of a flow φ in G* which is the amount of flow that is send by φ from s to t.

Definition 2.1.7 (Value of an $s-t$ flow φ in G)**.** *The value $k(\varphi)$ of an $s-t$ flow φ in $G = (V, E, k)$ is defined as*

$$k(\varphi) = \varphi(V \setminus \{t\}, t)$$

and corresponds to the amount of flow transported (by φ) from s to t.

It is easy to show by the flow conservation property that we have

$$k(\varphi) = \varphi(V_1, \overline{V_1}) \qquad \forall\, V_1 \subset V \setminus \{t\} \text{ with } s \in V_1 \tag{2.1}$$

This notion allows us finally to formulate the *maximum flow problem*. This problem is about finding, in the set of all flows from s to t in G, a flow of maximum value. To ensure that a maximum flow from s to t on a network G exists, we have to assume that G contains no path from s to t consisting solely of arcs of infinity capacity.

In a next step, we will introduce the *residual network of G with respect to the flow φ* which is an ancillary network that allows to handle easily the possibilities of changing the flow φ.

Definition 2.1.8 (Residual network of G with respect to φ)**.** *Let φ be a flow in G. The residual network $G_\varphi = (V, E, k_\varphi)$ of $G = (V, E, k)$ with respect to φ has the same vertices and edges as the original network G but different capacities. The capacity function of the residual network is called «residual capacity» and is defined in the following way:*

$$k_\varphi(e) = k(e) - \varphi(e) \qquad \forall\, e \in E$$

(3) Where $\omega^+(V_1, V_2) = \omega^+(V_1) \cap \omega^-(V_2)$. In an analogue manner we define $\omega^-(V_1, V_2) = \omega^-(V_1) \cap \omega^+(V_2) = \omega^+(V_2, V_1)$ and $\omega(V_1, V_2) = \omega(V_1) \cap \omega(V_2) = \omega^+(V_1, V_2) \cup \omega^-(V_1, V_2)$.

Any flow φ_r in the residual network G_φ corresponds to a possible change of the flow φ, that transforms φ into a flow φ' on G in the following way.

$$\varphi'(v,w) = \varphi(v,w) + \varphi_r(v,w) \quad \forall\ (v,w) \in E$$

and we write

$$\varphi' = \varphi \oplus \varphi_r$$

One can easily verify that this way of transforming flows from G_φ into G gives in fact a one-to-one correspondence between flows in G and in G_φ. Further we have the following property.

Property 2.1.9.

$$\varphi'(V_1,V_2) = \varphi(V_1,V_2) + \varphi_r(V_1,V_2) \quad \forall\ V_1,V_2 \subset V$$

In particular the value of the flow φ' is equal to the value of φ plus the value of φ_r (i.e. $k(\varphi') = k(\varphi) + k(\varphi_r)$).

So we can increase the value of an existing flow φ on G by finding a flow φ_r in the residual network G_φ with positive value and by transforming this flow into the flow φ'. Such a flow φ_r is called an *augmenting flow*. A simple way of applying this technique is by finding a path of non-saturated arcs in the residual network and by augmenting the flow along this path. Such a path is called an *augmenting path* and the value of the flow which can be transported through this path is called the *capacity of the augmenting path*. The capacity of an augmenting path corresponds therefore to the minimal capacity of the arcs on the path.
By the relationship between flows in the residual network and flows in the original network one can easily verify that we have the following theorem.

Theorem 2.1.10. *Let φ be a flow in G. So*

$$\varphi \text{ is a maximum flow} \Leftrightarrow \nexists \text{ augmenting path in } G_\varphi$$

The following property shows a very direct relationship between the original and residual network concerning $s-t$ cuts:

Property 2.1.11. *Let φ be a flow in the network G and $C = [V_s, V_t]$ an $s-t$ cut in G. So we have*

$$k_\varphi(C) = k(C) - k(\varphi)$$

Proof. This property is a consequence of the definition of the residual network and 2.1.

$$\begin{aligned} k_\varphi(C) &= \sum_{e \in \omega^+(V_s)} k_\varphi(e) \\ &= \sum_{e \in \omega^+(V_s)} (k(e) - \varphi(e)) \\ &= \sum_{e \in \omega^+(V_s)} k(e) - \sum_{e \in \omega^+(V_s)} \varphi(e) \\ &= k(C) - k(\varphi) \end{aligned}$$

□

The previous property thus says that the values of $s-t$ cuts in G_φ differ by a constant from their values in G. Therefore, minimum $s-t$ cuts in G_φ are also minimum in G.

2.2 $s-t$ cuts

In this section we will introduce some notations and properties of $s-t$ cuts in a network $G=(V,E,k)$.

We begin by defining a partial order $\ll\preceq\gg$ on $s-t$ cuts.

Definition 2.2.1 (The partial order $\ll\preceq\gg$ on $s-t$ cuts)**.** *Let $C^1=[V_s^1,V_t^1]$ and $C^2=[V_s^2,V_t^2]$ be two $s-t$ cuts in G. If $V_s^2\subset V_s^1$, we say that C^1 is greater than C^2 (resp. C^2 is smaller than C^1) and we write $C^2\preceq C^1$.*

This way of comparing $s-t$ cuts, leads to a natural definition for unions and intersections of $s-t$ cuts.

Definition 2.2.2 (Union and intersection of $s-t$ cuts)**.** *Let $C^1=[V_s^1,V_t^1]$ and $C^2=[V_s^2,V_t^2]$ be two $s-t$ cuts in G. We define the union and intersection of two $s-t$ cuts in the following way*

$$C^1\cup C^2=[V_s^1\cup V_s^2,V_t^1\cap V_t^2]$$
$$C^1\cap C^2=[V_s^1\cap V_s^2,V_t^1\cup V_t^2]$$

One can easily verify that we have the following property.

Property 2.2.3. *Let $\widetilde{C}=[\widetilde{V}_s,\widetilde{V}_t]$ be a minimum $s-t$ cut in G and $C=[V_s,V_t]$ be any $s-t$ cut in G (not necessarily minimum). So we have*

$$k(\widetilde{C}\cap C)\leq k(C)\quad \textit{and}$$
$$k(\widetilde{C}\cup C)\leq k(C)$$

As a direct consequence of the previous property we have.

Property 2.2.4. *Let $\widetilde{C^1}=[\widetilde{V_s^1},\widetilde{V_t^1}]$ and $\widetilde{C^2}=[\widetilde{V_s^2},\widetilde{V_t^2}]$ be two minimum $s-t$ cuts in G. So $\widetilde{C^1}\cup\widetilde{C^2}$ and $\widetilde{C^1}\cap\widetilde{C^2}$ are also minimum $s-t$ cuts.*

This property implies that in the set of all solutions of a minimum $s-t$ cut problem there is always a maximal and a minimal solution with respect to the partial order $\preceq$. The maximal solution can be seen as the union of all minimum $s-t$ cuts and the minimal solution as their intersection. Usually we try to solve the problem 1.10 by finding the maximal minimum $s-t$ cut. We will often refer to this cut as the *minimum $s-t$ cut with the highest disparities* or *nearest minimum $s-t$ cut* because it corresponds to the $s-t$ cut with the lowest deepness.

2.3 Relationships between flows from s to t and $s-t$ cuts

This section gives a short summary of the main results concerning the relationships between flows from s to t and $s-t$ cuts.

Property 2.3.1. *Let $C=[V_s,V_t]$ be an $s-t$ cut in G and φ a flow in G. So we have*

$$k(\varphi)\leq k(C)$$

Proof. This property is an immediate consequence of the dual relation between the minimum $s-t$ cut problem and the maximum flow problem, but it can also be verified in a direct way.

$$k(\varphi)=\varphi(V_s,V_t)\leq k(V_s,V_t)=k(C)$$

□

Theorem 2.3.2. *Let $\widetilde{C}=[\widetilde{V}_s,\widetilde{V}_t]$ be a minimum $s-t$ cut in G and $\widetilde{\varphi}$ a maximum flow in G. So we have*

$$k(\widetilde{\varphi})=k(\widetilde{C})$$

This theorem shows the relation of strong duality between the minimum $s-t$ cut and the maximum flow problem. It can also be verified by a more direct way without the use of linear programming. The interested reader finds a proof in [9].

Property 2.3.3.

a) Let $\widetilde{\varphi}$ be a maximum flow from s to t in G and $\widetilde{C}=[\widetilde{V}_s,\widetilde{V}_t]$ a minimum $s-t$ cut in G. So $\widetilde{\varphi}$ saturates every arc in $\omega^+(\widetilde{V}_s,\widetilde{V}_t)$ and as a consequence of the skew symmetry of a flow, we have that no flow is passing on the arcs $\omega^-(\widetilde{V}_s,\widetilde{V}_t)$. In this situation we say that $\widetilde{\varphi}$ saturates the $s-t$ cut $\widetilde{C}$.

b) Conversely, if C is an $s-t$ cut that is saturated by a flow φ, so C is a minimum $s-t$ cut and φ is a maximum flow in G.

By combining the previous results we have that a flow φ from s to t is maximum if and only if it saturates an $s-t$ cut C. In this case, C is a minimum $s-t$ cut.

Passage from a maximum flow to a minimum $s-t$ cut

Finally we introduce a simple algorithm that allows, on the base of a maximum flow $\widetilde{\varphi}$ from s to t, finding the maximal (in the sense of the partial order $\ll\preceq\gg$) minimum $s-t$ cut $\widetilde{C}=[\widetilde{V}_s,\widetilde{V}_t]$.

This algorithm is a labelling algorithm in the residual graph $G_{\widetilde{\varphi}}$ which will successively label the vertices in $\widetilde{V}_t$. We begin by labelling the sink t and apply the following rule: if $v\in V$ is already labelled and there is an unlabelled vertex w, such that there exists

an edge (w, v) with strictly positive residual capacity, so we label w. This labelling rule can be applied for example in a breath-first-search or also in a depth-first-search way to obtain an algorithm of complexity $\mathcal{O}(|E|)$.

The verification that this algorithm effectively finds the maximal minimum $s-t$ cut is very direct. Firstly, the labelling algorithm will never label a vertex in $\widetilde{V}_s$ because thanks to property 2.3.3 we know that all edges in $\omega^+(\widetilde{V}_s, \widetilde{V}_t)$ are saturated and therefore have a residual capacity of zero. So if we denote by V_1 the vertices labelled at the end of the algorithm, then we have $V_1 \subset \widetilde{V}_t$. On the other hand we have that all edges in $\omega^-(V_1)$ have a residual capacity of zero (if not, it would have been possible to continue the labelling process). But that means that the $s-t$ cut constructed by the algorithm $C = [\overline{V_1}, V_1]$ is saturated by $\widetilde{\varphi}$ and thanks to the second part of property 2.3.3 we know that C is a minimum $s-t$ cut. Because $\widetilde{C}$ is maximal respective to the partial order $\ll\preceq\gg$, we have that $\widetilde{V}_t \subset V_1$. Finally we must have $V_1 = \widetilde{V}_t$ and therefore $C = \widetilde{C}$.

3 Algorithms for the resolution of the minimum $s - t$ cut problem

In this chapter we will give an overview of current maximum flow algorithms and analyze them for the particular case of reconstruction networks. Furthermore, we introduce two preprocessing algorithms we have developed for diminishing the worst-case complexities of various maximum flow algorithms on reconstruction networks and for accelerating them in practice.

3.1 Introduction

Currently, the most efficient algorithms for solving minimum $s - t$ cut problems solve first the dual problem of 1.10 which is the maximum flow problem (from s to t). In the previous chapter we saw how we can finally pass in $\mathcal{O}(|E|)$ time from a maximum flow to a minimum $s - t$ cut. The maximum flow problem can be solved in polynomial time, i.e. in a complexity of $\mathcal{O}(polynome(|V|, |E|))$, but we will also discuss some algorithms that are only pseudopolynomial (their worst-case complexity depends typically on capacities), because according to the problem we have to solve, it can be advantageous to use such an algorithm instead of a strongly polynomial one.

There exists numerous techniques for solving maximum flow problems. Nevertheless almost all flow algorithms up to date for general networks belong to one of the two following classes:

1. **Augmenting path algorithms.** These algorithms satisfy the flow conservation property at the end of every step. They incrementally increase the flow by augmenting paths (sometimes by several augmenting paths at ≪the same time≫).

2. **Preflow-push algorithms.** These algorithms do not satisfy the flow conservation property during their execution. They ≪flood≫ the network such that some vertices have flow excess. Then they try to send as much excess as possible to the sink. Excess that cannot reach the sink, will be sent back to the source to eliminate all excesses and thus finally satisfying the conservation property.

Before analyzing algorithms belonging to these two classes for reconstruction networks, we do some preliminary work concerning the complexity analysis in the next section.

3.2 Preliminaries concerning the complexity analysis

In this section, we will introduce some notations for complexity analysis in general networks as well as some special notations for reconstruction networks. Additionally, we will exploit some important properties of reconstruction networks that allow us to give better complexity bounds for some algorithms on these particular networks.

3.2.1 Basic notations

General networks

We introduce the following notations that will be useful for complexity analysis in general networks:

- $n = |V|$: number of vertices in the network
- $m = |E|$: number of edges in the network
- ζ : value of a maximum flow from s to t in the network

In network flow literature, it is often made the assumption that all capacities have to be finite. This is in general not restrictive because we can replace infinite capacities by a sufficiently large finite capacity without changing the minimum $s-t$ cuts. For example the value of an arbitrary finite $s-t$ cut [(1)]. So if necessary, we can always assume that the networks we analyze have finite capacities, and in this case we introduce the following notation:

- $U = \max\limits_{e \in E} k(e)$: highest capacity in the network

So if we work in a network with infinite capacities, then U is simply the maximal capacity of the corresponding network where we have eliminated the infinite capacities as described before.

Reconstruction networks

For reconstruction networks, we introduce the following notation:

- $U_{consistency} = \max\limits_{e \in E_{consistency}} k(e)$: the highest capacity in $E_{consistency}$

Additionally, it is important to note that a reconstruction network G is sparse i.e. $\mathcal{O}(m) = \mathcal{O}(n)$. This is a direct consequence of the fact that apart from the two terminals, every vertex has at most six neighbors.

[(1)] It is always possible to apply this procedure because we have assumed that the network on which we work, has no augmenting path of infinite capacity. It is easy to see that this is equivalent to the existence of an $s-t$ cut of finite value. Finding an $s-t$ cut of finite value can be done by a labelling algorithm in $\mathcal{O}(m)$.

3.3 Upper bounds of the maximum flow value and preprocessing in a reconstruction network

Several algorithms depend on the value of a maximum flow ζ, but this value is in most cases unknown and therefore normally replaced in the worst case complexity by a known upper bound of ζ easily calculable. By property 2.3.1, we can use the value of an arbitrary $s-t$ cut as upper bound of ζ.

In general networks ζ is typically bounded by the trivial $s-t$ cut $[\{s\}, V \setminus \{s\}]$ which has a value of at most nU (as in the worst case s has outgoing arcs of capacity U to every other vertex). But in reconstruction networks we can do better. One simple method is to fix an $\alpha \in \{0, 1, \ldots, n_c - 2\}$ and to choose the $s-t$ cut $C = [V_s, V_t]$ with $V_s = \{s\} \cup \{(a, b, c) \in V' \mid c \leq \alpha\}$. This cut contains exactly $n_a n_b$ arcs which all belong to $E_{consistency}$ and we have therefore

$$k(\widetilde{C}) \leq k(C) \leq n_a n_b U_{consistency}$$

This bound is clearly better than nU, especially because n was replaced by $n_a n_b$. Unfortunately, the term $U_{consistency}$ may be very big (recall that we made no assumptions on the capacities of the consistency arcs $E_{consistency}$).

Therefore we introduce now a method which allows eliminating the dependency on $U_{consistency}$ of the upper bound for the maximum flow value. As every $s-t$ cut of finite value in a reconstruction network contains at least one arc of $E_{consistency}$, we have to perform changes in the network to obtain upper bounds for the maximum flow value that are independent of $U_{consistency}$. This can be done by preliminary augmentation of the flow through augmenting paths that are easy to find. Hereafter we apply a maximum flow algorithm on the resulting residual network (by property 2.1.11 we are sure to still find a minimum $s-t$ cut in G by this preliminary step). We call this technique of preliminary augmentation of the flow «*preprocessing*». The preprocessing algorithms we use, are fast algorithms, i.e., their complexity is inferior to the complexity of the flow algorithm applied afterwards and therefore they do not contribute to the worst case complexity of the combined algorithm.

We will now introduce two preprocessing algorithms.

3.3.1 Preprocessing algorithm 1

This algorithm sends as much flow as possible from s to t without using smoothness arcs. So flow is only sent straight forward to the sink. Algorithm 1 gives a pseudocode that describes how this preprocess algorithm constructs such a maximum flow (that we will call φ) on a reconstruction network $G = (V, E, k)$.

The first for-loop will be repeated $n_a n_b$ times, line 3 needs to compare $n_c - 1$ values, line 4 and 5 can be performed in constant time, the second for-loop will be repeated $n_c - 1$ times

Algorithm 1 First preprocessing algorithm on reconstruction networks

1: **procedure** PREPROCESS 1(G) ▷ where $G = (V, E, k)$ is a reconstruction network
2: **for** $(a, b) \in \mathcal{D}^G$ **do**
3: $\lambda = \min_{e \in E_{consistency}(a,b)} k(e)$
4: $\varphi(s, (a, b, 0)) = \lambda$
5: $\varphi((a, b, n_c - 1), t) = \lambda$
6: **for** $e \in E_{consistency}(a, b)$ **do**
7: $\varphi(e) = \lambda$
8: **end for**
9: **end for**
10: $G = G_\varphi$
11: **end procedure**

and finally line 7 can be executed in constant time. Therefore we have a complexity of $\mathcal{O}(n_a n_b) \cdot \mathcal{O}(n_c) = \mathcal{O}(n)$ for this preprocessing algorithm. So this preprocessing algorithm will not be a bottleneck operation when applying afterwards a maximum flow algorithm.

Bounding the value of the remaining flow

The main idea of this preprocessing algorithm is that after having performed it, we have that for every $(a, b) \in \mathcal{D}^G$ there is an arc $e_{(a,b)} = ((a, b, c_{(a,b)}), (a, b, c_{(a,b)} + 1)) \in E_{consistency}(a, b)$ with capacity 0. These arcs correspond to the arcs that were saturated by the augmenting operations of the preprocessing. We define now the following valid $s - t$ cut C:

$$\begin{aligned} & C = [V_s, V_t] \\ \text{where} \quad & V_s = \{s\} \cup \{(a, b, c) \in V' \mid c \leq c_{(a,b)}\} \end{aligned}$$

The only consistency arcs in the cut C are arcs of the type $e_{(a,b)}$ which have capacity 0. Therefore, the only arcs in C with non-zero capacities are smoothness arcs (all of them have capacity K^G) and it is easy to show that we have

$$k(C) \leq 2nK^G$$

Thus after applying the first preprocessing algorithm, we know that the maximum flow value ζ of the new graph G is bounded by $\mathcal{O}(nK^G)$ which is independent of $U_{consistency}$. Note that the graph smoothness factor K^G is normally a very little value (usually between 1 and 10).

3.3.2 Preprocessing algorithm 2

The second preprocessing algorithm we introduce now, applies an idea similar to the first one. Here we will not ignore the smoothness arcs completely but we will take at least half of them into consideration.

This algorithm simply sends as much flow as possible from s to t without using smoothness arcs parallel to the b-axis. So smoothing only happens in a-direction.

Parallels with classical stereo algorithms

The solution of this problem is very similar to solutions constructed with the classical stereo algorithms using epipolar lines. To say that we get already a very interesting solution but with probably a lot of discontinuities in the b-direction. Because of this problem, numerous methods were proposed to ≪correct≫ solutions found by epipolar methods such that they show less discontinuities. Unfortunately, it was rather difficult to smooth the solutions in a global manner. The flow formulation shows here a big advantage. After having applied the second preprocessing algorithm, we do not only get a solution of the reconstruction problem (without smoothing in b-direction) but also a residual network on which we can do further work. After the preprocessing, we will simply continue with a maximum flow algorithm on the graph (this time with all smoothness arcs) that will smooth the solution and finally achieve the global optimum of the original flow problem.

Efficient implementation of the algorithm

The point of this preprocessing algorithm is that we can determine rapidly a maximum flow of the subproblem without smoothness arcs in b-direction. We will now discuss how to do this. Figure 3.1 a) shows the graph that corresponds to the mentioned subproblem. Each horizontal slice of this problem gives a new maximum flow problem independent of the others. In this way we get n_b maximum flow problems of the type as illustrated in figure 3.1 b). We will index these problems by their b-values. For a fixed $b \in \{0, 1, \ldots, n_b - 1\}$ we will denote the graph corresponding to the problem on level b by $\widehat{G}_b$.

These problems have the particularity that the corresponding graph is $s - t$ planar, i.e. the graph is planar and it is possible to give a planar representation where the source s and the sink t lie both on the boundary of the unbounded face (figure 3.1 b) gives such a representation for our problem) [(2)].

Let $\widehat{G} = (\widehat{V}, \widehat{E}, \widehat{k})$ be such an $s-t$ planar network as illustrated in figure 3.1 b). We will now see how we can solve this maximum flow problem by a shortest path technique on a directed dual-like graph [(3)] of $\widehat{G}$ obtaining a complexity of $\mathcal{O}(n \log(n))$. The method we introduce, uses the same idea as the algorithm explained in [9] for undirected $s - t$ planar graphs.

[(2)] For more information about planarity of graphs, see [15].

[(3)] For more information on dual graphs, see [15].

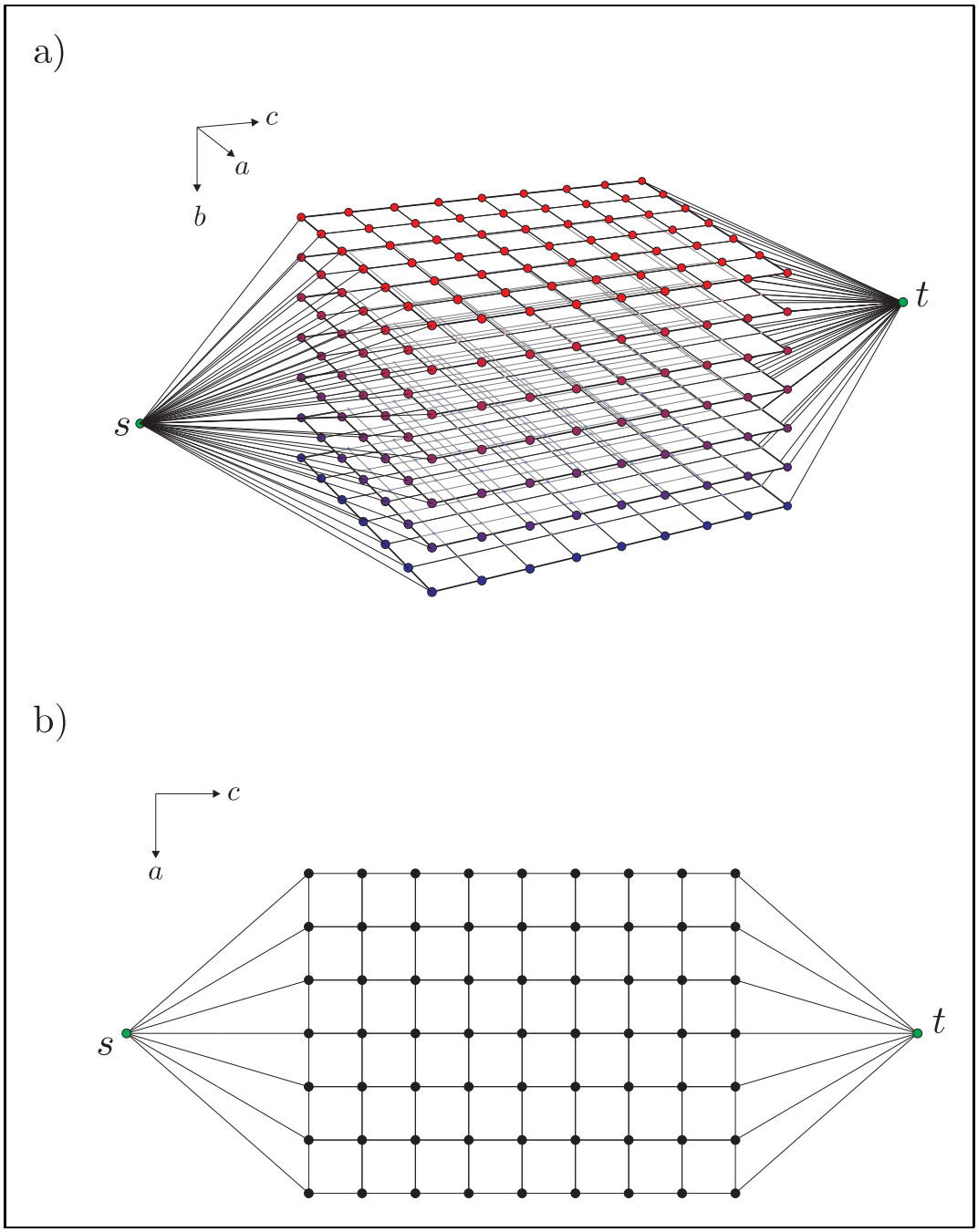

Figure 3.1: Image *a*) shows the graph of the subproblem that corresponds to the second preprocessing algorithm. This problem can be easily divided into n_b problems of type *b*).

We begin by constructing a network $G^* = (V^*, E^*, l^*)$ (where $l^* : E^* \longrightarrow \mathbb{R}^+$ is a length function) as illustrated in figure 3.2.

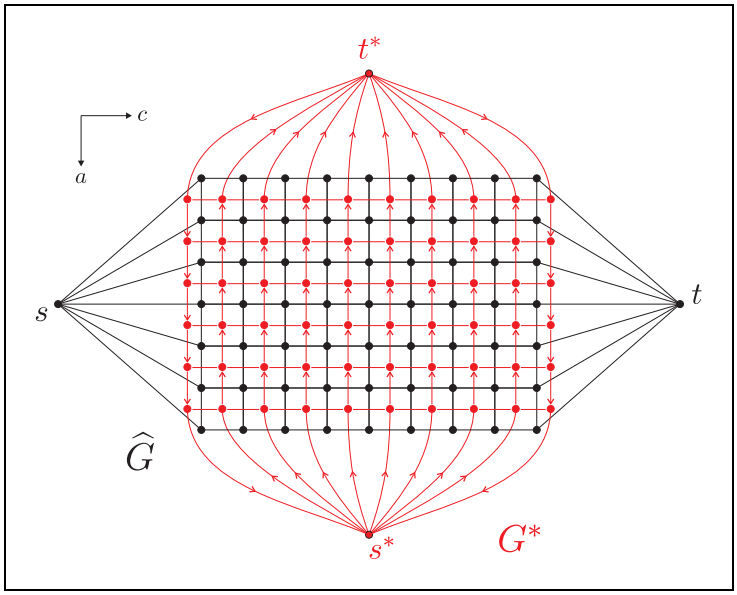

Figure 3.2: The graph G^* that corresponds to $\widehat{G}$.

We have a one-to-one correspondence between the edges of finite value in $\widehat{G}$ and the edges in G^* in the following way. To an edge $e \in \widehat{E} \subset E$ with $k(e) < \infty$, we associate the edge in $e^* \in E^*$ that crosses e (in the geometric representation as illustrated in figure 3.2) such that (e, e^*) is right-handed. It is easy to see that this rule effectively defines a one-to-one correspondence. Finally, we define the length function l^* in the following manner:

$$l^*(e^*) = k(e) \quad \forall\, e^* \in E^*$$

In words, the length of an edge in G^* is equal to the capacity of its corresponding edge in $\widehat{G}$. Note that l^* is effectively a positive length function as k is a positive capacity function.

Now we calculate the distances from s^* to all other vertices in G^* [(4)]. For $v^* \in V^*$, we denote by $d^*(v^*)$ the distance from s^* to v^*. With an analogue reasoning as in [9] p.260 ff we have the following property:

Property 3.3.1. *The distance $d^*(t^*)$ from s^* to t^* is the value of a minimum $s-t$ cut in $\widehat{G}$. Furthermore a minimum $s-t$ cut $\widehat{C} = [\widehat{V}_s, \widehat{V}_t]$ in $\widehat{G}$ can be constructed by defining $\widehat{V}_s$ as the set of the left side of a shortest path $\widehat{p}$ from s^* to t^* as illustrated in figure 3.3.*

[(4)] The *distance* from s^* to $v^* \in V^*$ in G^* is defined as the length of the shortest path from s^* to v^* in G^*.

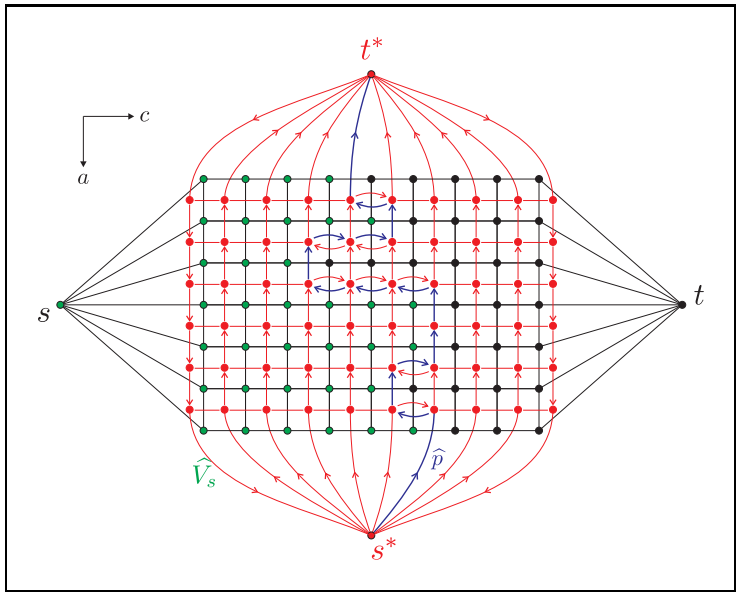

Figure 3.3: Finding a minimum $s-t$ cut $\widehat{C} = [\widehat{V}_s, \overline{\widehat{V}_s}]$ in $\widehat{G}$ by a shortest path $\widehat{p}$ from s^* to t^* in G^*.

This property says that we can find rapidly a minimum $s-t$ cut in $\widehat{G}$ by a shortest path in G^*, but to be able to construct the residual network after the preprocessing, we have to find not only a minimum $s-t$ cut, but also a corresponding maximum flow.
Here too, we can use the same idea that is already known for undirected planar networks and is explained in [9]. With an analogue proof as in [9], we can show that the following property is valid:

Property 3.3.2. *The flow φ as defined below is a maximum flow in G.*

$$\varphi(e) = \begin{cases} d^*(v_2^*) - d^*(v_1^*) & \forall\, e \in E \text{ with } k(e) < \infty \\ -\varphi(e^R) & \forall\, e \in E \text{ with } k(e) = \infty \end{cases}$$

where $(v_1^*, v_2^*) = e^*$

This property explains finally how we can get the augmenting flow φ with this method.

The second preprocessing algorithm can thus be resumed by the pseudocode described in algorithm 2. The complexity of this algorithm is $\mathcal{O}(n \log(n_a n_c))$. The first for-loop is repeated n_b times. The construction of the graph G^* in line 3 can easily be done in $\mathcal{O}(n_a n_c)$ time by taking advantage of the regular grid structure of $\widehat{G}_b$. By applying Dijkstra's algorithm for shortest paths, we can perform line 4 in $\mathcal{O}(n_a n_c \log(n_a n_c))$ time [(5)]. The second for-loop will be repeated $\mathcal{O}(n_a n_c)$ times and line 6 needs only constant

[(5)] We can achieve this bound by implementing Dijkstra's algorithm with a binary heap or a Fibonacci heap. See [9] for more information on Dijkstra's algorithm and its implementation.

time for each execution. Finally, we need $\mathcal{O}(n)$ to determine the residual network G_φ. So the bottleneck operation in the first for-loop is line 4 and we get a complexity of $\mathcal{O}(n_b n_a n_c \log n_a n_c) = \mathcal{O}(n \log(n_a n_c))$ for the whole preprocessing algorithm. Note that this complexity is inferior to $\mathcal{O}(n \log(n))$ and will not slow down the combined complexity if we apply afterwards a maximum flow algorithm.

Algorithm 2 Second preprocessing algorithm on reconstruction networks

```
1: procedure PREPROCESS 2(G)   ▷ where G = (V, E, k) is a reconstruction network
2:     for b = 0 to n_b − 1 do
3:         Construct the graph G* = (V*, E*, l*) corresponding to Ĝ_b = (V̂_b, Ê_b, k̂_b).
4:         Calculate the distances d* between s* and all other vertices in G*.
5:         for e ∈ Ê_b do
6:             φ(e) = d*(v*_2) − d*(v*_1)   where (v*_1, v*_2) = e*
7:         end for
8:     end for
9:     G = G_φ
10: end procedure
```

Some remarks on the resolution method Note that in a more general manner, we can simply construct the $s-t$ dual of the graph $\widehat{G}$ and define a one-to-one correspondence between all the edges (not only those with finite capacity) of $\widehat{G}$ and its $s-t$ dual G^* [(6)]. An edge $e^* \in E^*$ that corresponds to an edge $e \in \widehat{E} \subset E$ with $k(e) = \infty$, will simply get a length of ∞. So this will not change the distances from s^* to any other vertex in V^*. Note that this dual-technique can be applied to any $s-t$ planar directed graph. But if we do not work on reconstruction networks, it may be more difficult to construct the $s-t$ dual graph G^*.

Bounding the value of the remaining flow

In the same way as in the first preprocessing algorithm, we apply the second preprocessing procedure and then introduce an $s-t$ cut that allows to bound the remaining maximum flow from s to t. As for every $b \in \{0, 1, \ldots, n_b - 1\}$ we have solved the maximum flow problem on $\widehat{G}_b$, we can determine for each of these networks a minimum $s-t$ cut $\widehat{C}_b = [\widehat{V}_s^b, \widehat{V}_t^b]$ by the algorithm introduced in 2.3. We now define the following valid $s-t$ cut C on G:

$$C = [V_s, V_t]$$
$$\text{where} \quad V_s = \bigcup_{b=0}^{n_b-1} \widehat{V}_s^b$$

[(6)] The $s-t$ dual of the graph $\widehat{G}$ is the same as the graph G^* we introduced before, with the difference that it is completed with the missing reversal arcs, i.e. we have the property that if $e^* \in G^*$ so $e^{*R} \in G^*$.

The $s-t$ cut C contains no smoothness arcs parallel to a with positive capacity as well as no consistency arcs with positive capacity. This is a direct consequence of the fact that the cuts $\widehat{C}_b$ are minimum in their corresponding network $\widehat{G}_b$.
Now it is easy to show that we have

$$k(C) \leq nK^G$$

So with this preprocessing too, we know that after applying it, the remaining maximum flow value ζ is bounded by $\mathcal{O}(nK^G)$. Even though this is the same complexity bound as with the first preprocess algorithm, the second one gives in practice much better results.

Variations of the algorithm

The second preprocessing algorithm we discussed was based on the preliminary elimination of smoothness edges parallel to the b-direction. This algorithm can be applied in an analogue manner by preliminary elimination of smoothness edges parallel to the a-direction (with a complexity of $\mathcal{O}(n\log(n_b n_c))$). If we use the first version of this algorithm so we say that we apply the *second preprocessing algorithm with respect to the b-direction* and in the second case we call it the *second preprocessing algorithm with respect to the a-direction.*
An interesting practical application is to apply first the second preprocessing algorithm with respect to the b-direction and then with respect to the a-direction (note that the complexity of this combination is still inferior to $\mathcal{O}(n\log(n))$). Unfortunately, it is in general not possible to give better bounds on the remaining maximum flow value ζ than nK^G but the empirical behavior of this algorithm shows to be very interesting.

By repeating the second preprocessing algorithm alternatively with respect to the b-direction and to the a-direction we could try to find better initial flows, but there are several problems with this method. Theoretically, even if we continue this procedure till we cannot augment the flow anymore by the second preprocessing algorithm, we will finally obtain in general a non-optimal solution and we cannot give a better bound on the remaining maximum flow value than nK^G. And practically, the iterations after having applied the second preprocessing algorithm once with respect to the b-direction and then with respect to the a-direction find in general very little additional flow.
Table 3.3.2 gives the results of three empirical tests on different problems. One can see that applying two iterations of the second preprocessing algorithm seems to be a very interesting method. Here we can hope to get an initial flow of 95% of the optimum. Note that the quality of the preprocessing results depends on the graph smoothness factor K^G. The smaller K^G is, the better will be the result. Our tests were made with smoothness factors around 5, which is a very typical choice.

	Optimum	Preprocess1	Preprocess 2			
			i=1	i=2	i=3	i=4
Problem 1	853047	458758	718351	811148	811783	811799
	100%	53.8%	84.2%	95.1%	95.2%	95.2%
Problem 2	288477	183278	248176	278636	278707	278708
	100%	63.5%	86.0%	96.6%	96.6%	96.6%
Problem 3	934674	740437	839591	911959	912144	912144
	100%	79.2%	89.8%	97.6%	97.6%	97.6%

Table 3.1: Empirical results of the first and second preprocessing algorithm (where i represents the number of iterative applications of the second preprocessing algorithm). The upper values represent the flow values found by the algorithms and the lower values their percentage with respect to the maximum flow value.

3.4 Augmenting path algorithms

3.4.1 Introduction

Augmenting path algorithms are known to be the first algorithms developed for the resolution of the maximum flow problem. They try to augment flow by finding augmenting paths (see 2.1 for information about augmenting paths). The primary question with this technique is how we have to choose augmenting paths to get efficient algorithms. Augmenting path algorithms typically apply one of the following methods:

1. Choose the first augmenting path you find.
2. Choose paths that allow to transport a ≪sufficiently large≫ amount of flow.
3. Choose shortest augmenting paths such that the distance between source and sink monotonically increases during the algorithm.

The first method is the simplest one and allows finding augmenting paths very quickly. Unfortunately, it is possible that we have to find a very large number of augmenting paths to obtain finally a maximum flow. Because the augmenting paths we use have no particular structure, we typically cannot do better than to limit the number of augmenting paths we have to find by ζ because each augmenting path augments the flow of at least one unit (since the capacities are integral) and the total augmentation that will be performed is ζ. Therefore the complexities of these algorithms typically depends on U and thus are only pseudopolynomial.

Algorithms of the second type try to limit the dependence on capacities. By choosing augmenting paths with relatively large capacities, we have to perform less augmentations. We will not get rid off the problem that the complexity depends on U, but while

algorithms of the first type typically depend linearly on U, we can reduce this dependency to $\log(U)$. On the other hand, it is normally more expensive to find paths with large capacities.

With the third method, we are able to eliminate the dependence upon U of the complexity and thus to obtain polynomial time algorithms. These algorithms typically work through a number of phases where at each phase, we try to find a family of shortest augmenting paths with the property that after having augmented the flow on these paths, the distance between source and sink in the new residual network increases. The main idea of these algorithms is that we can bound the number of phases by n, because if the distance between source and sink is bigger than $n-1$, so there will be no augmenting path from s to t in the residual network. Additionally, finding an appropriate family of augmenting paths as explained before, as well as the augmentation on this family, can be done in strongly polynomial time. Therefore we are able to obtain strongly polynomial maximum flow algorithms with this method.

We will now rapidly introduce some important augmenting path algorithms with their worst-case complexity in general networks and reconstruction networks.

3.4.2 General labelling algorithm and Edmonds-Karp algorithm

The *general labelling algorithm* simply tries to find an augmenting path from the source to the sink by applying a labelling technique. Finding an augmenting path in this way has thus a complexity of $\mathcal{O}(m)$. By the previous discussion, we know that the number of iterations for finding augmenting paths is bounded by the maximum flow value ζ. In general networks, this algorithm has therefore a complexity of

$$\mathcal{O}(m\zeta) \leq \mathcal{O}(nmU)$$

In reconstruction networks, we can combine this with one of the presented preprocessing algorithms to get a complexity of

$$\mathcal{O}(n\zeta) \leq \mathcal{O}(n^2 K^G)$$

By applying a breath-first-search rule for the labelling subroutine used for finding an augmenting path, we can find a shortest augmenting path in $\mathcal{O}(m)$. This rule leads to the *Edmonds-Karp algorithm*[7] which is an augmenting path algorithm of the third type. The Edmonds-Karp algorithm was developed in 1970 and is the oldest known strongly polynomial time algorithm for the resolution of the maximum flow problem. The worst case complexity of this algorithm on general networks is

$$\mathcal{O}(nm^2)$$

[7] The Edmonds-Karp algorithm is also called *successive shortest augmenting path algorithm.*

For additional information see for example [9] or [10]. Because the Edmonds-Karp algorithm is a particular case of the general labelling algorithm, we know that its computational time is also bounded by $\mathcal{O}(m\zeta)$. Therefore we have the following complexity bounds:

$\mathcal{O}(\min\{nm^2, m\zeta\})$	$\leq$	$\mathcal{O}(\min\{nm^2, mnU\})$	on general networks
$\mathcal{O}(\min\{n^3, m\zeta\})$	$\leq$	$\mathcal{O}(\min\{n^3, n^2K^G\})$	on reconstruction networks in combination with a presented preprocessing algorithm

As in reconstruction networks, we have in general $K^G \leq 10 \ll n$, we see that the fact that the Edmonds-Karp algorithm is strongly polynomial does not improve the theoretical worst-case complexity on reconstruction networks.

3.4.3 Algorithm of Dinic

The algorithm of Dinic is of the third type and can be seen as a speed up version of the algorithm of Edmonds and Karp. Dinic introduced the idea to work with shortest paths networks, called *layered networks*. To construct the corresponding layered network $G_l(V, E_l, k_l)$ to a network $G = (V, E, k)$, we begin with calculating the distances from each node $v \in V$ to t [8]. We denote the distance from $v \in V$ to t by $d(v)$. With this notation we can define the layered network G_l as follows:

$$E_l = \{(v, w) \in E \mid d(v) = d(w) + 1\}$$
$$k_l(e) = k(e) \quad \forall\, e \in E_l$$

It is easy to see that G_l is acyclic and contains all the shortest paths from s to t in G. On a layered network G_l, we can construct a so called *blocking flow* which is defined as follows:

Definition 3.4.1 (Blocking flow)**.** *A flow φ on a layered network $G_l = (V, E_l, k_l)$ is called a blocking flow, if for every path p from s to t in G_l, we have that p contains at least one arc e that is saturated by φ, i.e. $\varphi(e) = k(e)$.*

Note that a blocking flow does not have to be a maximum flow, but inversely a maximum flow on a layered network is always also a blocking flow. The reason for the introduction of the blocking flow lies in the following property:

Property 3.4.2. *Let φ be a blocking flow in G_l. So the distance from s to t in G_φ is strictly greater than the distance from s to t in G.*

The algorithm of Dinic works in phases and can be explained in the following way. At the beginning of a phase i we have a residual network G_i (where we initialize with $G_1 = G$) and we construct its corresponding layered network $(G_i)_l$. Then we determine a blocking

[8] The length function we use for calculating distances, gives a length of 1 to each arc $e \in E$ with $k(e) > 0$ and ∞ to all other arcs.

flow φ_i in $(G_i)_l$ and pass to the next residual network through this blocking flow, i.e. $G_{i+1} = (G_i)_{\varphi_i}$.

By property 3.4.2 we can deduce that Dinic's algorithm finds a maximum flow in at most $n-1$ phases, because at the first phase the distance between s and t is ≥ 1 and therefore $\geq n$ at phase n. Thus at phase n, there exists no simple augmenting path [(9)] in the current residual network G_n and therefore no augmenting path in G_n. By theorem 2.1.10 we are sure to have found a maximum flow.
In each phase, Dinic's algorithm has to find a blocking flow. Here the algorithm profits from the fact that it is easier to find blocking flows than maximum flows in acyclic networks. It is possible to implement an algorithm that finds a blocking flow in an acyclic network in $\mathcal{O}(nm)$ [(10)].

Therefore Dinic's algorithm has the following worst-case complexities:

$$\begin{aligned} \mathcal{O}(n^2 m) &\quad \text{on general networks} \\ \mathcal{O}(n^3) &\quad \text{on reconstruction networks} \end{aligned}$$

Even if the theoretical complexities of Dinic's algorithm and the Edmonds-Karp algorithm are identical in sparse networks, it is in general preferable to apply Dinic's algorithm, as it has a lower constant factor in most cases.

Dynamic tree implementation for finding blocking flows

A dynamic tree is a rather complex data structure that was developed to improve the worst-case complexity of several network algorithms (in particular algorithms for finding blocking flows). By using dynamic trees, one can find a blocking flow in a complexity of

$$\mathcal{O}(m \log(n))$$

This allows to implement Dinic's algorithm with the following complexities:

$$\begin{aligned} \mathcal{O}(nm \log(n)) &\quad \text{on general networks} \\ \mathcal{O}(n^2 \log(n)) &\quad \text{on reconstruction networks} \end{aligned}$$

Unfortunately, the dynamic tree data structure normally slows down the algorithm in practice because of two reasons. At first this structure has a large overhead (a large constant factor of work is associated with each operation in this structure). And still more important is that it needs more time for operations that are in practice bottleneck operations (whereas it is faster in the theoretical worst-case bottleneck operations). For more information on dynamic trees, see [14].

[(9)] A path is called *simple* if it contains each arc of the network at most once.
[(10)] See for example [9] for additional information.

3.4.4 Capacity scaling algorithm

Capacity scaling is a simple approach to ensure that we find augmenting paths with large capacities. It is more a technique than simply a specific algorithm and finds numerous applications in different algorithms. Nevertheless, we will introduce this technique by the capacity scaling algorithm which was the first one using capacity scaling.
The capacity scaling algorithm passes through a number of phases where at each phase it augments flow by augmenting paths. We introduce a value Δ that corresponds to the minimum amount of flow that an augmenting path has to be able to transport from the source to the sink. For the first phase, we set $\Delta = 2^{\lfloor \log U \rfloor}$ and we work on the network $G_\Delta = (V, E, k_\Delta)$ which only considers capacities greater than Δ i.e.

$$k_\Delta : E \longrightarrow \mathbb{R}_+$$
$$k_\Delta(e) = \begin{cases} k(e) & \text{if } k(e) \geq \Delta \\ 0 & \text{if } k(e) < \Delta \end{cases}$$

This network is called the Δ-*network of* G [(11)]. We then apply the general labelling algorithm to find a maximum flow in the network G_Δ. Having determined a maximum flow φ_1, we terminate the first phase, set $\Delta = \frac{\Delta}{2}$ and we start the second phase where we have to find a maximum flow by the general labelling algorithm in the new Δ-network $(G_{\varphi_1})_\Delta$, and so on and so forth. In the last step, we have to find a maximum flow in a residual network of the network G_1 which is equal to G since we work with integer capacities. This proves that we will effectively find a maximum flow with this algorithm. It is easy to see that we will go through $1 + \lfloor \log(U) \rfloor$ phases and additionally it can be shown very directly that the maximum number of augmentations per phase is bounded by $\mathcal{O}(m)$ [(12)]. Taking into consideration that finding an augmenting path takes $\mathcal{O}(m)$ time, we thus get an algorithm with a worst-case complexity on general networks of

$$\mathcal{O}(m^2 \log(U))$$

If we use capacity scaling in Dinic's algorithm, so we even can decrease the complexity bound on general network to

$$\mathcal{O}(nm \log(U))$$

This variant of the algorithm is known as *Gabow's algorithm* (see [9] for more information). Anyway, on reconstruction networks the complexity reduces in both cases thanks to their sparseness to

$$\mathcal{O}(n^2 \log(U))$$

Here too, we can improve the worst-case complexity on reconstruction networks by a preliminary application of one of the presented preprocessing algorithms. We have

[(11)] In literature you will often find the expression Δ-*residual network* of G with respect to a flow φ on G. This corresponds in our context to the Δ-network of G_φ.
[(12)] See [9] or [10] for further information.

shown that after having applied such a preprocessing algorithm, we can find an $s-t$ cut $C = [V_s, V_t]$ where the only arcs with positive capacities in C are smoothness arcs. Therefore at this moment it is impossible to find an augmenting path with a capacity $\geq K^G$. So we can directly begin in the first phase of the capacity scaling algorithm with $\Delta = K^G$, which implies a worst-case complexity bound in reconstruction networks of

$$\mathcal{O}(n^2 \log(K^G))$$

As already mentioned, capacity scaling can be used for other maximum flow algorithms too. Another positive point of this technique is that it is in general easy to implement.

3.4.5 MA-ordering algorithm

The MA-ordering algorithm was introduced by the two Japanese researchers Satoru Fjishige and Shigueo Isotani in 2003. We will just give the main ideas of the algorithm (for further information see [12]).
The MA-ordering algorithm is an algorithm of the second type that iteratively increases flow through augmenting flows. The abbreviation ≪MA≫ stands for *maximum adjacency* and describes the way how this algorithm constructs augmenting flows. At each iteration the algorithm begins by ordering the vertices from s to t by maximum adjacency which is done as follows. Let $G = (V, E, k)$ be the current residual graph. We introduce a set $W \subset V$ which contains all vertices that have already been ordered. At the beginning of the MA-ordering we set $v_0 = s$ and $W = \{v_0\}$. We will then iteratively introduce a vertex $v \in V \setminus W$ into W. At the step i we have $W = \{v_0, v_1, \ldots, v_{i-1}\}$. We associate to each vertex $v \in V$ an adjacency value $b(v)$ that is defined as follows:

$$b(v) = \begin{cases} k(\{v_0, v_1, \ldots, v_{j-1}\}, v_j) & \text{if } v = v_j \in W \\ k(W, v) & \text{if } v \in V \setminus W \end{cases} \tag{3.1}$$

We then introduce a vertex $v_i \in V \setminus W$ into W with the highest adjacency value, i.e.

$$v_i = \underset{v \in V \setminus W}{argmax}\, b(v)$$

As defined in 3.1, a vertex that was introduced in W does not change its adjacency value anymore. But we have to update the adjacency values of the neighbors of v_i in $V \setminus W$ for the next step of the MA-ordering.
We will stop the ordering at the moment when we introduce the sink t, for example $t = v_k$. Then we will build an augmenting flow on the partial graph of G that contains only edges connecting vertices in W to vertices in W with higher indices. We thus obtain an acyclic graph. It is possible to find in $\mathcal{O}(m)$ an augmenting flow φ in this network of value $k(\varphi)$ equal to

$$\delta = \min_{v \in W} b(v) \quad {}^{(13)}$$

[13] See [12] for more information.

It can also be shown that δ is at least as big as the capacity of any augmenting path (and therefore also of the augmenting path with the highest capacity). This property in combination with the flow decomposition theorem [14] allows to give a complexity bound on the number of augmentations of $\mathcal{O}(m \log(U))$ (to get this bound we use a typical reasoning that can be found for example in [9] p.211).
The inventors of the algorithm were able to show that the number of augmentations can also be bounded by $\mathcal{O}(n \log(\zeta)) \leq \mathcal{O}(n \log(nU))$.

To efficiently implement the above described algorithm we need an appropriate data structure for the MA-ordering. One can observe that the operations done by an MA-ordering are the same as the ones used for finding a shortest path with Dijkstra's algorithm. Therefore the Fibonacci-Heap for example allows to perform an MA-ordering in $O(m + n \log(n))$ time.
We finally get the following running times of the MA-ordering algorithm:

$\mathcal{O}((m + n \log(n))n \log(\zeta)) \leq \mathcal{O}((m + n \log(n))n \log(nU))$	on general networks
$\mathcal{O}(n^2 \log(n) \log(\zeta)) \leq \mathcal{O}(n^2 \log(n) \log(nK^G))$	on reconstruction networks in combination with a presented preprocessing algorithm

In [12] an interesting scaling version of the above presented algorithm is introduced which does not need sophisticated data structure like the Fibonacci heap and achieves the following running times:

$\mathcal{O}(nm \log(U))$	on general networks
$\mathcal{O}(n^2 \log(K^G))$	on reconstruction networks in combination with a presented preprocessing algorithm

3.4.6 Growing-trees algorithm

In September 2004, Yuri Boykov and Vladimir Kolmogorov (two researchers in the domain of computer vision) introduced a new algorithm for solving maximum flow problems and tested it with success on various flow problems appearing in computer vision. Because of its structure, we will refer to this algorithm as the *growing-trees algorithm*. This is an algorithm of the first type and we will only give a brief description, for further information see [7].

The growing-tree algorithm is similar to the general labelling algorithm but has two important differences. Whereas the general labelling algorithm uses one search tree beginning at the source to find augmenting paths, we will use two search trees in the growing-trees algorithm. One T_{source} with root at the source and another one T_{sink} with

[14] See [9] p.80 for more information on the flow decomposition theorem.

root at the sink. We will alternately grow the trees in the same manner as in the general labelling algorithm (we simply have to adapt the labelling rule for the sink tree T_{sink}). When the two trees ≪touch≫ each other (i.e. when they contain a common vertex), then we have found an augmenting path. The second difference between the growing-tree algorithm and the general labelling algorithm is that in the growing-trees algorithm, after increasing the flow by the augmenting path found, we will not rebuild the two trees T_{source} and T_{sink} from scratch on but try to keep as much information as possible. For information how this is done, we refer to [7].

Thus the growing-trees algorithm repeats iteratively the following three stages:

1. **Growth stage:** The two trees T_{source} and T_{sink} grow until they touch.
2. **Augmentation stage:** The flow is increased through the augmenting path. found in the growth stage
3. **Adoption stage:** The trees T_{source} and T_{sink} are restored.

It is easy to see that one execution of the growth or augmentation stage takes $\mathcal{O}(m)$ time on general networks. By studying the implementation of the adoption stage presented in [7] one can examine that its worst-case complexity per execution is bounded by $\mathcal{O}(n^3)$ on general networks and will be achieved in only very particular conditions. On reconstruction networks the adoption stage has a complexity that is bounded by $\mathcal{O}(n^2)$. We therefore get the following complexity bounds for the growing-trees algorithm:

$$\begin{aligned} \mathcal{O}(n^3\zeta) \leq \mathcal{O}(n^4 U) &\quad \text{on general networks} \\ \mathcal{O}(n^2\zeta) \leq \mathcal{O}(n^3 K^G) &\quad \text{on reconstruction networks} \end{aligned}$$

From a theoretical point of view, these are very poor complexities, but practically the time needed in the adoption phase is usually much better than the theoretical worst-case. On reconstruction networks the growing-tree algorithm in general clearly outperforms the general labelling algorithm (which has no adoption phase).

The authors of this algorithm only present a non-scaling version. But in the same spirit as in the capacity scaling, we can introduce scaling by working with Δ-networks to get complexity bounds of

$$\begin{aligned} \mathcal{O}(mn^3 \log(U)) &\quad \text{on general networks} \\ \mathcal{O}(n^3 \log(K^G)) &\quad \text{on reconstruction networks} \end{aligned}$$

3.4.7 Algorithm of Goldberg and Rao

In 1997 Goldberg and Rao introduced a maximum flow algorithm using a new idea. Like Dinic's algorithm they iteratively construct blocking flows but for the construction

of the corresponding layered network they use a different length function. While in Dinic's algorithm the length of any arc with positive residual capacity is equal to one, the algorithm of Goldberg and Rao gives a length of zero to arcs with large residual capacities and inversely a length of one to arcs with relatively small residual capacities. Applying this idea, the corresponding layered network contains augmenting paths with quite high capacities. This algorithm can be seen as a mixture between type 2 and type 3 that tries to profit from both advantages because on the one hand we introduce a binary length function for finding augmenting paths with high capacities and on the other hand we still work with blocking flows to increase the distance from the source to the sink.

The exact description of the algorithm is rather complicated as it has to treat several technical difficulties (as for example cycles of zero length when constructing a layered network) and we refer to [13] for details. On general networks this algorithm has a complexity bound of

$$\mathcal{O}(\min\{n^{\frac{2}{3}}, m^{\frac{1}{2}}\} m \log(\frac{n^2}{m}) \log(U))$$

From a theoretical point of view, this algorithm is the fastest known algorithm on a large variety of maximum flow problems (in particular on reconstruction networks). On reconstruction networks we achieve a complexity of

$$\mathcal{O}(n^{\frac{3}{2}} \log(n) \log(U)) \quad (15)$$

Unfortunately, this algorithm is unlikely to be efficient in practice because it has substantial overhead (large constant time is needed to perform steps of the algorithm) and especially because it is very likely that the empirical running time will be near the theoretical worst-case complexity.

3.5 Preflow-push algorithms

3.5.1 Introduction and generic preflow-push algorithm

Algorithms in the preflow-push family are among the most efficient maximum flow algorithms in practice. Additionally, they are all strongly polynomial. Preflow-push algorithms use a rather different technique than augmenting path algorithms. Flow will be sent locally along arcs whereas augmenting path algorithms always send flow directly from the source to the sink. By sending flow along arcs, we will not satisfy the flow conservation property at intermediate stages anymore. Therefore preflow-push algorithms work with so called *preflows* instead of flows that are defined as follows:

(15) Unfortunately, a preliminary application of a preprocessing algorithm does not allow to improve easily the complexity bound.

Definition 3.5.1 (Preflow). *A preflow φ in a network $G = (V, E, k)$ is a function*

$\varphi : E \longrightarrow \mathbb{R}$ *that satisfies*

i)	*Skew symmetry:*	$\varphi(v,w) = -\varphi(w,v)$	$\forall\ (v,w) \in E$
ii)	*Capacity constraint:*	$\varphi(v,w) \leq k(v,w)$	$\forall\ (v,w) \in E$
iii)	*Relaxed flow conservation:*	$\sum_{e \in \omega^-(v)} \varphi(e) \geq 0$	$\forall\ v \in V \setminus \{s\}$

To simplify notations, we will use the same short-notations for preflows as for flows. Additionally, we introduce residual networks with preflows in the same way as with flows.

So the only difference between a preflow and a flow is the relaxation of the flow conservation property. By working with preflows, it is possible that a vertex $v \in V \setminus \{t\}$ has more flow entering than exiting. This imbalance is measured by the excess:

Definition 3.5.2 (Excess of a vertex). *The excess $ex(v)$ of a vertex $v \in V$ is defined as the flow entering in v minus the flow exiting from v, i.e.*

$$ex(v) = \varphi(V \setminus \{v\}, v)$$

By the relaxed flow conservation property, we have that all vertices except t have positive excesses. A vertex with strictly positive excess is a vertex from which we can send flow to a neighbor, such a vertex is called an *overflowing vertex*.
When sending flow locally on arcs, we have to ensure that on the whole, flow will be sent closer to the sink. For this reason, we introduce a height function $h : V \longrightarrow \mathbb{N}$ that will give estimations of the distances[16] from each vertex to the sink. Formally we define a height function in the following way:

Definition 3.5.3 (Height function). *Let φ be a preflow in the network G. So $h : V \longrightarrow \mathbb{N}$ is a height function for G_φ if*

i)	$h(s) = n$	
ii)	$h(t) = 0$	
iii)	$h(v) \leq h(w) + 1$	$\forall\ (v,w) \in E$ *with* $k_\varphi(v,w) > 0$

We call $h(v)$ the height of the vertex v.

It is easy to see that by property *iii)* of a height function we have that for every $v \in V \setminus \{s\}$, $h(v)$ is a lower bound of the distance from v to t that we will denote by $d(v)$.
Preflow-push algorithms always send flow downhill, i.e. from higher vertices to lower ones in the hope to push the excess nearer to the sink. Arcs on which flow can be pushed are called admissible arcs and are defined as follows:

[16] In this context every arc with strictly positive residual capacity has a length of 1 and all other have a length of 0.

Definition 3.5.4 (Admissible arc)**.** *An arc $(v, w) \in E$ is called admissible if it satisfies*

$$\begin{aligned} i) \quad & h(v) = h(w) + 1 \\ ii) \quad & k_\varphi(v, w) > 0 \end{aligned}$$

So having an overflowing vertex $v \in V$ and an admissible arc $(v, w) \in E$, we can push as much excess as possible from v to w. This procedure is called $Push(v, w)$ and is one of the basic operations of a preflow-push algorithm. Algorithm 3 gives a pseudocode for $Push(v, w)$.

Algorithm 3 The basic operation $Push(v, w)$ on a residual network G_φ where φ is a preflow

▷ **Applies when:** v is overflowing and $(v, w) \in E$ is admissible.

```
1: procedure PUSH(v, w)
2:     λ = min{ex(v), k_φ(v, w)}
3:     φ(v, w) = φ(v, w) + λ
4:     φ(w, v) = φ(w, v) − λ
5:     ex(v) = ex(v) − λ
6:     ex(w) = ex(w) + λ
7: end procedure
```

To ensure that the algorithm will not lose the global picture of the problem, we have to update regularly the height function h. This can be done in the following situation. If $v \in V$ is an overflowing vertex such that for all neighbor $w \in V$ with $k_\varphi(v, w) > 0$ we have $h(v) \leq h(w)$ so we can increase the height of v to $1 + \min\{h(w) \mid k_\varphi(v, w) > 0\}$ [17]. This procedure is called $Relabel(v)$ and is the second basic operation in a preflow-push algorithm. A pseudocode of it is described in algorithm 4.

Algorithm 4 The basic operation $Relabel(v)$ on a residual network G_φ where φ is a preflow

▷ **Applies when:** v is overflowing and for all neighbors w with $k_\varphi(v, w) > 0$ we have $h(v) \leq h(w)$.

```
1: procedure RELABEL(v)
2:     h(v) = 1 + min{h(w) | k_φ(v, w) > 0}
3: end procedure
```

It can be proven that when we perform applicable Push and Relabel operations, so h will always be a height function as defined in 3.5.3. Furthermore we will effectively send as much flow as possible to the sink and by performing Push and Relabel operations as long as possible, we finally send the excess, that cannot reach the sink anymore, back to the source and our preflow therefore transforms into a flow. We refer to [9] and [10] for

[17] This updating technique is also known in the label-correcting algorithm for the determination of shortest distances. See [9] p.136 ff for more information on this technique.

the proofs of these properties and for additional details.

This algorithm which does not specify a rule how to choose an applicable operation (either Push or Relabel) is known as the *generic preflow-push algorithm*. To complete the description of this algorithm we have to indicate how we initialize it, i.e. which initial preflow φ and height function h will be used. As initial preflow φ we use the preflow that saturates all arcs outgoing of the source and has a value of 0 on all other arcs, i.e. for $(v, w) \in E$ we have

$$\varphi(v,w) = \begin{cases} k(v,w) & \text{if } v = s \\ -k(w,v) & \text{if } w = s \\ 0 & \text{if } v \neq s \text{ and } w \neq s \end{cases}$$

For the initial height function h we use the following one:

$$h(v) = \begin{cases} n & \text{if } v = s \\ 0 & \text{if } v \in V \setminus \{s\} \end{cases}$$

The generic preflow-push algorithm has the following complexities:

$\mathcal{O}(n^2 m)$	on general networks
$\mathcal{O}(n^3)$	on reconstruction networks

For additional information on the complexity bound on general networks we refer to [9] and [10].

The different preflow-push algorithms that we will discuss afterwards, work exactly in the same way as the generic preflow-push algorithm with the only difference that they specify rules how choosing an applicable operation.
All the specifications that we will discuss apply the following rule. They choose an overflowing vertex $v \in V \setminus \{t\}$ and apply as long as possible applicable Push(v, w) and Relabel(v) operations on v till v has lost all of its excess [18]. This procedure for eliminating all the excess of v is called *Discharge*(v). Algorithm 5 gives a pseudocode for it.
Therefore the specifications of the generic preflow-push algorithm that we will discuss afterwards simply give rules that determine an overflowing vertex that has to be discharged. Before presenting these algorithms we will introduce two heuristics that have shown to be crucial in making implementations of the preflow-push algorithms perform well in practice.

[18] It can easily be verified that it is always possible to get rid of all the excess of v in this way.

Algorithm 5 The operation $Discharge(v)$ on a residual network G_φ where φ is a preflow

```
   ▷ Applies when: v is overflowing
 1: procedure DISCHARGE(v)
 2:     while ex(v) > 0 do
 3:         if ∃w ∈ V such that Push(v, w) is applicable then
 4:             Push(v, w)
 5:         else
 6:             Relabel(v)
 7:         end if
 8:     end while
 9: end procedure
```

Global relabelling and gap heuristic

During the development of the preflow-push algorithms, several heuristics were proposed to speed up the algorithms in practice [19]. Especially two of them, called *global relabelling* and *gap heuristic*, have shown to be very useful.

They both profit from the following property:

Property 3.5.5. *If we have a preflow φ in the network G and an $s-t$ cut $C = [V_s, V_t]$ such that φ saturates C then we have the following conclusions:*

a) There exists a minimum $s-t$ cut $\widetilde{C}$ with $C \preceq \widetilde{C}$.

b) It is possible to get a preflow of maximum value only by effectuating Push and Relabel operations on the vertices in V_t.

So if we recognize that we are, during execution of a preflow-push algorithm, in a situation as described in property 3.5.5 so we can limit our attention to the vertices in V_t for finding a minimum $s-t$ cut respectively a maximum preflow. There are still the questions how to find out being in such a situation and how to transform finally a maximum preflow into a maximum flow. The global relabelling algorithm and the gap heuristic give answers to the first question.
The global relabelling algorithm simply calculates the exact distance $d(v)$ from every vertex $v \in V \setminus \{s\}$ to t by a breadth-first search from the sink on and actualizes the height function h as follows:

$$h(v) = \begin{cases} n & \text{if } v = s \\ d(v) & \text{if } v \in V \setminus \{s\} \end{cases}$$

Note that if we have a preflow φ in G that saturates an $s-t$ cut $C = [V_s, V_t]$, so after a global relabelling all vertices in V_s have a height that is $\geq n$ since there is no path from

[19] But they do not improve theoretical worst-case complexities.

a vertex $v \in V_s$ to t consisting of arcs with strictly positive residual capacity. Inversely the cut $[\{v \in V \mid h(v) \geq n\}, \{v \in V \mid h(v) < n\}]$ is always saturated by the preflow φ.
One can easily verify that after applying the global relabelling algorithm the $s-t$ cut $[\{v \in V \mid h(v) \geq n\}, \{v \in V \mid h(v) < n\}]$ will be the maximal $s-t$ cut that is saturated by φ.
The global relabelling algorithm is additionally very useful because it sets the heights to the exact distances (except for the vertex s). This will help to send the excesses more directly to the sink and thus accelerates the preflow-push algorithm. Global relabelling is something we will effectuate only from time to time to ensure that it will not become a bottleneck operation. By studying more deeply the preflow-push algorithm one can see that applying the global relabelling algorithm after each n Relabel operations will not have an effect on the worst-case complexity.

The gap heuristic looks more locally for situations as described in property 3.5.5. It is based on the following observation. If at some point in the execution of a preflow-push algorithm there exists a height $k \in \{1, 2, \ldots, n-1\}$ such that no vertex has height k. Then the $s-t$ cut $C = [V_s, V_t]$ as defined below is saturated by the preflow φ:

$$V_s = \{v \in V \mid h(v) > k\}$$

The justification of this is very direct. By the point *iii*) of the definition of a height function 3.5.3 we know that there are no arcs from V_s to V_t with strictly positive residual capacity. But this is equivalent to say that φ saturates C.

When we can find such a height k as explained before so we say that there is a *gap* at height k. By using an appropriate data structure it is easy to check if at a certain point of the algorithm, we have a gap or not [(20)]. Therefore the gap heuristic is less expensive in computational time than global relabelling but on the other hand we can typically only eliminate a small number of vertices and even if we have a saturated $s-t$ cut in the network, in general we will not find it directly by the gap heuristic. This explains also that these two heuristics can be complementary.

Finally for finding a maximum flow we will have to transform the maximum preflow into a maximum flow. There are algorithms that do this passage in $\mathcal{O}(nm)$ time on general networks [(21)]. On reconstruction networks this passage can even be done in $\mathcal{O}(n)$ because we can simply send the remaining excesses back to the source by the arcs $E_{backward}$ which all have infinity capacity. But if we are only interested in the minimum $s-t$ cut we can even skip the passage from a maximum preflow to a maximum flow.

We will now give an overview of different specifications of the generic preflow-push algorithm. More details can be found in [9] or [10].

[(20)] Checking if we have a gap or not can simply be done after each Relabel operation.
[(21)] See [9] for more information.

3.5.2 FIFO algorithm

The FIFO algorithm simply discharges overflowing vertices in a first-in first-out order, i.e. during the execution of the algorithm we maintain a list containing all the overflowing vertices. At the beginning, this list contains all the vertices which got excess by the initial preflow. Then we iteratively discharge the first vertex of this list and remove it from the list. All vertices that became overflowing through this operation will be added at the end of the list and so on and so forth.

The FIFO implementation of the generic preflow-push algorithm has a running time of

$\mathcal{O}(n^3)$ On both general networks and reconstruction networks

The FIFO algorithm is very simple to implement and achieves in general good practical running times (when implemented with global relabelling and gap heuristic). It has very similar characteristics as the so called *Relabel to front* implementation that is explained in [10].

3.5.3 Highest label algorithm

The highest label algorithm simply discharges always the highest overflowing vertex, i.e. it discharges an overflowing vertex v that satisfies $h(v) \geq h(w)\ \forall\, w \in V$ with $ex(w) > 0$. With this rule we first discharge vertices that are relatively far away from the sink. The advantage of this rule can be seen in the example on figure 3.4.

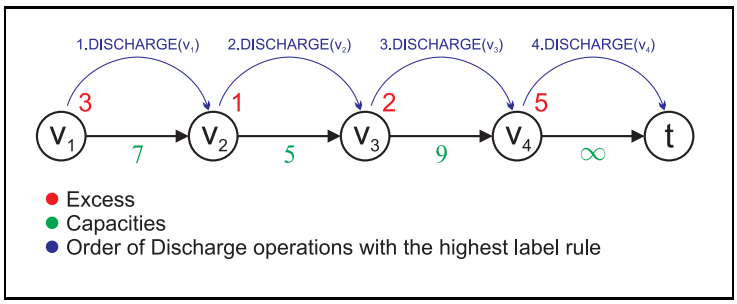

Figure 3.4: An example that shows the idea behind the highest label selection rule. Blue arrows indicate the transportation of the excess throught the Discharge operations.

If we begin by discharging vertices far away from the sink we first discharge v_1 then v_2, v_3 and finally v_4. In this way it is possible to get all the excess in only 4 Discharge operations to the sink (which is optimal).

The highest label algorithm has the following complexity bounds:

$\mathcal{O}(n^2\sqrt{m})$ on general networks

$\mathcal{O}(n^{\frac{5}{2}})$ on reconstruction networks

This algorithm has not only a very attractive strongly polynomial complexity bound but is also known to be one of the most efficient maximum flow algorithms in practice.

A typical characteristic of the highest label algorithm is that it works very locally because the vertices with great heights often are near to each other. This is also the main reason why the gap heuristic is very efficient in combination with this algorithm. But on the other hand this is a reason why the global relabelling heuristic is often less efficient with the highest label algorithm as it is unlikely to find great $s-t$ cuts that are saturated by the preflow. We will study empirical results of this algorithm (implemented with the gap heuristic) on reconstruction networks later.

3.5.4 Wave algorithm

The wave algorithm uses a similar idea as the highest label algorithm but tries to favor the global relabelling heuristic instead of the gap heuristic. It can be seen as a hybrid version of the highest label and FIFO algorithm. It performs passes over the overflowing vertices where at each pass all overflowing vertices will be discharged in non-increasing order of their heights. In other words, at the beginning of a pass the algorithm looks for the highest overflowing vertex. Lets say that this vertex has a height of k. Then it discharges all the vertices with height k, afterwards all vertices with height equal to $k-1$ will be discharged and so on till we have discharged all the vertices with height equal to 1. Then the pass terminates and we begin with the next pass.

The difference between the wave algorithm and the highest label algorithm is the following. While discharging an overflowing vertex with height k, new overflowing vertices with height $> k$ may be created. The highest label algorithm would then discharge the highest overflowing vertex which is one of the newly created ones and has a height $> k$. The wave algorithm instead will continue to discharge vertices with height k and ignores during a pass newly created vertices with superior height.
As the wave algorithm always examines all overflowing vertices during a path, it acts more globally than the highest label algorithm. This favors the global relabelling heuristic. Unfortunately the gap heuristic is less efficient in combination with the wave algorithm because of the following reason. Lets suppose that we would get locally a saturated $s-t$ cut $C=[V_s,V_t]$ such that the heights of the vertices in V_s are $< k$ (for a $k \in \{1,2,\ldots,n-1\}$) and the heights of the vertices in V_t that are adjacent to the vertices in V_s are $> k$. Even in such a case it is unlikely that the height k will be a gap because the global working style of the wave algorithm makes it very probable that we have a vertex $v \in V_t$ not adjacent to the vertices in V_s that has a height of $h(v)=k$.
On the other hand it is possible to find rapidly great saturated $s-t$ cuts through global relabelling. Later will test empirically the performance of the wave algorithm (with different combinations of the heuristics) on reconstruction networks.

The complexity of the wave algorithm is

$\mathcal{O}(n^3)$ On both general networks and reconstruction networks

3.6 Performances of the different flow algorithms on reconstruction networks

In this section we discuss the performance of the previously introduced maximum flow algorithms on reconstruction networks. In our search for efficient algorithms we begin by a first short analyze that allows us to figure out algorithms that are unlikely to be efficient in practice.
In a next step we compare the performances of the remaining algorithms with empirical tests and discuss situations in which they are particularly strong or weak.

3.6.1 Sorting out algorithms unlikely to be efficient in practice

After having effectuated some tests on the time it takes to find augmenting paths from scratch on, it was clear that augmenting path algorithms using this technique are unlikely to be efficient in practice. Among these types of algorithms are namely the general labelling algorithm, the Edmonds-Karp algorithm, the MA-ordering algorithm, the capacity-scaling algorithm as well as the algorithm of Dinic. In the past, numerous tests in general networks with these types of algorithms where effectuated with the conclusion that their performance is rather poor in practice compared to other algorithms. In [7] the algorithm of Dinic was tested on reconstruction networks. These tests enforce the assumption that the algorithms mentioned above are unlikely to be efficient on reconstruction networks. The capacity scaling technique does not help a lot because the main problem is not large capacities (because we simply apply a preprocessing algorithm) but the construction of augmenting paths.

The algorithm of Goldberg and Rao is currently the algorithm with the best worst-case complexity for maximum flow problems on reconstruction networks. By analyzing the algorithm we see that on the one hand it is probable that the practical performance of this algorithm will be near the worst case complexity bound and on the other hand the algorithm has extremely large constant factors. Especially the numerous subroutines and the use of complex data structures (we need for example the dynamic tree data structure to find blocking flows) slow down the algorithm in practice.
Apart of this, a short empirical analyze of a preflow-push algorithm on reconstruction networks in [1] shows that the empirical complexity of these types of algorithms can be clearly inferior to the complexity of the algorithm of Goldberg and Rao. Therefore it is unlikely that the latter algorithm will be efficient in practice.

For further analyzes and empirical tests we thus concentrate on the following algorithms:

- Preflow-push algorithms
- Growing-tree algorithms

3.6.2 Empirical tests and analyzes

We have implemented and tested the following algorithms working with the growing-trees technique:

- GT1 : growing-trees algorithm with first preprocessing algorithm
- GT2 : growing-trees algorithm with the second preprocessing algorithm
- GT2_S : a scaling version of the growing-trees algorithm in combination with the second preprocessing algorithm

The second preprocessing algorithm is always used with two iterations. Additionally we have implemented and tested the three previously presented preflow-push algorithms (FIFO, highest label and wave algorithm) with global relabelling and gap heuristic. First test results have shown that as expected the global relabelling heuristic is of great importance for the wave algorithm and the FIFO algorithm but slows down the highest label-algorithm in all of our experiments. The gap heuristic is crucial for the highest label algorithm as well as for the FIFO-algorithm. In the wave algorithm the usefulness of this heuristic was less clear. We therefore give experimental results of the following preflow-push algorithms that we abbreviate as follows:

- PPW : wave algorithm with global relabelling
- PPW_G : wave algorithm with global relabelling and gap heuristic
- PPH_G : highest label algorithm with gap heuristic
- PPF_G : FIFO algorithm with global relabelling and gap heuristic

We tried to implement the different flow algorithms carefully and to optimize the underlying data structures for reconstruction networks [(22)]. We have also paid attention to the fact that preflow-push implementations are running significantly faster on reconstruction networks when we favor Push operations on consistency arcs to pushes on smoothness arcs. And in the highest label algorithm, when different overflowing vertices are on the currently maximum height, so we try to break ties by choosing overflowing vertices in the same region as previously discharged vertices to provoke more gaps. More details of our implementations can be obtained by reading the comments of our source code.

We have tested these algorithms on three problems (that we denote by P1, P2 and P3) with four different graph smoothness factors (1,3,6 and 10). All the reconstruction networks have about $5 \cdot 10^6$ vertices. The Hardware we used was a Pentium M 1.6 GHz with 512 MB of RAM. Table 3.6.2 shows some obtained results.
All of the algorithms presented in the table solve typical reconstruction problems in a reasonable time. We see that problems become more difficult when using higher smoothness factors. Naturally the preflow-push algorithms are less sensible on the smoothness

[(22)] For example our wave algorithm often needs only between half and a third of the executional time compared to an implementation by the author of [1] (Version 1.07).

Problem	K^G	PPW	PPW_G	PPH_G	PPF_G	GT1	GT2	GT2_S
P1	1	42.9	59.5	42.8	80.3	15.5	22.6	22.6
	3	47.7	60.2	39.6	91.3	25.3	25.7	29.7
	6	50.2	62.9	45.2	100.9	50.7	35.6	39.6
	10	51.1	69.1	62.0	107.0	111.9	64.6	56.3
P2	1	22.2	23.7	19.2	34.1	10.0	12.9	12.9
	3	18.8	24.2	17.7	41.5	25.0	18.9	20.2
	6	22.8	29.0	27.4	47.9	48.8	28.2	27.0
	10	23.3	35.0	37.2	55.2	125.1	82.8	54.8
P3	1	37.0	45.5	22.2	62.8	13.2	21.5	21.5
	3	38.8	53.6	18.9	77.9	28.9	27.6	31.4
	6	45.4	56.5	25.1	94.0	57.5	39.4	43.1
	10	47.6	63.6	30.2	92.6	98.4	56.8	56.0

Table 3.2: Empirical results of flow algorithms on three reconstruction problems (P1,P2 and P3) with four different graph smoothness factors (1,3,6 and 10). The numbers in the columns of the algorithms represent the computational time in seconds for solving the problem.

factor than the augmenting path algorithms. The PPW_G algorithm always needed more computational time than PPW.
As already assumed in the theoretical part, we see that the gap heuristic does not improve the performance of the wave algorithm on reconstruction networks. The wave algorithm is even slowed down by the gap heuristic. The reason for this is that we need more complex data structures when using the gap heuristic because we have to be able to rapidly detect gaps, whereas the wave algorithm without gap heuristic can be implemented with relatively simple data structures. The wave algorithm is particularly rapid on reconstruction networks because in such a network we usually have large capacities on consistency arcs that are far away of the minimum $s-t$ cuts. Thus we can rapidly push flow near to the minimum $s-t$ cuts by the wave system and we already get a high probability that the global relabelling algorithm will be able to eliminate a huge part of the network.
The highest label algorithm also shows very interesting performances on reconstruction networks and its computational time does not change too drastically when we augment the smoothness factor. This algorithm provokes extremely often gaps.
The FIFO algorithm is the slowest one of the presented preflow-push algorithms. The main problem is that the efficiency of neither global relabelling nor gap heuristic is comparable to the case of PPW or PPH_G. It seems that also on reconstruction networks the main advantage of the FIFO algorithm is that it is very easy to implement because we do not need complex data structures.

Growing-trees algorithm show to be extremely fast when using little smoothness factors.

But when we do not implement techniques for handling large smoothness factors, so the computational time increases drastically as we can see by the results of GT1. As we can see in the table, using the second preprocessing algorithm is a possibility to diminish the computational time for reconstruction problems with large smoothness factors. When working with little smoothness factors, the additional time needed to perform two iterations of the second preprocessing algorithm is not covered by the gain in resolution time for the remaining problem. The scaling version GT2_S is an attractive option for limiting the computational time when working with large smoothness factors, and on the other hand we can still profit from finding a minimum $s-t$ cut relatively rapidly in the case of small smoothness factors.

3.6.3 Choice of an appropriate algorithm

As reconstruction problems with small smoothness factors are anyway not a big problem, it seems to be reasonable to choose an algorithm whose computational time does not explode when increasing the smoothness factors. Therefore the algorithms PPW, PPH_G, GT2 and GT2_S seem to be good choices to handle a large variety of reconstruction problems. When we only have to solve problems with relatively small smoothness factors, so GT1 is probably the fastest one.

4 Disparity reduction

In this chapter we develop methods for handling reconstruction problems where we have local reduction of the allowed disparities for the surfaces. This allows us in particular to impose internal and border conditions. Furthermore, theoretical results concerning this new formulation are given that provide a base for numerous applications. As an application of disparity reduction we propose an efficient pyramidal approach improving previous ones.

4.1 Introduction

Until now, a graph disparity mapping $f^G \in \mathcal{F}^G$ was a function of the type $f^G : \mathcal{D}^G \longrightarrow \{0, 1, \ldots, n_c - 2\}$ without further restrictions, i.e. independent of the point $q \in \mathcal{D}^G$ we choose, $f^G(q)$ can be any value in the codomain $\{0, 1, \ldots, n_c - 2\}$ (which corresponds to the disparities). In this chapter we will analyze formulations where we can forbid to reconstruct on certain disparities, depending on the point $q \in \mathcal{D}^G$ we choose.
Formally we fix for each $q \in \mathcal{D}^G$ a nonempty set $R_q^G \subset \{0, 1, \ldots, n_c - 2\}$ of admissible disparities for the point q. We resume these sets by the function R^G defined as follows:

$$\begin{aligned} R^G : \quad \mathcal{D}^G &\longrightarrow \mathcal{P}(\{0, 1, \ldots, n_c - 2\}) \setminus \emptyset \quad {}^{(1)} \\ q &\longmapsto R_q^G \end{aligned}$$

We call such a function R^G a *disparity reduction* or *height reduction*. The set of all *admissible graph disparity mappings with respect to* R^G will be denoted by $\mathcal{F}_{R^G}^G$ and is defined as follows:

$$\mathcal{F}_{R^G}^G = \{f^G \in \mathcal{F}^G \mid f^G(q) \in R_q^G \; \forall \, q \in \mathcal{D}^G\}$$

The set of all graph surfaces that correspond to mappings in $\mathcal{F}_{R^G}^G$ will be denoted by $\mathcal{S}_{R^G}^G$.
The optimization problem that corresponds to the disparity reduction R^G can now be formulated as follows:

$$\underset{S_{R^G}^G \in \mathcal{S}_{R^G}^G}{argmin} \; \left(\mathbf{E}_{\kappa^G}^G(S_{R^G}^G)\right)$$

Because the sets R_q^G are nonempty, we are sure that there exist a solution to this problem.

In numerous applications we do not need the possibility to reduce the disparities on arbitrary nonempty subsets of $\{0, 1, \ldots, n_c - 2\}$, but only on intervals of this set. Such a disparity reduction R^G satisfies

(1) For any set A, $\mathcal{P}(A)$ denotes the set of all subsets of A.

$\forall\, q \in \mathcal{D}^G \;\exists\, c^{R^G}_{min}(q), c^{R^G}_{max}(q) \in \{0, 1, \dots n_c - 2\}$ with $c^{R^G}_{min}(q) \leq c^{R^G}_{max}(q)$ such that
$R^G_q = \{c^{R^G}_{min}(q), c^{R^G}_{min}(q) + 1, \dots, c^{R^G}_{max}(q)\}$

Therefore the two functions $c^{R^G}_{min}, c^{R^G}_{max}$ are both of the type $\mathcal{D}^G \longrightarrow \{0, 1, \dots, n_c - 2\}$ and represent the lower respectively upper bounds for the allowed disparities. Disparity reduction on intervals will be a key ingredient for numerous applications we propose afterwards.

In the next section we will introduce graph formulations for the resolution of problems with disparity reduction.

4.2 Graph formulations for disparity reduction

The aim of this section is to give minimum $s - t$ cut formulations for the resolution of reconstruction problems with a disparity restriction R^G. For applications introduced afterwards, we only use disparity reductions on intervals. Therefore we will concentrate on this case.

A first simple idea to formulate disparity reduction is to modify the capacities of the original network (without disparity reduction) in the following way. We give an infinite capacity to all consistency arcs that correspond to forbidden disparities. To say if $(a, b, c) \in V'$ is a forbidden disparity so the arc $((a, b, c), (a, b, c + 1))$ gets a capacity of ∞. The justification of this construction is straight forward. Thanks to the fact that in the modified network exist $s - t$ cuts of finite values (because the set $\mathcal{S}^G_{R^G}$ is nonempty and the $s - t$ cuts corresponding the this graph surfaces have finite values), we know that a minimum $s - t$ cut cannot contain arcs of infinite capacity. Thus a graph surface corresponding to a minimum $s - t$ cut must respect the disparity reduction R^G. Furthermore the values of the $s - t$ cuts that respect R^G have not changed.

This simple idea allows to handle arbitrary disparity reduction. But it suffers from the fact that we still have to work with the whole graph even when we have a very restrictive reduction. This problem can be tackled by *contracting* adjacent vertices that are linked by arcs of infinite capacity. Contracting a subset of vertices $A \in V$ in a network $G = (V, E, k)$ means that we replace the set A by a single vertex v_A and every edge between a vertex $v \in V \setminus A$ and a vertex in A is replaced by an edge (with the same capacity as before contraction) from v to v_A. The algorithm 6 gives a formal description of this procedure.
One can see that contractions of this type does not change the nature of finite $s - t$ cuts in the network and it is still easy to pass from a finite $s - t$ cut to the corresponding valid graph surface. In fact the contraction of vertices linked by arcs of infinite capacity corresponds to the elimination of the consistency edges and corresponding vertices that

Algorithm 6 Contraction of a subset of vertices $A \in V$ in a network $G = (V, E, k)$

1: **procedure** CONTRACT(A)
2: Take the subnetwork of G on the vertices $V \setminus A$.
3: Add a vertex v_A (This vertex is a representative for the set A).
4: Add for every edge (v, w) in the original network with $v \in V \setminus A$ and $w \in A$ a vertex from v to v_A with the same capacity as (v, w). Analogously we add for every vertex (v, w) in the original network with $v \in A$ and $w \in V \setminus A$ an edge from v_A to v with the same capacity as (v, w).
5: **end procedure**

are forbidden by the disparity reduction R^G. Unfortunately this formulation changes the topology of the graph (we do not have anymore a grid structure) and often does not eliminate much edges. Therefore fast algorithms that were especially implemented for grid structures does not apply anymore.

That's why we introduce now another formulation for disparity reductions on intervals that keeps the grid structure of the graph and allows to reduce the graph to the region where the reconstruction respects R^G. The idea is to work only with the vertices where reconstruction is allowed by R^G. Formally we reduce the set V' to the following set:

$$\{(a, b, c) \in V' \mid c^{R^G}_{min}(a, b) \leq c \leq c^{R^G}_{max}(a, b) + 1\} \text{ (2)}$$

We then connect the source and the sink by vertices of infinite capacities to the new graph front and graph back to obtain a graph as illustrated in figure 4.1.
Working on such a network for solving the problem with disparity reduction on intervals is a very intuitive idea and was already used in [11]. Unfortunately the values of some valid $s - t$ cuts have changed in this formulation as shown by the figure 4.2. Thus the values of finite $s - t$ cuts in the network are not anymore equal to the energy of their corresponding cuts and we only can hope to get approximative solutions by working with minimum $s - t$ cuts on the network.
Therefore we investigated in this problem and developed a method that allows to handle this inconvenience by changing the capacities of some consistency arcs and eliminating some arcs in the network illustrated in figure 4.1.

For explaining our formulation we begin by a simple reconstruction network in 2D which only contains two elements in its graph domain D^G. In figure 4.3 we show how we change the disparities and edges on such a network. As illustrated in figure 4.2, the problem we have to cover is that in some $s - t$ cuts not all the smoothness arcs are taken into consideration. Our formulation compensates the values of these missing smoothness arcs by changing the capacities of some consistency arcs. This formulation can be justified

(2) The $\ll+1\gg$ after $c^{R^G}_{max}(a, b)$ corresponds to the additional slice in the formulation without disparity reduction and is necessary to get a consistency arc $((a, b, c^{R^G}_{max}(a, b)), (a, b, c^{R^G}_{max}(a, b)+1))$ that corresponds to $(a, b, c^{R^G}_{max}(a, b))$.

by using geometric arguments or arguments of graph theory. We present both type of reasonings and begin with the geometric one.

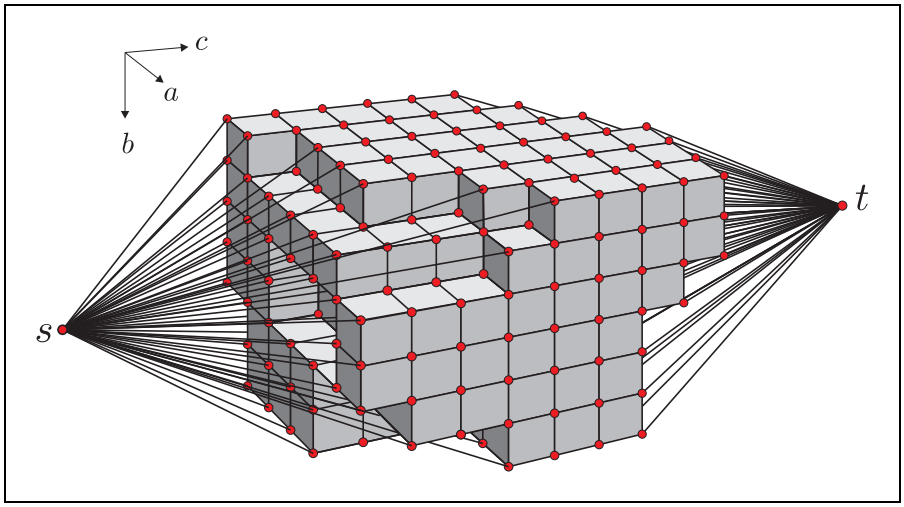

Figure 4.1: A representation of the graph for the resolution of a problem with disparity reduction on intervals.

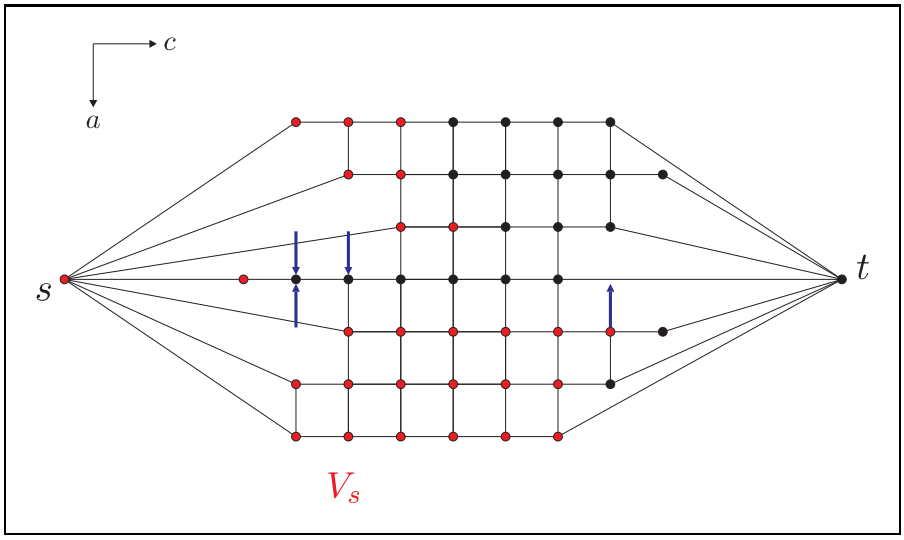

Figure 4.2: The $s-t$ cut $C = [V_s, \overline{V_s}]$ has an inferior value in the modified graph (which is shown in the figure) than in the original graph because C does not contain the blue arcs in the modified graph. Therefore we do not have anymore the property that the value of a valid $s-t$ cut equals the energy of the corresponding graph surface.

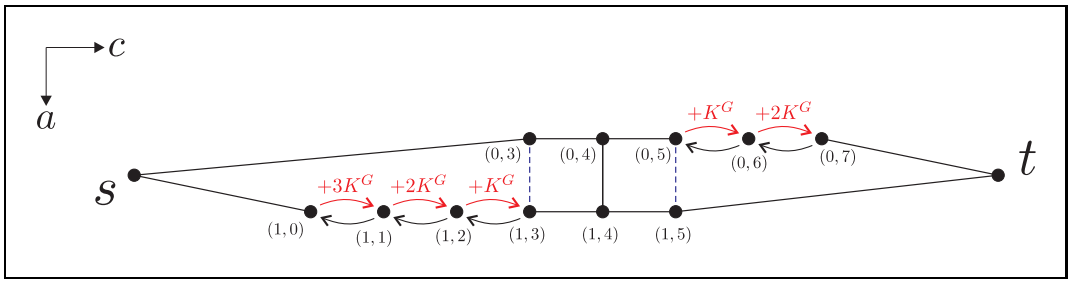

Figure 4.3: A simple problem with reduced disparities in 2D. Arcs whose capacities have been changed are drawn in red as well as the value by which we have changed the capacities. The eliminated edges are drawn as dashed blue lines.

When reconstructing on the vertex $(1,0)$ we are sure that we have a smoothness energy of at least $3K^G$ because the first disparity on which we can construct on the other disparity line is $(0,3)$. This energy corresponded in the original network without disparity reduction to the three smoothness arcs $((0,1),(1,1))$, $((0,2),(1,2))$ and $((0,3),(1,3))$ that do not exist anymore. This missing energy can be compensated by adding a value of $3K^G$ to the capacity of $((1,0),(1,1))$ as shown in figure 4.3. In the same manner we can justify the other changes.
We can also justify the changes by constructing the final graph in the following way. In a first step we construct the graph where we give a capacity of ∞ to forbidden consistency arcs and perform the vertex contraction as we have done in the previously introduced formulation. We finally obtain the network as illustrated in figure 4.4 a). Having performed this operations we are still sure that the value of a finite $s-t$ cut equals the energy of its corresponding surface. We will now step by step apply changes to the capacities and eliminate edges in the obtained network that do not touch the value of any $s-t$ cut till we get finally the graph as illustrated in figure 4.3. Note that we will exclusively eliminate edges with finite values and only change finite capacities to other finite capacities. Therefore the $s-t$ cuts of infinite values remain always exactly the same after the changes. So it will suffice to show that the values of finite $s-t$ cuts will not be modified by our subsequent changes.
The first change we do is to eliminate the edges $((0,3),(1,0))$,$((0,7),(1,5))$ and their reversals. This will not affect the value of any finite $s-t$ cut $C=[V_s,V_t]$ because C has to satisfy $(0,3),(1,0)\in V_s$ and $(1,5),(0,7)\in V_t$ and thus the eliminated edges cannot be in C. Furthermore because $(0,3)\in V_s$ and $(1,5)\in V_t$ we know that the edges $((1,1),(0,3))$, $((1,2),(0,3))$, $((1,3),(0,3))$, $((1,5),(0,6))$ and $((1,5),(0,5))$ cannot be in any finite $s-t$ cut. Therefore we eliminate them too and obtain the network illustrated in 4.4.
It can be easily seen that the arc $((0,3),(1,1))$ is in a finite $s-t$ cut C if and only if the arc $((1,0),(1,1))$ is in the cut C. The same is valuable for the two arcs $((0,6),(1,5))$ and $((0,6),(0,7))$. We can therefore eliminate the arcs $((0,3),(1,1))$ and $((0,6),(1,5))$ and add their capacities to $((1,0),(1,1))$ respectively to $((0,6),(0,7))$ as shown in figure 4.4 c).

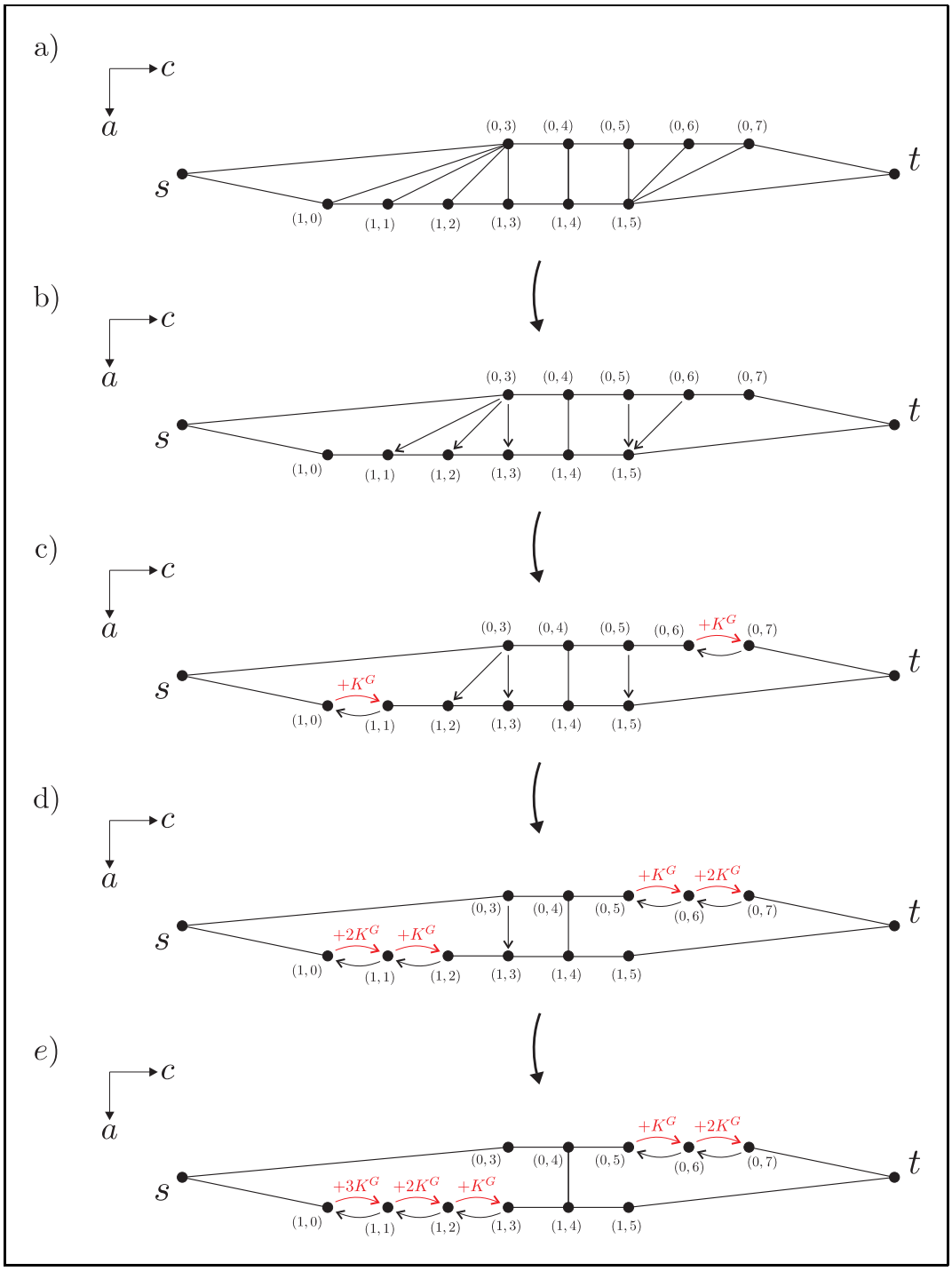

Figure 4.4: A step by step reasoning of the network modification for problems with reduced disparities on intervals.

Finally one can easily verify that the arc $((0,3),(1,2))$ is in a finite $s-t$ cut C if and only if one of the arcs $((1,0),(1,1))$ or $((1,1),(1,2))$ is in C. We can therefore eliminate the arc $((0,3),(1,2))$ and add its capacity K^G to $((1,0),(1,1))$ and $((1,1),(1,2))$ without changing the value of any finite $s-t$ cut. We can apply the same changes on the sink side of the graph as shown in the figures to obtain the network shown in figure 4.4 d). With an analog reasoning as before we see that the arc $((0,3),(1,3))$ is in a finite $s-t$ cut C if and only if one of the arcs $((1,0)(1,1)),((1,1)(1,2))$, of $((1,2),(1,3))$ is in C. We can therefore eliminate the edge $((0,3),(1,3))$ and add its capacity K^G to the other three edges mentioned above to obtain finally the graph illustrated in figure 4.4 e) which is the same as the one in figure 4.3.

There is just one special case of disparity reduction on intervals that can appear in the simple 2D graph on which we work. It is about the situation illustrated in figure 4.5 when the two disparity lines are not connected to each other. In this case we can simply apply the changes as shown in the figure. Additionally we have to add the constant $3K^G$ to the value of every finite $s-t$ cut to obtain its value before the modifications. This value is explained by the fact that we have a smoothness energy of at least $3K^G$ in our example due to the disparity reduction since there is a gap of three disparities between the restrictions on the two disparity lines. But for finding the minimum $s-t$ cut (and therefore the optimal surface) we evidently do not have to consider this constant energy addition.

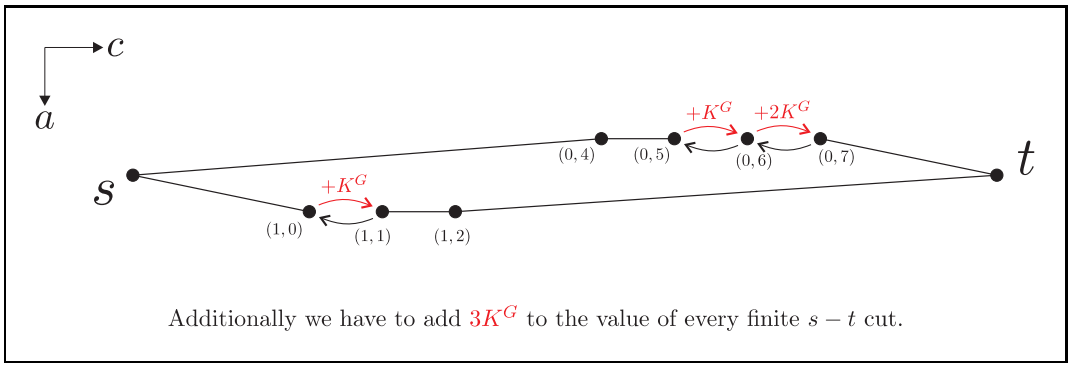

Figure 4.5: A special case that can arrive in disparity reduction by intervals.

This method for formulating disparity reductions on intervals on simple 2D graphs with two elements in the graph domain can be easily extended to arbitrary reconstruction networks. We simply have to apply for every two neighboring points in the graph domain the previously described modifications. Figure 4.6 shows how we apply this formulation on a more general example in 2D. The 3D case is analogue.

We have implemented the formulation with reduced disparities on intervals. Further

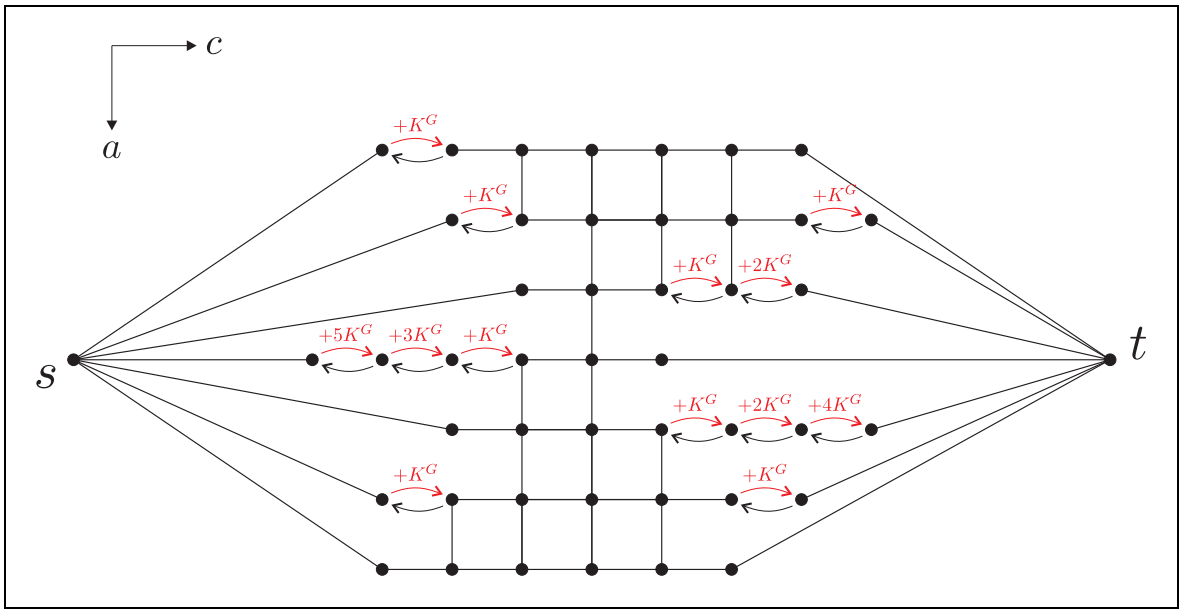

Figure 4.6: A 2D example of our formulation for disparity reduction on intervals.

information how to solve such problems with our code can be found in chapter about the code.

In the next section we will discuss an important special case of disparity reduction on intervals.

4.3 Internal and border conditions

An important special case of the disparity reduction on intervals are internal and border conditions. We talk about *internal conditions* when we fix some disparities for the reconstruction, i.e. for some chosen points $q \in D \subset \mathcal{D}^G$ we associate a priori disparities $c_q \in \{0, 1, \ldots, n_c - 2\}$ and we impose that a graph disparity mapping $f^G \in \mathcal{F}^G$ for the reconstruction has to satisfy $f^G(q) = c_q \; \forall \, q \in D$.
Internal conditions can easily be formulated as a problem with disparity reduction on intervals R^G by defining the functions $c^{R^G}_{min}$ and $c^{R^G}_{max}$ as follows:

$$c^{R^G}_{min}(q) = \begin{cases} c_q & \text{if } q \in D \\ 0 & \text{if } q \in \mathcal{D}^G \setminus D \end{cases}$$

$$c^{R^G}_{max}(q) = \begin{cases} c_q & \text{if } q \in D \\ n_c - 2 & \text{if } q \in \mathcal{D}^G \setminus D \end{cases}$$

We will now pass to the border conditions. When solving a reconstruction problem without disparity reduction, the smoothness effect on the border of the graph domain is less than in the middle as we do not have a smoothing influence from outside the graph domain. But when we impose a solution just outside the graph domain, so we can calculate the smoothing effect that this solution will have on the borders of our graph. This is what we call *border conditions*, i.e. we impose a solution for some regions outside the graph domain and we want to find the optimal solution inside the graph domain with respect to these border conditions.
An easy method to formulate border conditions is to enlarge the graph domain on the border regions where we impose conditions and to treat the border conditions as internal conditions. Therefore border conditions can be seen as a special case of internal conditions.
Having changed the network in this manner for solving a problem with border conditions, one can see that the added border region will not be connected to the vertices in the interior region by smoothness arcs anymore. We therefore can again re-eliminate the added border region before solving the problem. So the only changes we have done to the original network are changes of the capacities of the vertices at the border. Therefore it is not necessary to work with a reconstruction graph where we added border points.

In the next section we will discuss some important properties concerning internal and border conditions.

4.4 Some important properties concerning internal and border conditions

In this section some theoretical results, concerning internal and border conditions, that we developed are presented. We have three key properties. The first one simply says that when we fix internal conditions on the optimal disparities so the formulation of the problem with these disparity conditions allows to find the global optimal solution. We will call this property «completion property» because it allows to complete a part of a global optimal solution. The second one is about the independence of subproblems separated by internal conditions and will be called «independence property». The third one deals with the comparison of two solution on the same region but with different border and internal conditions and will be called «monotonicity property». These properties will be of great importance for our further work.

4.4.1 Completion property

Let $A \subset \mathcal{D}^G$ be a subset of the graph domain and let $\widetilde{f}^G \in \mathcal{F}^G$ be an optimal graph disparity mapping. Furthermore we introduce an internal condition R^G on the set A that imposes that the disparity for any point in A must be on the optimal value, i.e. $\forall\, a \in A$ we have $R^G(a) = \{\widetilde{f}^G(a)\}$. So the conclusion of the completion property is the following: An optimal solution $f^G \in \mathcal{F}^G_{R^G}$ for this problem with reduced disparities is an

optimal solution for the problem without reduction.

The justification is straight forward. The graph disparity mapping $\widetilde{f}^G$ has minimal energy among all the mappings in $\mathcal{F}^G$. By the particular choice of the internal conditions we have furthermore $\widetilde{f}^G \in \mathcal{F}^G_{R^G} \subset \mathcal{F}^G$. Therefore $\widetilde{f}^G$ has also minimal energy among all graph disparity mappings in $\mathcal{F}^G_{R^G}$. As the problem with reduced disparities chooses a graph disparity mapping in $\mathcal{F}^G_{R^G}$ with minimal energy, we therefore have that the energy of f^G is equal to the energy of $\widetilde{f}^G$ and is therefore a globally optimal solution.
Note that if we only work with nearest optimal solutions so we must have $\widetilde{f}^G = f^G$.

4.4.2 Independence property

The independence property can be stated in the following way. When we solve a problem with internal conditions such that the graph domain contains two regions without conditions separated by the internal conditions, so we can get the optimal solution by solving these two regions independently with border conditions imposed by the previously internal conditions. Figure 4.7 shows such a constellation. Here we suppose to have internal conditions R^G on the set $C \subset \mathcal{D}^G$. The problem with these internal conditions can now be solved by solving two independent subproblems. One is the problem on the region $A \subset \mathcal{D}^G$ where we impose border conditions R^G on C (we simply use the previously internal conditions on C as border conditions). The second problem is the one on the region $B \subset \mathcal{D}^G$ with border conditions R^G on C.

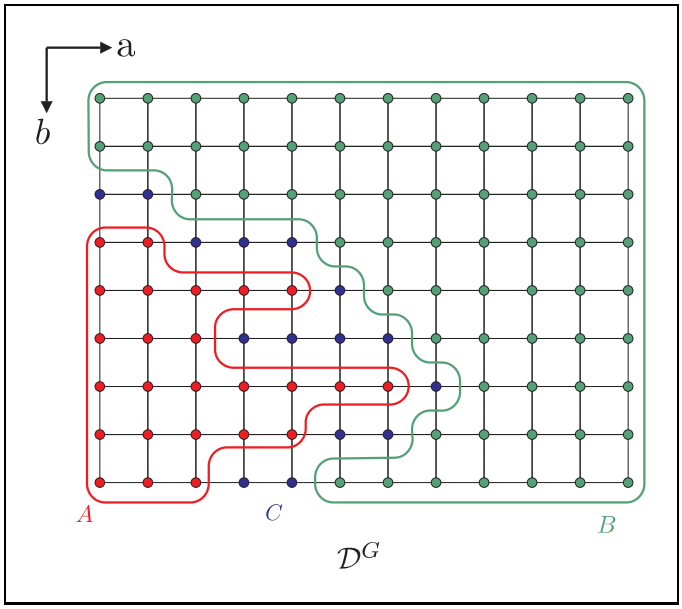

Figure 4.7: Having imposed internal conditions on the region $C \subset \mathcal{D}^G$ we can solve the non connected subproblems on $A \subset \mathcal{D}^G$ and $B \subset \mathcal{D}^G$ with border conditions on C.

The independence property is very intuitive. By regarding the general energy function we use, as formulated in 1.4, we see that energy is only attributed locally and thus distant regions do not have a direct influence to each other. A more mathematical way for justifying the independence property is to observe that when we construct the network G_{R^G} for solving the whole problem with internal conditions R^G on C, so we have the following situation. When we take the subgraph of G_{R^G} consisting of all vertices except the two terminals so this subgraph has two connected components, one that corresponds to the region A and the other to the region B. Therefore the minimum $s-t$ cut problem can be solved separately on these two regions.

4.4.3 Monotonicity property

The monotonicity property is about the comparison of two solutions on the same region but with different border and internal conditions. Suppose that we will solve a reconstruction problem on a subset of the graph domain $A \subset \mathcal{D}^G$ as illustrated in figure 4.8 where we impose two different conditions (internal and border). Once we impose conditions on a set $B_1 \subset \mathcal{D}^G$ and once on a set $B_2 \subset \mathcal{D}^G$ [(3)]. Let R_1^G and R_2^G be two disparity reductions on the sets B_1 respectively B_2 and we suppose that R_1^G *is higher than* R_2^G which is defined as follows:

Definition 4.4.1 ($R_1^G \geq R_2^G$)**.** *The reduction R_1^G on B_1 is higher than the reduction R_2^G on B_2 (and we write $R_1^G \geq R_2^G$) if for every $q \in B_1 \cup B_2$ we have:*

1) If $q \in B_1 \cap B_2$ then R_1^G and R_2^G impose conditions on q whereas the condition imposed by R_1^G is higher (in terms of disparities) than the one imposed by R_2^G, i.e.

$$R_1^G(q) = \{c_1(q)\} \text{ and } R_2^G(q) = \{c_2(q)\} \text{ with } c_1(q) \geq c_1(q)$$

2) If $q \in B_1 \setminus B_2$ then R_1^G imposes a condition on q where it sets its disparity as high as possible (i.e. on the disparity $n_c - 2$) and R_2^G imposes no condition, i.e.

$$R_1^G(q) = \{n_c - 2\} \text{ and } R_2^G \text{ imposes no condition on } q$$

3) If $q \in B_2 \setminus B_1$ then R_1^G imposes no condition on q and R_2^G imposes a condition on q where it sets its disparity as low as possible (to say on 0), i.e.

$$R_1^G \text{imposes no condition on } q \text{ and } R_2^G(q) = \{0\}$$

When having $R_1^G \geq R_2^G$ we also say that the condition R_1^G is nearer than R_2^G (in terms of depths).

[(3)] In our example B_1 and B_2 only contain points in A and adjacent to A. Conditions outside of this region have anyway no influence on the problem on A.

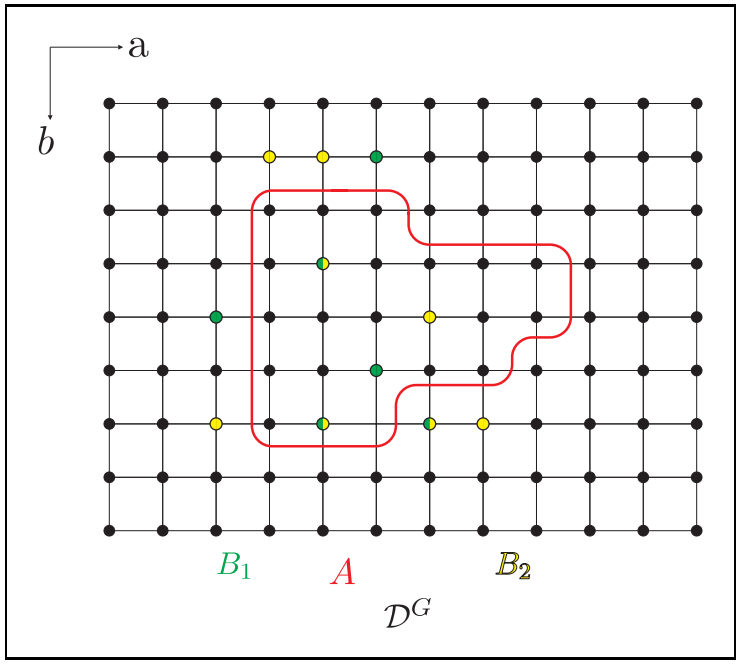

Figure 4.8: Possible constellation for the monotonicity property.

Let $\widetilde{f}_1^G \in \mathcal{F}^G_{R_1^G}$ be the nearest optimal graph disparity mapping for the reconstruction problem on A with conditions R_1^G on B_1 and $\widetilde{f}_2^G \in \mathcal{F}^G_{R_2^G}$ the nearest optimal graph disparity mapping for the reconstruction problem on A with conditions R_2^G on B_2. So we have the following property that we call *monotonicity property*:

Property 4.4.2 (Monotonicity property)**.** *Under the situation described above we have:*

$$\widetilde{f}_1^G(q) \geq \widetilde{f}_2^G(q) \quad \forall\, q \in A \tag{4.1}$$

I.e. the solution $\widetilde{f}_1^G$ is always nearer (i.e. higher in terms of disparities) than the solution $\widetilde{f}_2^G$. In words we can resume the monotonicity property by saying that nearer conditions give nearer solutions, which is very intuitive. By reasons of symmetry the monotonicity theorem is also valuable by reversing all inequalities (i.e. $\ll \leq \gg$ becomes $\ll \geq \gg$ and $\ll < \gg$ becomes $\ll > \gg$).

We will now give a proof of the monotonicity property.

Proof of the monotonicity property

Note that the conclusion of the monotonicity property is trivial when we have $A \setminus (B_1 \cup B_2) = \emptyset$. We therefore assume $A \setminus (B_1 \cup B_2) \neq \emptyset$. This proof contains two parts. In a first part of the proof we will show a weakened version of the inequality 4.1, namely

that we have

$$\exists\, \widehat{q} \in A \setminus (B_1 \cup B_2) \quad \text{with } \widetilde{f}_1^G(\widehat{q}) \geq \widetilde{f}_2^G(\widehat{q}) \tag{4.2}$$

We call this property *weak monotonicity*. In a second part we deduce the monotonicity property from the weak monotonicity property.

The weak monotonicity property will be proven by contradiction. So we suppose that

$$\forall\, q \in A \setminus (B_1 \cup B_2),\ \widetilde{f}_1^G(q) < \widetilde{f}_2^G(q) \tag{4.3}$$

In a first step we will show how we can reduce our problem to the case $A \cap B_1 = A \cap B_2$, i.e. R_1^G and R_2^G impose internal conditions on exactly the same set. We simply extend the disparity reduction R_1^G to the set $A \cap (B_1 \cup B_2)$ by imposing

$$R_1^G(q) = \{\widetilde{f}_1^G(q)\} \quad \forall\, q \in (A \cap B_2) \setminus B_1$$

By the completion property we know that the problem with the new reduction R_1^G on $B_1 \cup (B_2 \cap A)$ still gives $\widetilde{f}_1^G$ as nearest optimal solution. By doing the same changes with R_2^G we get into the case $A \cap B_1 = A \cap B_2$ and still violate the weak monotonicity. We therefore suppose this case and set $B = A \cap B_1 = A \cap B_2$.

To simplify notations we write $\mathbf{E}(\widetilde{f}_1^G)$ for the energy of the first solution with the conditions R_1^G (and $\mathbf{E}(\widetilde{f}_2^G)$ represents the energy for the second solution). The first energy can be divided into the following three parts (this division is also valuable analogously for the second energy):

1) The energy of the solution on the region $A \setminus B$ that we will denote by $\mathbf{E}(f_1^G\big|_{A\setminus B})$ [(4)].

2) The energy of the solution on the region B that we will denote by $\mathbf{E}(\widetilde{f}_1^G\big|_B)$.

3) The smoothness energy of the solution created between the region $A \setminus B$ and the condition R_1^G on B_1 that we will denote by $\mathbf{E}_s(\widetilde{f}_1^G\big|_{A\setminus B}, R_1^G)$.

With this notations we thus have

$$\mathbf{E}(\widetilde{f}_1^G) = \mathbf{E}(\widetilde{f}_1^G\big|_{A\setminus B}) + \mathbf{E}(\widetilde{f}_1^G\big|_B) + \mathbf{E}_s(\widetilde{f}_1^G\big|_{A\setminus B}, R_1^G)$$

We will now construct a new (not necessarily optimal) solution of the problem on A with conditions R_1^G by combining the first and the second solution. We use the first solution for the region B and the second solution for the region $A \setminus B$. We denote this solution by $[\widetilde{f}_1^G\big|_B, \widetilde{f}_2^G\big|_{A\setminus B}]$ and its energy with respect to R_1^G can be written as follows:

[(4)] $\widetilde{f}_1^G\big|_{A\setminus B}$ stands for the restriction of the domain of the function $\widetilde{f}_1^G$ to $A \setminus B$. We will use this notation in an analogue manner in other cases.

$$\mathbf{E}(\widetilde{f}_1^G|_B) + \mathbf{E}(\widetilde{f}_2^G|_{A\setminus B}) + \mathbf{E}_s(\widetilde{f}_2^G|_{A\setminus B}, R_1^G)$$ [5]

Because the solution $\widetilde{f}_1^G$ is optimal for the problem with conditions R_1^G we have that the energy $\mathbf{E}(\widetilde{f}_1^G)$ is less or equal than the energy of the combined solution. After simplification we get:

$$\mathbf{E}(\widetilde{f}_1^G|_{A\setminus B}) + \mathbf{E}_s(\widetilde{f}_1^G|_{A\setminus B}, R_1^G) \leq \mathbf{E}(\widetilde{f}_2^G|_{A\setminus B}) + \mathbf{E}_s(\widetilde{f}_2^G|_{A\setminus B}, R_1^G) \tag{4.4}$$

By interchanging the roles of the first and second solution we get, with the same procedure as before, the following equation:

$$\mathbf{E}(\widetilde{f}_2^G|_{A\setminus B}) + \mathbf{E}_s(\widetilde{f}_2^G|_{A\setminus B}, R_2^G) \leq \mathbf{E}(\widetilde{f}_1^G|_{A\setminus B}) + \mathbf{E}_s(\widetilde{f}_1^G|_{A\setminus B}, R_2^G) \tag{4.5}$$

We will now exploit the hypothesis that 4.2 is false and show that it implies the following inequality:

$$\mathbf{E}_s(\widetilde{f}_1^G|_{A\setminus B}, R_1^G) + \mathbf{E}_s(\widetilde{f}_2^G|_{A\setminus B}, R_2^G) \geq \mathbf{E}_s(\widetilde{f}_2^G|_{A\setminus B}, R_1^G) + \mathbf{E}_s(\widetilde{f}_1^G|_{A\setminus B}, R_2^G) \tag{4.6}$$

We prove this inequality by showing that for every pair of neighboring points r, q with $r \in A \setminus B$ and $q \in B_1 \cup B_2$ we have that the smoothness energy associated to them by the term on the left side of the inequality is bigger or equal to the smoothness energy associated to them by the right side of the above inequality. The inequality then follows by summing up over all such pairs of points r and q. We will treat three cases, namely $q \in B_1 \cap B_2$, $q \in B_1 \setminus B_2$ and $q \in B_2 \setminus B_1$.

1.case: $q \in B_1 \cap B_2$

Because $R_1^G \geq R_2^G$ we have that $R_1^G(q) = \{c_1(q)\}$, $R_2^G(q) = \{c_2(q)\}$ and $c_1(q) \geq c_2(q)$. In this case inequality 4.6 formulated on the points r and q transforms to

$$K^G|\widetilde{f}_1^G(r) - c_1(q)| + K^G|\widetilde{f}_2^G(r) - c_2(q)| \geq K^G|\widetilde{f}_2^G(r) - c_1(q)| + K^G|\widetilde{f}_1^G(r) - c_2(q)|$$

This inequality can be shown rather easily algebraically by using appropriately the two properties $\widetilde{f}_1^G(r) < \widetilde{f}_2^G(r)$ and $c_1(q) \geq c_2(q)$.

2.case: $q \in B_1 \setminus B_2$

By the definition of $R_1^G \geq R_2^G$ we have that $R_1^G(q) = \{n_c - 2\}$ and R_2^G imposes no condition on q. Because R_1^G and R_2^G impose internal conditions on the same set B, we now that R_1^G imposes a border condition on q. Therefore the two neighbors r and q do not contribute to the energies $\mathbf{E}_s(\widetilde{f}_1^G|_{A\setminus B}, R_2^G)$ and $\mathbf{E}_s(\widetilde{f}_2^G|_{A\setminus B}, R_2^G)$ because R_2^G has no border condition on q. In this case the inequality 4.6 formulated on the points r and q transforms to

$$K^G|\widetilde{f}_1^G(r) - (n_c - 2)| \geq K^G|\widetilde{f}_2^G(r) - (n_c - 2)|$$

[5] The term $\mathbf{E}_s(\widetilde{f}_2^G|_{A\setminus B}, R_1^G)$ represents the smoothness energy between the region $A \setminus B$ and the condition R_1^G on B_1 of the combined solution.

By using the facts that $\widetilde{f}_1^G(r) \leq n_c - 2$ and $\widetilde{f}_2^G(r) \leq n_c - 2$ (because $n_c - 2$ is the highest disparity) as well as $\widetilde{f}_1^G(r) < \widetilde{f}_2^G(r)$ we see that the previous inequality is true.

3.case: $q \in B_2 \setminus B_1$

This case can be handled analogue to the case 2.

By summing over all neighbors $r \in A \setminus B$ and $q \in B_1 \cup B_2$ we have thus proved 4.6.

Furthermore by summing the two inequalities 4.4 and 4.5 and simplifying we get

$$\mathbf{E}_s(\widetilde{f}_1^G\big|_{A\setminus B}, R_1^G) + \mathbf{E}_s(\widetilde{f}_2^G\big|_{A\setminus B}, R_2^G) \geq \mathbf{E}_s(\widetilde{f}_2^G\big|_{A\setminus B}, R_1^G) + \mathbf{E}_s(\widetilde{f}_1^G\big|_{A\setminus B}, R_2^G) \tag{4.7}$$

This is exactly the inverse inequality of 4.6. We thus must have equality in 4.7. Furthermore we must have equalities in 4.4 and 4.5 as a strict inequality in one of these equations would provoke a strict inequality in 4.7 which is impossible as we just observed that we must have equality there.
Having equality in 4.4 means that the combined disparity mapping $[\widetilde{f}_1^G\big|_B, \widetilde{f}_2^G\big|_{A\setminus B}]$ (which respects R_1^G) has the same energy with respect to R_1^G as $\widetilde{f}_1^G$ and is therefore an optimal solution. Because we assumed that the weak monotonicity property does not hold we have that the combined disparity mapping is everywhere as near (i.e. has higher disparities) as $\widetilde{f}_1^G$ and on at least one point (the point $\widehat{q}$ in 4.3) even nearer. This contradicts the assumption that $\widetilde{f}_1^G$ is the nearest optimal solution for the problem with conditions R_1^G and finally proofs the weak monotonicity property.

We will now deduce the monotonicity property from the weak monotonicity property. Again we prove by contradiction and thus suppose that the monotonicity property is false. Therefore $\exists\, \widehat{q} \in A$ with $\widetilde{f}_1^G(\widehat{q}) < f_2^G(\widehat{q})$. Because R_1^G is higher than R_2^G, we must have $\widehat{q} \in A \setminus (B_1 \cup B_2)$. We now define terms that will allow us to apply the weak monotonicity property.
We define A' as the set of all points in $A \setminus (B_1 \cup B_2)$ where we violate the monotonicity property, i.e.

$$A' = \{q \in A \setminus (B_1 \cup B_2) \mid \widetilde{f}_1^G(q) < \widetilde{f}_2^G(q)\}$$

Furthermore we define two reductions $R_1'^G$ and $R_2'^G$ that are extension of R_1^G and R_2^G to the sets $B_1 \cup (A \setminus A')$ and $B_2 \cup (A \setminus A')$, in the following way:

$$\begin{aligned} R_1'^G(q) &= \{\widetilde{f}_1^G(q)\} \quad \forall\, q \in B_1 \cup (A \setminus A') \\ R_2'^G(q) &= \{\widetilde{f}_2^G(q)\} \quad \forall\, q \in B_2 \cup (A \setminus A') \end{aligned}$$

Because we extend the two reductions on points that are not in A', it is easy to verify that we still have $R_1'^G \geq R_2'^G$. Furthermore by the completion property we know that the problems with reductions on $R_1'^G$ respectively $R_2'^G$ have $\widetilde{f}_1^G$ and $\widetilde{f}_2^G$ as nearest optimal solutions. Additionally the set A' is not empty as $\widehat{q} \in A'$.

Thus the problems with the solutions $\widetilde{f}_1^G$ and $\widetilde{f}_2^G$ of the problems with the two reductions $R_1'^G$ respectively $R_2'^G$ violate the weak monotonicity property and give a contradiction. □

We will now discuss some applications of the above introduced properties.

4.4.4 Application: Error characterization of subproblems

When having to solve a large reconstruction problem on a graph domain $\mathcal{D}^G$ it is often interesting to solve different subproblems on subsets of the graph domain. We define a subset of graph domain $A \subset \mathcal{D}^G$ for the resolution of the subproblem as shown in figure 4.9. Furthermore as shown in the figure, the set ∂A contains the points in $\mathcal{D}^G \setminus A$ adjacent to points in A. We call these points the outer border of A [(6)]. Let $\widetilde{f}^G$ the nearest optimal solution of the reconstruction problem over the whole graph domain and let $\widetilde{f}_A^G$ be the nearest optimal solution of the reconstruction problem over A with arbitrary border conditions on ∂A (it is also allowed to set no border conditions or only on some points of ∂A). Let $A_1 \subset A$ be the points where the two solutions coincide and A_2 the set where they do not, i.e.

$$A_1 = \{q \in A \mid \widetilde{f}^G(q) = \widetilde{f}_A^G(q)\}$$
$$A_2 = \{q \in A \mid \widetilde{f}^G(q) \neq \widetilde{f}_A^G(q)\} = A \setminus A_1$$

As we usually try to determine the optimal solution of the problem over the whole graph domain, we say that the reconstruction of the subproblem is erroneous on the set A_2. We have the following characterization of the set A_2.

Property 4.4.3 (Error characterization of subproblems)**.** *For every point of the graph domain $q \in A_2$ that was reconstructed erroneous by the subproblem we have that there exists a path on the domain grid from q to a point of ∂A that only passes through points in $A_2 \cup \partial A$.*

This property can be resumed by saying that every error of a subproblem originates from the outer border of the region over which we solve the problem.
This property is equivalent to say that we cannot have an island of errors $A_2' \subset A_2$ surrounded by points $\partial A_2' = A_1' \subset A_1$ where we have correctly reconstructed as illustrated in figure 4.10.
We will justify the error characterization by contradiction. So we suppose having an island of errors as illustrated in figure 4.10. Let R^G be internal conditions who set the disparities in A_1' to the disparities found by solving the subproblem (which are the same on the set $A_1' \subset A_1$ as the disparities found by solving the whole problem). By the completion property we know that the nearest solution of the subproblem with reduction R^G is the same as without the reduction. In addition, by the independence property the part on A_1' of the subproblem with reduction on R^G can be solved independently of

[(6)] In general for any set $B \subset \mathcal{D}^G$ we use ∂B for the outer border of B, i.e. ∂B is the set of all points in $\mathcal{D}^G \setminus B$ adjacent to points in B.

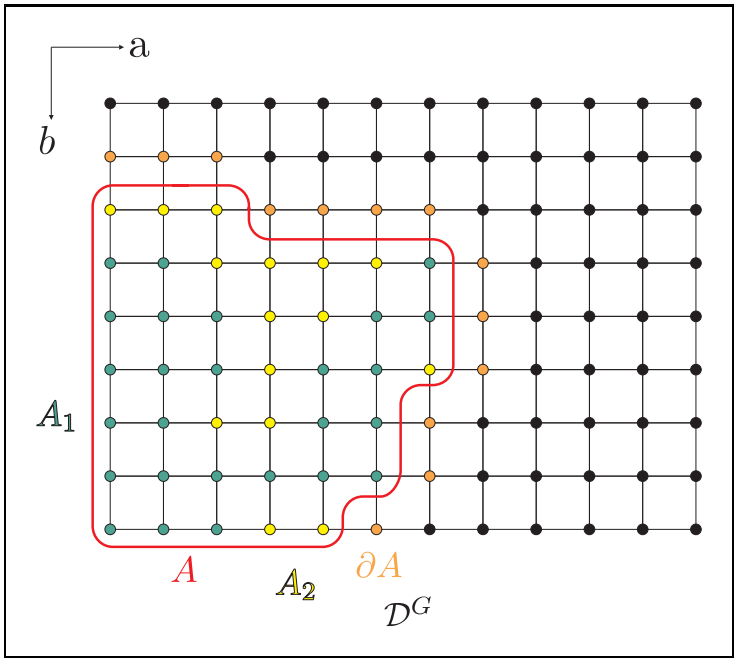

Figure 4.9: Example for showing the type of errors by solving a subproblem over $A \subset \mathcal{D}^G$. The set A_1 is the set of all the points where the nearest solution of the subproblem is the same as the nearest solution of the problem over $\mathcal{D}^G$. A_2 is the set of all the points where the two solutions differ, i.e. the points where we have not achieved the global optimal soluton by solving the subproblem. We have the property that for every point $q \in A_2$ we can find a path on the grid consisting entirely of points in $A_2 \cup \partial A$ that goes from q to a point of ∂A.

the rest. But as R^G sets the disparities of A'_1 on the global optimal solution, the completion property implies that the nearest optimal solution on A'_2 with border conditions R^G must be the same as the global optimal solution. Which is a contradiction as we supposed that the points in A'_2 were reconstructed erroneously by the subproblem.

One can show that the conclusion of the error characterization property is still valid if we set

$$A_1 = \{q \in A \mid \widetilde{f}^G(q) \leq \widetilde{f}^G_A(q)\}$$
$$A_2 = \{q \in A \mid \widetilde{f}^G(q) > \widetilde{f}^G_A(q)\} = A \setminus A_1$$

or

$$A_1 = \{q \in A \mid \widetilde{f}^G(q) < \widetilde{f}^G_A(q)\}$$
$$A_2 = \{q \in A \mid \widetilde{f}^G(q) \geq \widetilde{f}^G_A(q)\} = A \setminus A_1$$

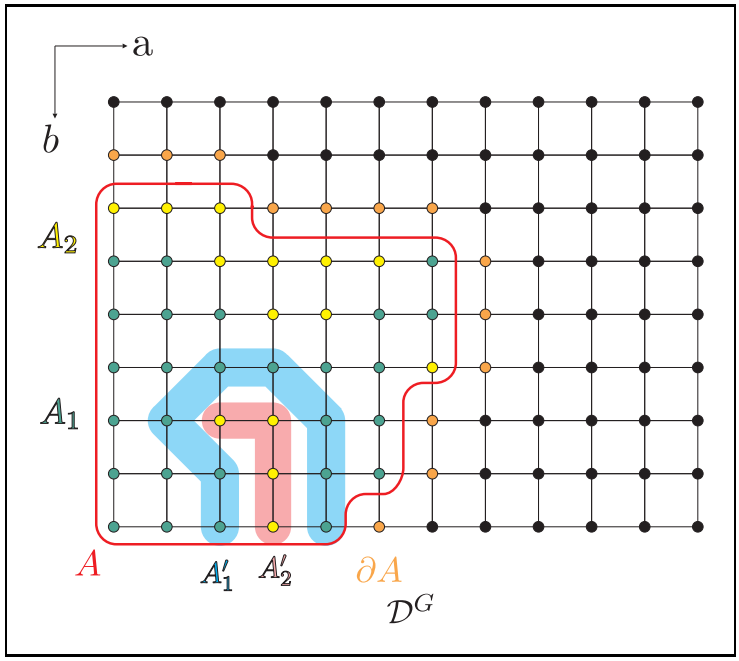

Figure 4.10: Errors as illustrated in this figure are impossible as there is a erroneous subregion A'_2 surrounded by the set A'_1 which is a subset of A_1 and was therefore reconstructed correctly. We call this impossible constellation an *island of errors*.

and also when we change the inequalities above. This can be justified in exactly the same way but by using the monotonicity property instead of the completion property.

4.4.5 Application: Constructing locally upper and lower bounds for the nearest optimal solution

We will now present a simple technique for constructing locally upper and lower bounds for the nearest optimal solution. Let $\widetilde{f}^G$ be the nearest optimal reconstruction solution on the entire graph domain $\mathcal{D}^G$ and let $A \subset \mathcal{D}^G$ be a subset of the graph domain on which we want to construct an upper bound (constructing a lower bound is analog) for $\widetilde{f}^G$. Let R^G be the following border condition on ∂A:

$$R^G(q) = \{n_c - 2\} \quad \forall\, q \in \partial A$$

To say R^G is the highest possible border condition on ∂A. Let $\widetilde{f}^G_A$ be the solution of the problem on A with border conditions R^G. So $\widetilde{f}^G_A$ is an upper bound on the optimal solution.

This result can be justified in the following way. Let $\widetilde{R}^G$ be the border condition on ∂A that sets its disparities on the optimal solution, i.e.

$$\widetilde{R}^G(q) = \{\widetilde{f}^G(q)\} \quad \forall\, q \in \partial A$$

By the completion property we know that the nearest optimal solution of the subproblem on A with border conditions $\widetilde{R}^G$ is $\widetilde{f}^G\big|_A$. As the border condition R^G is higher than $\widetilde{R}^G$, we thus have by the monotonicity property that $\widetilde{f}^G_A$ will be higher than $\widetilde{f}^G$, i.e.

$$\widetilde{f}^G_A(q) \geq \widetilde{f}^G(q) \quad \forall\, q \in A$$

Therefore $\widetilde{f}^G_A$ is effectively an upper bound of $\widetilde{f}^G$.
We can find in the same way a lower bound on A by setting the border condition R^G on ∂A as low as possible (i.e. $R^G(q) = 0\ \forall\, q \in \partial A$). This method is particularly interesting because we do not have to work on the whole network to get a lower or upper bound of the nearest optimal solution. Additionally, tests we have performed, showed that bounds of this type are often very accurate. Figure 4.11 shows a typical example of our tests. The error images show the accuracy of the upper bound on a subregion of the graph domain of 100x100 points. On the binary error map we see that on a large zone, the upper bound is even on the optimal solution.

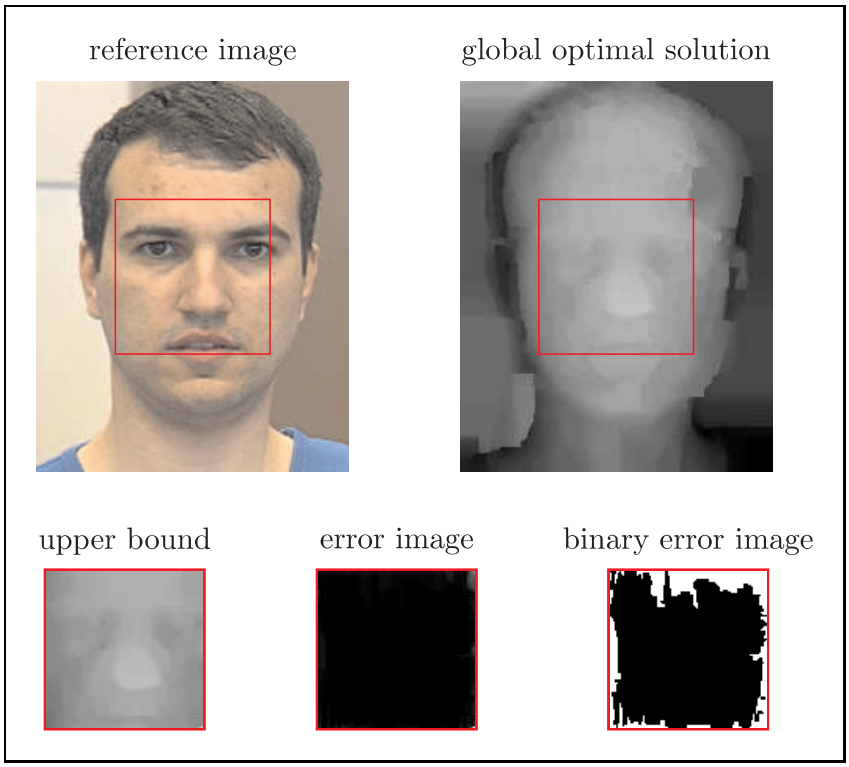

Figure 4.11: An empirical test for determining the quality of an upper bound. The red square on the reference image indicates the region (100x100 points) on which we construct an upper bound with the introduced technique. The error image shows a representation of the errors done from black ($\widehat{=}$ no error) to white ($\widehat{=}$ maximal error made of 36 disparities). The reconstruction was done over 100 disparities, i.e. $n_c = 101$.

4.5 Application: Pyramidal approach

In this section we will discuss an interesting application of disparity reduction, namely a pyramidal approach for solving approximately a reconstruction problem. The main idea of a pyramidal approach is to begin by solving a relatively oversimplified version of the original problem and to use the solution found there for creating a new problem that formulates the original problem more accurately and so one and so forth. In this way we never have to work on the whole graph and are thus capable to solve much bigger reconstruction problems as with a direct method. We apply the pyramidal idea in the following way.

We begin by solving the original reconstruction problem with a coarse discretization. The solution found by solving this problem will be used to determine a region in which we are trying to improve the solution with a finer discretization. This region will be chosen as a band that surrounds the previous solution. We then pass to the next iteration where we solve a reconstruction problem restricted on this band but with a finer discretization. This procedure can be applied iteratively. On the last iteration we solve a problem on a band in which we have the same discretization as the original problem. The solution found in this problem is the final solution of the pyramidal algorithm. Figure 4.12 shows an example of this pyramidal approach.

The article [11] was the first one that developed the pyramidal approach for the reconstruction problem formulated by minimum $s-t$ cuts. In that paper the above mentioned pyramidal approach is combined by an iterative adaption of the graph domain to be able to handle large problems.
Unfortunately, this adaption provokes reconstruction errors. Additionally the pyramidal algorithms in that paper seem not to apply a correct disparity reduction formulation and therefore do not handle the problem of missing smoothness arcs as described in figure 4.2. Our pyramidal algorithm does not suffers from these problems. The drawback of not adapting the graph domain is that we cannot solve problems with very large dimensions in a and b. The approach we have chosen to handle this problem is to break the original problem into several subproblems on different regions of the graph domain, that can be solved independently (we therefore can solve them simultaneously on several computers). We will discuss this technique in details in the chapter about parallelization.

We will now discuss some details concerning our implementation of the above proposed pyramidal algorithm.

4.5.1 Some details on the implementation

An implementation of the above proposed pyramidal algorithm can be found in our code. We have introduced two parameters λ, n'_c that allow to adjust the algorithm.

The parameter n'_c fixes the maximum number of disparities per point in the graph domain

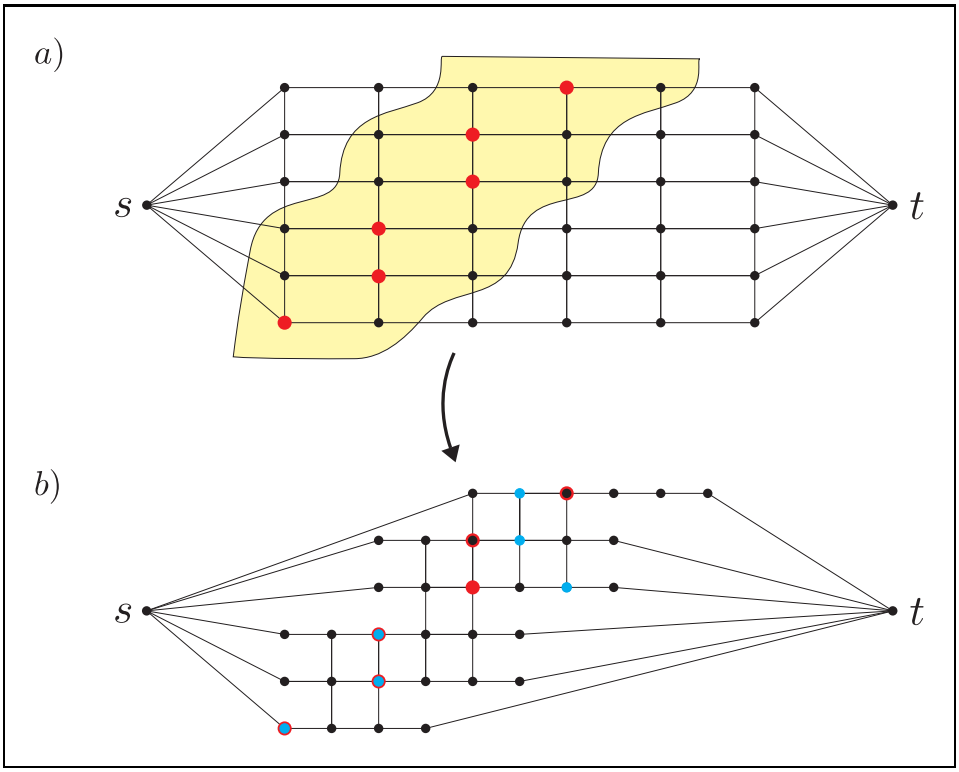

Figure 4.12: An example of the explained pyramidal approach with two iterations. In a first step we solve the initial problem with a coarse discretization of the disparities as illustrated in a) (the red points represent the nearest optimal solution found). We then fix a band surrounding the optimal solution and pass to the next iteration where we solve a problem with the original discretization of the disparities on this band as illustrated in b). The blue points represent the nearest optimal solution found in this problem.

that we consider in the reduced problems. Or in other words, the maximum number of vertices in the intervals for the disparity reduction. In example 4.12, we have $n'_c = 6$.
The second parameter λ allows to determine how the width of the band changes from one iteration to the next. For example $\lambda = 3$ means that the width of the band in the next iteration is a third of the current width of the band. So λ can be seen as a sort of «zooming-factor». In the example 4.12, we have $\lambda = 2$.
Because the number of disparities per interval is constant on each iteration, namely n'_c, we have also that λ indicates the refinement of the disparity discretization. To say $\lambda = 2$ means that by passing from one iteration to the next one, the distance between consecutive two vertices on the same disparity line halves.
When implementing our pyramidal algorithm, we paid attention that the vertices in the intermediate networks are always a subset of the vertices in the original network. This has the advantage that every intermediate solution of the algorithm can be used as an

approximate solution of the original problem. On the other hand this restriction implies that the parameter λ has to be a natural number ≥ 2, i.e. $\lambda \in \mathbb{N} \setminus \{1\}$.

We have chosen these two parameters for the adjustment of our pyramidal algorithm because of the following reasons. The parameter n'_c forces that in all iterations we never have more that $n_a \cdot n_b \cdot n'_c$ vertices. This allows to handle memory problems because we can fix n'_c on a value such that we are sure that we can handle a graph of $n_a \cdot n_b \cdot n'_c$ vertices with the available virtual memory, and we are sure that we can apply our pyramidal algorithm without memory problems. On the same time, thanks to the parameter n'_c we ensure that all the problems appearing during the pyramidal algorithm have approximately the same size.

The parameter λ allows to determine which reconstruction errors done in one iteration can still be corrected on the subsequent iterations. To say if $\lambda = 2$, so reconstructed points must have a distance to the optimal solution of more then half of the actual band width to provoke nonreversible errors in the subsequent iteration, because the next bands do not contain anymore the optimal solution. When having a problem where reconstruction is rather easy so we can choose a big λ. In the case of very sensible problems, a small λ is adequate.

In our code we use the variable `max_c_resolution` for the parameter n'_c (respectively `max_z_resolution` in the .ini-file) and `stretchfactor` for the parameter λ.

The presented pyramidal approach is also very useful to limit the number of consistency capacities that have to be calculated. Limiting these calculations may be crucial as many interesting energy functions use very time expensive methods for calculating consistency capacities. Sometimes preliminary optimization problems have to be solved for determining the consistency capacities. To be able to calculate the number of consistency capacities that have to be calculated in our pyramidal approach, we first have to evaluate the number of iterations N done by the pyramidal algorithm in function of the two parameters n'_c and λ.

For determining the number of iterations N of the pyramidal algorithm, we will observe the iterations in inverse order and determine at each step, how many disparities per disparity line of the original graph are covered by the current band. The iteration 1 will then be the first one for which the band covers all the $n_c - 1$ disparities per disparity line.

At the last iteration (i.e. iteration number N) the disparity discretization of the current problem and the original problem are the same. Additionally, as we observe $n'_c - 1$ disparities per disparity line in this iteration, we also observe $n'_c - 1$ points per disparity line of the original graph. In the previous iteration (iteration $N-1$) the band was by a factor of λ bigger. We therefore observed $\lambda(n'_c - 1)$ points per disparity line of the original graph. By repeating this argument we will cover at the first iteration $\lambda^{(N-1)}(n'_c - 1)$ disparities per disparity line of the original graph. As the first iteration has to cover all

the disparities of the original graph, we must have

$$\lambda^{(N-1)}(n'_c - 1) \geq n_c - 1 \tag{4.8}$$

So as explained above, N is the first natural number satisfying 4.8. Therefore N can be written in the following way:

$$N = \left\lceil \frac{\log(n_c - 1) - \log(n'_c - 1)}{\log(\lambda)} \right\rceil + 1 \quad {}^{(7)} \tag{4.9}$$

We can now calculate the number of consistency capacities that have to be calculated during the pyramidal algorithm (where we suppose that at each iteration we calculate all the necessary capacities from scratch on).
At each iteration of the pyramidal algorithm we have $n_a n_b n'_c$ vertices in the network. We therefore have to calculate $n_a n_b (n'_c - 1)$ consistency capacities per iteration. So during the whole pyramidal algorithm we calculate $n_a n_b (n'_c - 1)N$ capacities. When solving the original problem directly we have to calculate $n_a n_b (n_c - 1)$ consistency capacities. Therefore the ratio between the calculations in the original network and the ones in the pyramidal approach is:

$$\begin{gathered}(n_c - 1) : ((n'_c - 1)N) = \\ (n_c - 1) : \left((n'_c - 1)(\left\lceil \frac{\log(n_c - 1) - \log(n'_c - 1)}{\log(\lambda)} \right\rceil + 1) \right)\end{gathered} \tag{4.10}$$

We will now illustrate with a practical example how one can profit from the pyramidal approach for calculating less consistency capacities. Suppose we want to solve a problem with 200 disparity steps, i.e. $n_c = 201$. We fix the two parameters on $n'_c = 51$ and $\lambda = 4$ [(8)]. By equation 4.9, the pyramidal algorithm takes two iterations, i.e. $N = 2$. Equation 4.10 allows to determine the ratio between the consistency calculations by the direct approach and the pyramidal approach which is

$$200 : 100$$

To say, we will calculate exactly the half of consistency capacities with the pyramidal approach. Note that this ratio may be even considerably bigger when working with a greater n_c and λ.

Note also that the computational time for solving the flow problems that appear in a pyramidal approach is usually inferior to the computational time for solving the original flow problem directly, thanks to the relatively little size of the flow problems in the pyramidal algorithm. Using a pyramidal approach, we thus reduce the complexity of the problem.

[(7)] For any real number $x \in \mathbb{R}$ we write $\lceil x \rceil$ for rounding up the value of x to the next number in $\mathbb{Z}$.
[(8)] This is a very reasonable choice. We have solved some problems with exactly these parameters and usually got results with no significant reconstruction errors.

5 Parallelization

In this chapter we introduce algorithms that can be parallelized for solving large scale problems that are intractable with the classical minimum $s-t$ cut formulation. We will typically divide the problem into different subproblems, that can be solved independently, and thus simultaneously on different computers. Having solved the subproblems, we will merge them together to get a solution of the original problem. Most of the proposed methods are algorithms that find accurate approximations of the optimal solution and have even good chances to find the global optimal solution. In particular, we propose methods that allow to give an upper bound of the error made. Additionally, we propose an exact parallelizable algorithm for finding the global optimal solution.

5.1 Introduction

One of the major drawbacks of the minimum $s-t$ cut formulation we use for solving the stereo problem, is that we rapidly get huge reconstruction networks. For example when we want to calculate a disparity map for an image of size 2048×1536 (almost all common digital cameras have at least such a resolution) with a disparity range of $n_c = 300$ we get a reconstruction network with about one billion vertices. When we want to solve such large scale problems, the following principal problems appear:

- Memory problems
- Very high computational time for solving the flow problem
- Very high computational time for calculating the consistency capacities

The pyramidal approach is one method for handling these problems, but unfortunately it is not entirely satisfactory for very big instances as we have to choose a very little n'_c to handle such problems, what provokes significant errors. Also the computational time is still very high by solving huge problems with the pyramidal approach.

That is why we introduce in this chapter algorithms that solve approximately big problems by combining the pyramidal approach with the idea of parallelization. We will cut the original problem into different subproblems that we can solve independently. This technique allows us to handle the three mentioned problems above.

To simplify the explanations we will begin with the case where we divide the original problem into two subproblems. This will be done by defining two overlapping regions $A, B \subset \mathcal{D}^G$ that cover the graph domain as illustrated in figure 5.1. We will then solve

the two subproblems over A and B. We denote by $\widetilde{f}_A^G$ the nearest optimal solution on the region A and by $\widetilde{f}_B^G$ the nearest optimal solution over the region B. Finally we will merge the solution in the following way. On the part $A \setminus B$ we will use $\widetilde{f}_A^G$, and on part $B \setminus A$ we will use $\widetilde{f}_B^G$. Now we have to specify how we construct the missing part of the solution on the middle region $A \cap B$. This can be done in various ways. We will propose and analyze different methods for merging the middle region later. This method allows to diminish the a and b dimensions of the subproblems. For handling the c dimension we can use the proposed pyramidal approach when solving the subproblems.

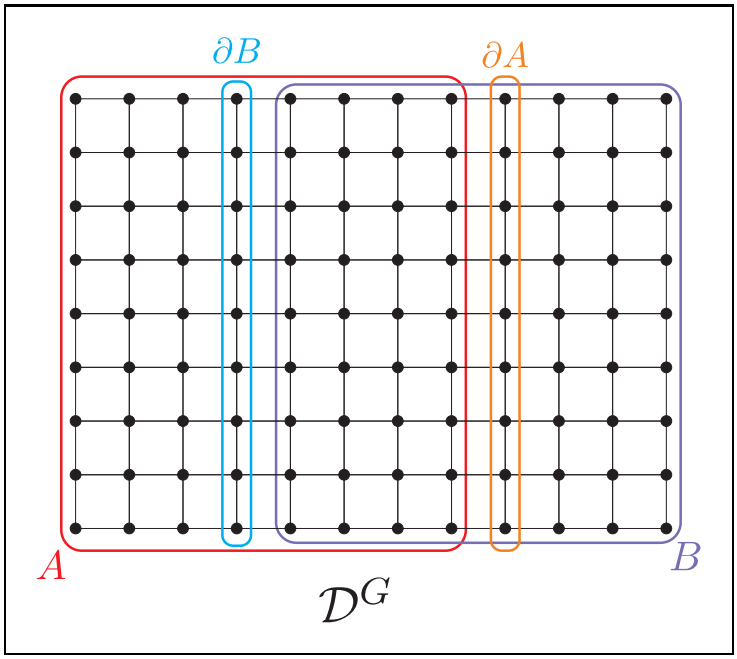

Figure 5.1: We define two subproblem on the graph domain, one over A and one over B. We will then construct a global solution by taking on the set $A \setminus B$ the solution found by solving the problem over A and analogously we use the solution found by solving the problem over B for the part $B \setminus A$. We discuss later how we complete this solution on the middle region $A \cap B$.

This idea of parallelization is motivated by the following observation. When solving the subproblem over the region A (the same discussion is valuable for the region B) so we know by the error characterization property that all the errors we do in the reconstruction must originate from the outer border ∂A. But in practice the influence of this border is only limited. This comes from the fact that a local region of the object we reconstruct does not contain much information about another remote region. Therefore the errors provoked by ∂A normally do not have a big influence on the region $A \setminus B$, and often they even do not reach this region (in such a case we even found the optimal solution on $A \setminus B$).

The next section gives empirical results allowing to quantify the reconstruction errors on the set A and therefore allowing to determine a reasonable width of the overlapping region to limit the errors made on $A \setminus B$ and $B \setminus A$.

5.2 Error analysis for determining the sets A and B

For analyzing errors in subproblems we first solve a subproblem on a region $A \subset \mathcal{D}^G$ as illustrated in figure 5.1 (we denote the nearest solution of this problem by $\widetilde{f}_A^G$). We then compare this solution to the nearest global optimal solution $\widetilde{f}^G$. The error will be measured in difference of disparities on the graph, i.e. the error made on a point $q \in A$ is $|\widetilde{f}_A^G(q) - \widetilde{f}^G(q)|$.
We made tests on several frameworks for analyzing these errors. They were all very similar. We therefore discuss in this section examples with a typical framework. Figure 5.2 a) shows the reference image, with a resolution of 271 x 181, of a face that we will reconstruct and b) is a representation of the nearest optimal solution when solving the problem directly on hundred disparity steps (i.e. $n_c = 101$).

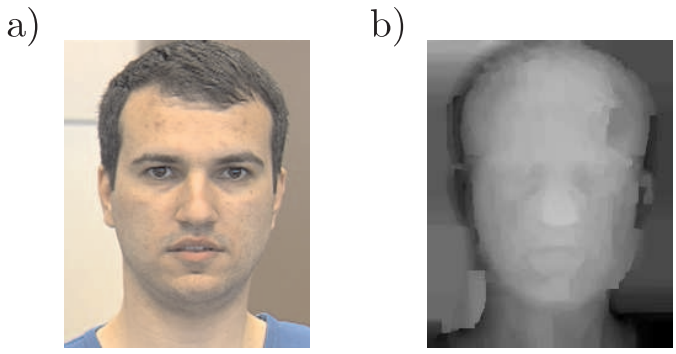

Figure 5.2: **a)** Reference image of our example. **b)** Disparity map obtained by solving the whole problem directly.

We will now discuss the results when reconstructing only the first k columns of the graph domain for $k \in \{150, 125, 100, 75, 50\}$. Figure 5.3 shows in the first line the reconstruction results as disparity maps. Already here, we see that the errors made on the right border are very limited. On the second line we give a binary error map where each graph point with a difference to the global optimal solution is drawn in white. As predicted by the error characterization property, all errors originate from the right border. Furthermore the influence of the error is fairly locally. The third and forth line of figure 5.3 give a quantification of the errors made. The third line shows in grayscales the amplitude of the errors, where black corresponds to no error and white to the maximum error made which is indicated in the fourth line.

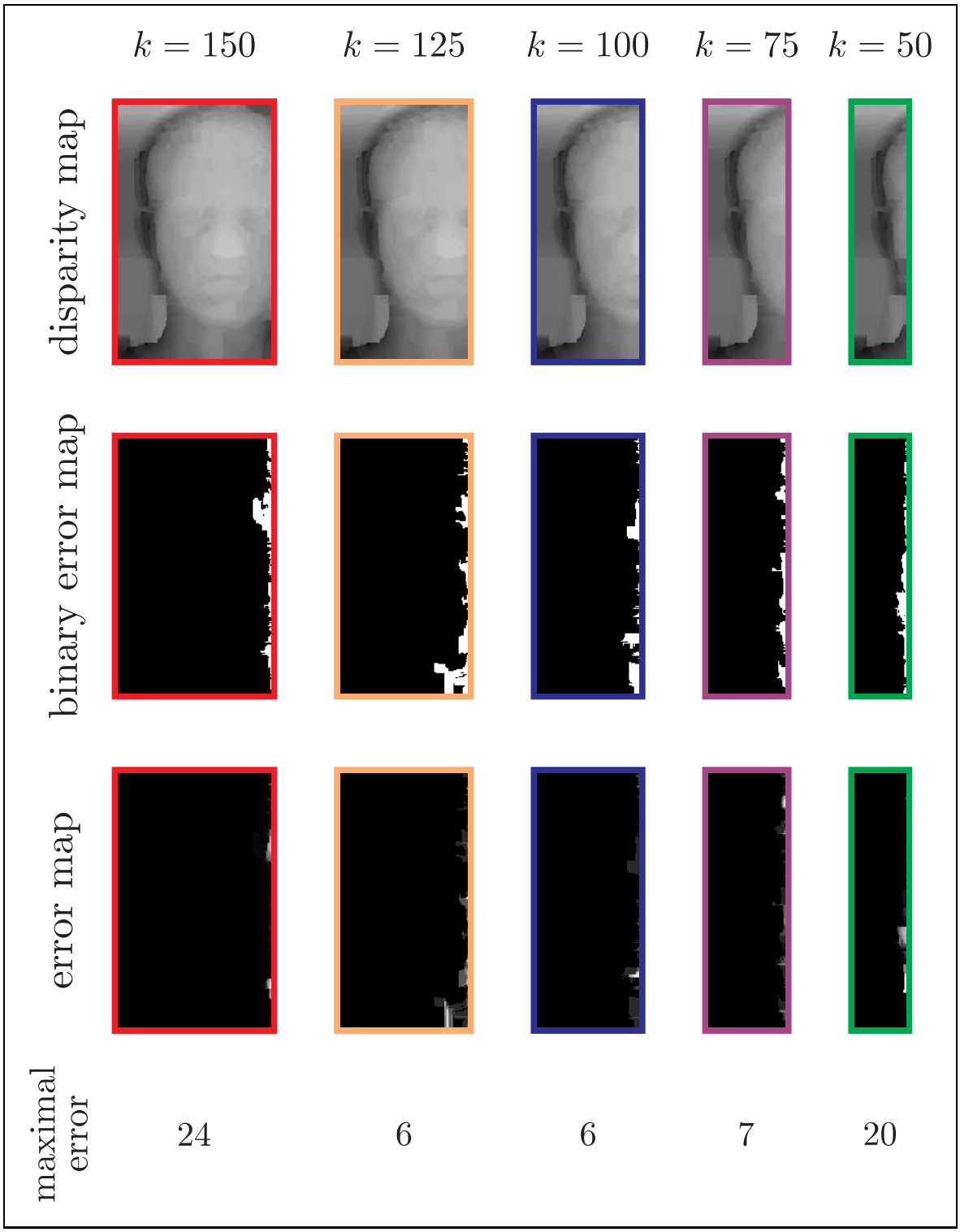

Figure 5.3: A testserie for errors on subproblems.

All the five illustrated subproblems can be seen as possible choices for the left subproblem for our parallelization idea (i.e. this corresponds to the problem over region A in figure 5.1). We will concentrate on the question how large to choose the overlapping region between the sets A and B as shown in figure 5.1 to limit the errors made in $A \setminus B$ (respectively $B \setminus A$).
Therefore we analyze the errors left when cutting some columns on the right side of our subproblems in figure 5.3. The region left after cutting some columns would correspond to $A \setminus B$. Therefore the number of columns cut corresponds to the width of the overlapping region $A \cap B$.
One method to observe the quality of the remaining region after cutting is the maximum error in this region. Figure 5.4 shows a diagram where we have plotted the remaining maximum error of the different subproblems in function of the columns we cut on the right side. We see that already by cutting 8 columns, we have in none of the five subproblems errors superior to 5 disparity values (which corresponds to an error of 5% as we work with 100 disparities). By comparing the disparity maps of the subproblems with the original disparity map, one can see that errors of this amplitude do not create large, disturbing discontinuities. Furthermore these remaining errors are typically on very local regions. It is also interesting to note that only one problem has errors that are more than 20 points away from the right border.

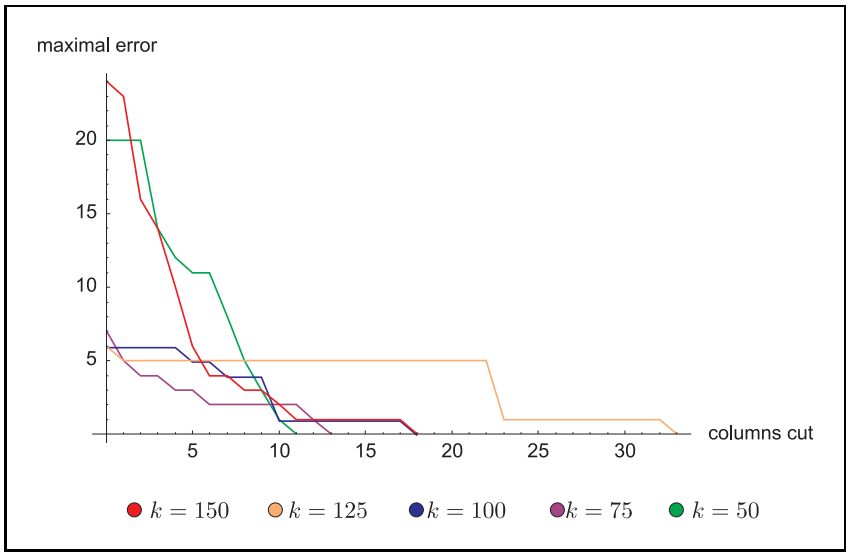

Figure 5.4: Diagram representing the maximal reconstruction errors we still have when cutting columns on the right side.

Another method for discussing the remaining error when cutting on the right side, is to analyze the nature of the sum of the remaining errors. Figure 5.5 shows a diagram where this values are represented. This diagram shows that errors decrease in general very fast when cutting columns on the right side. In such a diagram the nature of the curves (how they decrease) seems to be more interesting than the effective values, as

these values depend on parameters like the image size and the disparity resolution.

Resuming, in most cases cutting between 10 and 20 columns yields already very good results in practice. Going under 10 columns may be dangerous and can result in significant discontinuities depending on the particular case. Therefore the overlapping region in our parallelization approach should contain in general at least 10 columns. Evidently, reconstruction errors typically occur in regions that are difficult to reconstruct. To diminish the damage when cutting at such a region we think that an overlapping width of 20 columns would be adequate. Note that we determined the overlapping width in function of the reconstruction quality on the sets $A \setminus B$ and $B \setminus A$. Depending on how we complete the solution in the middle region $A \cap B$ it may be necessary to adapt this width. Techniques for completion on $A \cap B$ as well as their dependence on the overlapping width will be discussed afterwards. Before passing to this topic we will shortly discuss in the next subsection a method for diminishing the overlapping width.

Finally, note that we have to calculate more consistency capacities in our parallel approach as there exist overlapping regions. This can be easily compensated by using our pyramidal approach for solving the subproblems over A and B. In this way, even less consistency capacities have to be calculated in the parallel approach as by solving the problem with the classical $s-t$ cut formulation directly.

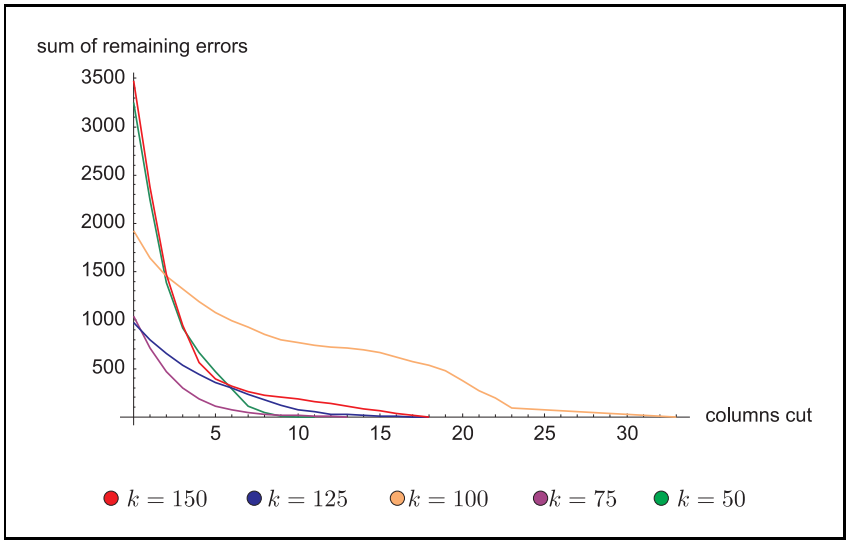

Figure 5.5: Diagram representing the sum of the remaining disparity errors we still have when cutting columns on the right side.

5.2.1 Diminishing the overlapping width through intelligent image-cutting

The risk of obtaining large reconstruction errors in a subproblem depends directly on the region where we have cut the image to get the subproblem. Therefore it may be

interesting to cut the image in a region that is easy to reconstruct, instead of blindly cutting on a vertical line. This would allow us to diminish the overlapping width. Regions that are comfortable to reconstruct satisfy typically the following characteristics:

- very textured
- almost no light reflections
- visible from almost all cameras

The first two properties can be determined by image processing techniques. The last one is more difficult to exploit. An interesting idea is to study the capacities of some consistency arcs over the region to analyze, that are well distributed over the disparity range. If some of these consistency arcs have relatively little capacities, so it is probable that the correspondent region is rather easy to reconstruct.
Having once determined a measure of reconstruction difficulty for the graph domain, we can try to cut the graph domain in two parts such that the cut passes through regions with good reconstruction measure. Here formulations using shortest paths could be very interesting.

5.3 Some completion methods

In this section we discuss different possibilities to complete the solution on the region $A \cap B$. The quality of a completion method can be measured by the following factors:

1. Little energy value of the merged solution
2. No large discontinuities created
3. Even with a little width of the overlapping region we obtain good results
4. Speed of the completion method

The first point is a measure of global quality of the solution, whereas the second point measures the local quality of reconstruction. Point three and four are important to limit the computational time. Especially when working with complex consistency capacities, it may be of importance to limit the width of the overlapping region.

We will discuss the following five methods for completion on $A \cap B$:

a) Simple completion through straight division of the region $A \cap B$

b) Completion by solution merging

c) Completion through merging path

d) Energy-optimal completion

e) Variation of the energy-optimal completion

Simple completion through straight division of the region $A \cap B$

One of the most simplest methods for completion is to divide the region $A \cap B$ vertically into two regions as illustrated in figure 5.6. On the left side of the division line we use the solution $\widetilde{f}_A^G$ and on the right side $\widetilde{f}_B^G$.
This method has the advantage that it is extremely simple and fast. Unfortunately, we risk to get large discontinuities at the line where we merged. To cover this problem, we typically have to double the width of the overlapping region. Finally, we often cannot profit from the speed of the algorithm as the broadening of the overlapping region implies more calculations for solving the two subproblems over A and B, especially when the used energies are complex. Nevertheless, when using relatively simple energy functions, this method can be interesting in practice.

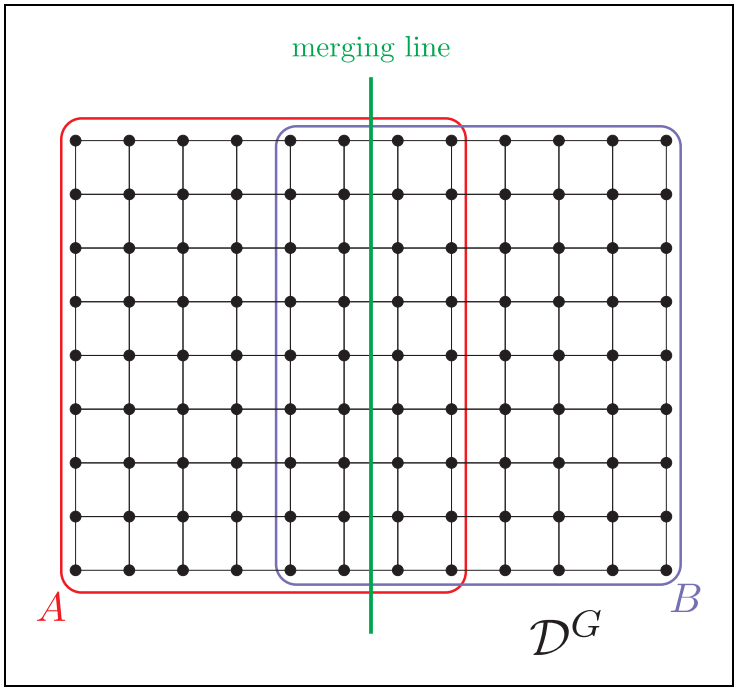

Figure 5.6: One of the most simplest ways to complete the solution in $A \cap B$ is to divide the region $A \cap B$ with a vertical line. On the left side we use the solution found in A and on the right side the solution found solution B.

Completion by solution merging

Another simple method for completion is to merge the two solutions on the overlapping region. As we know that the errors of the solutions $\widetilde{f}_A^G$ respectively $\widetilde{f}_B^G$ originate from its right respectively left border it is natural to give more weight to the solution $\widetilde{f}_A^G$ on the left side of $A \cap B$ and to favor solution $\widetilde{f}_B^G$ near the right side of $A \cap B$. For example

we can construct a solution on $A \cap B$ by a linear merging of $\widetilde{f}_A^G$ and $\widetilde{f}_B^G$. Formally, if N is the number of columns of the overlapping region, so on column number k of $A \cap B$ we use the solution

$$\left[\frac{1}{N+1}((N+1-k)\widetilde{f}_A^G + k\widetilde{f}_B^G)\right] \ ^{(1)}$$

This method is very fast and yields very smooth surfaces. Thus we do not need to broaden drastically the width of the overlapping region as in the previous method to avoid discontinuities. On the other hand almost all reconstruction errors we made in $\widetilde{f}_A^G$ and $\widetilde{f}_B^G$ have a negative influence on the merged solution. Therefore even without having large discontinuities, the energy value of the merged solution may be rather high. This problem is amplified by the fact that the method constructs a solution on the overlapping region without consulting the corresponding consistency arcs. Anyhow the presented merging method is an interesting technique to get smooth solutions without a large overlapping width.

Completion through merging path

An interesting idea for completion is to find an adequate path ρ as illustrated in figure 5.7 instead of a simple straight line as in the previous method. We denote by A_ρ the part of the graph domain on the left side of the path and by B_ρ the part on the right side. Having found a path ρ we will then merge the solution by using $\widetilde{f}_A^G$ on the region A_ρ and $\widetilde{f}_B^G$ on the region B_ρ.

An intuitive criterion for the merging path would be to choose a path that minimizes the smoothness energy we will get between the two regions A_ρ and B_ρ. More formally such a path ρ satisfies:

$$\rho = \underset{r \text{ merging path}}{argmin} \mathbf{E}_s(\widetilde{f}_A^G\big|_{A_r}, \widetilde{f}_B^G\big|_{B_r}) \tag{5.1}$$

This path can be determined in the following way. We construct a dual-like graph on the set $A \cap B$ as illustrated in figure 5.8. A merging path corresponds to a path from α^* to β^* on the green graph (that we call *merging graph*). To every edge e^* of the merging graph we associate the edge $e = (q_1, q_2)$ of the grid over the graph domain such that e crosses e^* and (e, e^*) is right-handed. We use the notation $e^* = (q_1, q_2)^*$.
We then introduce the following length function l^* on the merging graph. Let e^* be an edge in the merging graph and $e = (q_1, q_2)$ its corresponding edge on the graph domain, so we set

$$l^*(e^*) = K^G|\widetilde{f}_A^G(q_1) - \widetilde{f}_B^G(q_2)|$$

This length corresponds to the smoothness energy that will be created between q_1 and q_2 when the merging path uses edge e^*. It is thus easy to see that the shortest path from

(1) For a real number $x \in \mathbb{R}$, $[x]$ denotes the neares integer value to x (we normally break ties by rounding up).

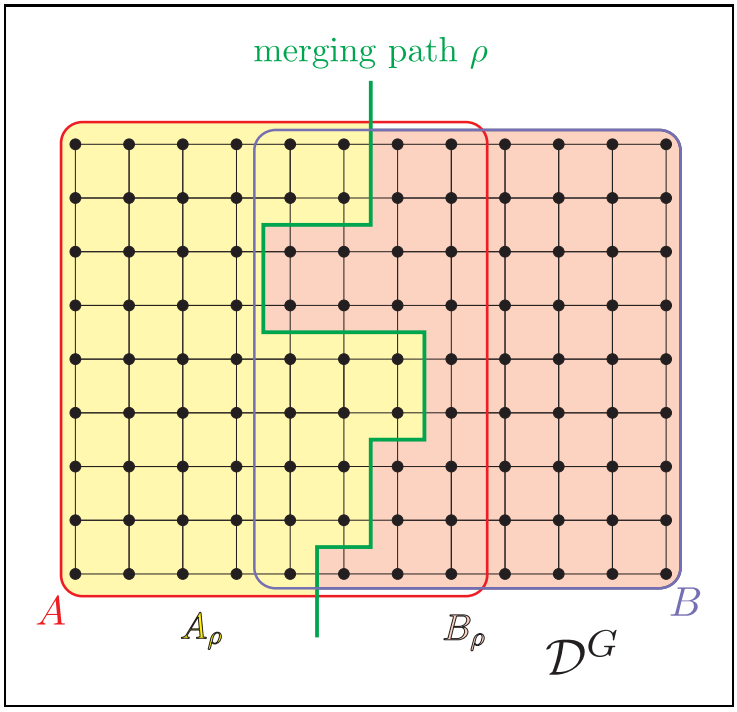

Figure 5.7: We try to find an adequate path ρ for merging the solution on the overlapping region.

α^* to β^* with respect to l^* satisfies 5.1.

This merging method is quite fast and gives solutions with rather good quality concerning local discontinuities and energy value. Additionally, even with a relatively small width of the overlapping region, we get good results. Therefore, this method is very efficient in practice even with complicated energies.

Energy-optimal completion

The optimal completion in terms of energy can be obtained by solving a reconstruction problem on the set $A \cap B$, where we impose border conditions on the left and right outer border who are set on the disparities given by the solutions $\widetilde{f}_A^G$ and $\widetilde{f}_B^G$. Formally we define a condition R^G on the set $\partial A \cup \partial B$ as follows:

$$R^G(q) = \begin{cases} \{\widetilde{f}_B^G(q)\} & \text{if } q \in \partial A \\ \{\widetilde{f}_A^G(q)\} & \text{if } q \in \partial B \end{cases}$$

We then solve the reconstruction problem over the region $A \cap B$ with border conditions R^G to obtain a solution $\widetilde{f}_{A\cap B}^G$. By construction, we have that this solution is the best completion in terms of energy.

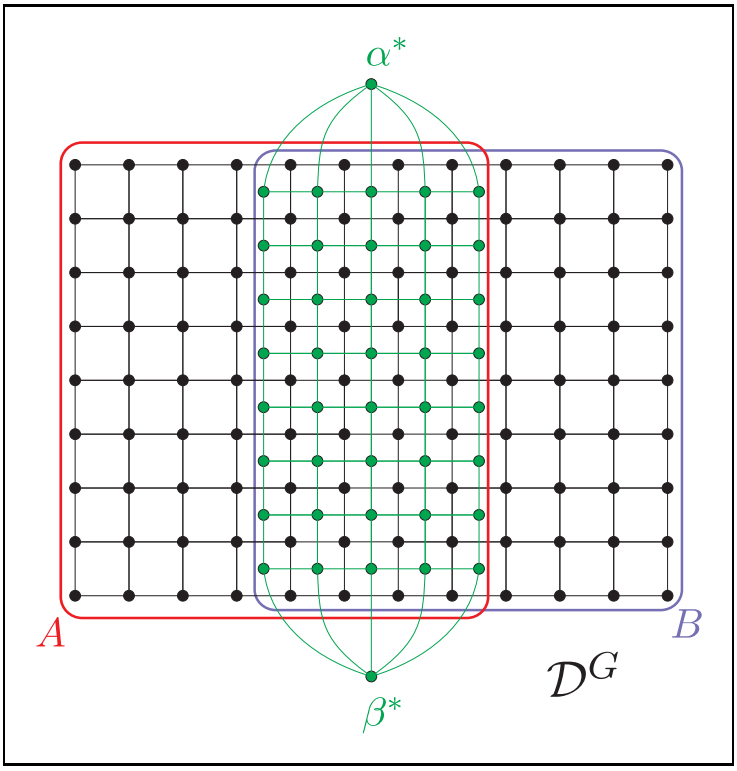

Figure 5.8: A merging path corresponds to a path from α^* to β^* in the dual-like graph in green that we call *merging graph*. By introducing an adequate length function in the merging graph we can choose a shortest path from α^* to β^* as the merging path.

This method has some interesting characteristics. The reconstructed surface is very smooth and has an excellent energy value. Furthermore we can use a very little width of the overlapping region (for example 10 to 20 points). On the other hand, this way of completion is rather time expensive as we have to solve another reconstruction problem over the whole disparity range.

Variation of the energy-optimal completion

We can speed up the energy-optimal completion technique by imposing disparity reductions (on intervals) on the reconstruction problem over $A \cap B$ in the following way. For a point $q \in A \cap B$ we only allow disparities between $\widetilde{f}_A^G$ and $\widetilde{f}_B^G$. As this two solutions are normally near to each other, we can reduce the resolution time drastically. The drawback is that in general we will not find the energy-optimal completion anymore. On the other hand, the obtained solution is still better in terms of energy than any of the previously proposed methods except the energy-optimal completion. Additionally, the solution we get through this method will not show large discontinuities in general and

even with a small overlapping width, we will get good results. Therefore we think that this method is probably one of the most interesting in practice when the determination of the consistency capacities is time consuming.

5.4 Completion method allowing to give an error bound on the energy

In this section we present a completion method that allows to give an upper bound on the difference in energy between the global optimal solution and merged solutions. This method uses the merging path technique as introduced before with the following differences. We construct an upper bound $\widetilde{f}_A^G$ on the region A and a lower bound $\widetilde{f}_B^G$ on the region B with the border condition technique introduced in 4.4.5, instead of solving these subproblems without border conditions. And we use another length function $l^{*\prime}$ on the merging graph defined as follows. Let e^* be an edge in the merging graph and $e = (q_1, q_2)$ the corresponding edge in the graph domain. We then set

$$l^{*\prime}(e^*) = K^G |\widetilde{f}_A^G(q_1) - \widetilde{f}_B^G(q_1)|$$

So $l^{*\prime}$ is just a little modification of the length function l^*. The motivation of this energy function $l^{*\prime}$ lies in the following theorem.

Theorem 5.4.1. *When merging on a merging path ρ, so the difference between the energy of the merged solution and the energy of the global optimal solution is at most*

$$2l^{*\prime}(\rho)$$

where $l^{*\prime}(\rho)$ is the length of the path ρ with respect to $l^{*\prime}$.

Proof. Let $\widetilde{f}^G$ be the nearest global optimal solution and $[\widetilde{f}_A^G|_{A_\rho}, \widetilde{f}_B^G|_{B_\rho}]$ the merged solution found by the above merging method. We define the following two conditions R_A^G on ∂A_ρ and R_B^G on ∂B_ρ:

$$R_A^G(q) = \{\widetilde{f}_A^G(q)\} \quad \forall\, q \in \partial A_\rho$$
$$R_B^G(q) = \{\widetilde{f}_B^G(q)\} \quad \forall\, q \in \partial B_\rho$$

We begin by showing the following lemma:

Lemma 5.4.2.

1. $\boldsymbol{E}_s(\widetilde{f}^G|_{A_\rho}, R_A^G) - \boldsymbol{E}_s(\widetilde{f}_A^G|_{A_\rho}, R_A^G) \leq K^G \sum_{(q_1,q_2)^* \in \rho} \left(\widetilde{f}_A^G(q_1) - \widetilde{f}^G(q_1)\right)$
2. $\boldsymbol{E}_s(\widetilde{f}_B^G|_{B_\rho}, \widetilde{f}_A^G|_{A_\rho}) - \boldsymbol{E}_s(\widetilde{f}_B^G|_{B_\rho}, R_B^G) \leq l^{*\prime}(\rho)$
3. $\boldsymbol{E}_s(\widetilde{f}^G|_{B_\rho}, R_B^G) \leq \boldsymbol{E}_s(\widetilde{f}^G|_{B_\rho}, \widetilde{f}^G|_{A_\rho}) + K^G \sum_{(q_1,q_2)^* \in \rho} \left(\widetilde{f}^G(q_1) - \widetilde{f}_B^G(q_1)\right)$

Proof of lemma 5.4.2
All the three points of this lemma use the property that $\widetilde{f}_A^G$ is an upper bound and $\widetilde{f}_B^G$ a lower bound of the nearest global optimal solution $\widetilde{f}^G$ on the regions where they are defined.
The first point can be justified in the following way. Let $e = (q_1, q_2)$ be an edge from A_ρ to B_ρ. So the difference between the contribution of this edge to the smoothness energy $\mathbf{E}_s(\widetilde{f}^G\big|_{A_\rho}, R_A^G)$ and the contribution of e to $\mathbf{E}_s(\widetilde{f}_A^G\big|_{A_\rho}, R_A^G)$ can be at most $K^G|\widetilde{f}_A^G(q_1) - \widetilde{f}^G(q_1)| = K^G(\widetilde{f}_A^G(q_1) - \widetilde{f}^G(q_1))$ [(2)]. By summing over all such edges e, we get the desired result. Note that the characterization of the condition R_A^G is of no importance for this point.
The second point uses the same argument as the first one. This time, the two terms differ on the region ∂B. Once we have R_B^G and once $\widetilde{f}_A^G$. As before fix an edge $e = (q_1, q_2)$ from A_ρ to B_ρ. The difference between the contribution of e to the term $\mathbf{E}_s(\widetilde{f}_B^G\big|_{B_\rho}, \widetilde{f}_A^G\big|_{A_\rho})$ and to the term $\mathbf{E}_s(\widetilde{f}_B^G\big|_{B_\rho}, R_B^G)$ is bounded by $K^G(\widetilde{f}_A^G(q_1) - \widetilde{f}_B^G(q_1))$. By summing over all edges e from A_ρ to B_ρ and identifying the right side as $l^{*\prime}(\rho)$ we get the desired result.
For proving the third point we also choose an edge $e = (q_1, q_2)$ from A_ρ to B_ρ. The contribution of this edge to the energy term $\mathbf{E}_s(\widetilde{f}^G\big|_{B_\rho}, R_B^G)$ is $K^G|\widetilde{f}_B^G(q_2) - \widetilde{f}^G(q_1)|$. Through a little transformation using the triangle inequality we get

$$\begin{aligned} K^G|\widetilde{f}^G(q_2) - \widetilde{f}_B^G(q_1)| &= K^G|\widetilde{f}^G(q_2) - \widetilde{f}^G(q_1) + \widetilde{f}^G(q_1) - \widetilde{f}_B^G(q_1)| \\ &\leq K^G|\widetilde{f}^G(q_2) - \widetilde{f}^G(q_1)| + K^G|\widetilde{f}^G(q_1) - \widetilde{f}_B^G(q_1)| \end{aligned}$$

Finally by summing over all edges e from A_ρ to B_ρ we get the desired result.
□(Lemma 5.4.2)

We will now develop some relations concerning energy terms related with $\widetilde{f}^G, \widetilde{f}_A^G$ and $\widetilde{f}_B^G$.
By the completion theorem we know that $\widetilde{f}_A^G\big|_{A_\rho}$ is the nearest optimal solution on the region A_ρ with border conditions R_A^G. As $\widetilde{f}^G\big|_{A_\rho}$ is another solution for the same problem we must have that its energy cannot be better than the one of the solution $\widetilde{f}_A^G\big|_{A_\rho}$ for this problem, i.e.

$$\mathbf{E}(\widetilde{f}_A^G\big|_{A_\rho}) + \mathbf{E}_s(\widetilde{f}_A^G\big|_{A_\rho}, R_A^G) \leq \mathbf{E}(\widetilde{f}^G\big|_{A_\rho}) + \mathbf{E}_s(\widetilde{f}^G\big|_{A_\rho}, R_A^G)$$

This equation can be transformed to

$$\mathbf{E}(\widetilde{f}_A^G\big|_{A_\rho}) \leq \mathbf{E}(\widetilde{f}^G\big|_{A_\rho}) + \mathbf{E}_s(\widetilde{f}^G\big|_{A_\rho}, R_A^G) - \mathbf{E}_s(\widetilde{f}_A^G\big|_{A_\rho}, R_A^G)$$

[(2)] Because $\widetilde{f}_A^G$ is an upper bound for the nearest optimal solution, we know that the right side of the equality is positive.

and by applying point 1 of lemma 5.4.2 we get

$$\mathbf{E}(\widetilde{f}_A^G\big|_{A_\rho}) \leq \mathbf{E}(\widetilde{f}^G\big|_{A_\rho}) + K^G \sum_{(q_1,q_2)^* \in \rho} \Big(\widetilde{f}_A^G(q_1) - \widetilde{f}^G(q_1)\Big) \tag{5.2}$$

Analogously as before, we know by the completion theorem that $\widetilde{f}_B^G\big|_{B_\rho}$ is the optimal nearest solution on the region B_ρ with border conditions R_B^G. As before we compare this solution to $\widetilde{f}^G\big|_{B_\rho}$ to get

$$\mathbf{E}(\widetilde{f}_B^G\big|_{B_\rho}) \leq \mathbf{E}(\widetilde{f}^G\big|_{B_\rho}) + \mathbf{E}_s(\widetilde{f}^G\big|_{B_\rho}, R_B^G) - \mathbf{E}_s(\widetilde{f}_B^G\big|_{B_\rho}, R_B^G)$$

We now add $\mathbf{E}_s(\widetilde{f}_B^G\big|_{B_\rho}, \widetilde{f}_A^G\big|_{A_\rho})$ to both sides of the previous inequality to obtain

$$\begin{aligned}\mathbf{E}(\widetilde{f}_B^G\big|_{B_\rho}) + \mathbf{E}_s(\widetilde{f}_B^G\big|_{B_\rho}, \widetilde{f}_A^G\big|_{A_\rho}) \leq \\ \mathbf{E}(\widetilde{f}^G\big|_{B_\rho}) + \underbrace{\mathbf{E}_s(\widetilde{f}^G\big|_{B_\rho}, R_B^G)}_{\text{I}} + \underbrace{\mathbf{E}_s(\widetilde{f}_B^G\big|_{B_\rho}, \widetilde{f}_A^G\big|_{A_\rho}) - \mathbf{E}_s(\widetilde{f}_B^G\big|_{B_\rho}, R_B^G)}_{\text{II}}\end{aligned}$$

We can now apply point 2 of lemma 5.4.2 to the term II and point 3 of lemma 5.4.2 to I to obtain

$$\begin{aligned}\mathbf{E}(\widetilde{f}_B^G\big|_{B_\rho}) + \mathbf{E}_s(\widetilde{f}_B^G\big|_{B_\rho}, \widetilde{f}_A^G\big|_{A_\rho}) \leq \\ \mathbf{E}(\widetilde{f}^G\big|_{B_\rho}) + \mathbf{E}_s(\widetilde{f}^G\big|_{B_\rho}, \widetilde{f}^G\big|_{A_\rho}) + K^G \sum_{(q_1,q_2)^* \in \rho} \Big(\widetilde{f}^G(q_1) - \widetilde{f}_B^G(q_1)\Big) + l^{*\prime}(\rho)\end{aligned} \tag{5.3}$$

Finally by summing the inequalities 5.2 and 5.3 we get

$$\begin{aligned}&\underbrace{\mathbf{E}(\widetilde{f}_A^G\big|_{A_\rho}) + \mathbf{E}(\widetilde{f}_B^G\big|_{B_\rho}) + \mathbf{E}_s(\widetilde{f}_B^G\big|_{B_\rho}, \widetilde{f}_A^G\big|_{A_\rho})}_{=\mathbf{E}[\widetilde{f}_A^G\big|_{A_\rho}, \widetilde{f}_B^G\big|_{B_\rho}]} \leq \\ &\underbrace{\mathbf{E}(\widetilde{f}^G\big|_{A_\rho}) + \mathbf{E}(\widetilde{f}^G\big|_{B_\rho}) + \mathbf{E}_s(\widetilde{f}^G\big|_{B_\rho}, \widetilde{f}^G\big|_{A_\rho})}_{=\mathbf{E}(\widetilde{f}^G)} + \underbrace{K^G \sum_{(q_1,q_2)^* \in \rho} \Big(\widetilde{f}_A^G(q_1) - \widetilde{f}_B^G(q_1)\Big)}_{=l^{*\prime}(\rho)} + l^{*\prime}(\rho)\end{aligned}$$

Be replacing the indicated terms we finally get

$$\mathbf{E}[\widetilde{f}_A^G\big|_{A_\rho}, \widetilde{f}_B^G\big|_{B_\rho}] \leq \mathbf{E}(\widetilde{f}^G) + 2l^{*\prime}(\rho)$$

what finishes the proof.

□

Therefore we try to merge on the shortest path (with respect to $l^{*\prime}$) since this gives us the best error bound. When the length of the shortest merging path is zero, so we are sure that we have achieved the global optimum. When choosing a relatively wide overlapping region, this case appears often.
This method has very similar characteristics as the first merging path method. An interesting application is to use this method for bounding the error of the energy-optimal completion or its variation in the following way. We calculate an upper bound solution on A and a lower bound solution on B as before. We then merge the two solutions by the energy-optimal completion or its variation. Both of these methods achieve solutions with a better energy than the merging path method introduced in this section. Therefore, the error bound of the merging path method is also valid for the energy-optimal completions.

5.5 Generalization of the division into two subproblems

An important question is how to generalize the above suggestions to get more than two subproblems. One idea is to use the above procedures recursively, i.e. for the resolution of both subproblems on A and B we use again one of the proposed methods. This recursive call can be continued as long as desired (typically till the subproblems are little enough to be solved directly, respectively till we have a number of subproblems in the same range as the number of computers that we use for the resolution).
Unfortunately the error bound of the previously introduced method will not be valuable anymore when using recursive calls as it is impossible to bound the error propagation efficiently (nasty examples are easy to find). To cover this problem we can divide the graph domain at the beginning. Figure 5.9 shows a grid division of the graph domain in four overlapping regions. We construct alternately upper and lower bound solutions on the different regions i.e. we construct upper bound solutions on A and D and lower bound solutions on B and C. For merging we have to find a *merging structure* as illustrated in the figure. Here the search for the best merging structure is more difficult. Nevertheless, the result of the error bounding can easily be generalized on the grid and as before the error bound is valuable even if we do not find the best merging structure.

Note that the idea of preliminary division of the graph domain can also be of interest for other proposed methods because it has the following advantages:

- Size and number of the regions can be fixed very easily.
- The implementation for the parallelization is much easier.

5.6 A parallelizable method for finding the optimal solution

In this section we present a parallelizable method that can be applied when it is important to get the global optimal solution. The method consists of two phases. In a first phase, we construct locally upper and lower bounds as explained in 4.4.5. This phase can be easily parallelized. During the first phase we maintain a global upper bound and

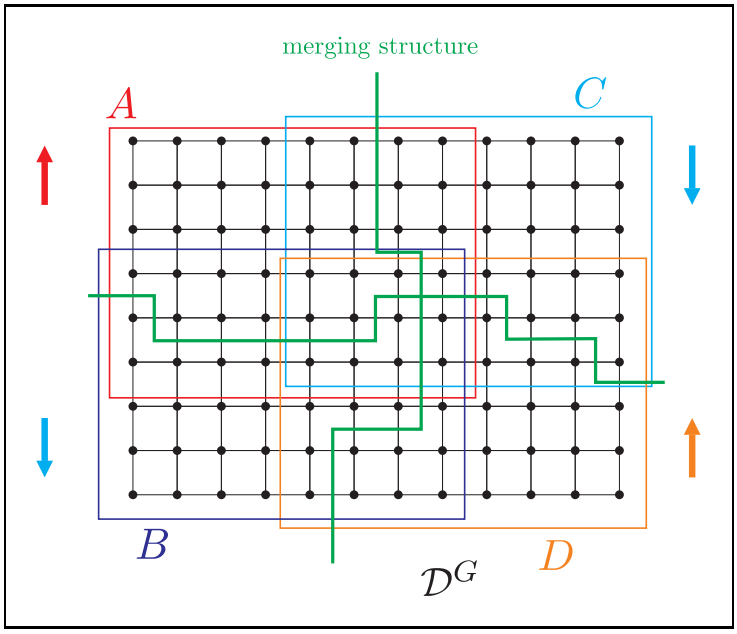

Figure 5.9: Here the graph domain was divided at the beginning in four overlapping regions A, B, C, D. To apply a merging path method we must find a *merging structure*.

a global lower bound which can be deduced by combining the local upper and lower bounds calculated. On points of the graph domain where the global upper bound equals the global lower bound we can already fix the solution. The region of the graph domain where we can fix the solution in this way is called *fixed region*. The other points are on the *non fixed region*.

We try to obtain one of the following two situations during the first phase:

- The non fixed region consists of several regions that are not connected to each other as illustrated in figure 5.10. In this case we continue the algorithm on every such region separately.
- The reconstruction problem on the remaining vertices between the global upper and lower bound can be solved by one computer. We therefore can stop the first phase and pass to the second phase which consists of determining the solution directly.

Theoretically it would be possible that we will never get in one of the two situations described above. Fortunately, in practice this cases are extremely rare as the local upper and lower bounds that we determine in the first phase often touch the optimal solution as seen in 4.4.5. Therefore this technique allows in general to get the optimal solution

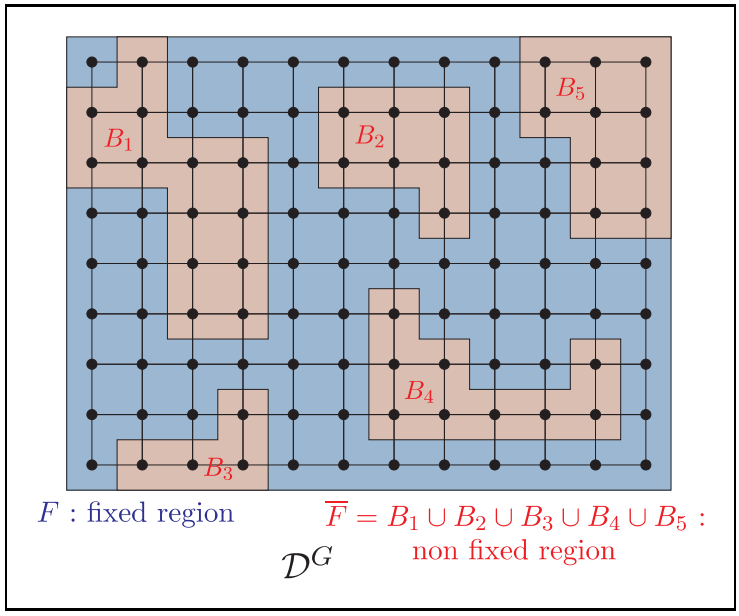

Figure 5.10: B_1, B_2, B_3, B_4, B_5 are zones in the non fixed region that are not connected to each other. Thus they can be solved independently.

for very large problems. The drawback is that this method is very time consuming and it is difficult to determine the computational time at the beginning.

6 Code

In this chapter we will shortly discuss the code that is provided with this work. The code is written in C++ and can be used as a library for solving reconstruction problems with different of the proposed methods. Namely, we have implemented various flow algorithms and capacity functions. Furthermore the proposed disparity reduction on intervals is provided, as well as functions for applying the proposed pyramidal approach and a simple parallelization method. Additionally, we introduced a format for handling reconstruction solutions (the «dcm» format) and implemented different functions that allow to manipulate solutions and save them under different formats (dcm, pts, or as an image).

To simplify the use of some methods of high practical interest, we provide three programs. The main program is named `rstereo.exe` and allows to solve reconstruction problems directly, or by parallelization. Also the pyramidal approach can be used with this program for solving problems. It can be executed from a command line and takes one argument, which is the name of an initialization file in which parameters corresponding to the problem and the resolution method are specified. The file `rstereo_template.ini` is a template of such an initialization file and explains how to enter the parameters and use the program.
The second program `dcm_to_boundary.exe` allows to extract a horizontal or vertical line of a solution and to save it in a file with the dcm format (information of how to use this program can be found in the file `dcm_to_boundary.cpp` or simply by executing the program without arguments). This file can be used later as boundary condition in the `rstereo.exe` program (further explanations of how to introduce a boundary condition can be found in the file `rstereo_template.ini`).
The third program we provide is named `rcompare.exe` and compares two solutions (in dcm format) or images (in pgm format) and saves their difference as an image with pgm format (information of how to use the program can be found in the file `rcompare.cpp` or simply by executing the program without arguments).

Below, we give a listing of the modules we programmed with a short description.

rstereo.h/rstereo.cpp The principal module containing the `main` function.

global.h/global.cpp Module containing global constants and all preprocessor definitions made. Here one can change the algorithm and capacity function that will be used. Also the dumping of different comments can be activated and deactivated in this module. For debugging, the DEBUG constant can be introduced which performs

numerous tests for finding the first apparition of an irregular situation. DEBUG should not be activated for the final compilation as it slows down the program.

exceptionhandling.h/exceptionhandling.cpp A module defining classes for exception handling.

4x4matrixalgebra.h/4x4matrixalgebra.cpp Some algebraic manipulations of 4x4 matrices are defined here. They are used for the projection of points to the images.

image.h/image.cpp Classes for images and reconstruction frameworks are defined here.

vertex1.h/vertex1.cpp Introduces a class for vertices.

flow1.h/flow1.cpp This is one of the most important modules. It defines a class for reconstruction problems and the functions to solve them.

pyramidaldisparity.h/pyramidaldisparity.cpp Module where the proposed pyramidal approach is implemented.

parallelization.h/parallelization.cpp A parallelization algorithm using preliminary grid-division of the graph domain and straight merging is defined here.

distancequeue.h/distancequeue.cpp Defines a data structure used by preflow-push algorithms for calculating exact distances to the sink.

heightlist.h/heightlist.cpp Defines a data structure for an efficient implementation of the gap heuristic.

heightstockage.h/heightstockage.cpp Defines a data structure for an efficient implementation of the wave algorithm.

verticeslist.h/verticeslist.cpp Data structure used by the growing-trees algorithm.

disparitycmap.h/disparitycmap.cpp Defines a class for working with solution.

ini.h/ini.cpp Defines a class for initialization files.

fibonacciheap.h/fibonacciheap.cpp Implementation of the Fibonacci heap (can be used for the shortest path algorithm of Dijkstra).

dcm_to_boundary.cpp Here, the program `dcm_to_boundary.exe` is implemented and explained.

rcompare.cpp Implementation of the program `rcompare.exe` and explanations for its use.

More details about our code can be found as commentaries in the corresponding modules. For using our code as a library you can simply include the file ≪rstereo.h≫.

The software provided with this work contains furthermore a set of images and projection matrices that allow to test our code and the above mentioned programs. Further information can be found in the file `readme.txt`.

Conclusion

At first, different algorithms for the resolution of the minimum $s-t$ cut problem on reconstruction networks were studied theoretically and empirically. This allowed us to determine the most promising algorithms for our problem. We introduced two preprocessing algorithms speeding up some of the maximum flow algorithms on reconstruction networks.
Additionally, we developed a formulation that allows to impose disparity restrictions on the problem and reduces the size of the network. In particular we can use this formulation to impose internal and border conditions. The idea of using a pyramidal approach discussed in [11] was improved by using the introduced disparity reduction techniques. Furthermore, some interesting theoretical results for problems with internal and border conditions were introduced. They are of practical importance in numerous proposed methods.
We developed methods for parallelization allowing to tackle problems that were till now untractable with the classical use of the $s-t$ cut formulation. By combining these methods with the proposed pyramidal approach we have not only reduced the computational time thanks to the parallelizing process but we also reduced the complexity of the resolution method and still get very accurate solutions.
Finally we have written a code in C++ allowing to apply and test most of the proposed methods and algorithms. This code can be used as a library for further work.

There are a lot of possibilities for continuing this work. A topic of practical interest would be to develop methods for cutting the graph domain on regions where reconstruction is simple. This would allow to improve the proposed parallelization methods because the obtained subproblems have a smaller influence to each other. In particular we can choose a very little overlapping width for the subproblems. It would also be interesting to generalize and test the obtained results and algorithms on other types of networks (usually not grid like) that are used for solving the reconstruction problem.

Finally, I would like to thank Professor Dominique de Werra, Tınaz Ekim and Pascal Lagger for their help and assistance during the last months.

Bibliography

[1] ROY S., Stereo Without Epipolar Lines: A Maximum-Flow Formulation. *International Journal of Computer Vision*, 34(2/3):147-162. August 1999

[2] ROY S., COX I.J., A Maximum-Flow Formulation of the n-Camera Stereo Correspondence Problem. *IEEE Proc. of Int. Conference on Computer Vision*, 492-499. Bombay, 1998

[3] PARIS S., SILLION F., LONG Q., *A Volumetric Reconstruction Method from Multiple Calibrated Views using Global Graph Cut Optimization*. Montbonnot Saint Ismier, 2003

[4] PARIS S. *Extraction of Three-dimensional Information from Images*, doctoral thesis, Université Joseph Fourier. Grenoble, 2004

[5] ZENKLUSEN R., *Reconstruction de surfaces tridimensionnelles par des méthodes de flots dans les graphes*, Projet de semestre, EPFL. Lausanne, june 2004

[6] FAUGERAS O., *Three-Dimensional Computer Vision*. Cambridge : MIT Press, 1993

[7] KOLMOGOROV V., BOYKOV J., An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision. *IEEE Transactions on PAMI* 26(9):1124-1137. September 2004

[8] KOLMOGOROV V., ZABIH, R., What energy functions can be minimized via graph cuts. *IEEE Transactions on Pattern and Machine Intelligence*, 26(2):147-159. February 2004

[9] AHUJA K.R., MAGNANTI T.L., ORLIN, J.B., *Network Flows*. Upper Saddle River : Prentice Hall, 1993

[10] CORMEN T.H., LEISERSON C.E., RIVEST R.L., STEIN C., *Introduction to Algorithms* Second Edition. Massachusetts, 2001

[11] ROY S., DROUIN M., *Non-Uniform Pyramid Stereo for Larges Images*. Erlangen, 2002

[12] FUJISHIGE S., ISOTANI S., New Maximum Flow Algorithms by MA Orderings and Scaling. *Journal of the Operations Research Society of Japan*, Vol. 46, No. 3. 2003

[13] GOLDBERG A., RAO S., *Length Functions for Flow Computations*. Princeton, 1997

[14] SLEATOR D.D., TARJAN R.E., A data structure for dynamic trees. *Proc. Thirteenth Annual ACM Symp. on Theory of Computing* pp.114-122. 1981

[15] WEST D.B., *Introduction to Graph Theory* Second Edition. Upper Saddle River : Prentice Hall, 2001

[16] DE WERRA D., LIEBLING T.M., HECHE J.-F., *Recherche opérationnelle pour ingénieurs I*. Lausanne, Presses polytechniques et universitaires romandes : 2003