Network Flow Algorithms for Discrete Tomography

Proefschrift

ter verkrijging van
de graad van Doctor aan de Universiteit Leiden,
op gezag van de Rector Magnificus Dr. D. D. Breimer,
hoogleraar in de faculteit der Wiskunde en
Natuurwetenschappen en die der Geneeskunde,
volgens besluit van het College voor Promoties
te verdedigen op dinsdag 19 september 2006
klokke 16.15 uur
door

Kees Joost Batenburg

geboren te Rotterdam
in 1980.

Samenstelling van de promotiecommissie:

promotor:	Prof. dr. R. Tijdeman
copromotor:	Dr. ir. H. J. J. te Riele (CWI)
referent:	Prof. dr. A. Kuba (University of Szeged, Hungary)
overige leden:	Prof. dr. R. J. F. Cramer (CWI / Universiteit Leiden) Dr. W. A. Kosters Prof. dr. J. B. T. M. Roerdink (Rijksuniversiteit Groningen) Prof. dr. J. Sijbers (Universiteit Antwerpen) Prof. dr. S. M. Verduyn Lunel

The research in this thesis has been carried out at the national research institute for Mathematics and Computer Science in The Netherlands (CWI) and at the Universiteit Leiden. It has been financially supported by the Netherlands Organisation for Scientific Research (NWO), project 613.000.112.

Network Flow Algorithms
for Discrete Tomography

Batenburg, Kees Joost, 1980 –
Network Flow Algorithms for Discrete Tomography
Printed by Universal Press - Veenendaal
AMS 2000 Subj. class. code: 94A08, 90C90
NUR: 918
ISBN-10: 90-9020935-2
ISBN-13: 978-90-9020935-7

e-mail: kbatenbu@math.leidenuniv.nl

NWO
Netherlands Organisation for Scientific Research

THOMAS STIELTJES INSTITUTE
FOR MATHEMATICS

© K.J. Batenburg, Leiden 2006.

The illustrations on the cover of this thesis are based on a tomographic reconstruction of a diamond, recorded by DiamScan, Belgium. Part of the data processing was performed by Prof.dr. J. Sijbers of the University of Antwerp, Belgium.

All rights reserved. No part of this publication may be reproduced in any form without prior written permission from the author.

Contents

Outline **1**

1. Network flow algorithms for discrete tomography: an overview **3**

- 1.1. Introduction 3
- 1.2. Network flow formulation for two projections 4
- 1.3. Weighted reconstruction 9
- 1.4. Reconstruction from noisy projections 13
- 1.5. Algorithms and implementation 17
- 1.6. Extension to more than two projections 18
  - 1.6.1. Computing the start solution 19
  - 1.6.2. Computing the weight map 20
  - 1.6.3. Choosing the pair of directions 22
  - 1.6.4. Stop criterion 23
  - 1.6.5. Some results 23
- 1.7. Reconstructing 3D volumes 25
- 1.8. Extension to plane sets 28
- 1.9. Applications of discrete tomography 30
  - 1.9.1. Atomic resolution electron tomography of nanocrystals 31
- Bibliography 37

2. An evolutionary algorithm for discrete tomography **39**

- 2.1. Introduction 39
- 2.2. Preliminaries 41
- 2.3. Algorithmic approach 45
  - 2.3.1. Overview of the approach 45
  - 2.3.2. A new crossover operator 46
  - 2.3.3. The hillclimb operator 47
  - 2.3.4. The mutation operator 50
  - 2.3.5. Algorithm details 50
- 2.4. Computational results 52
  - 2.4.1. Reconstruction of hv-convex images 53
  - 2.4.2. Reconstruction from three projections 54
  - 2.4.3. Reconstruction using Gibbs priors 55

2.5. Conclusions 57
Bibliography 58

3. **A network flow algorithm for reconstructing binary images from discrete X-rays** **61**
3.1. Introduction 61
3.2. Preliminaries 63
3.3. Algorithm description 65
  3.3.1. Overview 65
  3.3.2. Network flow approach 65
  3.3.3. An iterative network flow algorithm 67
  3.3.4. Computing the start solution 67
  3.3.5. Computing the pixel weights 69
  3.3.6. Choosing the direction pair 70
  3.3.7. Stop criterion 71
  3.3.8. Time complexity 72
3.4. Results 72
  3.4.1. Experimental setup 72
  3.4.2. Qualitative description of algorithm performance 73
  3.4.3. Random polygons 75
  3.4.4. Random ellipses 77
  3.4.5. Random ellipses with noise 77
  3.4.6. Noisy projection data 78
  3.4.7. Comparison with alternative approaches 80
3.5. Discussion 82
3.6. Conclusions 83
Bibliography 84

4. **A network flow algorithm for 3D binary tomography of lattice images** **87**
4.1. Introduction 87
4.2. Preliminaries 88
4.3. The case $k = 2$ 89
4.4. An algorithm for $k > 2$ 92
  4.4.1. Peeling 94
  4.4.2. Computing the start solution 94
  4.4.3. Selecting the direction pair 94
  4.4.4. Computing the voxel weights 94
  4.4.5. Stop criterion 95
4.5. Results 95
4.6. Discussion and conclusions 96
Bibliography 99

5. An algorithm for the reconstruction of binary images without an intrinsic lattice 101
5.1. Introduction . . . 102
5.2. Preliminaries . . . 103
    5.2.1. Data collection geometries . . . 103
    5.2.2. Definitions and notation . . . 104
5.3. Two projections . . . 105
5.4. More than two projections . . . 113
    5.4.1. Choosing the pair of directions . . . 114
    5.4.2. Converting between different grids . . . 116
    5.4.3. Computing the weight map . . . 117
    5.4.4. Termination and final output . . . 118
5.5. Experimental results . . . 118
    5.5.1. Parallel beam projections . . . 119
    5.5.2. Fan beam projections . . . 122
    5.5.3. Noisy projection data . . . 123
    5.5.4. Real-world data . . . 124
5.6. Conclusions . . . 125
Bibliography . . . 127

6. On the reconstruction of crystals by discrete tomography 129
6.1. Introduction . . . 129
6.2. Preliminaries . . . 132
6.3. Algorithm . . . 136
    6.3.1. Block phase . . . 137
    6.3.2. Fitting phase . . . 138
    6.3.3. Culling phase . . . 140
    6.3.4. Noise . . . 141
6.4. Experimental results . . . 141
6.5. Discussion . . . 142
6.6. Conclusions . . . 143
Bibliography . . . 144

Nederlandse samenvatting 145

Curriculum Vitae 153

Other publications 155

Outline

Tomography is a powerful technique to obtain images of the interior of an object in a non-destructive way. First, a series of *projection images* (e.g., X-ray images) is acquired and subsequently a reconstruction of the interior is *computed* from the available projection data. The algorithms that are used to compute such reconstructions are known as *tomographic reconstruction algorithms*.

Discrete tomography is concerned with the tomographic reconstruction of images that are known to contain only a few different gray levels. By using this knowledge in the reconstruction algorithm it is often possible to reduce the number of projections required to compute an accurate reconstruction, compared to algorithms that do not use prior knowledge.

This thesis deals with new reconstruction algorithms for discrete tomography. In particular, the first five chapters are about reconstruction algorithms based on *network flow methods*. These algorithms make use of an elegant correspondence between certain types of tomography problems and network flow problems from the field of Operations Research.

Chapter 1 introduces the network flow approach and serves as an introduction to the remaining chapters. In Chapters 2, 3 and 4 we describe new algorithms for the reconstruction of *lattice images* from few projections. Lattice images can be used to model *nanocrystals*, which are of great interest in Materials Science. In Chapter 5, an algorithm is presented for reconstructing images that do not have a lattice structure. In the last chapter, Chapter 6, we study a problem that occurs in the application of discrete tomography to the reconstruction of nanocrystals from projections obtained by electron microscopy.

Chapter 2 deals with the reconstruction problem from only two projections. This problem is severely underdetermined and using prior knowledge is essential to obtain meaningful reconstructions. We describe an evolutionary algorithm that can incorporate various forms of prior knowledge. The algorithm uses a network flow algorithm as a subroutine to compute images that satisfy the projection constraints. It can also be used to reconstruct images from more than two projections. In that case the remaining projections — besides the first two — are considered as additional prior knowledge.

The evolutionary algorithm is not suitable for reconstructing large images, e.g., of size 256×256. In Chapter 3 we present an algorithm that *can* be used to reconstruct such larger images if more than two projections are available. The algorithm iteratively solves a series of network flow problems, each time using two of the projections. Chapters 2 and 3 deal with two-dimensional images. The algorithm from Chapter 3 is generalized to a three-dimensional setting in Chapter 4.

The network flow approach is naturally suited for lattice images. Most practical applications of tomography deal with images that do not have an intrinsic lattice structure. An extension of the network flow approach to images that do not have an intrinsic lattice structure is described in Chapter 5.

The algorithms from Chapters 2–4 can be used to compute reconstructions of nanocrystals at atomic resolution from projection data obtained by electron microscopy. However, for many interesting Materials Science samples, a lattice model does not correspond perfectly with physical reality. Deviations from the perfect lattice occur frequently in real microscopic samples. In Chapter 6 we present an algorithmic approach that can be used as a preprocessing step in order to apply discrete tomography in such cases.

The author has published several articles on discrete tomography that are not included in this thesis. The section “Other publications” provides references to publications where the methods from Chapters 1–6 have been applied. It also lists references to other related publications by the author.

Chapter 1

Network flow algorithms for discrete tomography: an overview

Part of this chapter will appear as a chapter of the book: G.T. Herman and A. Kuba, eds., "Advances in Discrete Tomography and its Applications", Springer, to appear (November 2006).

Abstract. There exists an elegant correspondence between the problem of reconstructing a 0-1 lattice image from two of its projections and the problem of finding a maximum flow in a certain graph. In this chapter we describe how network flow algorithms can be used to solve a variety of problems from discrete tomography. First, we describe the network flow approach for two projections and several of its generalizations. Subsequently, we present an algorithm for reconstructing 0-1 images from more than two projections. The approach is extended to the reconstruction of 3D images and images that do not have an intrinsic lattice structure. The chapter concludes with an introduction to the application of discrete tomography to the reconstruction of nanocrystals. The network flow approach that is introduced in this chapter forms the basis for the subsequent chapters.

1.1. Introduction

The problem of reconstructing a 0-1 image from a small number of its projections has been studied extensively by many authors. Most results deal with images that are defined on a lattice, usually a subset of $\mathbb{Z}^2$. Already in 1957, Ryser studied the problem of reconstructing an $m \times n$ 0-1 matrix from its row and column sums [21, 22]. He also provided an algorithm for finding a reconstruction if it exists. Ryser's algorithm is extremely efficient. In fact it can be implemented in such a way that it runs in linear time, $O(m+n)$, by using a compact representation for the output image [6].

The problem of reconstructing a 0-1 matrix from its row and column sums can also be modeled elegantly as a *network flow problem*. In 1957 Gale was the first to describe the two-projection reconstruction problem in the context of flows in networks, providing a completely different view from Ryser's approach [8]. In the latter work there was no reference to the algorithmic techniques for solving network flow problems. In 1956 Ford and Fulkerson published their seminal paper on an algorithm for computing a maximum flow in a network [7], which can be used to solve the two-projection reconstruction problem. Using the network flow model, Anstee derived several mathematical properties of the reconstruction problem [3].

The reconstruction problem from two projections is usually severely underdetermined. The number of solutions can be exponential in the size of the image. In practice the goal of tomography is usually to obtain a reconstruction of an *unknown original image*, not just to find any solution that has the given projections. If only two projections are available, additional prior knowledge must be used. Certain types of prior knowledge can be incorporated efficiently into the network flow approach, by using the concept of *min cost flows*.

The evolutionary algorithm that is described in Chapter 2 is capable of incorporating prior knowledge for the reconstruction of a 0-1 matrix from its row and column sums. It uses the fact that min cost flow problems can be solved efficiently.

A drawback of the network flow approach is that it cannot be generalized to the case of more than two projections. The reconstruction problem is NP-hard for any set of more than two projections [10]. In Chapter 3 we describe an iterative approach for reconstructing 0-1 images from more than two projections. In each iteration a reconstruction is computed from only two projections, using the network flow approach. The reconstruction from the previous iteration, which was computed using a different pair of projections, is used as prior knowledge such that the new reconstruction resembles the previous one.

In this chapter the network flow approach will be described, starting from the basic two-projection case. Section 1.2 describes the basic network flow formulation. In Section 1.3 the model is extended to incorporate prior knowledge in the reconstruction procedure. Section 1.4 describes how the network flow approach can be made tolerant to noise and other errors. The implementation of network flow algorithms for discrete tomography is discussed in Section 1.5. Several highly efficient implementations of network flow algorithms are available. This section also addresses the time complexity of the relevant network flow algorithms. The basic iterative algorithm for reconstructing from more than two projections is described in Section 1.6. This algorithm can be generalized to 3D reconstruction very efficiently, which is discussed in Section 1.7. So far, all sections deal with *lattice images*. In Section 1.8 we discuss how the algorithms from the previous sections can be adapted to the problem of reconstructing binary images that do not have a lattice structure.

1.2. Network flow formulation for two projections

The reconstruction problems of this chapter can be posed in several different forms. We mainly consider the reconstruction of a *subset* F of $\mathbb{Z}^2$ from its projections, but one can also formulate this problem in the context of reconstructing *binary matrices* or *black-and-white images*. In the case of binary matrices the set F is represented by the set of matrix-entries that have a value of 1. If we want to display a set $F \subset \mathbb{Z}^2$ and F is contained in a large rectangle $A \subset \mathbb{Z}^2$ (e.g., 256^2 elements), it is convenient to represent F as a *black-and-white* image. The white pixels correspond to the elements of F; the black pixels correspond to the remaining elements of A. *Continuous tomography algorithms*, such as the Algebraic Reconstruction Technique (see, e.g., Chapter 7 of [19]), represent the reconstruction as a *gray-level image*. At several points in this chapter we discuss how to utilize algorithms for continuous tomography for solving the discrete reconstruction problems. In these cases we use the black-and-white image representation of F, as this representation can easily be connected with the gray-level images from continuous tomography. Depending on the representation of the set F, points in A may also be called *entries* (in the context of binary matrices) or *pixels* (in the context of black-and-white images).

In this section we consider the problem of reconstructing a subset F of the lattice $\mathbb{Z}^2$ from its projections (defined below) in *two lattice directions*, $v^{(1)}$ and $v^{(2)}$. This is a generalization of the problem of reconstructing a binary matrix from its row and column sums. We assume that a *finite* set $A \subset \mathbb{Z}^2$ is given such that $F \subset A$. We call the set A the *reconstruction lattice*. As an illustration of the concept of the reconstruction lattice, consider the representation of F as a black-and-white image. The set A defines the *boundaries* of the image: all white pixels are known to be within these boundaries.

We denote the cardinality of any finite set F by $|F|$. Define $\mathbb{N}_0 = \{x \in \mathbb{Z} : x \geq 0\}$. A *lattice direction* is a pair $(a,b) \in \mathbb{Z}^2$ such that a and b are coprime and $a \geq 0$. Let $v^{(1)}$, $v^{(2)}$ be two given lattice directions. A *lattice line* is a line in $\mathbb{Z}^2$ parallel to either $v^{(1)}$ or $v^{(2)}$ that passes through at least one point in $\mathbb{Z}^2$. Any lattice line parallel to $v^{(k)}$ ($k = 1,2$) is a *set* of the form $\{nv^{(k)} + t : n \in \mathbb{Z}\}$ for $t \in \mathbb{Z}^2$. The sets $\mathcal{L}^{(1)}$ and $\mathcal{L}^{(2)}$ denote the sets of lattice lines for directions $v^{(1)}$ and $v^{(2)}$ respectively. For $k = 1,2$, put $L^{(k)} = \{\ell \in \mathcal{L}^{(k)} : \ell \cap A \neq \emptyset\}$. Note that $L^{(1)}$ and $L^{(2)}$ are finite sets. We denote the elements of $L^{(k)}$ by $\ell_{k,i}$ for $i = 1, \dots, |L^{(k)}|$. As an example, Figure 1.1 shows the reconstruction lattice for $A = \{1,2,3\} \times \{1,2,3\}$, $v^{(1)} = (1,0)$, $v^{(2)} = (1,1)$. For this example, the sets $L^{(1)}$ and $L^{(2)}$ contain three and five lattice lines respectively.

For any lattice set $F \subset \mathbb{Z}^2$ its projection $P_F^{(k)} : L^{(k)} \to \mathbb{N}_0$ in direction $v^{(k)}$ is defined as

$$P_F^{(k)}(\ell) = |F \cap \ell| = \sum_{x \in \ell} f(x) ,$$

where f denotes the characteristic function of F. The reconstruction problem can now be formulated as follows:

Problem 1 *Let $v^{(1)}$, $v^{(2)}$ be given distinct lattice directions and let $A \subset \mathbb{Z}^2$ be a given reconstruction lattice. Let $p^{(1)} : L^{(1)} \to \mathbb{N}_0$ and $p^{(2)} : L^{(2)} \to \mathbb{N}_0$ be given functions. Construct a set $F \subset A$ such that $P_F^{(1)} = p^{(1)}$ and $P_F^{(2)} = p^{(2)}$.*

Define $S^{(k)} = \sum_{\ell \in L^{(k)}} p^{(k)}(\ell)$. We call $S^{(k)}$ the *projection sum for direction $v^{(k)}$*. Note that if F is a solution of Problem 1, we have $S^{(k)} = |F|$ for $k = 1,2$. In Section 1.4, a generalization of Problem 1 will be described for which the prescribed projections $p^{(1)}$ and $p^{(2)}$ may contain errors. In that case the projection sum for direction $v^{(1)}$ may be different from the projection sum for direction $v^{(2)}$.

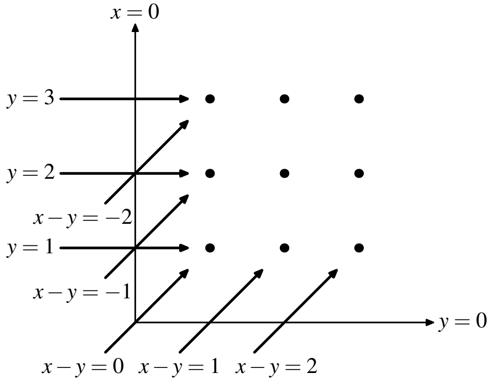


Figure 1.1: *Example lattice:* $A = \{1,2,3\} \times \{1,2,3\}$, $v^{(1)} = (1,0)$, $v^{(2)} = (1,1)$.

With the triple $(A, v^{(1)}, v^{(2)})$ we associate a *directed* graph $G = (V, E)$, where V is the set of nodes and E is the set of edges. We call G the *associated graph* of $(A, v^{(1)}, v^{(2)})$. The set V contains a node s (the *source*), a node t (the *sink*), one node for each $\ell \in L^{(1)}$ and one node for each $\ell \in L^{(2)}$. The node that corresponds to $\ell_{k,i}$ has label $n_{k,i}$. We call the nodes $n_{k,i}$ *line nodes*.

Nodes $n_{1,i}$ and $n_{2,j}$ are connected by a (directed) edge $(n_{1,i}, n_{2,j})$ if and only if $\ell_{1,i}$ and $\ell_{2,j}$ intersect in A. We call these edges *point edges* and denote the set of all point edges by $E_p \subset E$. There is a bijective mapping $\Phi : E_p \longrightarrow A$ that maps $(n_{1,i}, n_{2,j}) \in E_p$ to the intersection point of $\ell_{1,i}$ and $\ell_{2,j}$. We call Φ the *edge-to-point mapping* of G. For $e \in E_p$ we call $\Phi(e)$ the *corresponding point of* e and for $x \in A$ we call $\Phi^{-1}(x)$ the *corresponding edge of* x.

Besides the point edges the set E contains the subsets $E_1 = \{(s, n_{1,i}) : i = 1, \ldots, |L^{(1)}|\}$ and $E_2 = \{(n_{2,j}, t) : j = 1, \ldots, |L^{(2)}|\}$ of directed edges. We call the elements of E_1 and E_2 the *line edges of* G. The complete set of edges of G is given by $E = E_p \cup E_1 \cup E_2$. Figure 1.2 shows the associated graph for the triple $(A, v^{(1)}, v^{(2)})$ from Figure 1.1.

Note that the structure of the associated graph is independent of the projections $p^{(1)}$ and $p^{(2)}$. To use the associated graph G for solving a particular instance of the reconstruction problem, we assign *capacities* to the edges of G. A *capacity function for* G is a mapping $E \longrightarrow \mathbb{N}_0$. We use the following capacity function U:

$$\begin{array}{lcl}
\text{for } i = 1, \ldots, |L^{(1)}|, & & j = 1, \ldots, |L^{(2)}| : \\
U((n_{1,i}, n_{2,j})) & = & 1 \, , \\
U((s, n_{1,i})) & = & p^{(1)}(\ell_{1,i}) \, , \\
U((n_{2,j}, t)) & = & p^{(2)}(\ell_{2,j}) \, .
\end{array}$$

A *flow* in G is a mapping $Y : E \longrightarrow \mathbb{R}_{\geq 0}$ such that $Y(e) \leq U(e)$ for all $e \in E$ and such that for all $v \in V \backslash \{s, t\}$:

$$\sum_{w:\,(w,v) \in E} Y((w,v)) = \sum_{w:\,(v,w) \in E} Y((v,w)) \, .$$

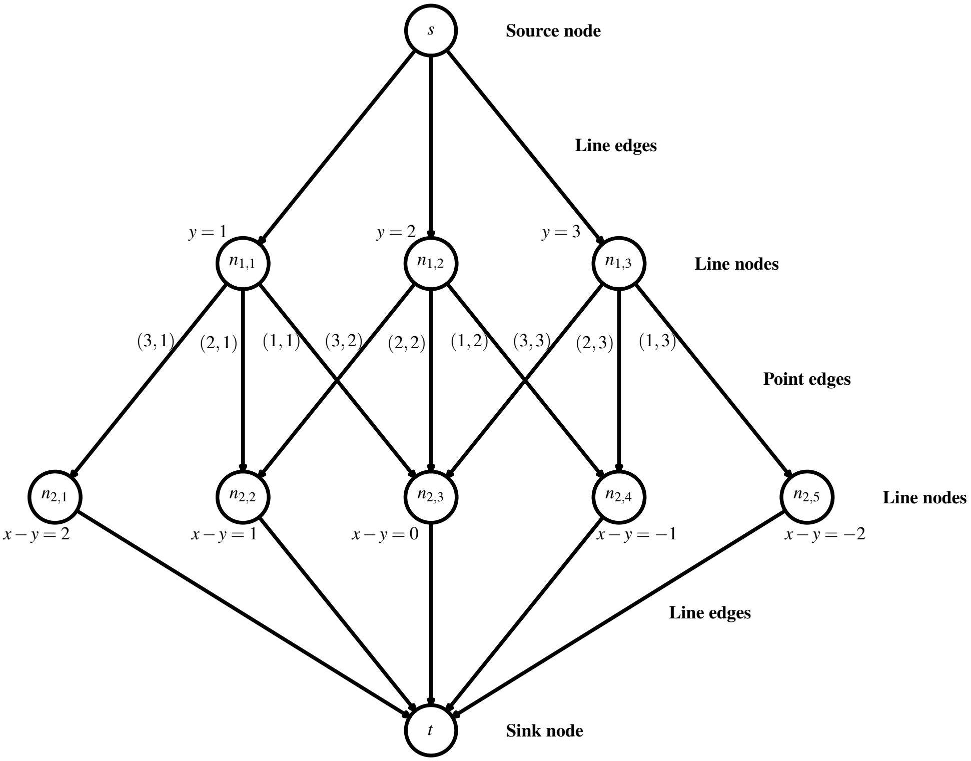


Figure 1.2: *Associated graph G for the triple* $(A, v^{(1)}, v^{(2)})$ *from Figure 1.1.*

The latter constraint is called the *flow conservation constraint*. Flows in graphs are also known as *network flows* in the literature. Let $\mathcal{Y}$ be the set of all flows in G. For a given flow $Y \in \mathcal{Y}$ the *size* $T(Y)$ of Y is given by $T(Y) = \sum_{(s,v)\in E} Y((s,v))$. If we consider G as a network of pipelines, carrying flow from s to t, the size of a flow is the net amount of flow that passes through the network. Due to the flow conservation constraint we also have $T(Y) = \sum_{(v,t)\in E} Y((v,t))$. The associated graph G has a layered structure: all flow that leaves the source s must pass through the point edges. This yields the equality $T(Y) = \sum_{e\in E_p} Y(e)$. If $Y(e) \in \mathbb{N}_0$ for all $e \in E$, we call Y an *integral flow*. Note that for any integral flow Y in the associated graph G, we have $Y(e) \in \{0,1\}$ for all $e \in E_p$, as the capacity of all point edges is 1.

There is an elegant correspondence between the solutions of the reconstruction problem and the integral flows of maximal size (*max flows*) in the associated graph G:

Theorem 1 *Suppose that $S^{(1)} = S^{(2)} =: \bar{T}$. Problem 1 has a solution if and only if there exists an integral flow in G of size $\bar{T}$. Moreover, there is a 1-1 correspondence between the solutions of Problem 1 and the integral flows of size $\bar{T}$ in G.*

Proof. We show first that any integral flow in G of size $\bar{T}$ corresponds to a unique solution of Problem 1. Let Y be a flow in G of size $\bar{T}$. For each $e \in E_p$ we have $Y(e) \in \{0,1\}$. Put $F_Y = \{\Phi(e) : e \in E_p \text{ and } Y(e) = 1\}$, where Φ is the edge-to-point mapping of G. The set F_Y contains all lattice points for which the corresponding point edge in G carries a flow of 1. We call F_Y the *corresponding point set of Y*. We claim that F_Y is a solution of Problem 1. We show that $P^{(1)}_{F_Y} = p^{(1)}$; the proof for direction $v^{(2)}$ is completely analogous.

From the capacity constraints on the line edges of G and the fact that $T(Y) = S^{(1)}$ it follows that all line edges of G must be filled completely by Y. Therefore we have $Y((s,n_{1,i})) = p^{(1)}(\ell_{1,i})$ for all $i = 1,\dots,|L^{(1)}|$. Because of the flow conservation constraint at the line nodes of G we have

$$\sum_{j=1}^{|L^{(2)}|} Y((n_{1,i},n_{2,j})) = p^{(1)}(\ell_{1,i}) \quad \text{for } i = 1,\dots,|L^{(1)}|$$

and therefore

$$|\{\Phi((n_{1,i},n_{2,j})) : Y((n_{1,i},n_{2,j})) = 1\}| = p^{(1)}(\ell_{1,i}) \quad \text{for } i = 1,\dots,|L^{(1)}| \,.$$

From the structure of G it follows that

$$F_Y \cap \ell_{1,i} = \{\Phi((n_{1,i},n_{2,j})) : Y((n_{1,i},n_{2,j})) = 1\} \,.$$

which yields $P^{(1)}_{F_Y}(\ell_{1,i}) = p^{(1)}(\ell_{1,i})$ for $i = 1,\dots,|L^{(1)}|$. To prove that every flow Y of size $\bar{T}$ in G corresponds to a *unique solution* of Problem 1, we note that Y is completely determined by its values on the point edges of G. Therefore a flow $Y' \neq Y$ of size $\bar{T}$ must be different from Y at at least one of the point edges, hence $F_{Y'} \neq F_Y$.

We will now show that the mapping from flows of size $\bar{T}$ in G to solutions of Problem 1 is surjective. For any solution F of Problem 1 define the *corresponding flow* Y_F:

$$Y_F((n_{1,i}, n_{2,j})) = \begin{cases} 1 & \text{if } \Phi((n_{1,i}, n_{2,j})) \in F \ , \\ 0 & \text{otherwise} \ . \end{cases}$$

Specifying Y_F on the point edges completely determines the flow through the remaining edges by the conservation of flow constraint. We call Y_F the *corresponding flow of* F. It is easy to verify that Y_F satisfies all edge capacity constraints. By definition F is the corresponding point set of Y_F. We have $T(Y_F) = \sum_{(v,w) \in E_p} Y((v,w)) = |F|$, so Y_F is a flow of size $|F| = S^{(1)} = \bar{T}$. This shows that the mapping $Y \rightarrow F_Y$ is a bijection. □

The proof of Theorem 1 shows that we can find a solution of Problem 1 by finding an integral flow of size $\bar{T} = S^{(1)} = S^{(2)}$ in the associated graph. This flow is a *maximum flow* in G, because all line edges are completely saturated. Finding a maximum integral flow in a graph is an important problem in operations research and efficient algorithms have been developed to solve this problem, see Section 1.5.

The equivalence between the reconstruction problem for two projections and the problem of finding a maximum flow in the associated graph was already described by Gale in 1957 [8] in the context of reconstructing binary matrices from their row and column sums. Theorem 1 generalizes this result to the case of any reconstruction lattice A and any pair of lattice directions $(v^{(1)}, v^{(2)})$. This is a new result, although it is very similar to the original result from Gale.

In the next sections we will see that the network flow approach can be extended to solve more complex variants of the reconstruction problem and that it can be used as a building block for algorithms that compute a reconstruction from more than two projections.

1.3. Weighted reconstruction

Problem 1 is usually severely underdetermined: the number of solutions can be exponential in the size of the reconstruction lattice A. In practical applications of tomography, the projection data are usually obtained by measuring the projections of an unknown object (the *original object*) and it is important that the reconstruction closely resembles this object. One way to achieve this is to use *prior knowledge* of the original object in the reconstruction algorithm. One of the first attempts to incorporate prior knowledge in the network flow approach was described in [25], in the context of medical image reconstruction.

In this section we consider a weighted version of Problem 1:

Problem 2 *Let A, $v^{(1)}$, $v^{(2)}$, $p^{(1)}$, $p^{(2)}$ be given as in Problem 1. Let $W : A \rightarrow \mathbb{R}$ be a given mapping, the* weight map. *Construct a set $F \subset A$ such that $P_F^{(1)} = p^{(1)}$ and $P_F^{(2)} = p^{(2)}$ and the* total weight *$\sum_{x \in F} W(x)$ is maximal.*

As a shorthand notation we refer to the total weight of F as $W(F)$. Problem 2 is a generalization of Problem 1. Through the weight map one can express a preference for a particular solution if the reconstruction problem has more than one solution. This preference is specified independently for each $x \in A$. The higher the weight $W(x)$, the stronger is the preference to include x in the reconstruction F. Note that a preference for image features that involve several pixels cannot be specified directly through the weight map.

The associated graph G can also be used to solve the weighted version of the reconstruction problem. Define the mapping $C : E \to \mathbb{R}$ as follows:

$$C(e) = \begin{cases} -W(\Phi(e)) & \text{for } e \in E_p \, , \\ 0 & \text{otherwise} \, . \end{cases}$$

The *cost* $C(Y)$ of a flow Y in G is defined as $\sum_{e \in E} C(e)Y(e)$. The *min cost flow problem* in G deals with finding an integral flow Y of a prescribed size $\bar{T}$ in G such that the cost $C(Y)$ is minimal. If we choose $\bar{T} = S^{(1)} = S^{(2)}$, any integral flow Y of size $\bar{T}$ is a maximum flow in G and corresponds to a solution of Problem 1. The total weight of the solution that corresponds to a flow Y equals $-C(Y) = W(F_Y)$. Therefore solving the integral min cost flow problem in G yields a solution of the reconstruction problem of maximum weight, solving Problem 2.

Just as for the max flow problem, efficient algorithms are available for solving the (integral) min cost flow problem. However, most such algorithms assume that the edge costs are integer values. If the edge costs are all in $\mathbb{Q}$, we can simply multiply all edge costs by the smallest common multiple of the denominators to obtain integer costs. If the edge costs are not in $\mathbb{Q}$ the solution of Problem 2 can be approximated by multiplying all edge costs with a large integer and rounding the resulting costs.

In [25] Slump and Gerbrands described an application of Problem 2 to the reconstruction of the left ventricle of the heart from two orthogonal angiographic projections. They used a min cost flow approach to solve a specific instance of Problem 2.

Having the ability to solve Problem 2 can be very helpful in solving a variety of reconstruction problems. We will describe two such problems. These problems deal with the reconstruction of *binary images*, i.e., images for which all pixels are either black or white. Each pixel in the image corresponds to a lattice point. A binary image corresponds to the lattice set $F \subset A$, where F contains the lattice points of all white pixels in the image.

Example 1 As an application of Problem 2, consider an industrial production line, where a large amount of similar objects has to be produced. Suppose that a *blueprint* is available, which specifies what the objects should look like. Occasionally, flaws occur in the production process, resulting in objects that don't match the blueprint. To check for errors, the factory uses a tomographic scanner that scans the objects in two directions: horizontal and vertical. To obtain a meaningful reconstruction from only two projections, the blueprint is used as a *model image*. For each object on the factory line, the reconstruction is computed that matches the blueprint in as many points as possible.

This problem can be formulated in the context of Problem 2. Suppose we want to reconstruct an $n \times n$ image. Put $A = \{1, \ldots, n\} \times \{1, \ldots, n\}$, $v^{(1)} = (1,0)$ and $v^{(2)} = (0,1)$. Let F_M be the lattice set that corresponds to the blueprint. We want to compute the solution F

of Problem 1 such that

$$|(F \cap F_M) \cup (A \backslash F \cap A \backslash F_M)| = |A| - |F \bigtriangleup F_M|$$

is maximal, where $\bigtriangleup$ denotes the symmetric set difference. The term $F \cap F_M$ represents the white pixels shared by F and F_M; the term $A \backslash F \cap A \backslash F_M$ represents the shared black pixels. To formulate this problem as an instance of Problem 2, put

$$W(x) = \begin{cases} 1 & \text{if } x \in F_M \,, \\ 0 & \text{otherwise} \,. \end{cases}$$

The solution of Problem 2 for this weight map maximizes $|F \cap F_M|$, the number of common elements of F and F_M, subject to the constraints $P_F^{(1)} = p^{(1)}$ and $P_F^{(2)} = p^{(2)}$.

For the symmetric difference $F \bigtriangleup F_M$, the following equality holds:

$$|F \bigtriangleup F_M| = (|F| - |F \cap F_M|) + (|F_M| - |F \cap F_M|) \,.$$

Noting that $|F| = S^{(1)}$ is constant for all solutions of Problem 1 yields

$$\begin{aligned} |(F \cap F_M) \cup (A \backslash F \cap A \backslash F_M)| &= \\ |A| - (S^{(1)} - |F \cap F_M|) - (|F_M| - |F \cap F_M|) &= \\ 2|F \cap F_M| + (|A| - |F_M| - S^{(1)}) \,. \end{aligned}$$

The term $(|A| - |F_M| - S^{(1)})$ is constant, which shows that maximizing $|(F \cap F_M) \cup (A \backslash F \cap A \backslash F_M)|$ is equivalent to maximizing $|F \cap F_M|$. We conclude that the given weight map indeed computes the reconstruction that corresponds to the blueprint in as many pixels as possible.

Figure 1.3a shows a blueprint image of 64×64 pixels that represents a semiconductor part. The white pixels correspond to the wiring; the black pixels correspond to the background. Suppose that the object shown in Figure 1.3b passes the scanner. The object clearly contains a gap that is not present in the blueprint and should be detected. Figure 1.3c shows a reconstruction computed from the horizontal and vertical projection data of the faulty object, using the blueprint image of Figure 1.3a. It has the same projections as the image in Figure 1.3b and corresponds to the blueprint in as many pixels as possible. Although the reconstruction is not perfect, the gap is clearly visible and the object can be easily identified as faulty. For comparison, consider the image in Figure 1.3d, which also has the same projections as the images in Figure 1.3b and 1.3c. This time, the reconstruction corresponds to the blueprint in as *few* pixels as possible. Comparing this reconstruction to the original image of the faulty part shows how severely underdetermined the reconstruction problem is when only two projections are available. Of course, using a blueprint image does not guarantee that the reconstruction resembles the scanned object, but it is likely that the reconstruction will be much better than if no prior knowledge is used at all.

In Chapter 2 an evolutionary algorithm is presented for the reconstruction of binary images from two projections. Reconstructing a binary image that adheres to two prescribed projections is a relatively easy task, but many solutions may exist. Therefore, additional

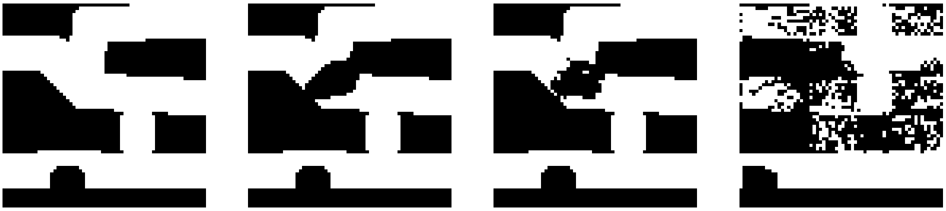


Figure 1.3: *(**a**) Blueprint image of a semiconductor part. (**b**) Test image, containing a gap in one of the wires. (**c**) Reconstruction of the test image from the horizontal and vertical projections, using the image from (a) as a model image. (**d**) Reconstruction using an inverted version of the blueprint image as a model image.*

prior knowledge has to be incorporated which makes the reconstruction more unique. Our evolutionary algorithm is capable of incorporating various types of prior knowledge. The algorithm makes extensive use of the network flow approach to find images that adhere to the prescribed two projections and that have few pixel differences with various model images. Both evolutionary operators that are used by the algorithm (*crossover* and *mutation*) consist of the computation of a model image, followed by the computation of a flow in the associated graph to obtain a new image.

Example 2 Another practical problem that can be formulated in the framework of Problem 2 is how to obtain a 0-1 reconstruction from an already computed real-valued reconstruction. Computing a 0-1 reconstruction from more than two projections is a computationally hard problem, but for computing a real-valued reconstruction several algorithms are available, such as the Algebraic Reconstruction Technique (see Chapter 7 of [19]). These algorithms typically require many projections to compute an accurate reconstruction. Figure 1.4a shows an ART reconstruction of the image in Figure 1.3b from 6 projections. If we want the reconstruction to be binary, this reconstruction can be "rounded", such that all pixel values less than $1/2$ become 0 and all pixel values of $1/2$ or more become 1. The result is shown in Figure 1.4b. A different way to obtain a binary reconstruction is to solve Problem 2 using the pixel values of the original image as the weight-map: the higher the gray value of a pixel in the continuous reconstruction, the higher the preference for this pixel to be assigned a value of 1 in the binary reconstruction. In this way the reconstruction will perfectly satisfy two of the projections, while "resembling" the continuous reconstruction. Figure 1.4c and 1.4d show two such reconstructions. The reconstruction in Figure 1.4c was obtained using $v^{(1)} = (1,0)$, $v^{(2)} = (0,1)$. For the second reconstruction the lattice directions $v^{(1)} = (0,1)$, $v^{(2)} = (1,1)$ were used. Both reconstructions are better than the one in Figure 1.4b at some features, but it is not clear how to detect automatically which one is better, or how the two solutions can be combined into one superior solution. In Section 1.6 we describe how the reconstructions for different pairs of lattice directions can be combined to compute a single, more accurate reconstruction (see Figure 1.9).

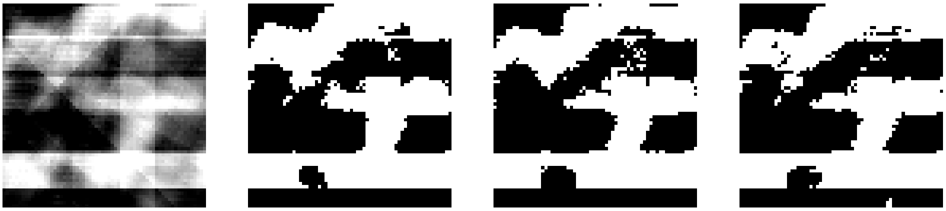


Figure 1.4: *(**a**) ART reconstruction of the image in Figure 1.3b from 6 projections. (**b**) Rounded ART reconstruction. (**c**) Solution of Problem 2, using the ART reconstruction as the weight map, for lattice directions* $(1,0)$ *and* $(0,1)$*. (**d**) Solution of Problem 2 using lattice directions* $(0,1)$ *and* $(1,1)$.

1.4. Reconstruction from noisy projections

The network model from Section 1.2 and 1.3 is only suitable for computing reconstructions from *perfect* projection data. In simulation experiments it is easy to compute perfect projections of a given image, but data that is obtained by physical measurements is usually polluted by noise. As an example of what happens in the network of Section 1.2 when the projection data contains errors, consider the possibility that $S^{(1)} \neq S^{(2)}$. In this case it is clear that no perfect solution of the reconstruction problem exists. One can still compute a maximum flow in the associated graph G. Due to the line arc capacity constraints such a flow will always have size at most $\min(S^{(1)}, S^{(2)})$. If the measured projection for a line ℓ is lower than the number of points on that line in the original object, that line will always contain too few points in the reconstruction, regardless of the measured line projections in the other direction, because of the capacity constraint on the corresponding line edge of ℓ.

In this section we consider a modification of the associated graph which can be used to compute a reconstruction F for which the norm of the residue, i.e., the difference between the projections of F and the two prescribed projections is minimal. This network does not have the drawbacks that we described above of the network from Section 1.2.

Let $F \subset A$. For $k = 1,2$, the projections $P_F^{(k)}$ of F have finite domains, so we can regard $P_F^{(k)}$ as a vector of $|L^{(k)}|$ elements. We denote the sum-norm of this vector by $|P_F^{(k)}|_1$. For a given prescribed projection $p^{(k)}$ the norm

$$|P_F^{(k)} - p^{(k)}|_1 = \sum_{\ell \in L^{(k)}} |P_F^{(k)}(\ell) - p^{(k)}(\ell)|$$

equals the total summed projection difference over all lines in $L^{(k)}$. Another norm that is often used in tomography is the Euclidean norm $|\cdot|_2$. The sum-norm is better suited for incorporation in the network flow approach. We now define a generalization of Problem 1 that allows for errors in the prescribed projections.

Problem 3 *Let* A, $v^{(1)}$, $v^{(2)}$, $p^{(1)}$, $p^{(2)}$ *be given as in Problem 1. Let* $\bar{T} \in \mathbb{N}_0$.
Construct a set $F \subset A$ *with* $|F| = \bar{T}$ *such that* $|P_F^{(1)} - p^{(1)}|_1 + |P_F^{(2)} - p^{(2)}|_1$ *is minimal.*

Problem 3 asks for a set F which has a prescribed number of $\bar{T}$ elements such that F

corresponds as well as possible to the two prescribed projections, according to the sum-norm. If Problem 1 has a solution, we can find all solutions by putting $\bar{T} = S^{(1)}$ and solving Problem 3. We will show that Problem 3 can be solved within the network flow model. For any n-dimensional vector $p \in \mathbb{R}^n$, define

$$|p|^+ = \sum_{i=1}^{n} \max(p_i, 0) .$$

To solve Problem 3 we need to make some modifications to the associated graph. Before introducing the modified graph we prove the following lemma.

Lemma 1 *Let $F \subset A$, $|F| = \bar{T}$. Then, for $k = 1, 2$,*

$$|P_F^{(k)} - p^{(k)}|_1 = 2|P_F^{(k)} - p^{(k)}|^+ + S^{(k)} - \bar{T} .$$

Proof. Let $k \in \{1, 2\}$. By definition we have

$$|P_F^{(k)} - p^{(k)}|_1 = |P_F^{(k)} - p^{(k)}|^+ + |p^{(k)} - P_F^{(k)}|^+ .$$

For each line $\ell \in L^{(k)}$ we have

$$P_F^{(k)}(\ell) = p^{(k)} + \max(P_F^{(k)}(\ell) - p^{(k)}(\ell), 0) - \max(p^{(k)}(\ell) - P_F^{(k)}(\ell), 0) .$$

Summing this equation over all lines $\ell \in L^{(k)}$, it follows that

$$\bar{T} = S^{(k)} + |P_F^{(k)} - p^{(k)}|^+ - |p^{(k)} - P_F^{(k)}|^+ ,$$

hence

$$|P_F^{(k)} - p^{(k)}|_1 = 2|P_F^{(k)} - p^{(k)}|^+ + S^{(k)} - \bar{T} . \tag{1.1}$$

□

Lemma 1 shows that solving Problem 3 is equivalent to finding a set F with $|F| = \bar{T}$ for which

$$|P_F^{(1)} - p^{(1)}|^+ + |P_F^{(2)} - p^{(2)}|^+$$

is minimal, since $S^{(1)}$, $S^{(2)}$ and $\bar{T}$ are constant.

We will now describe how the associated graph can be modified for solving Problem 3. The network from Section 1.2 forms the basis for the new network. From this point on we refer to the line edges of the network from Section 1.2 as *primary line edges*. As before, we denote the sets of all primary line edges for directions $v^{(1)}$ and $v^{(2)}$ by E_1 and E_2 respectively. Let $\ell \in L^{(k)}$ be any lattice line for direction $v^{(k)}$ and let $e \in E_k$ its corresponding primary line edge. The capacity of e imposes a hard upper bound on the number of points on ℓ in the network flow reconstruction. To relax this hard constraint we add a second edge for each lattice line, the *excess edge*. The excess edges are parallel to their corresponding primary line edges and have the same orientation. We denote the set of excess edges for directions $v^{(1)}$ and $v^{(2)}$ by E'_1 and E'_2 respectively. The resulting graph G' is shown in Figure 1.5. The capacities of the primary line edges remain unchanged. The excess edges have unbounded capacities. Effectively this means that the total flow through a primary line edge and its corresponding excess edge — both belonging to a line $\ell \in L^{(k)}$ — is bounded by $|A \cap \ell|$, as all outgoing flow from the line edges must pass through $|A \cap \ell|$ point edges that each have capacity 1. Therefore it is still possible to assign finite capacities to the excess edges.

The primary line edges of the new graph are still assigned a cost of 0, as in the original network. The excess edges are assigned a cost of K, where K is a positive constant. In this way it is possible to allow more points on a line ℓ than $p^{(k)}(\ell)$, but only at the expense of a cost increase.

Now consider the problem of finding a min cost flow in G' of size $\bar{T}$. Without computing such a flow, we can already be sure that any excess edge will only carry flow if its corresponding primary line edge is saturated up to its capacity. Otherwise the cost could be decreased by transferring flow from the excess edge to the primary edge.

Suppose that $Y : E \to \mathbb{N}_{\geq 0}$ is a min cost flow in G' of size $\bar{T}$. The total cost of Y given by

$$C(Y) = K\Big(\sum_{e \in E'_1} Y(e) + \sum_{e \in E'_2} Y(e)\Big) .$$

Let F_Y be the set of points for which the corresponding point edges in Y carry a positive flow, as in the proof of Theorem 1. For any line $\ell \in L^{(k)}$, the total flow through the primary and excess edge of ℓ must equal $P^{(k)}_{F_Y}(\ell)$, because of the flow conservation constraints. Therefore we have

$$\sum_{e \in E'_k} Y(e) = |P^{(k)}_{F_Y} - p^{(k)}|^+ ,$$

hence

$$C(Y) = K(|P^{(1)}_{F_Y} - p^{(1)}|^+ + |P^{(2)}_{F_Y} - p^{(2)}|^+) .$$

Applying Lemma 1, we conclude that a min cost flow in G' of size $\bar{T}$ yields a solution of Problem 3.
The new network can also be used to solve an extended version of Problem 2.

Problem 4 *Let A, $v^{(1)}$, $v^{(2)}$, $p^{(1)}$, $p^{(2)}$ be given as in Problem 2. Let $\bar{T} \in \mathbb{N}_0$, $\alpha \in \mathbb{R}_{>0}$. Construct a set $F \subset A$ with $|F| = \bar{T}$ such that*

$$\alpha(|P^{(1)}_F - p^{(1)}|_1 + |P^{(2)}_F - p^{(2)}|_1) - \sum_{x \in F} W(x)$$

is minimal.

Similar to the procedure for solving Problem 2, we set $C(e) = -W(\Phi(e))$ for all $e \in E_p$. Assuming that an excess edge only carries flow if its corresponding primary line edge is completely full, the total cost of an integral flow $Y \in \mathcal{Y}$ now becomes

$$C(Y) = K(|P^{(1)}_{F_Y} - p^{(1)}|^+ + |P^{(2)}_{F_Y} - p^{(2)}|^+) - \sum_{x \in F_Y} W(x) .$$

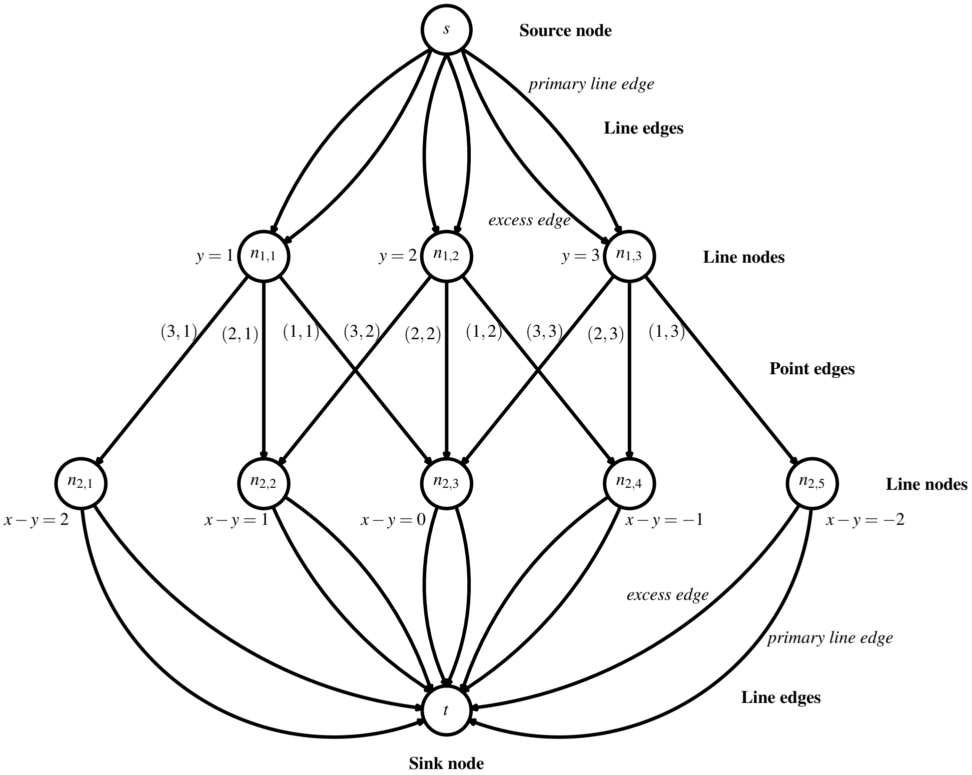


Figure 1.5: *Modified associated graph G' for the triple $(A, v^{(1)}, v^{(2)})$ from Figure 1.1.*

Setting $K = 2\alpha$ and using Equation (1.1) yields

$$C(Y) = \alpha(|P^{(1)}_{F_Y} - p^{(1)}|_1 + |P^{(2)}_{F_Y} - p^{(2)}|_1) - \sum_{x \in F_Y} W(x) - C_0 \,,$$

where C_0 is a constant. We conclude that if Y is an integral min cost flow of size $\bar{T}$ in G' then F_Y is a solution to Problem 4.

1.5. Algorithms and implementation

As described in the previous sections, Problem 1, 2, 3 and 4 can all be solved as instances of network flow problems. Both the max flow problem and the min cost flow problem have been studied extensively. The book [1] provides an overview of available algorithms. A survey of the time complexities of various network flow algorithms can be found in [23] (max flow: Chapter 10; min cost flow: Chapter 12).

We now assume that the reconstruction lattice A is a square of size $N \times N$ and we fix a pair $(v^{(1)}, v^{(2)})$ of lattice directions. It is clear that the number of points in A on each lattice line parallel to $v^{(1)}$ or $v^{(2)}$ is $O(N)$ and that the number of such lattice lines is also $O(N)$.

Problem 1 can be solved as an instance of the max flow problem in the associated graph. In [12], Goldberg and Rao describe an algorithm to compute a maximum flow in a graph with n nodes, m edges and maximum edge capacity c in $O(n^{2/3} m \log(n^2/m) \log c)$ time. The associated graph of the triple $(A, v^{(1)}, v^{(2)})$ has $n = O(N)$ nodes, $m = O(N^2)$ edges and a maximum edge capacity of $c = O(N)$. Therefore Problem 1 can be solved in $O(N^{8/3} \log N)$ time.

Problem 2 and 3 can both be solved as an instance of the min cost flow problem, i.e., the problem of finding a flow of fixed size that has minimal cost. The min cost flow problem can be reformulated as a minimum-cost circulation problem, by adding an edge from the sink node T to the source node S, see Section 12.1 of [23]. In [13], Goldberg and Tarjan describe an algorithm to compute a minimum-cost circulation in a graph with n nodes, m edges and maximum (integral) edge cost K in $O(nm \log(n^2/m) \log(nK))$ time. For the associated graph from Section 1.3, as well as for the modified associated graph from Section 1.4 this yields a time complexity of $O(N^3 \log(NK))$ for solving the min cost flow problem.

The problem of finding a maximum flow in the associated graph is known in the literature as *simple b-matching*. A flow that saturates all line edges is called a *perfect simple b-matching* and the weighted variant of finding a perfect b-matching is known as *perfect weighted b-matching*, see Chapter 21 of [23]. For these particular network flow problems, special algorithms have been developed that are sometimes faster than general network flow algorithms.

Implementing fast network flow algorithms is a difficult and time consuming task. The fastest way to use such algorithms is to use an existing, highly optimized implementation. Several network flow program libraries are available, some commercially and some for free. The ILOG CPLEX solver [16] performs very well for a wide range of network flow problems. The CS2 library from Goldberg [11] performs well and is free for non-commercial use. The same holds for the RelaxIV library from Bertsekas [5].

```
Compute the start solution $F^0$;
$i := 0$;
while (stop criterion is not met) do
begin
    $i := i + 1$;
    Select a pair of directions $v_a$ and $v_b$ $(1 \leq a < b \leq n)$;
    Compute a new weight-map $W^i$ from the previous solution $F^{i-1}$;
    Compute a new solution $F^i$ by solving Problem 2 for
    directions $v_a$ and $v_b$, using the weight map $W^i$;
end
```

Figure 1.6: *Basic steps of the algorithm.*

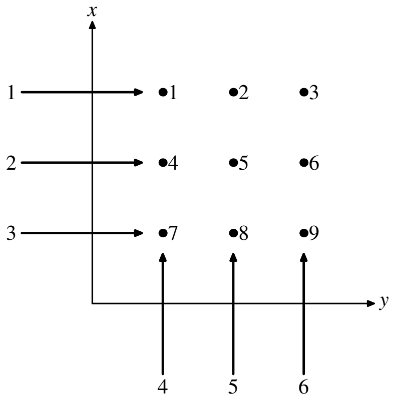
1.6. Extension to more than two projections

As shown in the previous sections, the reconstruction problem from two projections is well understood and can be solved efficiently. We now move to the case where more than two projections are available.

Problem 5 *Let $n > 2$ and let $v^{(1)}, \ldots, v^{(n)}$ be given distinct lattice directions. Let $A \subset \mathbb{Z}^2$ be a given lattice set. For $k = 1, \ldots, n$, let $p^{(k)} : L^{(k)} \to \mathbb{N}_0$ be given functions. Construct a set $F \subset A$ such that $P_F^{(k)} = p^{(k)}$ for $k = 1, \ldots, n$.*

When more projections are available the reconstruction problem is less underdetermined and we would like to be able to use the additional projections to increase the reconstruction quality. However, the reconstruction problem for more than two projections is NP-hard [10]. Therefore we have to resort to approximation algorithms. In this section we will describe an iterative algorithm that uses only two projections in each iteration. Within an iteration, a new pair of projections is first selected. Subsequently, an instance of Problem 2 is solved to obtain a reconstruction that satisfies the current two projections. The reconstruction from the previous iteration, which was computed using a different pair of projections, is used to construct the weight map of Problem 2, in such a way that the new reconstruction will resemble the previous one. In this way the other projections are incorporated in the reconstruction procedure in an implicit way.

Figure 1.6 describes the basic structure of the algorithm. In Chapter 3 we present a thorough description of the algorithm, along with extensive experimental results. In the next subsections we give a global description of each of the algorithmic steps. The algorithm relies heavily on the methods for solving two-projection subproblems, that we described in the previous sections.



$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{pmatrix}$$

Figure 1.7: *(***a***) (left) Numbering scheme for the lattice points and the lattice lines in a rectangular reconstruction lattice. (***b***) (right) System of equations corresponding to the numbering in (a).*

1.6.1. Computing the start solution

At the start of the algorithm there is no “previous reconstruction”; a start solution has to be computed for the iterative algorithm. Ideally, the start solution should satisfy two criteria:

- **Accuracy**. The start solution should correspond well to the prescribed projection data.
- **Speed**. The start solution should be computed fast (compared with the running time of the rest of the algorithm).

These are conflicting goals. Computing a highly accurate binary reconstruction will certainly take too much time, as the reconstruction problem is NP-hard.

There are several options for computing the start solutions, each having a different trade-off between speed and accuracy. The first option is to choose the empty set $F^0 = \emptyset$ as a start solution, i.e., an image that is completely black.

A better alternative is to use a very fast approximate reconstruction algorithm, such as one of the greedy algorithms described in [14]. The running time of these algorithms is comparable to the time it takes to solve a single network flow problem in the body of the main loop of our algorithm.

A third possibility is to start with a continuous reconstruction. A binary start solution can then be computed from the continuous reconstruction, as described in Example 2 of Section 1.3. One class of reconstruction algorithms that can be used consists of the *algebraic reconstruction algorithms* (see Chapter 7 of [19]). The basic idea of these algorithms is to describe Problem 5 as a system of linear equations:

$$Mx = b\,. \tag{1.2}$$

Figure 1.7 shows an example 3×3 grid with the corresponding system of equations for two directions, $v^{(1)} = (1,0)$ and $v^{(2)} = (0,1)$. Each entry of the vector x represents an element of A. The entries of the vector b correspond to the line projections for lattice directions $v^{(1)}, \ldots, v^{(n)}$. Each row of the binary matrix M represents a lattice line. The entry M_{ij} is 1 if and only if its corresponding lattice line i passes through point j.

The system (1.2) is usually underdetermined. The shortest solution of the system with respect to the Euclidean norm $|\cdot|_2$, which we denote as x^*, is a good choice for a start solution in discrete tomography. It can be shown that if Problem 5 has several solutions then the Euclidean distance of x^* to any of these solutions is the same, so x^* is "centered" between the solutions. In addition, if the system (1.2) has binary solutions, any of these solutions has minimal norm among all integral solutions, see [15]. Therefore a short solution is likely to be a good start solution. We refer to Chapter 3 for the details of these arguments. The shortest solution of (1.2) can be computed efficiently by iterative methods, as described in [26]. After this solution has been computed, a pair $(v^{(a)}, v^{(b)})$ of lattice directions has to be selected for computing the binary start solution. The start solution is computed by solving Problem 2, using the pixel values in x^* as the weight map.

1.6.2. Computing the weight map

In each iteration of the main loop an instance of Problem 2 is solved. The weight map for this reconstruction problem is computed from the reconstruction of the previous iteration; it does not depend on the selected pair of lattice directions.

The weight map should be chosen in such a way that the new reconstruction resembles the reconstruction from the previous iteration. In the new instance of Problem 2, only two of the projections are used. If the new reconstruction is similar to the previous reconstruction, which was computed using a different pair of projections, the new image will also approximately adhere to the prescribed two projections from the previous iteration. Repeating this intuitive argument we would hope that the new image also satisfies the projections from the iteration before the previous one, from the iteration before that one, etc.

The most straightforward way to make the new reconstruction resemble the previous one is to follow the approach from Example 1 in Section 1.3. If we put

$$W^i((x,y)) = \begin{cases} 1 & \text{if } (x,y) \in F^{i-1} , \\ 0 & \text{otherwise} , \end{cases}$$

the new reconstructed image F^i will have the same pixel value as F^{i-1} in as many pixels as possible. Unfortunately, this choice usually does not lead to good results. Typically, the main loop of the algorithm does not converge, making it difficult to decide when the algorithm should be terminated. This behaviour is by no means surprising. The reconstruction problem from a small number of projections is severely underdetermined. If no additional prior knowledge is used, a small number of projections (e.g., four, five) may not even yield nearly enough data to uniquely determine a reconstruction.

To deal with this problem we focus on the reconstruction of images that satisfy additional properties. *Smoothness* is a property that can often be observed in practical images: images consist of large areas that are completely black or completely white, instead of exhibiting completely random pixel patterns. A nice property of the smoothness concept is that it can be measured *locally*. We say that an image F is *perfectly smooth* at pixel $x \in A$

if all neighbouring points of x have the same value as x. Of course this notion requires the definition of a *neighbourhood* of x, which we will describe below.

From this point on we assume that the reconstruction lattice A is rectangular. If this assumption is not satisfied, we can use any square reconstruction lattice A' for which $A \subset A'$, as increasing the size of the reconstruction lattice does not affect the projections.

Let F^{i-1} be the reconstructed image from the previous iteration. As a neighbourhood of the point $p = (x_p, y_p) \in A$ we choose a square centered in (x_p, y_p). The reason for preferring a square neighbourhood over alternatives is that the required computations can be performed very efficiently in this case. Let $p = (x_p, y_p) \in A$. Let r be a positive integer, the *neighbourhood radius*. Put

$$N_p = \{ (x,y) \in A : x_p - r \leq x \leq x_p + r, y_p - r \leq y \leq y_p + r \} .$$

N_p contains all pixels in the neighbourhood of p, including p. In case p is near the boundary of A, the set N_p may contain fewer than $(2r+1)^2$ pixels. Let s_p be the number of pixels $q \in N_p$ for which $F(p) = F(q)$. Define

$$f_p = \frac{s_p}{|N_p|} .$$

We call f_p the *similarity fraction* of p. A high similarity fraction corresponds to a smooth neighbourhood of p.

Let $g : [0,1] \to \mathbb{R}_{>0}$ be a nondecreasing function, the *local weight function*. This function determines the preference for locally smooth regions. We compute the pixel weight $W(p)$ of p as follows:

$$W(p) = 2(F(p) - \frac{1}{2})g(f_p) .$$

Note that $2(F(p) - \frac{1}{2})$ is either -1 or $+1$.

When we take $g(f_p) = 1$ for all $f_p \in [0,1]$, there is no preference for local smoothness. To express the preference we make the local weight function g an increasing function of f_p. Now a pixel having a value of 1 that is surrounded by other pixels having the same value will obtain a higher weight than such a pixel that is surrounded by 0-valued pixels. A higher weight expresses a preference to retain the value of 1 in the next reconstruction. The same reasoning holds for pixels having a value of 0, except that in this case the pixel weights are negative. Three possible choices for the local weight function are

- $g(f_p) = f_p$.
- $g(f_p) = \sqrt{f_p}$.
- $g(f_p) = f_p^2$.

The latter choice results in a strong preference for pixels that are (almost) perfectly smooth. Of course many other local weight functions are possible. In Chapter 3 extensive results are reported for the local weight function

$$g(f_p) = \begin{cases} 1 & (f_p \leq 0.65)\ , \\ 4f & (0.65 < f_p < 1)\ , \\ 9 & (f_p = 1)\ . \end{cases} \tag{1.3}$$

The choice for this particular function is somewhat arbitrary. In each case, a preference is expressed for retaining the pixel value of p in the next reconstruction, instead of changing it. In the case that the whole neighbourhood of p has the same value as p, this preference is very strong. If the neighbourhood contains a few pixels having a different value, the preference is less. If there are many pixels in the neighbourhood that have a different value, the preference is even smaller.

So far we have not discussed how the neighbourhood radius should be chosen. If the start solution is already a good approximation of the final reconstruction, using a fixed value of $r = 1$ works well. For this neighbourhood radius the differences between consecutive reconstructions F^i are typically small. It is usually better to start with a larger neighbourhood radius, e.g., $r = 5$ or $r = 8$. This typically results in large changes between consecutive reconstructions. Only very large regions of pixels that have the same value obtain a strong preference to keep this value. Regions that are less smooth can easily change. A choice that works well for the range of images studied in Chapter 3 is to start the algorithm with $r = 8$ and to set $r = 1$ after 50 iterations.

1.6.3. Choosing the pair of directions

In each iteration of the main loop of the algorithm a new pair of lattice directions is selected. There is no selection scheme which is “obviously best” in all cases. Yet, there are several ways for choosing the direction pairs that perform well in practice.

A good choice for the new direction pair is to choose the lattice directions $v^{(a)}$, $v^{(b)}$, for which the total projection error

$$|P_F^{(a)} - p^{(a)}|_1 + |P_F^{(b)} - p^{(b)}|_1$$

is largest. After solving the new instance of Problem 2 the total projection error for these two lattice directions will be zero, assuming perfect projection data. This also guarantees that if at least two projections have a positive projection error after the previous iteration, both new lattice directions will be different from the ones used in the previous iteration.

If the number of projections is very small (e.g., four, five) the projection error is not a good criterion for selecting the new projection pair. For the case of four projections this scheme leads to cycling behaviour between two pairs of projections. The other projection pairs are not used at all. To avoid this behaviour it is better to use a fixed order of direction pairs, in which all pairs occur equally often. Such schemes, for four and five projections, are shown in Table 1.1.

iteration	1	2	3	4	5	6
1st dir.	1	3	1	2	1	2
2nd dir.	2	4	3	4	4	3

iteration	1	2	3	4	5	6	7	8	9	10
1st dir.	1	3	5	2	4	1	2	3	4	5
2nd dir.	2	4	1	3	5	3	4	5	1	2

Table 1.1: *(***a***) (top) Lattice direction scheme for four projections. Each projection pair is used equally often. No projection pair is used in two consecutive iterations. (***b***) (bottom) Lattice direction scheme for five projections.*

1.6.4. Stop criterion

In general it is not easy to determine when the iterative algorithm from Figure 1.6 should terminate, because there is no guaranteed convergence. Yet, the experiments in Chapter 3 show that if enough projections are available the algorithm often converges to the exact solution of Problem 5. Detecting that an exact solution has been found is easy and the algorithm always terminates in that case.

To measure the quality of the current reconstruction F the *total projection difference*

$$D(F) = \sum_{k=1}^{n} |P_F^{(k)} - p^{(k)}|_1$$

can be used. This distance is 0 for any perfect solution of Problem 5 and greater than 0 otherwise. The total projection difference can be used for defining termination conditions. If no new minimal value is found for the total projection distance during the last T iterations, where T is a positive integer, the algorithm terminates. We used $T = 100$ for all experiments in the next subsection.

1.6.5. Some results

We will now show some experimental results obtained by the iterative algorithm from Figure 1.6. The performance of the algorithm, as for any other general discrete tomography algorithm, depends heavily on the type of image that is being reconstructed and the number of available projections. In order to give extensive statistical results about the reconstruction quality, a *class of images* has to be defined first. All images in this class should have similar characteristics. The performance of the algorithm can then be measured for this particular image class. In Chapter 3 reconstruction results are reported for several different image classes. The results in this section show a varied set of test images with their reconstructions, rather than providing extensive quantitative results. Figure 1.8 shows six test images, each having different characteristics, and their reconstructions. The number of projections used is shown in the figure captions. The reconstructions of the first five images (a-e) are all *perfect reconstructions*, obtained using the weight function in Equation (1.3) from Section

1.6.2. For the first four images (a-d), the linear local weight function also works very well, even faster than the function in Equation (1.3). The image from Figure 1.8e contains many fine lines of only a single pixel thickness. In this case the local weight function $g(f_p) = \sqrt{f_p}$ works well. Which local weight function is best for a certain class of image depends strongly on characteristics of the particular class. This is also true for the *number of projections* that is required to reconstruct an image.

For reconstructing the image in Figure 1.8a, four projections suffice. The structure of the object boundary is fairly complex, but the object contains no "holes". The iterative algorithm reconstructed the image perfectly from four projections (horizontal, vertical, diagonal and anti-diagonal).

Figure 1.8b shows a much more complicated example. The object contains many cavities of various sizes and has a very complex boundary. Some of the black holes inside the white region are only a single pixel in size. In this case, our algorithm requires six projections to compute an accurate reconstruction. Some of the fine details in this image are not smooth at all. Still, the fine details are reconstructed with great accuracy. The image from Figure 1.8c is even more complex. It requires eight projections to be reconstructed perfectly.

The image from Figure 1.8d has a lot of local smoothness in the black areas, but it contains no large white areas. Still, the image is smooth enough to be reconstructed perfectly from only five projections. This example also illustrates that very fine, non-smooth details can be reconstructed by the algorithm, as long as the entire image is sufficiently smooth.

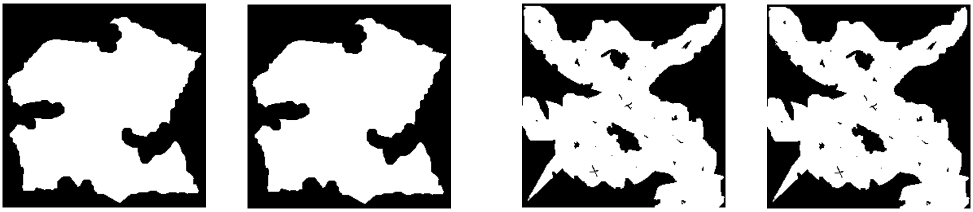
Figure 1.8e shows a vascular system, containing several very thin vessels. The iterative algorithm can reconstruct the original image perfectly from twelve projections. This is quite surprising, since the very thin vessels have a width of only one pixel, so they are not smooth. Still, the smoothness of the thicker vessels and the background area provides the algorithm with enough guidance to reconstruct the original image.

When the image contains no structure at all, the algorithm performs very badly. Figure 1.8f shows an image of random noise. The reconstruction from 12 projections shows that our algorithm has a preference for connected areas of white and black pixels. In this case, however, the smoothness assumption is obviously not satisfied by the original image. The distance between the projections of the image found by our algorithm and the prescribed projections is very small, however.

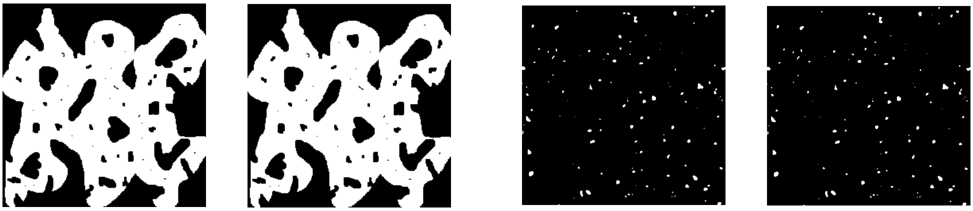
For reconstructing the images in Figure 1.8, a sufficiently large set of projections was used. Figures 1.9 and 1.10 demonstrate the result of using the algorithm if too few projections are available.

Figure 1.9 shows the results of reconstructing the semiconductor image of Figure 1.3b from three and four projections respectively. When using only three projections a reconstruction is found that has exactly the prescribed projections, but the reconstruction is very different from the original image.

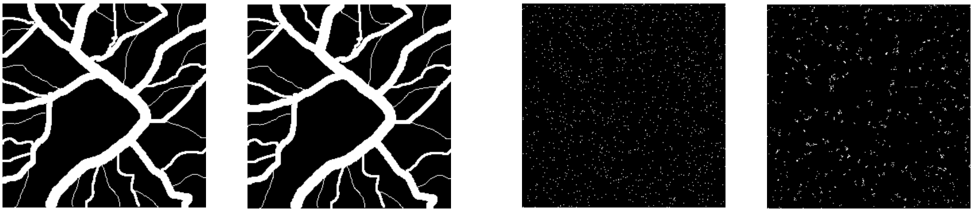
If too few projections are used the algorithm may also get stuck in a local minimum of the projection distance, which is shown in Figure 1.10. The original image can be reconstructed perfectly from five projections, but when only four projections are used the algorithm fails to find a good reconstruction. The projections of the reconstructed image are significantly different from the four prescribed projections, yet the algorithm is unable to find a better reconstruction.



(**a**) Original image. Reconstr., $n = 4$. (**b**) Original image. Reconstr., $n = 6$.



(**c**) Original image. Reconstr., $n = 8$. (**d**) Original image. Reconstr., $n = 5$.



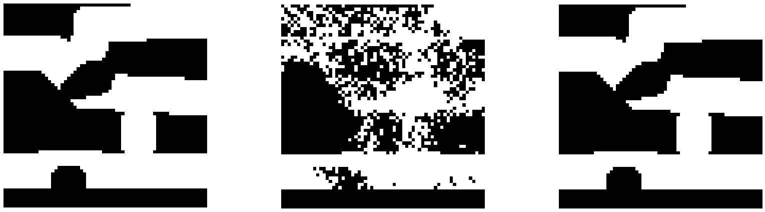
(**e**) Original image. Reconstr., $n = 12$. (**f**) Original image. Reconstr., $n = 12$.

Figure 1.8: *Six original images and their reconstructions. The number n of projections is shown in the figure caption.*

1.7. Reconstructing 3D volumes

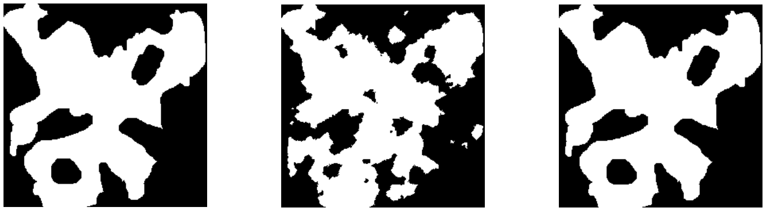
So far our approach has been concerned with the reconstruction of 2-dimensional images. In many practical applications of tomography it is important to obtain 3D reconstructions. Computing 3D reconstructions is usually a task that is computationally very demanding, as large amounts of data are involved. There is a slight difference in terminology between 2D and 3D reconstructions. *Pixels* in 2D images are usually called *voxels* in the context of 3D images, where they represent a unit cube in the 3D volume.

If there exists a plane H in $\mathbb{Z}^3$ such that all projection directions lie in H, all algorithms for 2D reconstruction can be used directly for 3D reconstruction as well, reconstructing the volume as a series of slices. All slices can be reconstructed in parallel, which allows for a large speedup if several processors are used. A disadvantage of reconstructing all slices independently is that certain types of prior knowledge cannot be exploited. For example, if we



Original image. Reconstruction, $n = 3$. Reconstruction, $n = 4$.

Figure 1.9: *(***a***) Original image. (***b***) Reconstruction from three projections (horizontal, vertical, diagonal) that has exactly the prescribed projections. (***c***) Perfect reconstruction of the original image from four projections (horizontal, vertical, diagonal, anti-diagonal).*



Original image. Reconstruction, $n = 4$. Reconstruction, $n = 5$.

Figure 1.10: *(***a***) Original image. (***b***) Reconstruction from four projections, which does not have the prescribed projections. (***c***) Perfect reconstruction of the original image from five projections.*

generalize the preference for local smoothness from Section 1.6 to 3D, voxels from adjacent slices are required to compute the neighbourhood density of a certain voxel. Therefore, the reconstruction computations of the slices are no longer independent.

If the projection directions are not co-planar, reconstructing the volume as a series of slices is not possible. This situation occurs, for example, in the application of atomic resolution electron microscopy. The crystal sample is tilted in two directions to obtain as many useful projections as possible.

Figure 1.11 shows an example of a $3 \times 3 \times 3$ volume A with its projections parallel to the lattice directions $v^{(1)} = (1,0,0)$ and $v^{(2)} = (0,1,0)$. Lattice points that have a value of 1 (i.e., lattice points included in the set F) are indicated by large dots. Similar to the associated network from Section 1.2, each two-projection problem in 3D also has an associated graph. The associated graph for the volume in Figure 1.11 is shown in Figure 1.12. Just as in the 2D case, the associated graph contains a line edge for every projected lattice line. The middle layer of edges contains one edge for every voxel, connecting the two line nodes for which the corresponding lines intersect with that voxel.

Figure 1.12 shows a nice property of the two-projection reconstruction problem. For any point edge $(n_{1,i}, n_{2,j}) \in E_p$ in the associated graph, the lines $\ell_{1,i}$ and $\ell_{2,j}$ have a nonempty intersection in A, so there is a plane in $\mathbb{Z}^3$ which contains both $\ell_{1,i}$ and $\ell_{2,j}$. Since $\ell_{1,i}$ and $\ell_{2,j}$

are translates of $v^{(1)}$ and $v^{(2)}$ respectively, this plane will always be a translate of the plane spanned by $v^{(1)}$ and $v^{(2)}$. If two lines $\ell_{1,i}$ and $\ell_{2,i}$ lie in different translates of this plane, there will be no voxel edge connecting the corresponding line nodes. Therefore, the max flow problem can be solved for each translate of the plane independently. In the example network of Figure 1.12 the subproblems for each of the planes $z = 0$, $z = 1$ and $z = 2$ can be solved independently. This property holds for any pair $(v^{(a)}, v^{(b)})$ of lattice directions, although the sizes of the subproblems depend on the direction pair. The number of point edges in each subproblem is bounded by the maximal number of voxels in A that lie in a single plane.

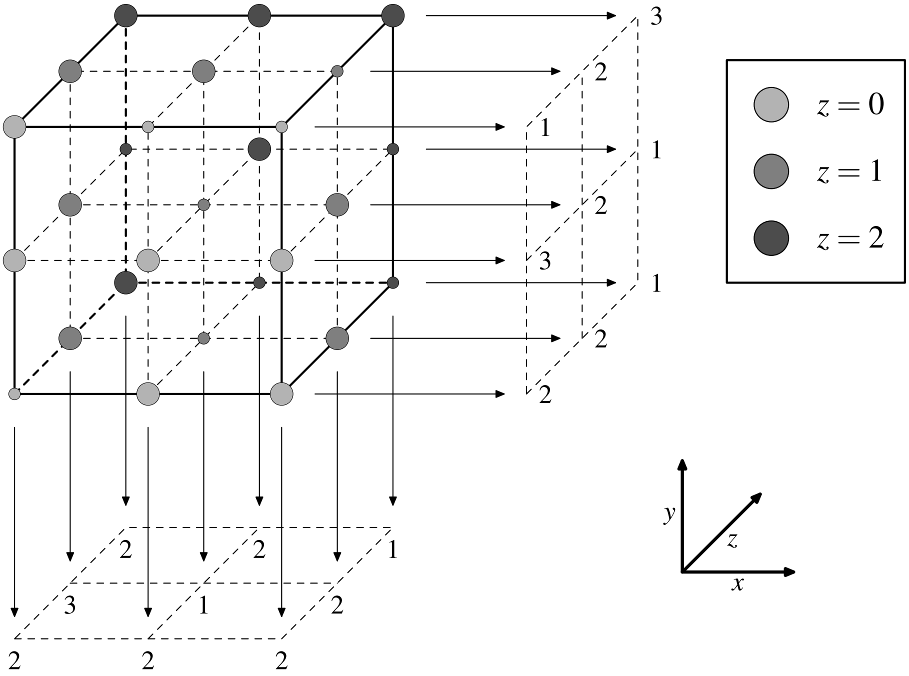


Figure 1.11: $3 \times 3 \times 3$ *binary volume with its projections in directions* $v^{(1)} = (1,0,0)$ *and* $v^{(2)} = (0,1,0)$. *A large circle indicates a value of* 1*; a small circle indicates a value of* 0.

Figure 1.13 shows a cubic test volume, displayed from three different viewing directions. The directions were selected to provide a clear view of the volume; they are not parallel to any of the projection directions. The iterative network flow algorithm can reconstruct the original image perfectly from projections along the four lattice directions $(1,0,0), (0,1,0), (0,0,1), (1,1,0)$. The image dimensions are shown in the figure. The test volume is surrounded by a black background, which is not counted in the image dimensions.

In Chapter 4 we describe the algorithm for 3D reconstruction in more detail and present various reconstruction results.

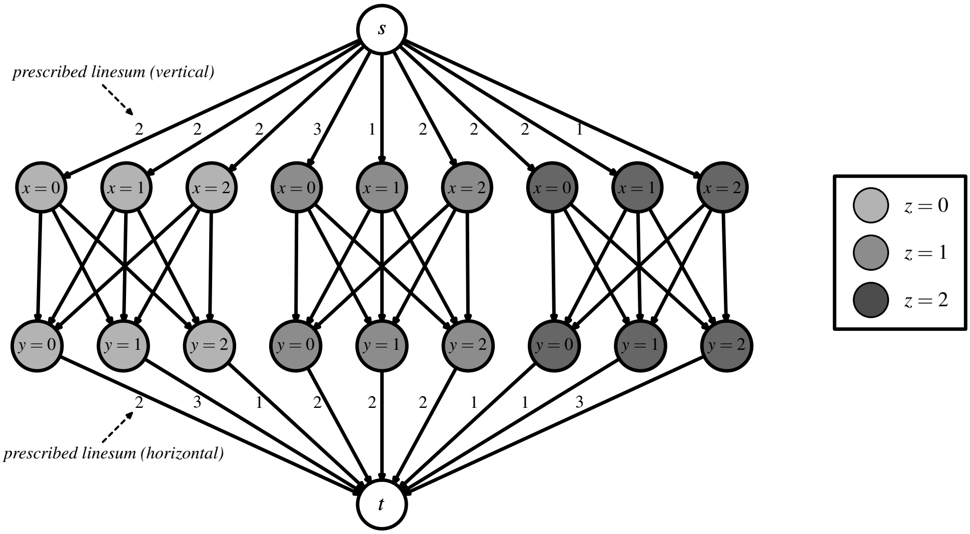


Figure 1.12: *Network corresponding to the two-projection reconstruction problem in Figure 1.11.*

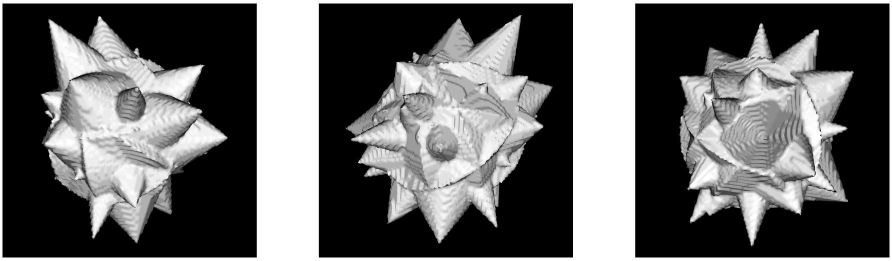


Figure 1.13: *Cones pointing out,* 128×128×128*: perfect reconstruction from* 4 *projections. The three images show the same volume from three different viewpoints.*

1.8. Extension to plane sets

So far we have considered the reconstruction of lattice sets in 2D and 3D. This model is well suited for the application of nanocrystal reconstruction at atomic resolution in electron microscopy [18]: atoms in a crystalline solid are arranged regularly in a lattice structure. In many other applications of tomography there is no "intrinsic lattice". In this section we consider the reconstruction of subsets of $\mathbb{R}^2$ from its projections in a small number of directions. We will also refer to such subsets as *planar black-and-white images*. The reconstruction problem for planar black-and-white images has also been studied in the field of *Geometric Tomography* (see, e.g., the book [9]). In this new context, the projection along a line is no longer a sum over a discrete set, rather it is a *line integral* or *strip integral* of a function $\mathbb{R}^2 \rightarrow \{0,1\}$, which yields a real value.

The iterative network flow algorithm can be adapted in such a way that it can be used for the reconstruction of planar black-and-white images. We will only give a high level overview of the algorithm. The details will be described in Chapter 5.

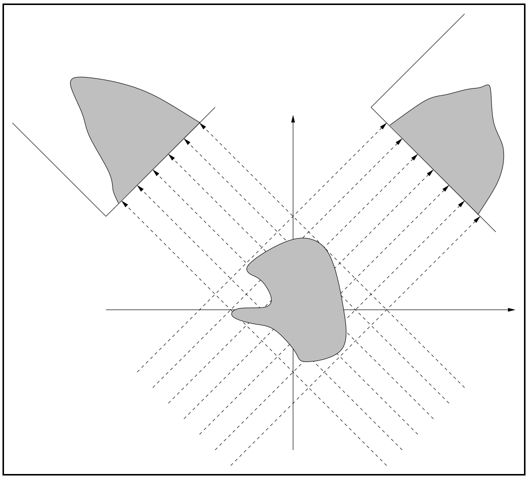


Figure 1.14: *A planar black-and-white image with two of its projections. If strip projections are used, the total amount of "black" in a set of consecutive strips parallel to the projection direction is measured.*

Figure 1.14 shows an example of a planar black-and-white image along with two of its projections. If strip projections are used, the total amount of "white" (or black) in a set of consecutive strips parallel to the projection direction is measured. As it is impossible to represent all possible planar images in a finite amount of computer memory, they are approximated by representing the images on a pixel grid. Each of the pixels can either have a value of 1 (white) or 0 (black).

The weighted reconstruction problem for two projections (i.e., Problem 2 in the context of lattice sets) can also be solved efficiently in the case of planar subsets. However, a specifically chosen pixel grid must be used, which depends on the two projection directions. Figure 1.15 shows how the grid is formed from two projections. Every pixel on the grid is the intersection of a strip in the first direction with a strip in the second direction. The network flow approach can be used for this pixel grid, to compute a 0-1 reconstruction from the given two projections.

In every iteration of the iterative network flow algorithm, a new pair of projections is selected. Therefore the pixel grid on which the new solution is computed is different in each iteration. To compute the weight map for the new pixel grid, the image from the previous iteration (defined on a different pixel grid) is first converted to a gray-scale image on the pixel grid by interpolation. Recall that the computation of the *similarity fraction* for a pixel p does not require that neighbouring pixels of p are *binary*. Therefore the new weight map can be computed in a straightforward way. An overview of the adapted version of the iterative

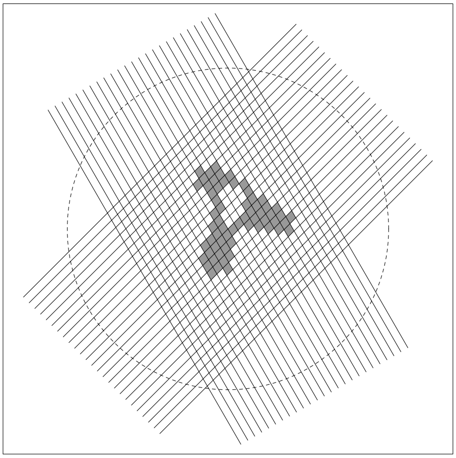


Figure 1.15: *Two parallel beams span a pixel grid. On this pixel grid, network flow methods can be used to solve the two-projection 0-1 reconstruction problem.*

algorithm is shown in Figure 1.16.

Just as for lattice images, the iterative network flow algorithm for plane sets is capable of computing very accurate reconstructions if a sufficient number of projections is available. However, if the original image does not have an intrinsic lattice structure we cannot expect a reconstruction that perfectly matches the projection data, as it is unlikely that the original image can be represented as a binary image on the pixel grid used by the algorithm.

1.9. Applications of discrete tomography

The main aim of this thesis is to present algorithms for the reconstruction of binary images from a small number of projections. The problem of reconstructing objects that do not have an intrinsic lattice structure, which is described in Chapter 5, has many potential applications. Many objects that are scanned in industry consist of a single material. Besides reconstruction results for simulated images, Chapter 5 contains reconstruction results for a real experimental dataset of a raw diamond that was scanned in a micro-CT scanner. The reconstruction results demonstrate that the techniques in this thesis can indeed be used for real world data.

Chapters 3 and 4 focus on the reconstruction of *lattice images*. Although lattice images may seem to be primarily of mathematical interest, the lattice model is also suitable for use in a practical application: the reconstruction of nanocrystals from projections obtained by electron microscopy. In the remainder of this section we will introduce this prominent application of discrete tomography. Chapter 6, the last chapter of this thesis, deals with one of the reconstruction problems that arise in the application of discrete tomography to the reconstruction of nanocrystals.

```
Compute the start solution $F^0$ on the standard pixel grid;
$i := 0$;
while (stop criterion is not met) do
begin
    $i := i + 1$;
    Select a new pair of directions $v^{(a)}$ and $v^{(b)}$ $(1 \leq a < b \leq n)$;
    Convert the previous solution $F^{i-1}$ to an image $\hat{F}^{i-1}$ which is defined
    on the pixel grid spanned by directions $v^{(a)}$ and $v^{(b)}$;
    Compute a new weight-map $W^i$ from the image $\hat{F}^{i-1}$;
    Compute a new solution $F^i$ on the grid spanned by $v^{(a)}$ and $v^{(b)}$
    by solving a variant of Problem 4, using the weight map $W^i$.
end
```

Figure 1.16: *Basic steps of the algorithm for plane sets.*

1.9.1. Atomic resolution electron tomography of nanocrystals

Three-dimensional imaging of extremely small crystals (nanocrystals) at atomic resolution is an important goal in Materials Science. Finding out exactly where each of the atoms reside inside different types of nanocrystals would provide a greater understanding of nanomaterials, which are abundant in many different applications.

Most quantities that are measured in physical experiments are *continuous quantities*. However, at the smallest scale, the atomic level, nature is actually discrete, as all materials are made out of a discrete set of atoms. If we focus on *crystalline solids*, there is also a different kind of discreteness involved: within a crystal the atoms are arranged in a highly regular structure, called the *crystal lattice*.

When looking at a small crystal (a *nanocrystal*), under a modern electron microscope, it seems possible to visually distinguish individual atoms in the sample, see Figure 1.17. However, the image that is seen under the microscope is actually a *projection image*. The round dots in Figure 1.17 represent projected columns of atoms, that stretch out in the direction orthogonal to the image. Such 2D projection images already provide substantial information about the sample, but it is not possible to determine the 3D position of the atoms from a single projection image.

When the first meeting on discrete tomography was organized in 1994, one of the main envisioned applications was 3D tomography of nanocrystals with atomic resolution. In fact, a novel technique in electron microscopy that had just been developed at that time, was one of the main reasons for starting research in discrete tomography. This technique, called QUANTITEM [20, 24], made it possible to count the number of atoms in each projected column of a nanocrystal. One of the main questions that arose from this new ability was, if it would be possible to reconstruct the 3D position of all atoms from the projection data by some tomographic reconstruction algorithm, provided that the atom counting could be

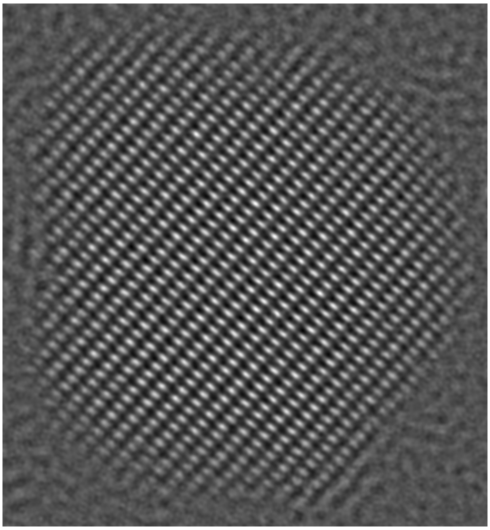


Figure 1.17: *Projection image of a nanocrystal from an electron microscope. The width of a pixel in this image is less than* 10^{-10}*m. Each round dot represents a projected column of atoms.* Courtesy of C. Kisielowski.

performed for several projection images.

For investigating samples that require a resolution of at most 1 nanometer (10^{-9}m), tomography has already been used extensively in electron microscopy. Just as in X-ray tomography, a series of projection images is recorded from a range of angles, called a *tilt series*. The reconstruction is subsequently computed by methods from continuous tomography, such as Filtered Backprojection or ART (see, e.g., [19]). Tomographic reconstruction from projections obtained by electron microscopy is known as *electron tomography*. Several factors limit the applicability of this type of tomography to atomic resolution reconstruction. Firstly, a large number of tilt angles is required to obtain reconstructions of sufficient quality. Typically, at least 50 images have to be recorded. Each time that the electron beam interacts with the sample to form a new image, the sample may be damaged by the electron beam. Therefore, recording many images is often impossible. Secondly, when recording images at atomic resolution, it is very difficult to properly align the sample for each new projection angle. In this aspect, electron tomography is somewhat different from X-ray tomography. In X-ray tomography the X-ray source is usually rotated around the object of interest. In electron microscopy, the sample is rotated within the microscope. Even if it is possible to rotate the sample holder with great accuracy, *sample drift* within the holder may still occur. At atomic resolution even the slightest changes in the sample position or orientation have major consequences for the acquired image. Finally, the interaction of the electron beam with the sample depends heavily on the orientation of the sample relative to the electron beam. If the electron beam is aligned parallel to one of the lattice directions of the crystal (called *zone axes*), the observed image will usually be very different from an image ob-

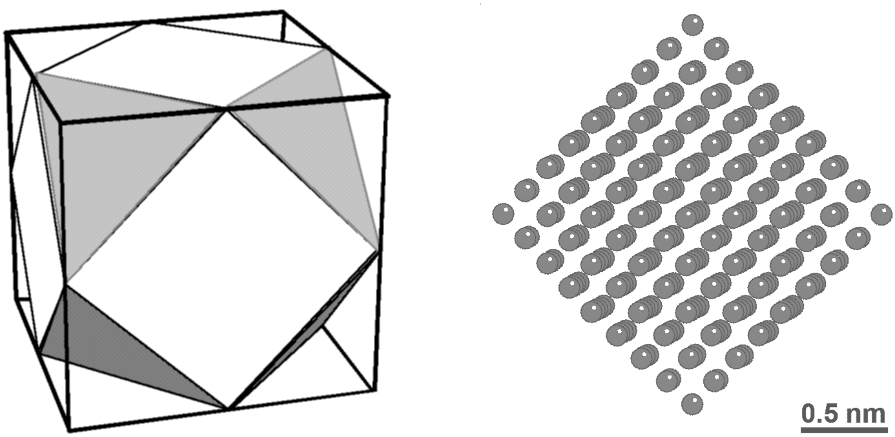


Figure 1.18: *(a) Left: Cuboctahedral shape of a simple nanocrystal (b) Right: Projection of a perfect nanocrystal consisting of 309 atoms*

tained without zone axis alignment. Mixing these two type of images in the reconstruction algorithm typically results in reconstruction artifacts.

Counting atoms

Figure 1.18 shows a 3D nanocrystal that consists of 309 atoms. Nanocrystals are highly regular structures. The atoms in a *perfect* crystal are arranged in a periodic pattern. By incorporating the regular structure of crystals in the reconstruction algorithm (i.e., reconstructing a *lattice image*) the special structure of nanocrystals can be exploited.

Not every orientation of the sample is equally suitable for counting the number of atoms. To observe *projected columns* of atoms in the microscopic images, the electron beam has to be aligned parallel to one of the zone axes of the crystal.

Unfortunately, the experimental results of QUANTITEM did not live up to the expectations. One of the main obstacles at that time that had to be overcome first were *aberrations*, that distorted the images from the microscope. Both *spherical aberrations* and *chromatic aberrations* are a main concern in electron microscopy. Spherical aberrations are caused by the use of a spherical lense, instead of an ideal hyperbolic lense. Chromatic aberrations are caused by the use of an electron beam that contains electrons of different energy levels.

Over the last ten years major improvements have been obtained in aberration correction. Special lense systems for correcting spherical aberrations have now been integrated into the newest microscopes. Besides correcting aberrations by adding hardware to the microscopes, new techniques have also been developed for improving the image quality by *computation*. Instead of recording a single image, a series of images is obtained using varying focus conditions. From a single image, only the *intensity* of the electron beam can be measured

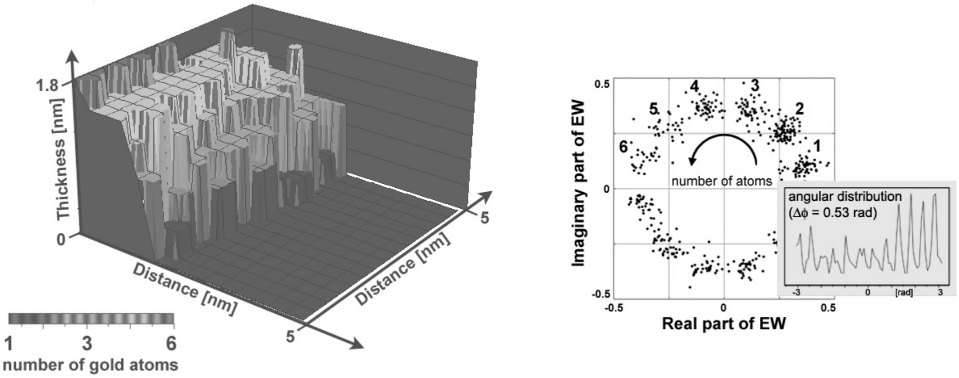


Figure 1.19: *(a) Left: experimentally determined exit wave phase for a very thin gold sample. (b) Right: The exit wave phases for the different atom columns cluster into groups, determined by the number of atoms in the column. The discrete step of the exit wave phase for each additional gold atom is 0.53 rad.*

as it exits the sample. By recording a sequence of images, it is also possible to compute the *phase* of the electron exit wave. This computation is known as *exit wave reconstruction*. The reconstructed exit wave of a crystalline sample turns out to be very useful for counting the number of atoms in each projected column.

Figure 1.19a shows the phase of the electron exit wave for a very thin gold sample, obtained by a real (physical) experiment. It can be seen in Figure 1.19b that a discrete jump in the exit wave phase occurs each time that an extra atom is added to a column.

Using the exit wave phase it is now possible to count the number of atoms in each column of a very thin sample. If the atom counting step can be performed for several different projection angles, a discrete tomography problem will have to be solved to compute a 3D reconstruction from these projections, see Figure 1.20.

Results of experiments using simulated microscopy data are reported in [17, 18]. At present, there is no dataset available yet that contains atom count data for several zone axis projections. The main obstacle towards obtaining such a dataset is the unavailability of a *double tilt-holder*. To obtain projection images along several zone axes, a single tilt axis for the sample is not enough. However, rotating the sample around more than one axis is technically difficult. Double tilt-holders that are capable of rotating along two axes are currently under construction.

Count errors

Most theory that has been developed for discrete tomography deals with reconstruction from perfect projections. Unfortunately, in most physical experiments, errors are very difficult to avoid. A major advantage of using the knowledge that atoms are discrete and that they occur in columns, is that this allows for some correction of errors. As can be seen in Figure 1.19,

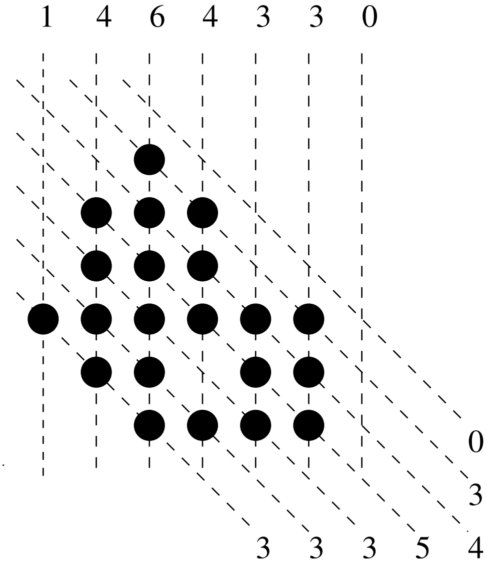


Figure 1.20: *Discrete tomography can be used to reconstruct the position of the atoms inside a nanocrystal from zone-axis projections, assuming that the atom positions are all lattice points.*

different columns that contain the same number of atoms don't result in equal measured exit wave phases. Suppose that an atom column contains k atoms and ϕ is the expected exit wave phase for this number of atoms. As long as the measured phase ϕ' is closer to ϕ than to the expected exit phases for other atom counts, the resulting atom count will still be correct. This kind of error correction is unusual in physics, as most measured quantities are *continuous*.

Now suppose that after the counting step the projection data still contains errors. In Chapters 2, 3 and 4 we will show that it is often possible to reconstruct binary images from just a few projections. If *perfect* projections from k directions are required to compute a perfect reconstruction of an image, but $k' > k$ projections are actually available, the projection data contains *redundancy*. Similar to the way that redundancy is exploited in Coding Theory to correct errors, it may be possible to use additional projections in discrete tomography for the correction of errors [4]. Unfortunately, the results in [2] show that the number of errors that can be corrected grows only linearly with the number of available projections.

For the application of electron microscopy, it may be possible to beat the theoretical bound on the number of correctable errors. Prior knowledge that comes from known crystallographic properties of crystals can be used to guide a reconstruction algorithm towards physically feasible reconstructions.

Defects

Assuming that all atoms in a nanocrystal are ordered in a regular lattice is mathematically very convenient. The resulting mathematical model of discrete tomography contains two forms of “discreteness”: each lattice point either contains one or zero atoms, i.e. the set of atoms is discrete, and the lattice itself is discrete. Unfortunately, microscopists are usually more interested in crystals that do *not* have a perfect lattice structure. Deviations from the regular lattice structure are known as *defects* in crystallography. The simplest of such deviations, which *can* be incorporated in the lattice model are *vacancies*. A crystal contains a vacancy if a lattice point is not occupied, where it should be occupied according to the periodic crystal structure. The ability to do 3D imaging of vacancies would already be of great interest in materials science. However, many defects occur in practical samples that severely distort the regular lattice structure. If it is not longer possible to find orientations in which projection *columns* of atoms can be observed, it is unclear how to use discrete tomography for such samples. Chapter 6 describes how it may still be possible to count atoms from the projection data, even though the atoms do not lie in regular columns.

Bibliography

[1] Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: *Network flows: theory, algorithms and applications*. Prentice-Hall (1993).

[2] Alpers, A., Gritzmann, P.: On stability, error correction and noise compensation in discrete tomography. *SIAM J. Discrete Math.*, **20**, 227–239 (2006).

[3] Anstee, R.P.: The network flows approach for matrices with given row and column sums. *Discrete Math.*, **44**, 125–138 (1983).

[4] Batenburg, K.J., Jinschek, J.R., Kisielowski, C.: Atomic resolution electron tomography on a discrete grid: atom count errors. *Microsc. Microanal.*, **11, suppl. 2** (2005).

[5] Bertsekas, D.P., Tseng, P.: RELAX-IV: a faster version of the RELAX code for solving minimum cost flow problems. *LIDS Technical Report LIDS-P-2276*, MIT, (1994).

[6] Dürr, C.: Ryser's algorithm can be implemented in linear time.
`http://www.lix.polytechnique.fr/˜durr/Xray/Ryser/`

[7] Ford, L.R., Fulkerson, D.R.: Maximal flow through a network. *Canad. J. Math.*, **8**, 399–404 (1956).

[8] Gale, D.: A theorem on flows in networks. *Pacific J. Math.*, **7**, 1073–1082 (1957).

[9] Gardner, R.J.: *Geometric tomography*. Cambridge University Press, New York (1995).

[10] Gardner, R.J., Gritzmann, P., Prangenberg, D.: On the computational complexity of reconstructing lattice sets from their X-rays. *Discrete Math.*, **202**, 45–71 (1999).

[11] Goldberg, A.V.: An efficient implementation of a scaling minimum-cost flow algorithm. *J. Algorithms*, **22**, 1–29 (1997).

[12] Goldberg, A.V., Rao, S.: Beyond the flow decomposition barrier. *J. ACM*, **45**, 783–797 (1998).

[13] Goldberg, A.V., Tarjan, R.E.: Finding minimum-cost circulations by successive approximation. *Math. Oper. Res.*, **15**, 430–466 (1990).

[14] Gritzmann, P., de Vries, S., Wiegelmann, M.: Approximating binary images from discrete X-rays. *SIAM J. Optim*, **11**, 522–546 (2000).

[15] Hajdu, L., Tijdeman, R.: Algebraic aspects of discrete tomography. *J. Reine Angew. Math.*, **534**, 119–128 (2001).

[16] *ILOG CPLEX*, `http://www.ilog.com/products/cplex/`

[17] Jinschek, J.R., Batenburg, K.J., Calderon, H.A., Van Dyck, D., Chen, F.-R., Kisielowski, C.: Prospects for bright field and dark field electron tomography on a discrete grid. *Microsc. Microanal.*, **10, suppl. 3** (2004).

[18] Jinschek, J.R., Calderon, H.A., Batenburg, K.J., Radmilovic, V., Kisielowski, C.: Discrete tomography of Ga and InGa particles from HREM image simulation and exit wave reconstruction. *MRS Proc.*, **839**, 4.5.1–4.5.6 (2004).

[19] Kak, A.C., Slaney, M.: *Principles of computerized tomographic imaging*. SIAM (2001).

[20] Kisielowski, C., Schwander, P., Baumann, F., Seibt, M. Kim, Y., Ourmazd, A.: An approach to quantitative high-resolution transmission electron microscopy of crystalline materials. *Ultramicroscopy* **58**, 131–155 (1995).

[21] Ryser, H.J.: Combinatorial properties of matrices of zeros and ones. *Can J. Math.*, **9**, 371–377 (1957).

[22] Ryser, H.J.: *Combinatorial mathematics*, The Carus Mathematical Monographs, **14**, Math. Assoc. of America, John Wiley and Sons, Inc., New York (1963).

[23] Schrijver, A.: *Combinatorial optimization: polyhedra and efficiency*. Algorithms and Combinatorics series, **24**, Springer, Heidelberg (2003).

[24] Schwander, P., Kisielowski, C., Baumann, F., Kim, Y., Ourmazd, A.: Mapping projected potential, interfacial roughness, and composition in general crystalline solids by quantitative transmission electron microscopy. *Phys. Rev. Lett.*, **71**, 4150–4153 (1993).

[25] Slump, C.H., Gerbrands, J.J.: A network flow approach to reconstruction of the left ventricle from two projections. *Comput. Gr. Im. Proc.*, **18**, 18–36 (1982).

[26] Tanabe, K.: Projection method for solving a singular system. *Numer. Math.*, **17**, 203–214 (1971).

Chapter 2

An evolutionary algorithm for discrete tomography

This chapter has been published as: K.J. Batenburg, “An evolutionary algorithm for discrete tomography”, Discrete Applied Mathematics, Vol. 151, 36–54 (2005).

Abstract. One of the main problems in Discrete Tomography is the reconstruction of binary matrices from their projections in a small number of directions. In this chapter we consider a new algorithmic approach for reconstructing binary matrices from only two projections. This problem is usually underdetermined and the number of solutions can be very large. We present an evolutionary algorithm for finding the reconstruction which maximises an evaluation function, representing the “quality” of the reconstruction, and show that the algorithm can be successfully applied to a wide range of evaluation functions. We discuss the necessity of a problem-specific representation and tailored search-operators for obtaining satisfactory results. Our new search-operators can also be used in other discrete tomography algorithms.

2.1. Introduction

Discrete Tomography (DT) is concerned with the reconstruction of a discrete image from its projections. One of the key problems is the reconstruction of a binary (black-and-white) image from only two projections, horizontal and vertical (see Figure 2.1). In 1957 Ryser [23] and Gale [11] independently derived necessary and sufficient conditions for the existence of a solution. Ryser also provided a polynomial time algorithm for finding such a solution. However, the problem is usually highly underdetermined and a large number of solutions may exist [27].

Among the applications of discrete tomography are the reconstruction of crystal lattices from projections obtained by electron microscopy [17, 25] and the reconstruction of angio-

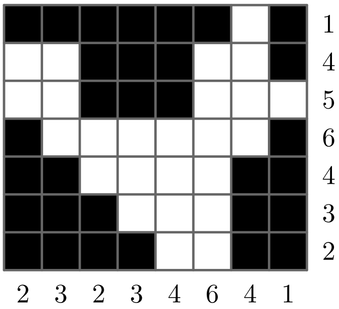


Figure 2.1: *A binary image with its horizontal and vertical projections.*

graphic images in medical imaging [22, 26]. In such applications the projection data are the result of measurements of a physical object and we are interested in finding a reconstruction which resembles the original image as closely as possible, not just one that corresponds to the given projections. Therefore it is necessary to use all available information about the class of images to which the measured image belongs.

In this chapter we assume that the given projection data are consistent (i.e., there is at least one image which has the given projections). In practice the situation is often more complicated, as noise and other errors may result in inconsistent data.

For certain classes of highly-structured images, such as hv-convex polyominoes (i.e., the white pixels in each row and column are contiguous), polynomial-time reconstruction algorithms exist (see, e.g., [4, 6]). On the other hand, there are classes of images, such as the more general class of hv-convex images (which may consist of many separate polyominoes), for which the reconstruction problem is NP-hard (see [19]). Instead of assuming specific properties of the image structure, we will focus on a more general approach. Suppose that we are able to define an evaluation function, which assigns a value to each of the solutions, reflecting how good a particular solution is in our context. An algorithm that maximises the evaluation over the set of all solutions will then yield the most desirable solution.

As an example, consider the class of hv-convex images. Suppose that the unknown original image belongs to this class. In [8] it is shown that when we define the evaluation of an image to be the number of neighbouring pairs of white pixels (either horizontal or vertical), the evaluation function is maximised by a reconstruction that has the prescribed projections if and only if the reconstruction is also hv-convex. Similarly, we can define an evaluation function for the reconstruction of hv-convex polyominoes, where we subtract a large penalty if the image consists of more than one polyomino.

Probably of greater practical relevance is the case where the unknown original image can be considered to be a random sample from a certain known probability distribution, defined on the set of all images. In this case the evaluation function reflects the likelihood that any given image is a random sample from this distribution.

Finally we remark that the NP-hard problem of reconstructing binary images from more than two projections fits into our model as well, as long as the horizontal and vertical projections are included in the set of known projections. In that case the evaluation function must include a term that indicates the deviation of the remaining projections (besides horizontal and vertical) of the reconstructed image from the prescribed projections. Note that our algorithm will always find a reconstruction for which the horizontal and vertical projections correspond *perfectly* to the prescribed projections, assuming that the prescribed projections in those two directions are consistent. Define the *projection deviation* of an image to be the total difference (as a sum of absolute values) between the image's projections and the prescribed projections. Because we only deal with maximisation problems, we use the negated projection deviation as the evaluation function.

The flexibility of our model comes at a price. In general, the problem of maximising the evaluation function over the set of all solutions is NP-hard, which follows directly from the NP-hardness of the problem of reconstructing binary images from more than two projections (see [12]). In order to deal with this intractability, we resort to the use of modern approximation algorithms; in particular, we propose an *evolutionary algorithm*, see [3]. The term evolutionary algorithm is generally used to denote a *class* of algorithms that are based on the concept of maintaining a "population" of candidate solutions, which evolves by means of evolutionary operators (e.g., mutation, crossover) and selection. In Section 2.3 we discuss the choices that we made in designing the algorithm. Because the problem at hand has a natural binary encoding it seems attractive to apply a classical genetic algorithm (GA) (see [21]), using the bitstring representation. For several reasons however, this does not lead to good results. We will discuss the problem features that cause the application of classical GA's to be inadequate and introduce new problem-specific mutation and crossover operators. We discuss the necessity of using a hillclimb operator as a post-mutation and post-crossover operator for improving the solution quality.

Section 2.4 presents our experimental results. The results suggest that our algorithm can be successfully applied to a wide range of evaluation functions, which makes it very versatile.

2.2. Preliminaries

We will now introduce some notation and define the DT problem mathematically. Parts of our algorithm are based on well-known theoretical results, which we will summarise. We assume that the reader is familiar with the theory of network flows [2] and the basic principles of evolutionary algorithms [3, 21].

Throughout this chapter we will assume that all binary matrices are of size $m \times n$. We first consider the basic problem of reconstructing a binary matrix from its horizontal and vertical projections.

Definition 1 *Let $R = (r_1, \ldots, r_m)$ and $S = (s_1, \ldots, s_n)$ be nonnegative integral vectors. We denote the class of all binary matrices $A = (a_{ij})$ satisfying*

$$\begin{aligned}\sum_{j=1}^{n} a_{ij} &= r_i, \quad i = 1, \ldots, m, \\ \sum_{i=1}^{m} a_{ij} &= s_j, \quad j = 1, \ldots, n,\end{aligned}$$

by $\mathcal{A}(R,S)$. The vectors R and S are called the row and column projections of any matrix $A \in \mathcal{A}(R,S)$.

Because DT is strongly related to digital image processing we often refer to binary matrices as *images* and call the matrix entries *pixels* with values *white* (1) and *black* (0).

From this point on we assume that the row and column projections are *consistent*, meaning that $\mathcal{A}(R,S)$ is nonempty. In particular, this implies that $\sum_{i=1}^{m} r_i = \sum_{j=1}^{n} s_j$. Necessary and sufficient conditions for the nonemptiness of $\mathcal{A}(R,S)$ are given in [18].

One of the basic problems in DT is the *reconstruction problem*:

Problem 6 *Let R and S be given integral vectors. Construct a binary matrix $A \in \mathcal{A}(R,S)$.*

Because the number of possible solutions of the reconstruction problem can be very large it is necessary to impose additional properties on the solution being sought. In this chapter we consider the following problem:

Problem 7 (MAXEVAL) *Let R and S be given integral vectors and let $f : \mathcal{A}(R,S) \rightarrow \mathbb{Z}$ be a given evaluation function. Find a binary matrix $A \in \mathcal{A}(R,S)$ such that $f(A)$ is maximal.*

Although in Problem 7 the domain of f is restricted to the class $\mathcal{A}(R,S)$, f is usually defined on the entire set $\{0,1\}^{m \times n}$.

The notion of *switching components* plays an important role in the characterisation of the class $\mathcal{A}(R,S)$.

Definition 2 *Let $A \in \mathcal{A}(R,S)$. A switching component of A is a 2×2 submatrix of the form*

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad \textit{or} \quad \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

Switching components have the property that if we interchange the 0's and 1's, the projections do not change. We call such an interchange operation an *elementary switching operation*. An important theorem of Ryser [24] describes how the class $\mathcal{A}(R,S)$ is characterised by a single element of this class and the set of switching components:

Theorem 2 *Let $A \in \mathcal{A}(R,S)$. There exists $B \in \mathcal{A}(R,S)$, $B \neq A$, if and only if A contains a switching component. Moreover, if such a matrix B exists, then A can be transformed into B by a sequence of elementary switching operations.*

1	1	0	**2**
0	1	0	**1**
1	0	1	**2**
2	**2**	**1**	

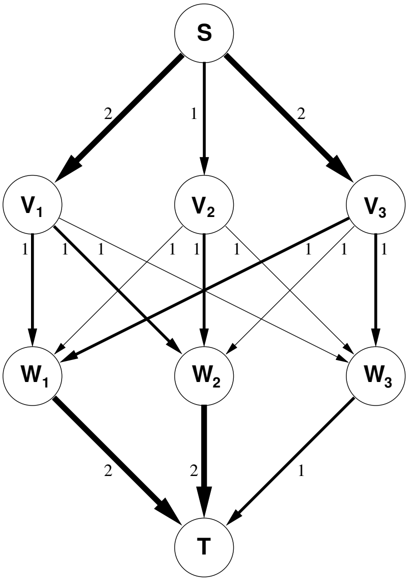


Figure 2.2: *A* 3 × 3 *image instance and its corresponding network flow.*

We remark that every matrix B that is the result of applying a sequence of elementary switching operations to A is also in $\mathcal{A}(R,S)$. The fact that we can transform a matrix in $\mathcal{A}(R,S)$ into any other matrix having the same projections by means of elementary switching operations, makes the use of elementary switching operations very suitable for local search procedures. In our evolutionary algorithm, we make extensive use of these operations.

An important operation in our algorithm is the computation of matrices in $\mathcal{A}(R,S)$, given R and S. We use a network flow approach for computing these matrices, which was first introduced by Gale [11]. First, we construct a directed graph N. The set V of nodes consists of a source node S, a sink node T, one layer $V_1,\dots,V_m$ of nodes that correspond to the image rows (*row nodes*) and one layer $W_1,\dots,W_n$ of nodes that correspond to the image columns (*column nodes*). The set A of arcs consists of arcs (S,V_i) for $i=1,\dots,m$, arcs (V_i,W_j) for $i=1,\dots,m, j=1\dots,n$, and arcs (W_j,T) for $j=1,\dots,n$. We remark that the two layers of row nodes and column nodes form a complete bipartite graph.

We also define a *capacity function* $C : A \to \mathbb{N}_0$ which assigns an integral capacity to every arc. All arcs between a row node and a column node are assigned capacity 1. We will denote these arcs as *pixel-arcs*, because each of them corresponds to a single, unique pixel of the image (determined by the row and column). The prescribed projections determine the capacities of the arcs from the source node and the arcs to the sink node. Every arc (S,V_i) has capacity r_i and every arc (W_j,T) has capacity s_j. Figure 2.2 shows the network N for an example 3 × 3 image.

It is well-known that a maximum flow from S to T can be found in polynomial time. From the theory of linear programming we know that there is a maximum flow which is integral. Suppose that this maximum flow fully saturates all outgoing arcs from the source (and all incoming arcs towards the sink). Every pixel-arc carries a flow of either 0 or 1. Assign a 1 to pixel (i, j) if arc (V_i, W_j) carries a flow of 1 and assign 0 otherwise. It is easy to verify that the resulting image has the prescribed projections in both directions. Conversely, if the horizontal and vertical projections are consistent, i.e., there is an image that satisfies them, the network must have a maximal flow that saturates all outgoing arcs from the source. We see that the problem of finding a maximum integral flow in N is equivalent to the reconstruction problem.

We can use algorithms for solving the max flow problem to compute solutions of the DT reconstruction problem. In our case we want to impose additional requirements on the resulting image. In particular, given a binary image $M \in \{0,1\}^{m\times n}$, we want to find the binary image $A \in \mathcal{A}(R,S)$ that differs from M in as few pixels as possible. We will refer to M as the *model image*. Our evolutionary algorithm computes reconstructions for many different model images. Each of those reconstructions adheres to the prescribed horizontal and vertical projections.

Problem 8 (BESTMATCH) *Let R and S be given integral vectors and let $M = (m_{ij})$ be a given binary matrix. Find $A = (a_{ij}) \in \mathcal{A}(R,S)$ such that*

$$\sum_{i=1}^{m}\sum_{j=1}^{n} |m_{ij} - a_{ij}|$$

is minimal.

Problem 8 can be solved efficiently using an extension of the network flow model, incorporating a cost function. We assign a cost $c_{ij} = -m_{ij}$ to every arc (V_i, W_j). The remaining arcs are all assigned a cost of 0. The process of finding a maximum flow of minimal cost now comes down to finding an integral flow that saturates the maximum number of pixel-arcs for which the corresponding model image pixel has value 1. In other words, if A is the image that corresponds to the maximal flow of minimal cost, then A has as many 1's in common with M as possible.

For any matrix $B = (b_{ij}) \in \{0,1\}^{m\times n}$, denote the number of 1's in B by n_B and define n_M analogously. Suppose that $B \in \mathcal{A}(R,S)$. Clearly, we have $n_B = \sum_{i=1}^{m} r_i = \sum_{j=1}^{n} s_j$. Put

$$v_B = |\{(i,j) : b_{ij} = 1 \wedge m_{ij} = 1\}|.$$

Then

$$|\{(i,j) : b_{ij} \neq m_{ij}\}| = (n_B - v_B) + (n_M - v_B) = n_B + n_M - 2v_B,$$

where $|\cdot|$ denotes the set cardinality. Hence

$$|\{(i,j) : b_{ij} = m_{ij}\}| = mn - (n_B + n_M - 2v_B),$$

which shows that maximising the number of common 1's between B and M is equivalent to solving Problem 8, because mn, n_B and n_M are constant for $B \in \mathcal{A}(R,S)$. Numerous standard algorithms are available for solving the min cost max flow problem in polynomial time. We can use any of these algorithms for solving Problem 8, see, e.g., [2].

2.3. Algorithmic approach

2.3.1. Overview of the approach

In this section we will describe a heuristic algorithm for solving Problem 7 (MAXEVAL). Because finding a reconstruction that maximises the evaluation function is an NP-hard problem, we have to resort to approximation algorithms. In our approach we make extensive use of the fact that Problem 8 (BESTMATCH) can be solved in polynomial time. In preliminary experiments we implemented and tested several approaches, based on simulated annealing [1], tabu search [13] and evolutionary algorithms [3]. On the basis of these preliminary experiments, we believe that simulated annealing and tabu search, which only use local search operators, are not well suited for this task. The main reason for this is that the DT problem usually has a great number of local optima and moving between different optima may require a large number of "uphill" steps.

Evolutionary algorithms can handle this problem in two ways. Firstly, the algorithm uses a diverse population of candidate solutions, instead of a single solution. Secondly, the crossover operator is capable of performing large steps in the search space.

Because the problem at hand allows for a natural bitstring representation of candidate solutions it fits directly into the framework of classical genetic algorithms on bitstrings: simply use $\{0,1\}^{mn}$ as the candidate solution space. We now have two optimisation criteria, the evaluation function and the deviation from the prescribed projections.

We found this approach to be inadequate for several reasons. In the first place, the crossover operator usually results in candidate solutions that deviate greatly from the prescribed projections. This is particularly common when the two parent solutions have many differences. This behaviour makes the crossover operator hardly effective at all, reducing the algorithm to a simple hillclimb procedure that only uses mutation to improve upon the solutions. In addition, the most common crossover operators, single-point crossover and uniform crossover, do not take into account the two-dimensional spatial structure of the candidate solutions. Intuitively, building blocks for this problem are made up of regions of pixels, not necessarily horizontally or vertically aligned. A problem-specific operator could benefit from this structure.

We have designed a problem-specific evolutionary algorithm to overcome the disadvantages of the classical genetic algorithm (GA). Instead of optimising over all bitstrings in $\{0,1\}^{mn}$, the algorithm optimises exclusively over $\mathcal{A}(R,S)$. At the end of each generation all candidate solutions have the prescribed horizontal and vertical projections. This requires a new crossover operator, that is not only capable of mixing features from both parents, but also of ensuring that the produced children adhere to the prescribed projections. Similar requirements apply to the mutation operator.

Our algorithm is a *memetic algorithm* (see [7]): after every crossover or mutation operation a stochastic hillclimb is performed until the solution has reached a local optimum. In this way, individuals always represent local optima in the search space. We chose this approach, because although the proposed crossover operator generates children that have the prescribed projections, the children are often of inferior quality with respect to the evaluation function (when compared to the parents). In order to fully exploit the explorative power

```
generate initial population $P_0$ of size $\lambda$, consisting of matrices in $\mathcal{A}(R,S)$;
perform a hillclimb operation on each matrix in $P_0$;
$t := 0$;
while (stop criterium is not met) do
begin
  $P'_t = \emptyset$;
  for $i := 1$ to $\mu$ do
  begin
    generate a child matrix $C$, by crossover or mutation;
    perform a hillclimb operation on $C$;
    $P'_t := P'_t \cup \{C\}$;
  end;
  select new population $P_{t+1}$ from $P_t \cup P'_t$;
  $t := t+1$;
end
output the best individual found;
```

Figure 2.3: *Outline of the evolutionary algorithm.*

of the crossover operator, the hillclimb procedure increases the solution quality while still remaining quite close to the solution that resulted from the crossover operation. Memetic algorithms typically require only a small number of generations (while performing a lot of work for each generation).

Figure 2.3 summarises our algorithm. The parameters λ and μ are the population size and the number of children that are created in each generation, respectively. In the next sections we will describe the various operators in more detail.

2.3.2. A new crossover operator

The crossover operator is one of the main parts of our algorithm. The input of the operator consists of two *parent images*. The output is a *child image*, which has certain features from both parents. Because all images in the population are members of $\mathcal{A}(R,S)$, the resulting image should have the prescribed projections. In order to enforce this constraint, we use the network flow formulation of the DT problem.

First, a *crossover mask* $X = (x_{ij}) \in \{0,1\}^{m \times n}$ is computed, which determines for each pixel from which parent image it inherits. In the following description, *assigning cell* (i,j) *to the first parent* means setting $x_{ij} = 0$ and *assigning cell* (i,j) *to the second parent* means setting $x_{ij} = 1$. By *neighbouring cells* of a cell (i,j) we mean the four horizontal and vertical neighbours (or fewer, if the pixel is at one of the image boundaries). As discussed in Section 2.3.1 we want the crossover operator to take the spatial structure of our image into account. Therefore the crossover mask should assign *areas* of the image to each of the parents. We use the following procedure for realising this. First, one random cell is selected in each of

```
while (there are border cells) do
begin
    randomly select a border cell $(x,y)$;
    mark all unassigned neighbouring cells of $(x,y)$ as border cells;
    assign these cells to the same parent as $(x,y)$;
    remove the border status of $(x,y)$;
end
```

Figure 2.4: *Main loop of the procedure that generates the crossover mask.*

the four quadrants (where the origin is considered to be in the center of the image). We call these cells the *seeds*. Two (randomly chosen) seeds are assigned to the first parent, the other two are assigned to the second parent. The four seeds are marked as *border cells*. Next, the algorithm enters a loop, shown in Figure 2.4. Because the seeds are each in a different quadrant, this procedure results in a crossover mask that roughly assigns the same number of cells to both parents.

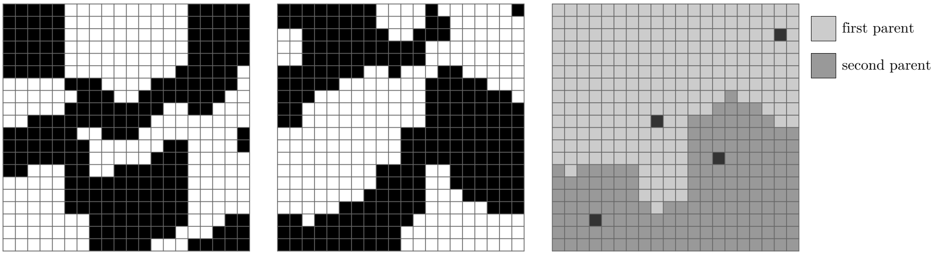
From the crossover mask and both parent images $P = (p_{ij})$ and $Q = (q_{ij})$, a *model image* $M = (m_{ij})$ is computed, as follows:

$$m_{ij} = \begin{cases} p_{ij} & \text{if } x_{ij} = 0 \\ q_{ij} & \text{if } x_{ij} = 1 \end{cases}$$

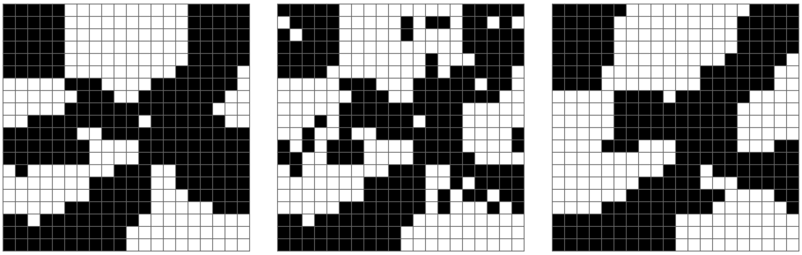
Subsequently, we construct the child image C by solving Problem 8, using M as the model image. This will result in a child image that is in $\mathcal{A}(R,S)$, resembles the first parent in a certain part and resembles the other parent in the rest of the image. Figure 2.5 shows two parent images (having the same projections), a crossover mask, the corresponding model image and the resulting child image. The four seeds are shown as dark cells in the crossover mask. In this example, we use the number of neighbouring pairs of white pixels as the evaluation function. Although the child image resembles both parents in their corresponding parts, it is clear that the child image is far from a local optimum with respect to the evaluation function. To ensure that the child image has sufficient quality, we apply a local hillclimb operator after the crossover operation, which we describe in the next section.

2.3.3. The hillclimb operator

The hillclimb operator applies a sequence of small modifications, all of which increase the evaluation function, until a local optimum is reached. Because we want the resulting image to be in $\mathcal{A}(R,S)$, we use elementary switching operations as local steps. The outline of the procedure is shown in Figure 2.6. It is of great importance that the switching component in each step is chosen randomly among all those yielding an increase of the evaluation function. We have performed experiments with implementations for which this was not the case: the search for switching components always started in the topleft corner and proceeded



a. first parent b. second parent c. crossover mask



d. model image e. child before hillclimb f. child after hillclimb

Figure 2.5: *The crossover operator combines two parent images into a child image.*

from left to right, top to bottom. However, this introduces a bias in the hillclimb process, resulting in "skew images", clearly showing the consequences of the biased operator.

Although the hillclimb operation can be easily described, its implementation offers several computational challenges. Finding all *applicable switching components* (switching components that are present in the image) is an $O((mn)^2)$ operation, making it computationally expensive. We store all applicable switching components in a hash table. Switching components are stored as two integer-pairs: (top-left, bottom-right). We can use the hash table to iterate over all applicable switching components. The search efficiency of hash tables is required to update the datastructure after a switching operation has been applied by first removing all switching components containing any of the four pixels involved and subsequently adding all new switching components that contain any of those four pixels. If the

```
while (a switching operation that can improve the evaluation exists) do
begin
  randomly select such a switching operation;
  apply the switching operation;
end
```

Figure 2.6: *Outline of the hillclimb procedure.*

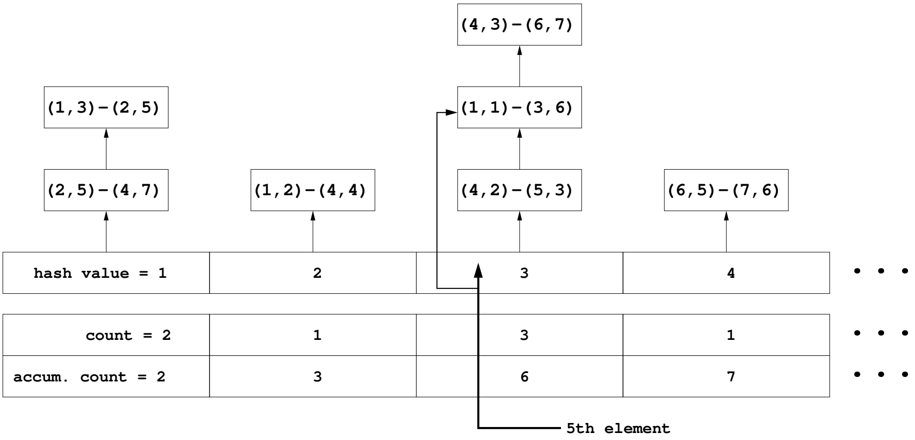


Figure 2.7: *The hash table of switching components is extended to allow binary search for an indexed element.*

hash table is large enough we can perform this update operation in $O(mn)$ time.

A second operation that our datastructure must support is efficient selection of a randomly chosen element. Each hash table entry points to a linked list of switching components. In a separate array, we store the number of switching components that reside in each list. In another array, we compute the accumulated number of switching components, up to and including that entry. Using this information, we can efficiently find the ith element in the hash table, given any index i: first perform a binary search to find the right table entry and then perform a (tiny) linear search to find the actual switching component. Figure 2.7 illustrates the approach.

Figure 2.8 shows a more precise formulation of the hillclimb operator. In every iteration of the outer loop the evaluation of the image is increased. If the evaluation function is bounded from above and from below (as is the case for the evaluation functions we consider in more detail later), these bounds yield a bound on the number of iterations of the outer loop. Note that the evaluation function of Problem 7 is discrete.

As the evaluation of the solution increases, the number of iterations in the inner loop until an improvement is found will generally increase. Because the efficiency of the inner loop contributes greatly to the efficiency of the hillclimb procedure we want to make the inner loop as efficient as possible. To improve efficiency, we can use the fact that for many evaluation functions the evaluation function after application of a switching component does not have to be computed from scratch, but can be derived from the previous evaluation (*delta updating)*. It is essential to exploit this feature, because the inner loop is executed so often. Despite the performance optimisations that we proposed, the hillclimb procedure is still the bottle-neck for the runtime of our algorithm.

```
initialize the hash table, storing all $s$ applicable switching components;
repeat
    initialise the accumulated count array;
    generate a random permutation $\pi$ of $1,\dots,s$;
    for $i := 1$ to $s$ do
    begin
        find switching component $\pi_i$ in the hash table;
        if (applying $\pi_i$ increases the evaluation) then
        begin
            apply $\pi_i$;
            update the hash table;
            break out of the for-loop and restart in the outer loop;
        end;
    end;
until (no improving switching component has been found);
```

Figure 2.8: *The hillclimb procedure.*

2.3.4. The mutation operator

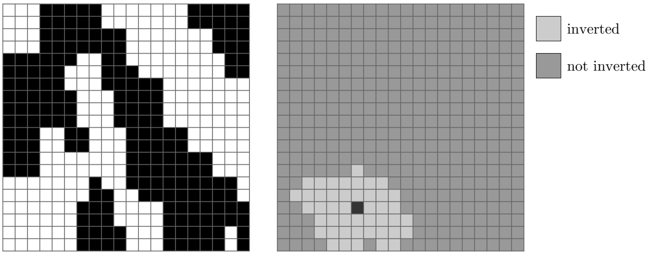
Besides the crossover operator, our algorithm also uses a mutation operator. This operator "distorts" a region of the image, while still adhering to the prescribed projections. The mutation operator is composed of several algorithmic steps, very similar to the crossover operator. Instead of the crossover mask, a *mutation mask* $X = (x_{ij}) \in \{0,1\}^{m\times n}$ is computed. First, a random number $k \in [k_{min}, k_{max}]$ and a random cell (i,j) (the *seed*) are selected. We assign $x_{ij} = 1$ and mark (i,j) as border cell. Subsequently, a similar procedure as in Figure 2.4 is executed. In this case however, the loop is only executed k times. All cells that are still unassigned after the loop terminates are assigned 0 in the mutation mask.

From the mutation mask X and a parent image P, a model image $M = (m_{ij})$ is computed by assigning $m_{ij} = p_{ij}$ if $x_{ij} = 0$ and assigning m_{ij} a random value (0 or 1) if $x_{ij} = 1$. In other words, the mutation mask determines which part of the parent image will be distorted in the model image.

After the model image has been computed, the same steps are performed as for the crossover operator: Problem 8 is solved using the weight mask as M. Subsequently, the hillclimb operator is applied, yielding the child individual. Figure 2.9 shows the the different steps of the mutation operator. The dark cell in the mutation mask indicates the position of the seed. As the figure shows, there may be differences between the child and the parent individual outside the distorted region, as a result of the projection constraints.

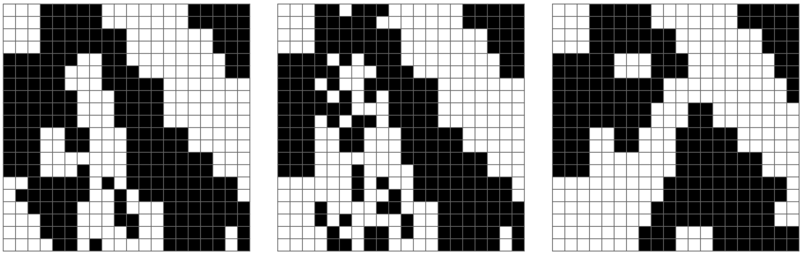
2.3.5. Algorithm details

In this section we address several details of our evolutionary algorithm that have not been discussed in the preceding sections.



a. parent

b. mutation mask



c. model image

d. child before hillclimb

e. child after hillclimb

Figure 2.9: *The mutation operator transforms a parent image into a child image.*

The initial population is generated by repeatedly solving Problem 8, each time using a new random binary matrix as the weight mask M. Although this procedure does not generate all matrices in $\mathcal{A}(R,S)$ with equal probability, it leads to sufficient diversity in the initial population.

The algorithm is designed to work with a variety of evaluation functions. We use tournament selection, because it is independent of the actual evaluation values (it only depends on their ordering) and because the selective pressure can be adjusted easily, by changing the tournament size.

Before applying the crossover operator, the two parents are randomly selected among the current population. The same applies to parent selection for the mutation operator. The probabilities of crossover vs. mutation are adjusted dynamically after each generation. For every generated child image, the algorithm keeps track of the operator that generated it. It also counts the number of children generated by crossover (a_c) and by mutation (a_m). After the new population has been selected, the algorithm counts the number of individuals in the new population that were created by crossover (b_c) and by mutation (b_m). From these numbers the average *crossover yield* $y_c = b_c/(a_c+1)$ and mutation yield $y_m = b_m/(a_m+1)$ are computed. (We add 1 to the denominator to avoid division by zero.) The probability that a child will be generated through crossover in the next generation is set to $y_c/(y_c+y_m)$. In this way, the algorithm creates more children by crossover in the next generation when the crossover operation in the current generation creates children of high quality. An explicit lower- and upperbound for y_c are used: $0.1 < y_c < 0.9$. If the computed value of y_c is outside these bounds, it is adjusted.

According to Section 2.3.4, the mutation operation randomly selects an integer k in the interval $[k_{min}, k_{max}]$. For all experiments in Section 2.4 we used $k_{min} = mn/64$ and $k_{max} = 5mn/64$.

Because the algorithm can be used with many different evaluation functions, it is difficult to design a universal termination condition. For some specific problems, an upper bound on the evaluation function value is known in advance and the algorithm can terminate as soon as an individual having that evaluation is found. Consider, for example, the reconstruction of an image from the horizontal and vertical projections and a third projection, see Section 2.4.2. In that case a projection deviation of 0 clearly corresponds to a global maximum of the evaluation function.

For such problems, having solutions for which optimality can easily be checked, separate termination procedures can be implemented in our system. In general, however, the algorithm terminates when no improvement on the best solution so far has been found in the last 20 generations.

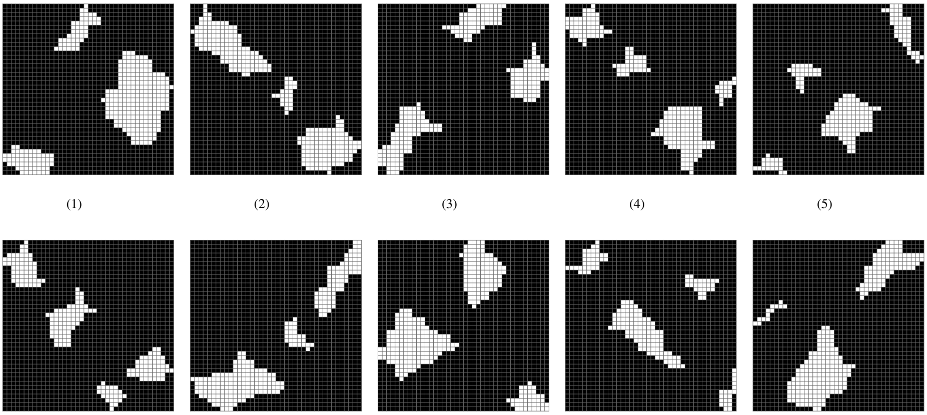
As indicated in Figure 2.3 the selection procedure selects individuals for the new population from both the old population and the group of new children. Individuals are allowed to be selected more than once for the new population. We performed experiments to find suitable values for the population size λ and the number of children μ that is created in each generation. We found that our algorithm performs well when $\lambda > \mu$, and with a large population. For all the experiments in Section 2.4 we used $\lambda = 1000$, $\mu = 500$ and a tournament size of 3. We remark that although we have chosen suitable values for the algorithm parameters, other values may work well too.

2.4. Computational results

We have implemented our algorithm in C++, using the gcc compiler. The implementation is fully object oriented. By using inheritance and virtual methods, different evaluation functions can be used without modifying the algorithm code. The evolutionary algorithm uses a base class *Individual* to represent a member of the population (a single image). For each evaluation function we implemented a derived class, which computes the evaluation function and performs delta updating computations. In this way new evaluation functions can be added with very little extra coding effort and without modifying existing code.

For the implementation of the network flow algorithm we used the *RelaxIV solver* [5], combined with the *MCFClass* library [10]. The source code of our algorithm (except the MCF library) is available from the author.

All our experiments were run on a Pentium IV 2800MHz PC with 512Mb of memory. We performed experiments with several evaluation functions. The evaluation functions are very different from each other, in order to demonstrate the versatility of the problem model and the algorithm. The experimental results are intended to demonstrate the feasibility and flexibility of our approach, not to provide extensive results on each of the individual problems that we consider.



(1) (2) (3) (4) (5)

(6) (7) (8) (9) (10)

Figure 2.10: *hv-convex test images*

2.4.1. Reconstruction of hv-convex images

The first class of test images consists of *hv-convex* images. Using methods from [16] we generated 10 hv-convex images of size 40 × 40. Each image consists of three or four hv-convex objects. There are no empty horizontal or vertical lines between the objects. The test images are shown in Figure 2.10. For this image class, the evaluation function is the total number of neighbouring white pixels (horizontal and vertical). We performed ten test runs for each of the images. Table 2.1 shows the results. We call a reconstruction *perfect* if it is the same as the original image. The table shows the number of runs that achieved a perfect reconstruction, the average runtime of the algorithm (in minutes) and the average number of generations after which the best reconstruction was found. When the algorithm did not find a perfect reconstruction, the reconstruction was always very different from the original image. Therefore, we do not report solution quality for those cases.

In the cases that the algorithm did not find the optimal solution, it converged to a local optimum. Figure 2.11 demonstrates the difficulties involved in this optimisation problem. It shows the second test image and another locally optimal image having the same projections. The images are very dissimilar, except for the object in the center. This problem is due to the presence of large blocks of switching components.

image	1	2	3	4	5	6	7	8	9	10
#perfect	9	5	8	8	9	6	6	6	6	9
average time (m)	25.6	16.6	15.5	8.8	8.5	10.8	17.7	17.7	8.8	15.7
average #gen.	24.6	17.3	18.5	14.8	15.8	18.5	30.2	15.0	21.0	23.6

Table 2.1: *Reconstruction results for the set of hv-convex images*

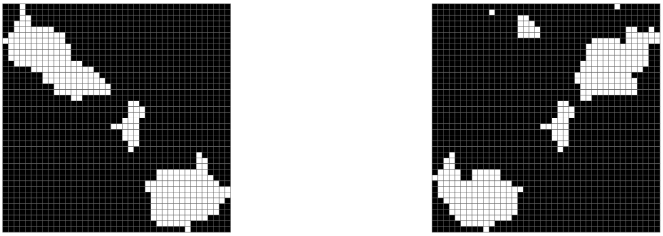
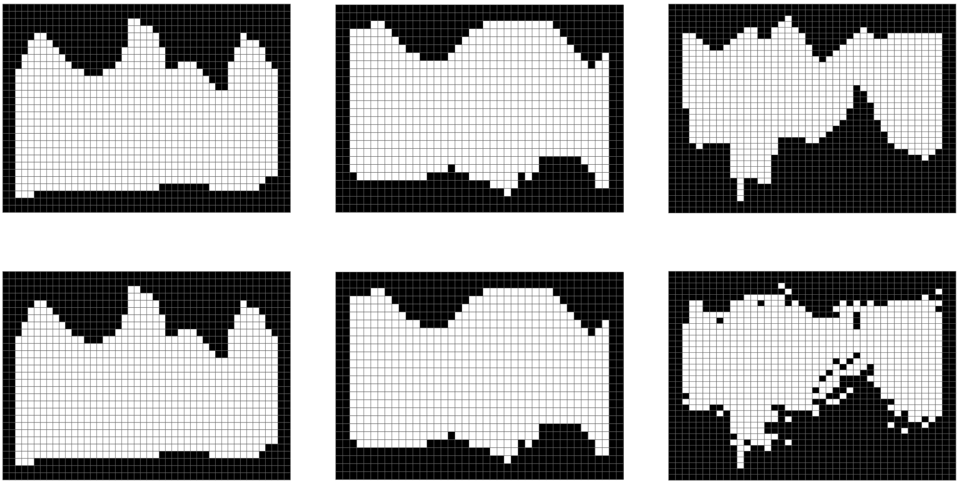


Figure 2.11: *Two locally optimal images having the same projections, yet being very dissimilar.*



(1) 29 × 46 (2) 26 × 41 (3) 36 × 42

Figure 2.12: *Three phantom images and their reconstructions.*

2.4.2. Reconstruction from three projections

The second group of test images are phantom images (i.e., simulated images) from [9]. These images have been used as test images in several publications ([14], [20]). In this case we reconstruct the images from three projections: horizontal, vertical and diagonal (top left to bottom right). Within our problem formulation, we consider the reconstruction problem from three projections as a special case of the reconstruction problem from two projections (horizontal and vertical), where the evaluation of an image depends on the deviation of its diagonal projection from the prescribed diagonal projection. The evaluation function to be maximised is the negated total deviation of the image's diagonal projection from the prescribed projection (as a sum of absolute values). The three test images and their reconstructions are shown in Figure 2.12. We performed a single test run for each of the images. The first two images were reconstructed perfectly (the reconstruction was equal to the original image) within 8 seconds. For the third image, however, the algorithm did not find a reconstruction with the given diagonal projection. The resulting reconstruction has a total difference of 4 from the prescribed diagonal projection, and took 30 seconds to compute.

The main strength of our algorithm is its flexibility. All test images are completely 4-connected and have no "holes" in them. When we assume this as *a priori information*, we can incorporate it in the evaluation function.

For an image A, put

$$f(A) = -c \times d(A) + b(A)$$

where $d(A)$ denotes the total difference between the diagonal projection of A and the prescribed projection, $b(A)$ denotes the total number of pairs of neighbouring white pixels in A and c is a very large constant. Maximising $f(A)$ will result in the reconstruction that satisfies the prescribed diagonal projection which has as many neighbouring white pixels as possible. Using this new evaluation function, the algorithm reconstructed the original images perfectly within one minute, even the third image. This experiment shows that reconstructions from three projections can be very accurate if appropriate a priori information is used and that such information can easily be incorporated in our algorithm.

2.4.3. Reconstruction using Gibbs priors

The third evaluation function involves a probability distribution, known as a *Gibbs* distribution. A method for using Gibbs priors in the reconstruction process was presented in [20]. We use similar definitions. We assume that the original image is a random sample of a certain known probability distribution. Let Π be such a probability distribution, on the set $\{0,1\}^{m\times n}$, which determines for each matrix the probability that it is sampled. The probability $\Pi(A)$ of a matrix A is given by

$$\Pi(A) = \frac{1}{Z} e^{\beta \sum_{i=1}^{m} \sum_{j=1}^{n} I_{ij}(A)}.$$

Here, Z is a normalising factor, β is a parameter defining the "peakedness" of the distribution and $I_{ij}(A)$ is the *local energy function* of cell (i,j). The local energy function of a cell is determined by the value of the cell and each of its eight neighbours. Border pixels, which have less than eight neighbours, are treated as if the lacking neighbours are black. Figure 2.14 shows the eight patterns for which the local energy is nonzero. The corresponding local energy values are k_w, k_b, k_e and k_c respectively. Each of the patterns corresponds to a particular local image feature, such as a homogeneous region or a corner. We denote a Gibbs distribution by the 5-tuple $(k_w, k_b, k_e, k_c, \beta)$, which we call the *parameters* of the distribution.

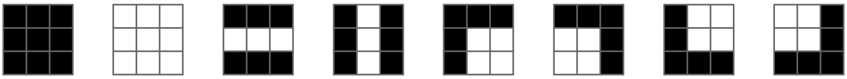
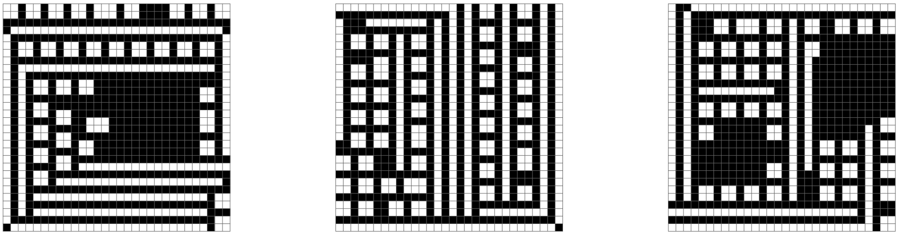


Figure 2.13: *The eight patterns having a nonzero local energy.*

As we know that the original image is a random sample from the given Gibbs distribution, we want to find a reconstruction that has maximal likelihood. We remark that this reconstruction is not guaranteed to be equal to the original image.



(1) (7,7,15,15,1.2) (2) (7,7,15,15,1.2) (3) (7,7,15,14,1.2)

Figure 2.14: *Three samples from Gibbs distributions.*

Maximising $\Pi(A)$ is equivalent to maximising $E(A) = \sum_{i=1}^{m} \sum_{j=1}^{n} I_{ij}(A)$, since $\Pi(A)$ is a monotonously increasing function of $E(A)$. Computing $E(A)$ involves only the known five parameters of the distribution and the image A, we do not need to compute Z.

We implemented a *Metropolis algorithm*, described in [20], for generating random samples from Gibbs distributions. Three such samples of size 30×30 and their parameters are shown in Figure 2.14. The reason that we selected three images, instead of showing results for a large number of images, is that many generated images are not tomographically challenging. For example, many contain several lines that only consist of black or white pixels. The fact that all three images consist of lines and blocks of four pixels is due to the selected patterns in Figure 2.13. Different patterns lead to other types of images. As our goal here is to illustrate the feasibility of our approach, we only show results for this set of patterns. We performed 10 test runs for each of the three images. Table 2.2 shows the reconstruction results. For the third image, the algorithm actually found a reconstruction that is slightly different from the original image, but has higher likelihood. We still marked this reconstruction as a perfect reconstruction. When the algorithm did not find a perfect reconstruction (image 2), the reconstruction was always very different from the original image. Therefore, we do not report solution quality for those cases. The results show clearly that perfect reconstruction from only two projections and a given Gibbs distribution is possible in nontrivial cases.

image	1	2	3
#perfect	10	5	10
average time (m)	16.4	19.5	14.4
average #generations	17.3	36.8	20.1

Table 2.2: *Reconstruction results for the Gibbs samples.*

2.5. Conclusions

We have presented a new algorithm for finding a binary image that satisfies prescribed horizontal and vertical projections and has an optimal or near-optimal evaluation function value. Our experimental results demonstrate that the algorithm is effective for several different evaluation functions, corresponding to diverse reconstruction problems.

As demonstrated by the results in Section 2.4.2 a main feature of our approach is its ability to incorporate various forms of a priori information in the evaluation function.

The bottleneck of the algorithm is the hillclimb operator. Because our hillclimb procedure involves keeping track of all applicable switching components, our algorithm is limited to images of size 50×50 or less. Perhaps making the hillclimb operation less exhaustive could result in improved time complexity of the operation.

Although we used the problem of reconstructing a binary image from more than two projections to demonstrate the flexibility of our approach, more effective algorithms are available that deal with this problem. In the next chapter we will present an algorithm for this problem that is capable of reconstructing much larger images.

In our experiments we often observed that really finding the global optimum of the evaluation function was necessary to find an image that resembled the unknown original image, which was used to generate the projection data. Near-optimal solutions were usually very different. In order to increase the stability of the search procedure, it is necessary to incorporate as much information as possible in the evaluation function. The various reconstruction results show that when sufficient information is incorporated, our algorithm is capable of finding accurate reconstructions.

A topic of further research will be the development of a variant of our evolutionary algorithm that is capable of reconstructing an image from noisy projection data. At present we assume that all projection data are perfect, which may not be realistic for practical applications.

Bibliography

[1] Aarts, E., Korst. J.: *Simulated annealing and boltzmann machines: a stochastic approach to combinatorial optimization and neural computing*, Wiley (1996).

[2] Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: *Network flows: theory, algorithms and applications*, Prentice-Hall (1993).

[3] Bäck, T., Fogel, D.B., Michalewicz, T., eds.: *Evolutionary computation 1*, Institute of Physics Publishing, Bristol and Philadelphia, (2000).

[4] Barcucci, E., Del Lungo, A., Nivat, M., Pinzani, R.: Reconstructing convex polyominoes from horizontal and vertical projections, *Theoret. Comp. Sci.*, **155**, 321–347 (1996).

[5] Bertsekas, D.P., Tseng, P.: RELAX-IV: a faster version of the RELAX code for solving minimum cost flow problems. *LIDS Technical Report LIDS-P-2276*, MIT, (1994).

[6] Brunetti, S., Del Lungo, A., Del Ristoro, F., Kuba, A., Nivat. M.: Reconstruction of 4- and 8-connected convex discrete sets from row and column projections, *Linear Algebra Appl.*, **339**, 37–57 (2001).

[7] Corne, D., Dorigo, M., Glover. F.: *New ideas in optimisation*, McGraw-Hill Education, (1999).

[8] Dahl, G., Flatberg, T.: Optimization and reconstruction of hv-convex (0,1)-matrices, *Electron. Notes Discrete Math.*, **12**, (2003).

[9] Fishburn, P., Schwander, P., Shepp, L., Vanderbei, R.: The discrete Radon transform and its approximate inversion via linear programming, *Discrete Appl. Math.* **75**, 39–61 (1997).

[10] Frangioni, A., Gentile, C.: The MCFClass Project (2003).
`http://www.di.unipi.it/di/groups/optimize/Software/MCF.html`

[11] Gale, D.: A theorem on flows in networks. *Pacific J. Math.*, **7**, 1073–1082 (1957).

[12] Gardner, R.J., Gritzmann, P., Prangenberg, D.: On the computational complexity of reconstructing lattice sets from their X-rays. *Discrete Math.*, **202**, 45–71 (1999).

[13] Glover, F., Laguna, M.: *Tabu Search*. Kluwer Academic Publishers (1998).

[14] Hajdu, L., Tijdeman, R.: An algorithm for discrete tomography. *Linear Algebra Appl.*, **339**, 147–169 (2001).

[15] Herman, G.T., Kuba, A., eds.: *Discrete tomography: foundations, algorithms and applications*. Birkhäuser, Boston (1999).

[16] Hochstättler, W., Loebl, M., Moll, C.: Generating convex polyominoes at random. *Discrete Math.*, **153**, 165–176 (1996).

[17] Kisielowski, C., Schwander, P., Baumann, F., Seibt, M. Kim, Y., Ourmazd, A.: An approach to quantitative high-resolution transmission electron microscopy of crystalline materials. *Ultramicroscopy* **58**, 131–155 (1995).

[18] Kuba, A., Herman, G.T.: Discrete tomography: a historical overview. *Chapter 1 of [15]*, 3–34 (1999).

[19] Del Lungo, A., Nivat, M.: Reconstruction of connected sets from two projections. *Chapter 7 of [15]*, 163–188 (1999).

[20] Matej, S., Vardi, A., Herman, G.T., Vardi. E.: Binary tomography using gibbs priors. *Chapter 8 of [15]*, 191–212 (1999).

[21] Michalewicz, Z.: *Genetic Algorithms + Data Structures = Evolution Programs; 3rd Revision edition*. Springer Verlag (1996).

[22] Onnasch, D.G.W., Prause, G.P.M.: Heart chamber reconstruction from biplane angiography. *Chapter 17 of [15]*, 385–401 (1999).

[23] Ryser, H.J.: Combinatorial properties of matrices of zeros and ones. *Can J. Math.*, **9**, 371–377 (1957).

[24] Ryser, H.J.: *Combinatorial mathematics*, The Carus Mathematical Monographs, **14**, Math. Assoc. of America, John Wiley and Sons, Inc., New York (1963).

[25] Schwander, P., Kisielowski, C., Baumann, F., Kim, Y., Ourmazd, A.: Mapping projected potential, interfacial roughness, and composition in general crystalline solids by quantitative transmission electron microscopy. *Phys. Rev. Lett.*, **71**, 4150–4153 (1993).

[26] Slump, C.H., Gerbrands, J.J.: A network flow approach to reconstruction of the left ventricle from two projections. *Comput. Gr. Im. Proc.*, **18**, 18–36 (1982).

[27] Wang, B., Zhang, F.: On the precise number of $(0,1)$-matrices in $\mathcal{A}(R,S)$. *Discrete Math.*, **187**, 211–220 (1998).

Chapter 3

A network flow algorithm for reconstructing binary images from discrete X-rays

This chapter has been accepted for publication in the Journal of Mathematical Imaging and Vision, Springer (2006).

Abstract We present a new algorithm for reconstructing binary images from their projections along a small number of directions. Our algorithm performs a sequence of related reconstructions, each using only two projections. The algorithm makes extensive use of network flow algorithms for solving the two-projection subproblems.

Our experimental results demonstrate that the algorithm can compute highly accurate reconstructions from a small number of projections, even in the presence of noise. Although the effectiveness of the algorithm is based on certain smoothness assumptions about the image, even tiny, non-smooth details are reconstructed exactly. The class of images for which the algorithm is most effective includes images of convex objects, but images of objects that contain holes or consist of multiple components can also be reconstructed very well.

3.1. Introduction

Discrete tomography (DT) is concerned with the reconstruction of a discrete image from its projections. In our context, the image is defined on a discrete set of lattice points and the set of possible pixel values is also discrete. The latter property distinguishes discrete tomography from continuous tomography. We will restrict ourselves to images that have only two possible pixel values, 0 (black) and 1 (white). Typically, the number of directions in which the projection data are available is very small.

Before practical applications of discrete tomography were considered, several problems of DT had already been introduced as interesting combinatorial problems. In 1957, Ryser [19] and Gale [7] independently presented necessary and sufficient conditions for the existence of a binary matrix with prescribed row and column sums. Ryser also provided a polynomial time algorithm for reconstructing such matrices.

Over the past ten years many of the basic DT problems have been studied extensively. A principal motivation for this research was a demand from material sciences for improved reconstruction techniques due to advances in electron microscopy [15–17, 21]. Since its inception around 1994, the field of DT has attracted many researchers, resulting in numerous publications and meetings. Discrete tomography is considered one of the most promising approaches to making atomic resolution 3D reconstructions of crystals in electron microscopy. Besides applications in electron microscopy DT has several other applications, such as medical imaging (particularly angiography [22]) and industrial tomography.

Although polynomial-time algorithms have been shown to exist for a number of specific DT problems, many of the more general DT problems are NP-complete. In particular it was shown by Gardner, Gritzmann and Prangenberg [8] that the problem of reconstructing a binary image from three or more projections is NP-complete.

In general, the problem of reconstructing binary images from a small number of projections is underdetermined. When only two projections are available, horizontal and vertical, the number of solutions can be very large [24]. Moreover, there can be solutions which are substantially different from each other. Similar difficulties occur when more than two projections are given: the number of solutions can be exponential in the size of the image for any set of projection directions.

Fortunately, images that occur in practical applications are rarely random; they usually exhibit a certain amount of "structure". Several authors have come up with algorithms for reconstructing binary matrices that satisfy additional constraints, such as certain forms of convexity or connectedness [3, 4]. In some cases a reconstruction can still be found in polynomial time. However, in practical situations such assumptions on the structure of the image are often too restrictive. Also, few proposed algorithms for the two-projection case can be generalized to the case where more than two projections are available.

Algorithmic approaches for more general DT problems have been proposed in [5, 10, 13, 25]. In [5] the authors propose a relaxation of the original problem that can be solved efficiently by linear programming. In [25] the approach from [5] is extended to include a smoothness prior. Several greedy heuristic algorithms for reconstructing binary images from more than two projections are described in [10]. The authors of [13] propose an iterative algorithm which starts at a real-valued solution and gradually moves toward a binary solution.

A well-known technique that can be used to improve reconstruction quality in some cases is the introduction of a smoothness prior. For the binary reconstruction problem this means that we try to find a reconstruction for which most local neighbourhoods of pixels are either completely black or completely white. For images that occur in practice, such smoothness assumptions are often satisfied. A disadvantage of this approach can be that very tiny local details tend to be neglected by the reconstruction algorithm, in favour of the smoothness constraints.

In this chapter we present a new algorithm for approximately reconstructing binary images from their projections. Although smoothness assumptions are used to guide the algorithm during the reconstruction procedure, they are not so dominant that they blur out the very fine details. In this way, the relative smoothness of many practical images can be used by the algorithm to obtain reconstructions, while the resulting reconstructions still show a very high detail level. Even single-pixel details are often reconstructed exactly.

The algorithm is very effective on a broad spectrum of binary images. We will show experimental evidence that the algorithm can — when used on images that contain large black or white areas — compute reconstructions that are almost or even completely identical to the original image, from a very small number of projections. The class of images for which the algorithm performs well includes the images of convex objects, but even objects that contain a large number of "holes" or consist of many separated components can be reconstructed with very high accuracy. We are not aware of any other algorithm for this reconstruction problem that achieves similar reconstruction quality on such test cases. A comparison with two alternative approaches is described in Section 3.4.7. Our algorithm is also very fast, which makes it suitable for reconstructing large images.

Our algorithm reconstructs an image from three or more projections by performing a sequence of related reconstructions, each using only two projections. We use a network flow approach for solving the two-projection problems.

In Section 3.2 we introduce some notation and describe our reconstruction problem in a formal context. Section 3.3 forms the main part of this chapter. We introduce our algorithm and describe each of its parts in detail. In Section 3.4 we present extensive experimental results on the performance of our algorithm and show how the algorithm can be adapted to work with noisy projection data. We also give a performance comparison between our algorithm and two alternative approaches.

3.2. Preliminaries

We restrict ourselves to the reconstruction of 2-dimensional images, although the algorithm that we propose can be used for d-dimensional images where $d > 2$ as well. The cases $d = 2,3$ are of most practical interest.

We denote the cardinality of a finite set V by $|V|$ and we denote the set $\{n \in \mathbb{Z} : n \geq 0\}$ by $\mathbb{N}_0$. Let m,n be positive integers. Put

$$A = \{(i,j) \in \mathbb{Z}^2 : 1 \leq i \leq n, 1 \leq j \leq m\}$$

and let $\mathcal{F}$ denote the collection of mappings $A \to \{0,1\}$. We call m and n the *height* and *width* of A respectively. We will sometimes regard the elements of $\mathcal{F}$ as *matrices* or *images*. Similarly, we will refer to the elements of A as *entries* or *pixels* respectively and to the values of pixels as colours, *black* (0) and *white* (1).

We call a vector $v = (a,b) \in \mathbb{Z}^2 \backslash \{(0,0)\}$ a *lattice direction* if a and b are coprime. For every lattice direction v, we define the *lattice line* $\mathcal{L}_v = \{nv \mid n \in \mathbb{Z}\}$. We denote the set of lines parallel to $\mathcal{L}_v$ by $\mathcal{P}_v = \{\mathcal{L}_v + t \mid t \in \mathbb{Z}^2\}$ and the subset of lines in $\mathcal{P}_v$ that contain at least one point of A by $\mathcal{S}_v = \{P \in \mathcal{P}_v \mid P \cap A \neq \emptyset\}$. Let $F \in \mathcal{F}$. We call the function $X_v F : \mathcal{S}_v \to \mathbb{N}_0$

defined by

$$X_vF(P) = |\{(x,y) \in \mathbb{Z}^2 : (x,y) \in P \cap A \text{ and } F(x,y) = 1\}|$$

for $P \in \mathcal{S}_v$ the *(discrete) 1-dimensional X-ray of F parallel to v*. We will also refer to the X-rays as *projections*.

In the inverse reconstruction problem, we want to construct a function $F \in \mathcal{F}$ which has prescribed 1-dimensional X-rays, parallel to a number of lattice directions $v_1, v_2, \dots, v_k$. More formally:

Problem 9 (Reconstruction problem).
Let $k \geq 1$ and $v_1, \dots v_k$ be given distinct lattice directions. Let $\phi_i : \mathcal{S}_{v_i} \to \mathbb{N}_0 (i = 1, \dots, k)$ be given functions. Find a function $F \in \mathcal{F}$ such that $X_{v_i}F = \phi_i$ for $i = 1, \dots, k$.

Because the functions ϕ_i all have finite domains, we sometimes consider ϕ_i to be a vector of $|\mathcal{S}_{v_i}|$ elements. We will refer to this vector as a *prescribed projection* and to its elements as *prescribed linesums*.

It is important to notice that there may be many $F \in \mathcal{F}$ having the prescribed X-rays even if the number of lattice directions is large [12]. However, the number of solutions depends heavily on the actual X-rays, which in turn depend on the “original image” that was measured.

Our algorithm is based on the fact that the problem of reconstructing binary images from two projections can be solved efficiently. In each iteration our algorithm takes one pair of projections and finds an image which satisfies those two projections exactly. We use polynomial time network flow algorithms for solving the two-projection problems. See [1] for an excellent reference on network flow algorithms. In the next sections we will assume that the reader is familiar with the basic ideas of network flows.

One can also regard the reconstruction problem as the problem of finding a 0-1 solution of a system of linear equations, which describes all the linesum constraints. We write this system as $Bx = b$, where every entry of the vector x corresponds to a pixel and every row in the matrix B corresponds to a projected lattice line. The column vector b contains the linesums for each of the k projections. We will use this notation several times in the next sections, referring to B as the *projection matrix* and to b as the *vector of prescribed linesums*. When we consider an image as a vector of pixel values, we refer to the *vector representation* of an image.

Let x be the vector representation of an image. We define the distance $\text{dist}(x)$ between the projections of x and the prescribed projections as

$$\text{dist}(x) = |Bx - b|,$$

where $|.|$ denotes the Euclidean norm.

3.3. Algorithm description

3.3.1. Overview

The purpose of our algorithm is to solve Problem 9 for more than two lattice directions. Our algorithm is iterative: in each iteration a new reconstruction is computed, using the reconstruction from the previous iteration.

To obtain a start solution, we solve a real-valued relaxation of the binary reconstruction problem, using all lattice directions. Every iteration consists of choosing two projection directions and then constructing an image that satisfies the linesums in those directions perfectly. Typically, this two-projection problem has many solutions. We use the reconstruction from the previous iteration (corresponding to a different pair of directions) to determine which of the solutions is selected. A weighted network flow model is used for solving the two-projection problems. By choosing appropriate weights, the reconstruction from the previous iteration can be incorporated into the new reconstruction process. The weights are chosen in such a way that the new reconstruction is not very different from the previous reconstruction. In this way information gathered from previous reconstructions is passed on to the new reconstruction.

The choice of the weights also depends on local smoothness properties of the image that resulted from the previous iteration. In this way, the assumption that the image is relatively smooth is used throughout the algorithm. As we will show in Section 3.4, this does not restrict the algorithm to the reconstruction of completely smooth images.

3.3.2. Network flow approach

Our algorithm makes extensive use of the network flow method for reconstructing binary images from two projections. This approach has been studied by several authors [2,22]. As an example, consider the two-direction problem in Figure 3.1, where the horizontal projection $r = (r_1, r_2, r_3)$ and the vertical projection $s = (s_1, s_2, s_3)$ of a 3×3 binary image F are given.

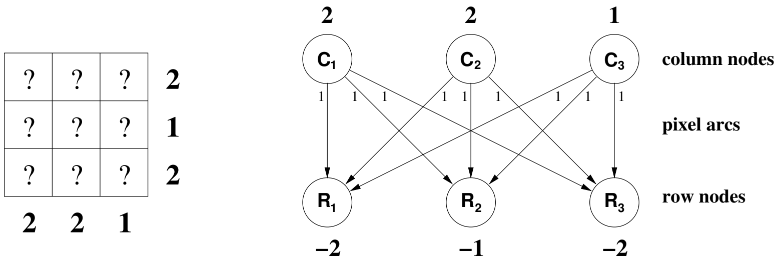
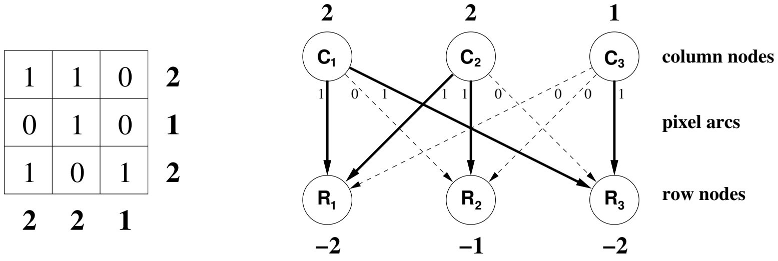


Figure 3.1: *A 3×3 reconstruction problem and the corresponding network.*

We construct a capacitated network G, as shown in Figure 3.1. The *column nodes* C_1, C_2, C_3 correspond to the image columns, the *row nodes* R_1, R_2, R_3 correspond to the image rows. There is an arc between every pair of column and row nodes. Each arc (C_i, R_j)



1	1	0	**2**
0	1	0	**1**
1	0	1	**2**
2	**2**	**1**	

Figure 3.2: *A solution of the transportation problem in G corresponds to a solution of the reconstruction problem. The number along each arc indicates the flow through that arc.*

corresponds to a single pixel of the image (the cell that intersects with column i and row j). Therefore we will refer to these arcs as *pixel arcs*. All pixel arcs are assigned a capacity of 1. The column nodes act as *sources*. The linesum for each column determines the (nonnegative) excess at the corresponding column node. The row nodes act as *sinks*. Every row node has a nonpositive excess (i.e., a shortage), which is the negated linesum of the corresponding row. The excess of the column nodes and row nodes are shown in Figure 3.1. We denote the flow through arc (C_i, R_j) by x_{ij}. We now consider the *transportation problem* in G, which is the problem of finding a flow $X = (x_{ij})$ such that

$$\begin{aligned} \sum_{i=1}^{3} x_{ij} &= r_j \quad \text{for } j = 1,2,3, \\ \sum_{j=1}^{3} x_{ij} &= s_i \quad \text{for } i = 1,2,3, \\ 0 \leq x_{ij} &\leq 1 \quad \text{for } 1 \leq i,j \leq 3. \end{aligned}$$

This transportation problem is a special case of the more general max flow problem. It is a well known fact that if all arc capacities are integral and if the problem is feasible (i.e., there is a solution), there exists a solution for which all the arc flows x_{ij} are integral. As all arcs of G have a capacity of 1, all arc flows in such a solution will be either 0 or 1. Efficient methods for finding a max flow are described in [1].

A reconstruction F' which has the same row and column sums as F can now be computed by first solving the transportation problem in the network G. Subsequently, each pixel of F' is assigned the flow through the corresponding pixel arc of G (either 0 or 1). Figure 3.2 illustrates the construction. The thickness of each arc depicts the flow through that arc (0 or 1).

The construction does not require the two directions to be horizontal and vertical; any pair of lattice directions can be used. In that case we refer to nodes in the graph G as *line nodes*. The resulting bipartite graph is not necessarily complete: a pair of line nodes is connected by an arc if and only if the two lines that correspond to these two nodes intersect within the image domain A. In the remainder of this section we will keep using the notation of the example, involving the horizontal and vertical projections. All presented concepts can be used for other pairs of lattice directions as well.

As noted before, the problem of reconstructing binary images from two projections can have a large number of solutions. When extra *a priori* information is available, such as an approximate reconstruction, we can use this information to select the most preferable solution by assigning a weight w_{ij} to every pixel arc (C_i, R_j). We will refer to these weights as *pixel weights*. We now try to find a solution of the transportation problem that maximizes the total weight:

$$\textbf{maximize} \sum_{(i,j)\in A} w_{ij}x_{ij}$$

$$\textit{such that } X = (x_{ij}) \textit{ is a solution of the transportation problem in } G.$$

The problem of finding the solution of maximal weight is a special case of the min-cost max-flow problem (in which the arc costs are the negated weights). Several efficient algorithms exist for this type of problem. This weighted network flow approach was already presented in [22] for reconstructing an image of the left ventricle using a priori information.

3.3.3. An iterative network flow algorithm

Figure 3.3 shows the basic steps of our algorithm. First, a start solution is computed by solving a real relaxation of the binary problem. This start solution is used to determine the pixel weights of the network G that corresponds to the first two directions. By solving a min-cost max-flow problem in G a binary solution is obtained which satisfies the projections in the first two directions exactly, while closely resembling the start solution. Subsequently, the new solution is used for determining the pixel weights of the network that corresponds to a different pair of directions. A preference for local smoothness of the image is also incorporated in this computation. The procedure is repeated until some termination condition is satisfied.

A slightly similar approach was briefly suggested in [11], but as far as we know it has never been further explored. It does not use a smoothness assumption, which plays in important role in our approach.

In the next subsections we will provide concrete algorithms for computing the start solution, choosing the pair of directions for every step of the algorithm and choosing the pixel weights.

3.3.4. Computing the start solution

At the start of the algorithm we do not have a solution of a previous network flow problem that we can use for choosing the pixel weights of the first network. One approach would be to assign equal weights to all pixels. We have chosen to solve a real-valued relaxation of the binary tomography problem and determine the pixel weights of the first network from the real solution. Consider the system of linear equations

$$Bx = b \tag{3.1}$$

```
Compute the real-valued start solution $X^* = (x^*_{ij})$ (see Section 3.3.4);
Compute the binary start solution $X^0$ by solving a min-cost
max-flow problem for directions $v_1$ and $v_2$, using $w_{ij} = x^*_{ij}$;
$i := 0$;
while (stop criterion is not met) do
begin
    $i := i + 1$;
    Select a new pair of directions $v_a$ and $v_b$ $(1 \leq a < b \leq k)$,
    see Section 3.3.6;
    Compute a new solution $X^i$ by solving the min-cost
    max-flow problem for directions $v_a$ and $v_b$, using
    the weight function from Section 3.3.5;
end
```

Figure 3.3: *Basic steps of the algorithm.*

where B is the projection matrix and b is the vector of prescribed linesums. We now allow the pixel values (the entries of the vector x) to have any real value instead of just 0 and 1. Because the system of equations is underdetermined it usually has an infinite number of solutions. We compute the shortest solution x^* with respect to the Euclidean norm. The motivation for using this particular solution comes from the following two observations, both from [12]:

Observation 1 *Suppose that the system (3.1) has a binary solution, $\tilde{x}$. Let $\hat{x}$ be any binary solution of (3.1). Then $|\hat{x} - x^*| = |\tilde{x} - x^*|$.*

This follows from the fact that x^* is the orthogonal projection of the origin onto the solution manifold $W = \{x \in \mathbb{R}^{mn} : Bx = b\}$. Because $\tilde{x}$ is also in W we can apply the Pythagorean formula:

$$|\tilde{x} - x^*|^2 = |\tilde{x}|^2 - |x^*|^2$$

But because $\tilde{x}$ is a binary vector we have

$$|\tilde{x}| = |\{1 \leq i \leq mn : \tilde{x}_i = 1\}| = \sum_{j=1}^{t} b_j$$

where $b_1, \ldots, b_t$ are the linesums corresponding to the first lattice direction. We observe that $|\tilde{x} - x^*|$ only depends on b. By substituting $\hat{x}$ for $\tilde{x}$ we find that $|\hat{x} - x^*| = |\tilde{x} - x^*|$. Observation 1 shows that x^* is “centered in between” all binary solutions.

Observation 2 *Suppose that the system (3.1) has a binary solution $\tilde{x}$. Let $\bar{x}$ be an integral solution of (3.1). Then we have $|\tilde{x}| \leq |\bar{x}|$.*

This follows from the fact that

$$\sum_{i=1}^{mn} \tilde{x}_i^2 = \sum_{i=1}^{mn} \tilde{x}_i = \sum_{j=1}^{t} b_j = \sum_{i=1}^{mn} \bar{x}_i \leq \sum_{i=1}^{mn} \bar{x}_i^2 .$$

where $b_1, \ldots, b_t$ are the linesums corresponding to the first lattice direction. We use the fact that 0 and 1 are both equal to their squares. Observation 2 shows that the binary solutions of (3.1) have the smallest norm among all integer solutions. Therefore the smallest real solution of the system seems to be a good first approximation to a binary solution.

We use the iterative projection method from [23] for solving the system (3.1). Because the matrix B is very sparse all computations can be performed efficiently. The accuracy that is required for our purpose is quite low, because we are only seeking a first approximation. Therefore we can terminate the iterative algorithm after a small number of iterations. For all experiments in Section 3.4 we used 300 iterations.

After the pair of projections for the first iteration of the reconstruction algorithm has been selected, the entries of x^* are used as pixel weights in the corresponding transportation problem.

3.3.5. Computing the pixel weights

In every iteration of the algorithm the pixel weights are recomputed for all pixels in the image. In this computation we incorporate the preference of smooth images over random images, having no particular structure. The new weight of a pixel depends not only on the corresponding pixel value in the reconstruction from the previous iteration, but also on the values of pixels in a neighbourhood.

Let $F \in \mathcal{F}$ be the reconstructed image from the previous iteration. As a neighbourhood of the pixel $p = (x_p, y_p)$ we choose a square centered in (x_p, y_p). The reason for preferring a square neighbourhood over alternatives is that the required computations can be performed very efficiently in this case. Let $p = (x_p, y_p) \in A$. Let r be a positive integer, the *neighbourhood radius*. Put

$$N_p = \{\, (x,y) \in A : x_p - r \leq x \leq x_p + r,\, y_p - r \leq y \leq y_p + r \,\}.$$

In case p is near the boundary of the image, the neighbourhood N_p may contain fewer than $(2r+1)^2$ pixels. Let s_p be the number of pixels in N that have the same colour (i.e., pixel value) as p, and let f_p be the relative fraction of such pixels:

$$\begin{aligned} s_p &= |\{\, (x,y) \in N_p : \quad F(x,y) = F(x_p, y_p) \,\}|, \\ f_p &= \frac{s_p}{|N_p|}. \end{aligned}$$

Let $g : [0,1] \to \mathbb{R}_{>0}$ be a nondecreasing function, the *local weight function*. This function determines the preference for locally smooth regions. We compute the pixel weight w_{x_p,y_p} of p as follows:

$$w_{x_p,y_p} = (F(x_p, y_p) - \frac{1}{2})\, g(f_p)$$

Note that $(F(x_p, y_p) - \frac{1}{2})$ is either $-\frac{1}{2}$ or $\frac{1}{2}$.

The vector of all pixel weights can be computed in time $O(mn)$, regardless of the neighbourhood radius r. First, the number of black pixels in all rectangles having $(0,0)$ as a corner is computed. Subsequently, the number of black pixels in each rectangular pixel neighbourhood can be computed as a linear combination of the values for rectangles having corner $(0,0)$. We omit the details here.

When we take $g(f_p) = 1$ for all $f_p \in [0,1]$, there is no preference for local smoothness. In that case the solution of the new network flow problem will simply have the same value as F in as many pixels as possible. For a constant weight function, the algorithm usually does not converge. We will show in Section 3.4 that incorporating a preference for local smoothness in the local weight function results (empirically) in good convergence for images that are relatively smooth.

We performed several preliminary experiments to determine a good local weight function. Several different functions appear to work equally well. For the experiments in Section 3.4 we used the following local weight function:

$$g(f_p) = \begin{cases} 1 & (f_p \leq 0.65) \\ 4f & (0.65 < f_p < 1) \\ 9 & (f_p = 1). \end{cases}$$

The choice for this particular function is somewhat arbitrary. In each of the three cases, a preference is expressed for retaining the pixel value of p in the next reconstruction, instead of changing it. In the case that the whole neighbourhood of p has the same colour as p, this preference is very strong. If the neighbourhood contains a few pixels having a different colour, the preference is smaller. If there are many pixels in the neighbourhood that have a different colour, the preference is even smaller.

The neighbourhood radius is not constant during the algorithm. In the experiments, we used two phases: during the first phase, which resolves the coarse structure of the image, we use $r = 8$. The large neighbourhood radius leads to a strong smoothing effect. After 50 iterations, the neighbourhood radius is set to 1. From this point on, the fine details in the image are gradually reconstructed.

Most min-cost max-flow algorithms (that can be used to find a maximal weight solution of the transportation problem) work with integer costs. We multiply all pixel weights by a large integer L and round the results to obtain integral weights. In the experiments of Section 3.4 we use $L = 10000$.

3.3.6. Choosing the direction pair

In every iteration of the algorithm a new pair of projections is selected which must be different from the pair of the previous iteration. It is allowed however that one of the two chosen projections is also used in the previous iteration.

When the number of prescribed projections is small (not greater than 6), we choose to iterate over all possible pairs of two directions. It is of great importance for the performance of the algorithm that all *pairs* of projections are used, not just all single projections. Tables

3.1 and 3.2 show the order in which the projection pairs are used for the case of four and five prescribed projections respectively, where the projections are numbered from one through four (five). These orders perform very well in practice, but several other orders which show similar performance can be chosen.

Table 3.1: *Projection-pair order for four projections*

iteration	1	2	3	4	5	6
1st direction	1	3	1	2	1	2
2nd direction	2	4	3	4	4	3

Table 3.2: *Projection-pair order for five projections*

iteration	1	2	3	4	5	6	7	8	9	10
1st direction	1	3	5	2	4	1	2	3	4	5
2nd direction	2	4	1	3	5	3	4	5	1	2

When the number of directions is larger it is no longer feasible to iterate over all possible direction pairs, which grows quadratically with the number of directions. Instead we use a heuristic to select the new direction pair. The heuristic first computes all projections of the current images. Then it compares these projections to the prescribed projection data and selects the pair of directions for which the total difference (the sum of the absolute values of linesum differences) between the image projections and the prescribed projections is the largest. We remark that unless the current reconstruction perfectly satisfies all projection constraints, these directions will always be different from the two directions used in the previous iteration. This heuristic is not suitable when the number of projections is very small. When reconstructing from four projections, for example, using the heuristic would result in using only two direction pairs, alternating between those pairs. Such cycling is much less likely to occur when the number of projections is larger.

3.3.7. Stop criterion

As we cannot evaluate how close the current reconstructed image is to the unknown original image, the only measure for the distance between those two images is the distance between their respective projections. In theory, this is not a good measure at all, since there may be two images which are very different, yet have similar projections. For the classes of smooth images that we focus on, however, this does not seem to be the case. In all our experiments using more than three projections we found that when the projections of the reconstruction got close to the prescribed projections, the reconstructed image was also close to the original image. Only when using three projections it sometimes occurred that the reconstructed image was quite different from the original image while the projections of both images were almost the same.

We use the distance between the projections of the current reconstructed image and the prescribed projections for defining termination conditions for our algorithm. When the reconstructed image has exactly the prescribed projections, the algorithm terminates. If no improvement is made in the distance between the prescribed projections and the projections of the reconstructed image in the last T iterations, where T is a positive integer, the algorithm also terminates. We used $T = 100$ for all experiments in Section 3.4.

Whenever the distance between the projections of the current image and the prescribed projections becomes less than a certain constant S the algorithm will always terminate after a maximum of N iterations, where N is a positive integer. This is to avoid cycling around the optimal solution. We used $S = 100$, $N = 50$ for all experiments. The algorithm always terminates after a maximum number U of iterations. We used $U = 1500$ in the experiments.

3.3.8. Time complexity

An important factor of the time complexity of our algorithm is determined by the complexity of the network flow solver that is used. The specific network flow problem that needs to be solved in each iteration is known in the literature as *(simple) capacitated b-matching* in a bipartite graph. Section 21.13a of [20] provides a complexity survey of this problem.

Let G be the graph for the transportation problem corresponding to lattice directions v_a and v_b. Let $M = mn$ be the number of pixel arcs in G and N be the total number of nodes. The number of nodes depends on v_a and v_b. For a fixed pair of lattice directions, we have $N = O(\max(m,n))$. By multiplying all pixel weights with a large integer and rounding the result we ensure that all pixel weights are integral. Let W be the maximal arc weight in the graph, i.e., the maximum of the pixel weights. In our case W is bounded by a constant. Gabow and Tarjan proposed an algorithm [6] having time complexity $O(N^{2/3}MC^{4/3}\log(NW))$, where C is the maximum of the arc capacities. For our problem we have $C = 1$, so this results in a time complexity of $O(\max(m,n)^{2/3} mn \log(\max(m,n)))$. If the image domain A is square, this yields a time complexity of $O(n^{8/3}\log n)$. As mentioned in Section 3.3.5, the computation of pixel weights can be performed in $O(mn)$ time.

Our algorithm is always terminated after at most U iterations (see Section 3.3.7). Ideally, one would hope for a result stating that the algorithm always converges to a solution of the reconstruction problem. In our case it seems unlikely that such a result is possible, as the convergence properties of the algorithm depend strongly on the particular (original) image that is reconstructed. To prove strong convergence results it seems necessary to consider only a very narrow class of images, instead of the wide range that we consider in this chapter.

3.4. Results

3.4.1. Experimental setup

We have implemented our algorithm in C++. We use the *CS2* network flow library by Andrew Goldberg for solving the min-cost max-flow problems (see [9]). This library is publicly available for noncommercial use. All results in this section were obtained using an AMD Athlon XP2800 PC.

The main criterion for evaluating the performance of any DT algorithm should be the resemblance of the reconstructed image to the measured physical object. Although in practical situations the “original” image is unknown, we can compare the original image with the reconstruction when working with artificial test images.

Different classes of images require a different number of projections to be reconstructed correctly. So far, we have no way to compute in advance how many projections will be enough. We use a fixed set of directions and do not focus here on which set of directions is best for our algorithm. Put

$$\begin{aligned} D := \{v_1,\ldots,v_{16}\} = \{ & (1,0),(0,1),(1,1),(1,-1),(1,2),(2,-1) \\ & (1,-2),(2,1),(2,3),(3,-2),(2,-3),(3,2) \\ & (1,3),(3,-1),(1,-3),(3,1)\} \end{aligned}$$

When reconstructing images from $k \leq 16$ projections, we use the set $\{v_1,\ldots,v_k\}$ of directions.

3.4.2. Qualitative description of algorithm performance

Our algorithm can be used for a wide range of images and a wide range of available projections. The number of projections that is required for accurate reconstruction greatly depends on the type of image. Before describing more detailed results for specific image classes, we will give a more qualitative description of the performance of our algorithm.

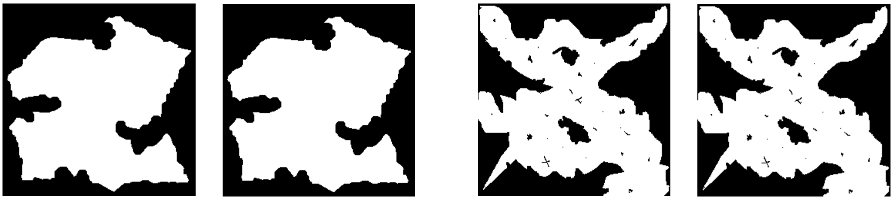
For reconstructing images such as the one shown in Figure 3.4a, four projections usually suffice. The structure of the object boundary is fairly complex, but the object contains no “holes”. Our algorithm reconstructed the image perfectly from four projections (horizontal, vertical, diagonal and anti-diagonal).

Figure 3.4b shows a much more complicated example. The object contains many cavities of various sizes and has a very complex boundary. Some of the black holes inside the white region are only a single pixel in size. In this case, our algorithm requires six projections to compute an accurate reconstruction. Some of the fine details in this image are not smooth at all. Still, the fine details are reconstructed with great accuracy. The image 3.4c is even more complex. It requires eight projections to be reconstructed perfectly.

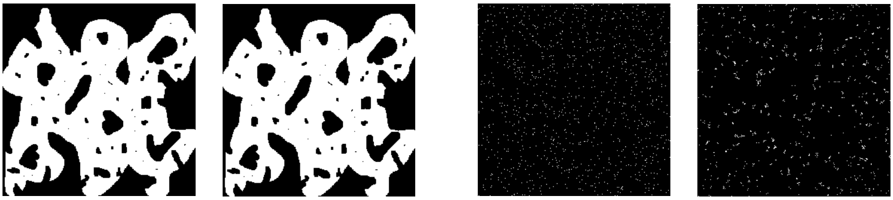
When the image contains no structure at all, our algorithm performs very badly. Figure 3.4d shows an image of random noise. The reconstruction from twelve projections shows that our algorithm has a preference for connected areas of white and black pixels. In this case, however, the smoothness assumption is obviously not satisfied by the original image. The distance between the projections of the image found by our algorithm and the prescribed projections is very small, however.

The image in Figure 3.4e has more structure, but it contains no large white areas. Still, the image is smooth enough to be reconstructed perfectly from only five projections. This example also illustrates that very fine, non-smooth details can be reconstructed by our algorithm, as long as the entire image is sufficiently smooth.

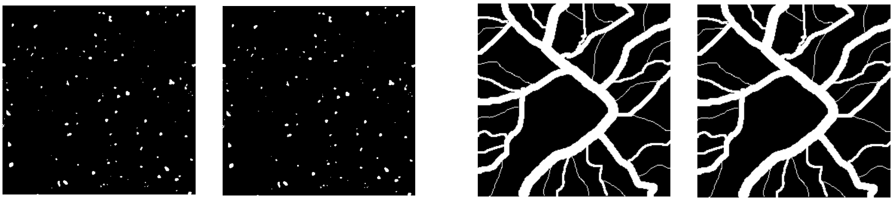
Figure 3.4f shows a vascular system, containing several very thin vessels. Our algorithm can reconstruct the original image perfectly from twelve projections. This is quite surprising, since the very thin vessels have a width of only one pixel, so they are not very smooth. Still, the smoothness of the thicker vessels provides the algorithm with enough guidance to reconstruct the original image.



a. Original image Reconstruction, $k = 4$ b. Original image Reconstruction, $k = 6$



c. Original image Reconstruction, $k = 8$ d. Original image Reconstruction, $k = 12$



e. Original image Reconstruction, $k = 5$ f. Original image Reconstruction, $k = 12$

Figure 3.4: *Six original images and their reconstructions.*

Although the examples in Figure 3.4 give an impression of the reconstruction capabilities of our algorithm, we cannot draw general conclusions from them. In the following two sections we propose three classes of images from which a random sample can be taken. We will report statistical data on the performance of our algorithm on a large number of test instances.

Our experiments with the algorithm show clearly that for each class of images, there is a certain minimum number of projections, k_{min} which is required to reconstruct all images correctly. When the number of prescribed projections is smaller than k_{min}, many reconstructions fail. When more than k_{min} projections are known, the algorithm usually converges faster to an accurate solution. Figure 3.5 shows what happens when the number of prescribed projections is too small. The algorithm still converges, but the resulting image is very different from the original image. Even though the projections of the reconstructed image are still different from the prescribed projections, the reconstructed image is a fixpoint of the iterative procedure. After adding an extra projection, the algorithm can reconstruct the original image perfectly, within a small number of iterations.

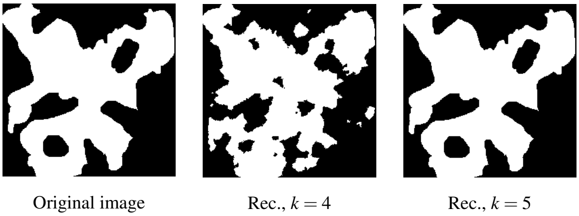

Original image Rec., $k = 4$ Rec., $k = 5$

Figure 3.5: *Using too few projections results in very bad reconstructions.*

3.4.3. Random polygons

For our first class of test images, we implemented a program which generates images that consist of a number of white polygons on a black background. The program takes as input the size of the image, the number n of polygons and the number p of points that are used to generate each polygon. In order to generate a polygon, a random set of p pixels is generated, using a uniform distribution. The convex hull of these pixels is used as the new polygon. The final image is a superposition of n polygons that have been generated in this way. Figures 3.6 shows several sample images that were generated by the program, using two different pairs of parameters (n, p).

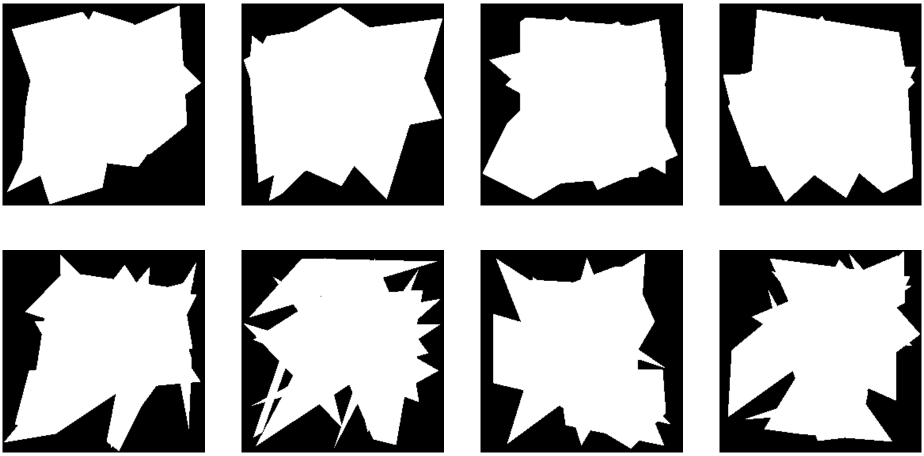


Figure 3.6: *Test images generated by the polygon program. Top row:* $n = 5, p = 8$. *Bottom row:* $n = 12, p = 4$.

For both these pairs (n, p) we performed 200 test runs, using a varying number of projections. All test images are of size 256×256. For more than three projections, the algorithm typically either converges to an image which (almost) perfectly satisfies the projection constraints, or it converges to an image which does not even come close to satisfying those constraints. In the former case, we always observed that there were very few differences between the reconstructed image and the original image. Only when using three projections, we observed that for many reconstructed images the difference between their projections and the prescribed projections was small, yet they were very different from the original images. For each set of test parameters, we report not only the average results over all test cases, but also the average results over the set of test cases for which the reconstruction was successful. By a *successful reconstruction* we mean that the distance between the projections of the image found by our algorithm and the prescribed projections is less than $20k$ (where k is the number of projections). In other words, the projections of the reconstructed image approximate the prescribed projections very well. The reason for letting the definition of a successful reconstruction depend on the number k of projections is that pixel errors in the resulting image (compared to the original image) typically result in errors in each of the projections. A single pixel error in the resulting image results in a larger distance from the prescribed projections when the number of projections is larger.

Table 3.3: *Experimental results for the "random polygons" test cases*

n	p	k	#success	#perfect	proj.error	pixel error	#iter.	time(s)
5	8	3	200	59	32	538	109	14
		4	200	200	0.0	0.0	66	11
		5	200	200	0.0	0.0	61	10
12	4	4	190	190	12	62	123	18
		4*			0.0	0.0	106	16
		5	200	200	0.0	0.0	108	18
		6	200	200	0.0	0.0	63	14

Table 3.3 shows the test results. The first three columns contain the test parameters n, p and k. The next two columns contain the number of successful and perfect reconstructions among the 200 test runs. We consider a reconstruction to be *perfect* if it is identical to the original image from which the projection data was computed. The next four columns contain the average projection error, average number of pixel differences with the original image, average number of iterations (network flow steps) and average running time. For those sets of test parameters for which only a subset of the images was reconstructed successfully, we also show the average results for this subset. We use an asterisk (*) to indicate that we consider only successful reconstructions.

When using our definition of a successful reconstruction, all images for the case $(n,p,k) = (5,8,3)$ are reconstructed "successfully". The results show clearly however, that although the average projection error is very small, the average pixel error is quite large. In this case, having a small projection distance is apparently not sufficient for reconstructing an image which is very similar to the original image. For more than three projections the situation is different. The results for the successful reconstructions (indicated by an asterisk) show that in these cases having a small projection distance implies that the reconstruction is very similar to the original image.

3.4.4. Random ellipses

For the second class of test images we implemented a program which generates a superposition of white ellipses on a black background. The parameters of the program are the size of the image, the number n of ellipses and the minimal and maximal radius (r_{min} and r_{max} respectively) of the ellipses (for both axes), measured in pixels. For each ellipse, a random point $(x_c, y_c) \in A$ is chosen as the center. Both radii r_x and r_y are selected randomly as integers from the interval $[r_{min}, r_{max}]$, using a uniform distribution. A random angle $\phi \in [0, \pi]$ is selected, over which the ellipse is rotated. All pixels (x, y) inside the ellipse are coloured white. Figure 3.7 shows five examples of images that were generated by this program, each using different parameter settings.

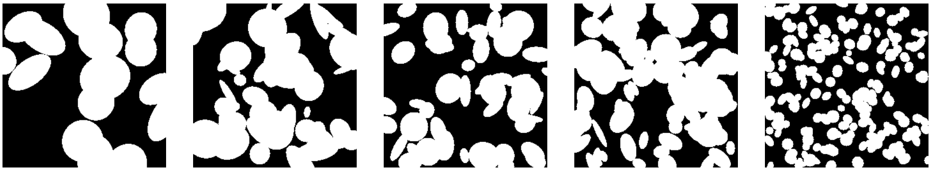

a. $(15,20,40)$ b. $(50,5,35)$ c. $(50,5,25)$ d. $(100,5,25)$ e. $(200,5,10)$

Figure 3.7: *Five images that were generated by the ellipse program, each using different parameters* (n, r_{min}, r_{max}).

By adjusting the program parameters we can sample many classes of images, each having their own characteristics. For each of the parameter settings shown in Figure 3.7 we performed 200 test runs. All test images are of size 256×256. The results are shown in Table 3.4.

The results show that for each image class there is a clear lower bound on the number of projections that is required by the algorithm to compute an accurate reconstruction. If enough projections are available, even the images from the most complex class $(200, 5, 10)$ are reconstructed with very high accuracy.

3.4.5. Random ellipses with noise

So far, the images that we described were very smooth. Surprisingly, our algorithm can also reconstruct very fine non-smooth details. We generated a relatively smooth image with fine non-smooth details by using the ellips program of the previous section — with parameters $(n, r_{min}, r_{max}) = (50, 5, 35)$ — and inverting a set of s random pixels afterwards, where s is a positive integer. We remark that we still use exact projection data: the original image is “polluted” by noise, not the projection data. Figure 3.8 shows three polluted images — for $s = 50, 100, 200$ — and their reconstructions from 9, 11 and 12 projections respectively. Although in none of the cases the reconstruction is perfect, most of the very fine details are accurately reconstructed.

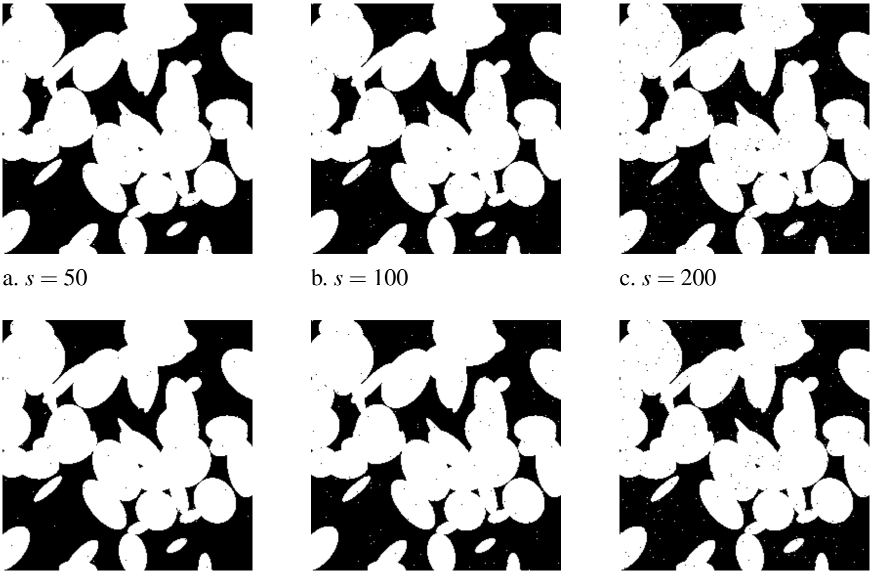
Table 3.4: *Experimental results for the "ellipse" test cases*

n	r_{min}	r_{max}	k	#suc.	#perf.	proj.err.	pixel err.	#iter.	time(s)
15	20	40	4	77	77	170	3257	286	36
			4*			0.0	0.0	180	24
			5	200	200	0.0	0.0	109	19
			6	200	200	0.0	0.0	64	13
50	5	35	5	9	9	737	7597	405	57
			5*			0.0	0.0	251	37
			6	121	113	577	2779	286	43
			6*			1.8	292	224	34
			7	200	162	5.9	1.2	103	22
			8	200	159	7.5	1.3	85	20
50	5	25	6	40	35	1202	6510	387	54
			6*			2.6	617	329	46
			7	147	109	521	1289	225	37
			7*			7.1	1.4	154	27
			8	200	162	6.3	1.1	97	21
			9	200	130	13.2	1.9	92	21
100	5	25	7	53	11	1589	4455	395	61
			7*			26	5.2	376	56
			8	186	63	270	457	240	40
			8*			35	5.9	215	37
			9	200	73	31	4.5	160	31
200	5	10	12	49	5	6699	6157	625	117
			12*			81	8.2	782	137
			14	200	27	107	9.0	356	68
			16	200	26	131	9.5	229	48

3.4.6. Noisy projection data

So far we have assumed that the prescribed *projections* are perfect, without any noise. The network flow approach from Section 3.3.2 cannot be used directly when the projection data is noisy, as the transportation problem that corresponds to any two-projection subproblem may not have an exact solution in that case. We need to replace the transportation problem by a slightly modified network flow problem, see Figure 3.9. Each arc in the network has an associated capacity u and cost c, which are shown in the figure as u/c.

The middle layer of the network still resembles the graph that corresponds to the transportation problem. The graph has been augmented with a source node S and a sink node T, a set of outgoing arcs from the source node and a set of incoming arcs to the sink node. For each column node C_i, there are two arcs from S to C_i. The first arc has capacity s_i (the prescribed column projection) and cost 0. The second arc, which we call the *overflow arc*, has capacity $m - c_i$ and cost K, where K is a very large integer. The set of arcs from the row nodes to the sink node has a similar structure, where the overflow arc for row j has capacity



a. $s = 50$ b. $s = 100$ c. $s = 200$

d. $s = 50, k = 9$ e. $s = 100, k = 11$ f. $s = 200, k = 12$

Figure 3.8: *Top: Polluted versions of an image generated by the ellips program. Bottom: Reconstructions of the polluted images.*

$n - r_j$. The pixel arcs all have capacity one and arc (C_i, R_j) has cost $c_{ij} = -w_{ij}$, where w_{ij} is the pixel weight of pixel (i, j).

The amount f of flow through the network, which equals the number of ones (white pixels) in the resulting image, is determined by averaging the sum of all linesums over all projections and rounding to the nearest integer. The number of white pixels is the same for all pairs of projections.

In each iteration of the algorithm the network that corresponds to the selected pair of directions is constructed and a flow of f units from S to T is computed that has minimal cost. Exceeding a prescribed linesum results in a high penalty cost of K per flow unit, but is allowed. Clearly, if $f \leq mn$, this min-cost max-flow problem always has a solution and as little flow as possible will go through overflow arcs. When the projection data is exact, the resulting min-cost max-flow solution will be identical to the solution of the original transportation problem and none of the overflow arcs will carry any flow.

To generate noisy projection data, we start with perfect projection data and generate for each linesum v a random sample r from a Gaussian distribution with average $\mu = 1$ and variance σ^2. The original linesum is replaced by rv. This model seems reasonable, since in many practical measurements the magnitude of the noise increases with the amount of material (white pixels) that an X-ray passes. Figure 3.10 demonstrates the performance of our algorithm for various noise levels and number of directions. The quality of the reconstruction degrades gradually as the noise level increases. A more accurate reconstruction can be obtained by increasing the number of projections. Even when $\sigma = 0.05$, the reconstruction

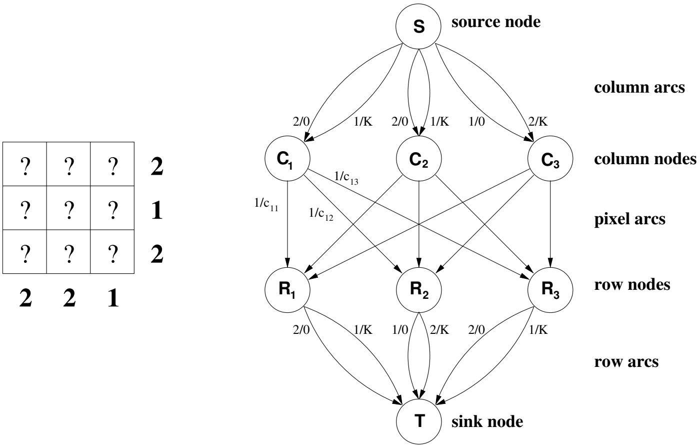
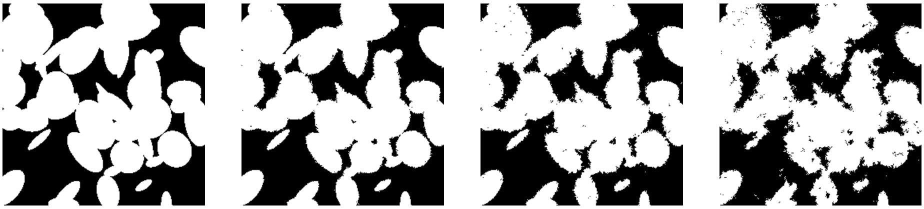
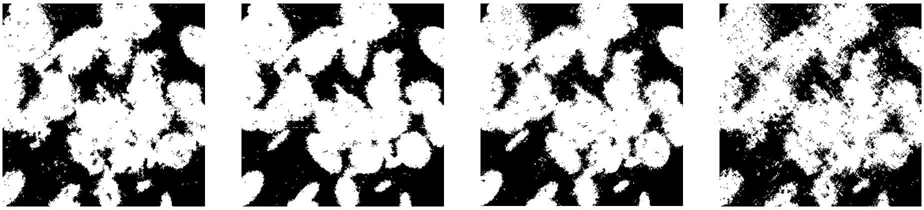


Figure 3.9: *Network for the case of noisy projection data.*

from 12 projections reveals almost all features of the original image.



a. Original image b. Rec., $k = 8, \sigma = 0.01$ c. Rec., $k = 8, \sigma = 0.02$ d. Rec., $k = 8, \sigma = 0.035$

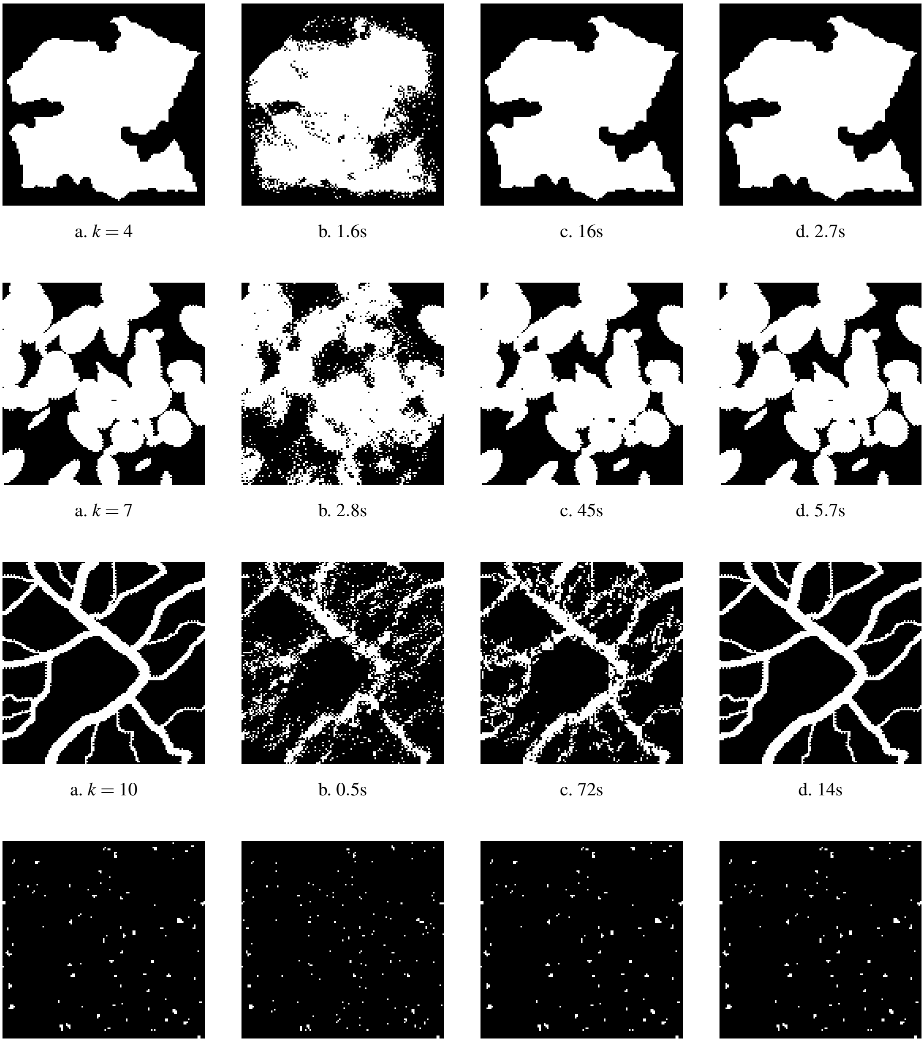


e. Rec., $k = 8, \sigma = 0.05$ f. Rec., $k = 12, \sigma = 0.035$ g. Rec., $k = 12, \sigma = 0.05$ h. Rec., $k = 12, \sigma = 0.1$

Figure 3.10: *Reconstruction results for noisy projection data.*

3.4.7. Comparison with alternative approaches

In this section we compare the reconstruction results of our algorithm to the results of two alternative algorithms for a set of four test images. There are few descriptions in the liter-



a. $k=4$ b. 1.6s c. 16s d. 2.7s

a. $k=7$ b. 2.8s c. 45s d. 5.7s

a. $k=10$ b. 0.5s c. 72s d. 14s

a. $k=5$ b. 0.1s c. 1.5s d. 4.6s

Figure 3.11: *Reconstruction results for four test images. The running times for each of the algorithms is indicated in the subfigure captions. The number of projections used is indicated for each original image. a. Original image; b. result of Gritzmann et al. [10]; c. result of Weber et al. [25]; d. result of our algorithm.*

ature of other reconstruction algorithms that are suitable for large images (i.e., 256×256 pixels or more).

In [5], Fishburn et al. describe a relaxation of the Reconstruction Problem (Problem 9) that can be solved by linear programming. The authors present reconstruction results for three small test images (around 40×40 pixels). The resulting reconstruction is real-valued, all pixel values are between 0 and 1. In [25], Weber et al. describe an extension of the linear program from [5] which incorporates a smoothness prior. We implemented the *FSSV2* approach which is described in their paper using the ILOG CPLEX interior point solver. The real-valued pixel values are rounded to binary values as a postprocessing step, which was also done in [25].

In [10], Gritzmann et al. describe a greedy algorithm for reconstructing binary images from their projections. The paper reports reconstruction results for images up to 512×512 pixels. We implemented the *Greedy-C* algorithm and the *Improvement* postprocessing step. The combination of these two procedures was reported to perform best compared to the other proposed algorithms in the paper. It is important to stress that this approach does not use any smoothness assumptions.

Figure 3.11 shows the reconstruction results for four test images. The test images all have a size of 128×128 pixels. The figure indicates the number of projections that was used for each test image and the running times for each of the algorithms.

The reconstruction results indicate that our algorithm is very fast compared to the linear programming approach from [25]. The greedy approach from [10] is even faster, but it clearly results in inferior reconstructions for all four test cases.

The reconstructions computed by our algorithm are perfect (identical to the original image) for all four test cases. Even for the vessel-like structure, which contains many thin lines, our algorithm computes a perfect reconstruction.

The linear programming approach yields a very good reconstruction for the first image, which is very smooth. Only 9 pixels are different from the original image. For the second image, which is less smooth, the reconstruction still looks reasonable, but several details are not reconstructed accurately. The approach does not work well for the third, vessel-like image. For the fourth image, the linear programming approach works very well. The required computation time is a major limitation for the linear programming approach when working with large images, at least for certain types of images. Note that for the very sparse fourth test image, the linear programming approach is very fast.

3.5. Discussion

The experimental results show that for a broad spectrum of images, our algorithm can compute high quality reconstructions from a small number of projections. This gives rise to optimism on the feasibility of reconstructing images from so few projections. Many important theoretical results have been focused on the reconstructibility of general images, for which no a priori assumptions are given. These results tend to be very poor. We have shown that rather soft smoothness assumptions are already sufficient to change the reconstructibility properties dramatically. We think that our soft smoothness assumptions are satisfied for many practical applications. In practice, most images are neither completely random, nor do they obey strict mathematical constraints, such as convexity. Our algorithm performs very well in this intermediate case, by using the assumption that the image is relatively smooth while not enforcing any rigid structure assumptions.

The grid model that we use, in which lattice lines only contain a discrete set of grid points, is widely used in the literature on discrete tomography (see, e.g., [14]). In particular, it serves as a model for the electron microscopy application, where atoms are arranged in a lattice. For several other applications, however, modelling images as plane sets is more appropriate (see, e.g., [18]). In medical imaging, for example, the measured linesums are actually line integrals of a certain density function. The basic principles of our algorithm can be generalized to cover a much wider range of discrete tomography models. We will report on two generalizations in the next chapters.

For each of the image classes that we used for testing, the results showed a clear lower bound on the number of required projections. When raising the number of projections above this lower bound, the running time of the algorithm decreases, up to certain point where adding additional projections no longer results in reduced reconstruction time.

Fortunately, the best known algorithms for solving minimum cost network flow problems have a low time complexity. Therefore our algorithm can even be used for reconstructing images as large as 1000×1000 pixels, although that may require several hours of computation time.

By adapting our algorithm as described in Section 3.4.6, it can also be used in the case of noisy projection data. Making a good comparison between our algorithm and alternative approaches is hard, because only few good alternatives are available. The results in Section 3.4.7 indicate that for relatively smooth images our approach is much faster than the linear programming approach from [25], which also uses a smoothness assumption. Yet our method yields comparable or better reconstructing quality. Our algorithm is slower than the greedy approach from [10], but the greedy approach yields inferior reconstruction quality.

3.6. Conclusions

We have presented a new algorithm for reconstructing binary images from a few projections. We demonstrated that the algorithm is capable of making very accurate reconstructions for a wide range of test images. The algorithm uses soft smoothness assumptions, yet it is capable of reconstructing very fine details. In its basic form the algorithm assumes perfect projection data, but it can be adapted to work in the case of noisy projection data as well. A comparison between our algorithm and two alternative approaches showed that for relatively smooth images the reconstruction quality of our algorithm is significantly better than for the other two algorithms. Our algorithm is very fast and can be used for reconstruction large images.

Bibliography

[1] Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: *Network flows: theory, algorithms and applications*, Prentice-Hall (1993).

[2] Anstee, R.P.: The network flows approach for matrices with given row and column sums. *Discrete Math.*, **44**, 125–138 (1983).

[3] Barcucci, E., Del Lungo, A., Nivat, M., Pinzani, R.: Reconstructing convex polyominoes from horizontal and vertical projections. *Theoret. Comp. Sci.*, **155**, 321–347 (1996).

[4] Brunetti, S., Del Lungo, A., Del Ristoro, F., Kuba, A., Nivat. M.: Reconstruction of 4- and 8-connected convex discrete sets from row and column projections. *Linear Algebra Appl.*, **339**, 37–57 (2001).

[5] Fishburn, P., Schwander, P., Shepp, L., Vanderbei, R.: The discrete Radon transform and its approximate inversion via linear programming. *Discrete Appl. Math.* **75**, 39–61 (1997).

[6] Gabow, H.N., Tarjan, R.E.: Faster scaling algorithms for network problems. *SIAM J. Comput.*, **18**, 1013–1036 (1989).

[7] Gale, D.: A theorem on flows in networks. *Pacific J. Math.*, **7**, 1073–1082 (1957).

[8] Gardner, R.J., Gritzmann, P., Prangenberg, D.: On the computational complexity of reconstructing lattice sets from their X-rays. *Discrete Math.*, **202**, 45–71 (1999).

[9] Goldberg, A.V.: An efficient implementation of a scaling minimum-cost flow algorithm. *J. Algorithms*, **22**, 1–29 (1997).

[10] Gritzmann, P., de Vries, S., Wiegelmann, M.: Approximating binary images from discrete X-rays. *SIAM J. Optim*, **11**, 522–546 (2000).

[11] Gritzmann, P., Prangenberg, D., de Vries,S., Wiegelmann, M.: Success and failure of certain reconstruction and uniqueness algorithms in discrete tomography, *Int. J. Image. Syst. Tech.*, **9**, 101–109 (1998).

[12] Hajdu, L., Tijdeman, R.: Algebraic aspects of discrete tomography. *J. Reine Angew. Math.*, **534**, 119–128 (2001).

[13] Hajdu, L., Tijdeman, R.: An algorithm for discrete tomography. *Linear Algebra Appl.*, **339**, 147–169 (2001).

[14] Herman, G.T., Kuba, A., eds.: *Discrete tomography: foundations, algorithms and applications*. Birkhäuser, Boston (1999).

[15] Jinschek, J.R., Batenburg, K.J., Calderon, H.A., Van Dyck, D., Chen, F.-R., Kisielowski, C.: Prospects for bright field and dark field electron tomography on a discrete grid. *Microsc. Microanal.*, **10, suppl. 3** (2004).

[16] Jinschek, J.R., Calderon, H.A., Batenburg, K.J., Radmilovic, V., Kisielowski, C.: Discrete Tomography of Ga and InGa Particles from HREM Image Simulation and Exit Wave Reconstruction. *MRS Proceedings*, **839**, 4.5.1–4.5.6 (2004).

[17] Kisielowski, C., Schwander, P., Baumann, F., Seibt, M. Kim, Y., Ourmazd, A.: An approach to quantitative high-resolution transmission electron microscopy of crystalline materials. *Ultramicroscopy* **58**, 131–155 (1995).

[18] Kuba, A.: Reconstruction of measurable plane sets from their two projections taken in arbitrary directions. *Inverse Problems*, **4**, 513–527 (1988).

[19] Combinatorial properties of matrices of zeros and ones. *Can J. Math.*, **9**, 371–377 (1957).

[20] Schrijver, A.: *Combinatorial optimization: polyhedra and efficiency*. Algorithms and Combinatorics series, **24**, Springer, Heidelberg (2003).

[21] Schwander, P., Kisielowski, C., Baumann, F., Kim, Y., Ourmazd, A.: Mapping projected potential, interfacial roughness, and composition in general crystalline solids by quantitative transmission electron microscopy. *Phys. Rev. Lett.*, **71**, 4150–4153 (1993).

[22] Slump, C.H., Gerbrands, J.J.: A network flow approach to reconstruction of the left ventricle from two projections. *Comput. Gr. Im. Proc.*, **18**, 18–36 (1982).

[23] Tanabe, K.: Projection method for solving a singular system. *Numer. Math.*, **17**, 203–214 (1971).

[24] Wang, B., Zhang, F.: On the precise number of $(0,1)$-matrices in $\mathcal{A}(R,S)$. *Discrete Math.*, **187**, 211–220 (1998).

[25] Weber, S., Schnörr, C., Hornegger, J.: A linear programming relaxation for binary tomography with smoothness priors. *Electron. Notes Discrete Math.*, **12** (2003).

Chapter 4

A network flow algorithm for 3D binary tomography of lattice images

This chapter has been published as: K.J. Batenburg, "A new algorithm for 3D binary tomography", Proceedings of the Workshop on Discrete Tomography and its Applications, New York, Electronic Notes in Discrete Mathematics, Vol. 20, 247–261 (2005).

Abstract. In this chapter we propose a new algorithm for reconstructing 3D binary images on a lattice from a small number of their projections. The algorithm is iterative; a new 3D image is computed in each iteration, using network flow methods. It is based on an algorithm for computing 2D reconstructions, which performs very well for a large class of images. We demonstrate the performance of our algorithm on a set of characteristic test images.

4.1. Introduction

Reconstructing 3D binary images from a small number of projections is one of the most important problems in discrete tomography. Research on discrete tomography was stimulated in the 1990s as a result of advances in electron microscopy that made it possible to count the number of atoms in each projected column of a crystal lattice, in several directions [7,9]. Unfortunately, the signal-to-noise ratio turned out to be prohibitively large for this technique. Recently, new techniques have been developed that provide a much better signal-to-noise ratio. The measured atom counts can be used as input for a 3D tomographic reconstruction procedure, which reconstructs the lattice positions of all individual atoms [5, 6].

The problem of reconstructing binary images on a lattice has been studied extensively. The book [4] provides an excellent overview. In 1999, Gardner et al. showed in [3] that for any set of more than two projections the general reconstruction problem is NP-hard, for both 2D and 3D images. When certain a priori knowledge of the image is available, it may be possible to restrict the reconstruction procedure to a specific class of images. For several classes of 2D images, such as the class of convex images, polynomial time reconstruction algorithms exist. However, such classes are often too restrictive to be used in practical applications.

In Chapter 3, we proposed a new algorithm for reconstructing 2D binary images on a square lattice from few projections, using soft smoothness assumptions. The algorithm performs very well for a wide range of images and the smoothness assumption seems to be practically realistic.

In this chapter we propose a generalization of the algorithm from Chapter 3 to 3D images, defined on a lattice. In principle any algorithm that computes 2D reconstructions can also be used for the 3D case by considering the volume as a series of 2D slices and reconstructing each slice independently. This approach requires that all projection directions lie in a single plane. For the electron microscopy application however, the projection directions must coincide with the orientation of the atom columns (called *zone axes*). Therefore the number of suitable directions is very small and these directions do not lie in a single plane. As a consequence, reconstructing a 3D image as a series of 2D slices is not possible. Our algorithm does not require the projection directions to lie in a single plane. The algorithm is iterative. In every iteration a 3D reconstruction is computed using only two of the prescribed projections and the reconstruction from the previous iteration. The subproblem that is solved within an iteration *can* be solved as a series of slices, yet the volume is split into a different set of slices for each new iteration.

4.2. Preliminaries

We denote the cardinality of a finite set V by $|V|$ and we denote the set $\{n \in \mathbb{Z} : n \geq 0\}$ by $\mathbb{N}_0$. Let w, h, d be positive integers, the volume dimensions. Put

$$A = \{(x,y,z) \in \mathbb{Z}^3 : 0 \leq x < w, 0 \leq y < h, 0 \leq z < d\}$$

and let $\mathcal{F}$ denote the collection of mappings $A \to \{0,1\}$. For the sake of convenience we will sometimes regard the elements of $\mathcal{F}$ as *3D images*. Similarly, we will refer to the elements of A as *voxels*. We sometimes use the shorthand notation F_{xyz} to denote $F(x,y,z)$.

We call a vector $v = (a,b,c) \in \mathbb{Z}^3 \backslash \{(0,0,0)\}$ a *lattice direction* if it satisfies $\gcd(a,b,c) = 1$. For every lattice direction v, we define the *lattice line* $\mathcal{L}_v = \{nv \mid n \in \mathbb{Z}\}$. We denote the set of all lines parallel to $\mathcal{L}_v$ by $\mathcal{P}_v = \{\mathcal{L}_v + t \mid t \in \mathbb{Z}^3\}$ and the subset of lines in $\mathcal{P}_v$ that contain at least one point of A by $\mathcal{S}_v = \{P \in \mathcal{P}_v \mid P \cap A \neq \emptyset\}$. Let $F \in \mathcal{F}$. We call the function $X_v F : \mathcal{S}_v \to \mathbb{N}_0$ defined by

$$X_v F(P) = \sum_{(x,y,z) \in P \cap A} F(x,y,z)$$

the *(discrete) 1-dimensional X-ray parallel to v*. We will also refer to $X_v F$ as the *projection* of F parallel to v. Because the function $X_v F$ has finite domain, we can consider it as a vector of $|\mathcal{S}_v|$ elements.

In the reconstruction problem, we want to construct a function $F \in \mathcal{F}$ which has prescribed 1-dimensional X-rays, parallel to certain lattice lines $v_1, \ldots, v_k$. More formally:

Problem 10 (Reconstruction problem).
Let $k \geq 1$ and $v_1, \ldots, v_k$ be given distinct lattice directions. Let $\phi_i : \mathcal{S}_{v_i} \to \mathbb{N}_0$ $(i = 1, \ldots, k)$ be given functions, the prescribed projections. *Find a function $F \in \mathcal{F}$ such that $X_{v_i}F = \phi_i$ for $i = 1, \ldots, k$.*

Clearly, the reconstruction problem does not always have a solution. In this chapter we assume that the prescribed projections are consistent, meaning that the reconstruction problem has at least one solution.

Let $v = (a, b, c)$ be a lattice direction. If $\max(a, b, c) \geq \max(w, h, d)$ then $|P \cap A| \leq 1$ for all $P \in \mathcal{S}_v$. If the set $\{v_1, \ldots, v_k\}$ contains such a lattice direction, solving the reconstruction problem is trivial. In practice it is usually not possible to choose such a lattice direction, because of physical constraints. In the electron microscopy application for example, atom projections are seen under the microscope as circles having a finite width. The lattice directions must be chosen in such a way that projected atom columns do not overlap, otherwise the number of atoms in each column cannot be counted. We are mainly interested in lattice directions for which $\max(a, b, c)$ is small, e.g., no more than 3.

4.3. The case $k = 2$

It is well known that the reconstruction problem in 2D from only two projections can be solved in polynomial time. In 3D this is also the case. For any two prescribed projections, the 3D reconstruction problem can be solved by considering the volume as a series of 2D slices and reconstructing each of the slices independently. Let v_a, v_b be the distinct lattice directions corresponding to the two prescribed projections. Consider the following linear programming problem (denoting $F(x, y, z)$ by F_{xyz}):

Problem 11

$$\begin{array}{ll}
\textbf{maximize} & \displaystyle\sum_{(x,y,z)\in A} F_{xyz} \qquad (F = (F_{xyz}) \in \mathbb{R}^{w\times h\times d}) \\
\textbf{subject to} & \displaystyle\sum_{(x,y,z)\in P_a} F_{xyz} \leq \phi_a(P_a) \quad \textit{for all } P_a \in \mathcal{S}_{v_a} \\
 & \displaystyle\sum_{(x,y,z)\in P_b} F_{xyz} \leq \phi_b(P_b) \quad \textit{for all } P_b \in \mathcal{S}_{v_b} \\
 & 0 \leq F_{xyz} \leq 1 \quad (x, y, z) \in A.
\end{array}$$

Clearly, any solution $\tilde{F} = (\tilde{F}_{xyz})$ of the reconstruction problem is a solution of Problem 11, since

$$\sum_{(x,y,z)\in A} \tilde{F}_{xyz} = \sum_{P\in\mathcal{S}_{v_a}} \phi_a(P) = \sum_{P\in\mathcal{S}_{v_b}} \phi_b(P).$$

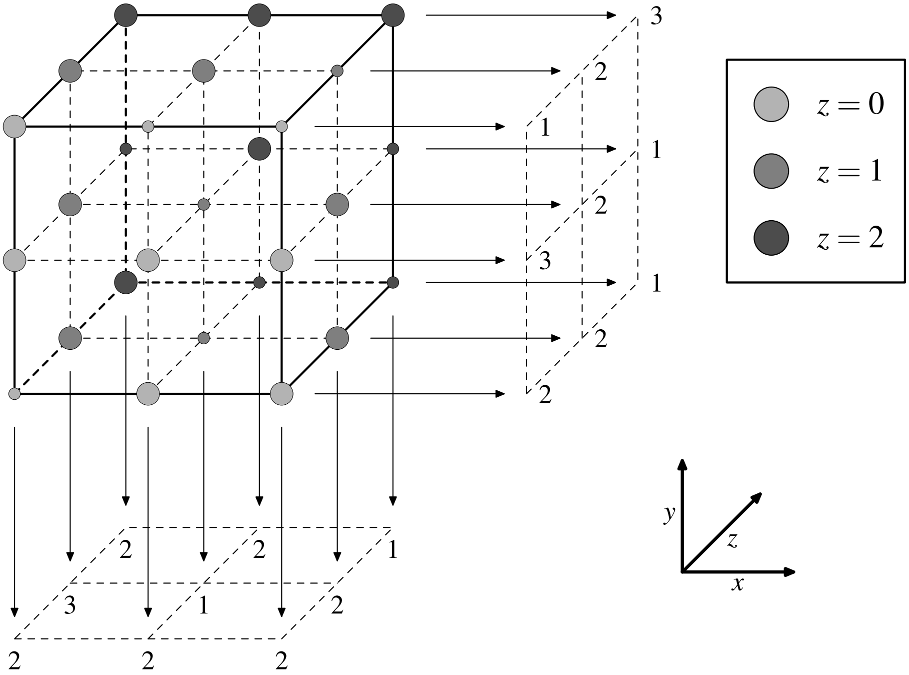


Figure 4.1: *A* $3 \times 3 \times 3$ *binary volume with its projections in directions* $v_a = (1,0,0)$ *and* $v_b = (0,1,0)$. *A large dot indicates a value of* 1, *a small dot indicates a value of* 0.

Figure 4.1 shows an example of a $3 \times 3 \times 3$ binary volume with its projections parallel to the lattice directions $v_a = (1,0,0)$ and $v_b = (0,1,0)$. Problem 11 is equivalent to a *max flow* problem in a directed graph G. Figure 4.2 shows the graph G that corresponds to the volume in Figure 4.1.

The set V of nodes of G consists of a node S (the *source*), one layer of nodes for each $P \in \mathcal{S}_{v_a}$, one layer of nodes for each $P \in \mathcal{S}_{v_b}$ and a node T (the *sink*). Every node in one of the two middle layers is labeled by its respective set P.

Let E be the set of arcs of G. Every arc in E has an associated *capacity*. The set E contains an arc (S, P_a) with capacity $\phi_a(P_a)$ for each $P_a \in \mathcal{S}_{v_a}$. Similarly, E contains an arc (P_b, T) with capacity $\phi_b(P_b)$ for each $P_b \in \mathcal{S}_{v_b}$.

For any $P_a \in \mathcal{S}_{v_a}$, $P_b \in \mathcal{S}_{v_b}$, E contains the arc (P_a, P_b) iff $P_a \cap P_b \cap A \neq \emptyset$. We call such an arc (P_a, P_b) a *voxel arc*. Every voxel arc has capacity 1. For every voxel $(x,y,z) \in A$ there is exactly one arc $(P_a, P_b) \in E$ such that $P_a \cap P_b = \{(x,y,z)\}$. We call (P_a, P_b) the *corresponding voxel arc* of (x,y,z) and (x,y,z) the *corresponding voxel* of (P_a, P_b).

The problem of finding a maximal flow from S to T that satisfies all capacity constraints in G is equivalent to the linear programming problem 11, considering F_{xyz} to be the flow through the corresponding voxel arc of $(x,y,z) \in A$. We refer to the book [1] for details. The

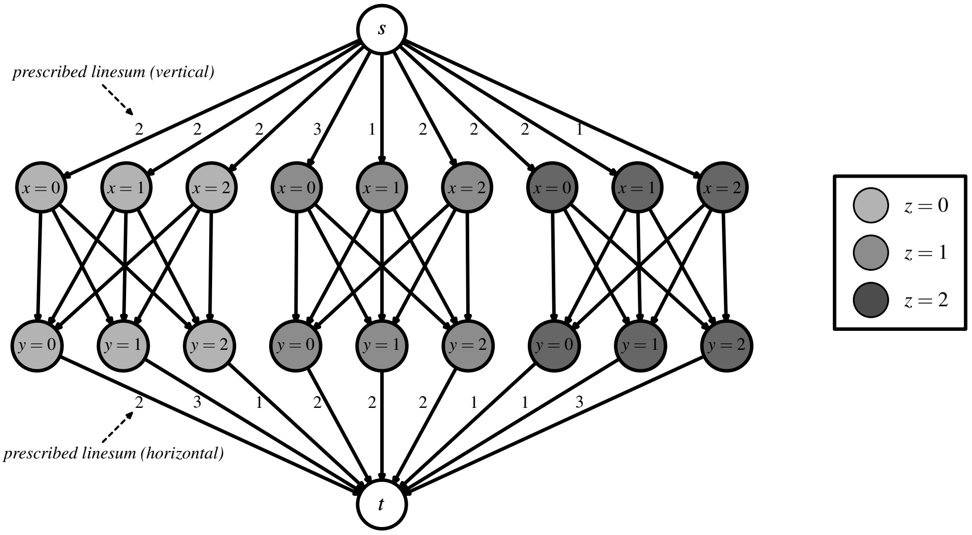


Figure 4.2: *Network corresponding to the two-projection reconstruction problem in Figure 4.1.*

flow through all voxel arcs completely determines the flow through all other arcs of G.

In particular, since all arc capacities are integral, an integral maximal flow can be found in polynomial time. For the voxel arcs, integrality of flow means that every F_{xyz} in the resulting flow is either 0 or 1.

Any solution $\tilde{F} = (\tilde{F}_{xyz})$ of the reconstruction problem corresponds to a valid flow in G, which completely saturates all outgoing arcs from S and all incoming arcs to T. This shows that $\tilde{F}$ corresponds to a *maximal flow* in G. Conversely, any maximal flow $F = (F_{xyz})$ in G must saturate all arcs from S and all arcs to T completely. It then follows directly from the flow conservation constraints in G that if F is an integral flow, it is a solution to the reconstruction problem.

Figure 4.2 also shows a nice property of the two-projection reconstruction problem. If $(P_a, P_b) \in E$ then P_a and P_b have a nonempty intersection in A, so there is a plane in $\mathbb{R}^3$ which contains both P_a and P_b. Since P_a and P_b are translates of $\mathcal{L}_a$ and $\mathcal{L}_b$ respectively, this plane will always be a translate of the plane spanned by v_a and v_b. Because there are no voxel arcs between voxels lying in different translates of this plane, the max flow problem can be solved for each translate of the plane independently. In the example network of Figure 4.2 the subproblems for each of the planes $z = 0$, $z = 1$ and $z = 2$ can be solved independently. This property is independent of the specific pair (v_a, v_b) of lattice directions, although the sizes of the subproblems depend on the direction pair. The number of voxel arcs in each subproblem is bounded by the maximal number of voxels in A that lie in a single plane.

We remark that in general the solution of a two-projection reconstruction problem is not unique. In practical applications it is important that the reconstructed image closely resembles the “original image” from which the projection data have been obtained. Two prescribed projections are often not sufficient to achieve this goal. In the next section we will propose an algorithm that can be used when more than two projections are available.

4.4. An algorithm for $k > 2$

In the previous section we showed that the 3D reconstruction problem from two projections can be solved efficiently. Moreover, the reconstruction problem can be split into a series of independent subproblems in this case, which can be solved in parallel.

Unfortunately, if $k > 2$ the reconstruction problem is NP-hard. If not all lattice directions are contained in a common plane, there is also no straightforward way to split the problem into smaller subproblems.

We propose an iterative algorithm for solving the reconstruction problem for $k > 2$. In every iteration a two-projection reconstruction problem is solved, corresponding to two of the prescribed projections. This two-projection problem typically does not have a unique solution. Using the reconstruction from the previous iteration, we define an ordering on the set of solutions, reflecting the *quality* of the new solution with respect to the previous reconstruction. Let $v_a, v_b \in \{v_1, \dots, v_k\}$ be two disjoint lattice directions. Define a *voxel weight* $w_{xyz} \in \mathbb{R}$ for each voxel $(x,y,z) \in A$. Put $t = \sum_{P \in \mathcal{S}_{v_a}} \phi_a(P) = \sum_{P \in \mathcal{S}_{v_b}} \phi_b(P)$. Consider the following linear programming problem:

Problem 12

$$
\begin{array}{lll}
\textbf{maximize} & \displaystyle\sum_{(x,y,z)\in A} w_{xyz} F_{xyz} & (F = (F_{xyz}) \in \mathbb{R}^{w \times h \times d}) \\
\textbf{subject to} & \displaystyle\sum_{(x,y,z)\in A} F_{xyz} = t & \\
 & \displaystyle\sum_{(x,y,z)\in P_a} F_{xyz} \leq \phi_a(P_a) \quad \textit{for all } P_a \in \mathcal{S}_{v_a} & \\
 & \displaystyle\sum_{(x,y,z)\in P_b} F_{xyz} \leq \phi_b(P_b) \quad \textit{for all } P_b \in \mathcal{S}_{v_b} & \\
 & 0 \leq F_{xyz} \leq 1 \quad (x,y,z) \in A. &
\end{array}
$$

Any integral solution $F = (F_{xyz})$ of Problem 12 has maximal total weight

$$
W(F) = \sum_{(x,y,z)\in A} w_{xyz} F_{xyz}
$$

among all solutions of the two-projection reconstruction problem for lattice directions (v_a, v_b). The linear programming problem 12 is equivalent to a min-cost max-flow problem on the graph G described in Section 4.3. An integral solution can be found in polynomial time.

We can express a preference for F_{xyz} to have a value of 1 or a value of 0, by choosing w_{xyz} to be (relatively) large or small respectively. In particular, for any 3D image $M = (M_{xyz}) \in \mathcal{F}$, choosing

$$
w_{xyz} = M_{xyz} \qquad (x,y,z) \in A
$$

```
Perform preprocessing step: peeling;
Compute the real-valued start solution $F^0$;
$i := 0$;
while (stop criterion is not met) do
begin
    $i := i + 1$;
    Select a new pair of directions $v_a$ and $v_b$;
    For each voxel $(x, y, z)$, compute a voxel weight $w_{xyz}$ that depends
    on the value of voxel $(x, y, z)$ and its neighbors in the previous
    reconstruction $F^{i-1}$.
    Solve a series of min-cost max-flow problems, one for each translate of
    the plane spanned by $v_a$ and $v_b$.
    Merge the solutions of the max-flow problems into a new solution $F^i$.
end
```

Figure 4.3: *Basic steps of our algorithm.*

and solving the linear programming problem 12 yields the solution F of the two-projection problem that has fewest voxel differences with M. We refer to Section 1.3 for a proof of this fact.

Now that we have described how to construct a solution for a two-projection subproblem, we can give a schematic overview of our algorithm for more than two projections, which solves a two-projection problem in each iteration.

Figure 4.3 shows a global description of our algorithm. It is similar to the algorithm from Chapter 3 for reconstructing 2D images. First, as a preprocessing step, lines containing only 0's or only 1's are removed from the volume in an iterative way until there are no such lines left. We call this the *peeling* step. Subsequently, a start solution is computed by solving a real-valued relaxation of the binary reconstruction problem. Next, the algorithm enters a loop.

In every iteration i a pair (v_a, v_b) of lattice directions is selected. A new solution F^i is computed that perfectly satisfies the prescribed projections for the directions (v_a, v_b) and closely resembles the solution F^{i-1} from the previous iteration. Such a solution can be found by first choosing a suitable set of voxel weights and then solving the linear programming problem 12. For computing the voxel weight w_{xyz} not only the value of voxel (x, y, z) in the previous reconstruction is used, but also the value of several surrounding voxels. Soft local smoothness assumptions are used here: a voxel is more likely to retain its value if many of its neighbors share this value. In this way, the assumption that the image is relatively smooth is used throughout the algorithm. In the next subsections we will describe the algorithmic steps in more detail.

4.4.1. Peeling

The peeling step is a very simple preprocessing step. While there is a line $P \in \mathcal{S}_{v_j}$ $(1 \leq j \leq k)$ such that either $X_{v_j}F(P) = 0$ or $X_{v_j}F(P) = \phi_j(P)$, the line is "removed" from the volume. This means that all voxels on the line are immediately assigned their final value (0 or 1 respectively) and their projections are subtracted from the prescribed projections.

Peeling can reduce the size of the residual volume significantly. For the test volume in Figure 4.4 of Section 4.5 for example, peeling is a very effective preprocessing step.

4.4.2. Computing the start solution

For computing the start solution the algorithm uses all k prescribed projections simultaneously. The system

$$\begin{aligned} X_{v_1}F(P_1) &= \phi(P_1) && \text{for all} \quad P_1 \in \mathcal{S}_{v_1} \\ X_{v_2}F(P_2) &= \phi(P_2) && \text{for all} \quad P_2 \in \mathcal{S}_{v_2} \\ & && \dots \\ X_{v_k}F(P_k) &= \phi(P_k) && \text{for all} \quad P_k \in \mathcal{S}_{v_k} \end{aligned} \tag{4.1}$$

is a system of linear equations in the variables F_{xyz}, $(x,y,z) \in A$. We remove the constraint that $F_{xyz} \in \{0,1\}$, allowing any real value F_{xyz}. As a start solution F^0 we use the shortest real-valued solution of (4.1) with respect to the Euclidean norm. We refer to Chapter 3 for a motivation of choosing this particular solution. For the pixel weights in the first two-projection iteration of the algorithm we use

$$w_{xyz} = F^0_{xyz} \qquad (x,y,z) \in A.$$

4.4.3. Selecting the direction pair

In every iteration a new pair of lattice directions is selected from the set $\{v_1,\dots,v_k\}$. It appears to be important that every *pair* of directions is used. For example, when computing a reconstruction from four projections, the results are much better when all pairs are used compared to cycling between the two pairs (v_1,v_2) and (v_3,v_4). For the reconstruction results in Section 4.5 we used a fixed pair ordering in which every pair occurs with the same frequency, see Chapter 3 for details.

4.4.4. Computing the voxel weights

Define the *neighborhood* N_{xyz} of a point $p = (x,y,z) \in A$ as

$$N_{xyz} = \{(x',y',z') \in A : |x-x'| \leq 1, |y-y'| \leq 1, |z-z'| \leq 1\}.$$

Let $F \in \mathcal{F}$ be the reconstruction from the previous iteration of the algorithm. The voxel weight of p for the new reconstruction depends on the value F_{xyz} of p and the values of its neighbors in the previous reconstruction. Define

$$\begin{aligned} s &= |\{(x',y',z') \in N: \ F(x',y',z') = F(x,y,z)\}|, \\ f &= s/|N|. \end{aligned}$$

We now choose the following pixel-weight $w_{x,y,z}$ of p:

$$w_{xyz} = \begin{cases} F(x,y,z) - 1/2 & (f \leq 0.65), \\ 4(F(x,y,z) - 1/2)f & (0.65 < f < 1), \\ 9(F(x,y,z) - 1/2) & (f = 1) \end{cases}$$

This is the same choice of pixel weights as in the 2D version of the algorithm from Chapter 3. It appears to be suitable for computing 3D reconstructions as well. Extensive experiments are necessary here to determine which choice of pixel weights works best. This choice may well depend on characteristics of the type of images that the algorithm is used for.

4.4.5. Stop criterion

For determining when the algorithm should terminate, we use the distance

$$D(F) = \sum_{i=1}^{k} |X_{v_i} F - \phi_i|,$$

where $|\cdot|$ denotes the Euclidean norm. The algorithm terminates if $D(F)$ has not decreased for T iterations, where T is a prescribed positive integer. We used $T = 10$ for the experiments in Section 4.5. The algorithm terminates immediately if the current reconstruction perfectly satisfies the prescribed projections. As a consequence, if we use the algorithm for the case $k = 2$ (which is possible, but not very useful), it always terminates after one iteration, provided that the prescribed projections are consistent.

4.5. Results

We implemented the algorithm for 3D reconstruction from more than two projections in C++. For solving the min-cost max-flow problems we use the RelaxIV solver within the MCF framework [2]. For visualizing the 3D volumes we use the Visualization Toolkit (VTK) [8]. To visualize a volume, the contours of the objects in the volume are first extracted into a polygon model. Subsequently, a surface rendering algorithm is used to render the contours.

Figure 4.4 shows the reconstruction results for an object consisting of a number of cones pointing outward, from two, three and four projections (top to bottom). Each 3D image is displayed from three different angles (left to right). When there are only two prescribed projections our algorithm terminates after the first iteration and the reconstructed image perfectly satisfies both prescribed projections. The results show that two projections are clearly not enough to obtain a reconstruction that resembles the original object. The reconstruction from three projections does resemble the original object, but the object surface in the reconstruction is rather rough. The reconstruction from four projections is completely identical to the original volume. As noted in Subsection 4.4.1, the peeling step is very effective here. A large part of the empty space (consisting of 0's) surrounding the object is peeled away during this step.

Figure 4.5 shows three different test volumes. The first volume is the union of 100 random cones with varying height, radius and orientation. The second volume contains 100 relatively large spheres with varying radius. For such a volume, peeling is far less effective than for the volume in Figure 4.4. The third volume contains 1000 small spheres of varying radius. Our algorithm can reconstruct each of these images from six projections, using lattice directions

$$(1,0,0),(0,1,0),(0,0,1),(1,1,0),(1,0,1) \text{ and } (0,1,1).$$

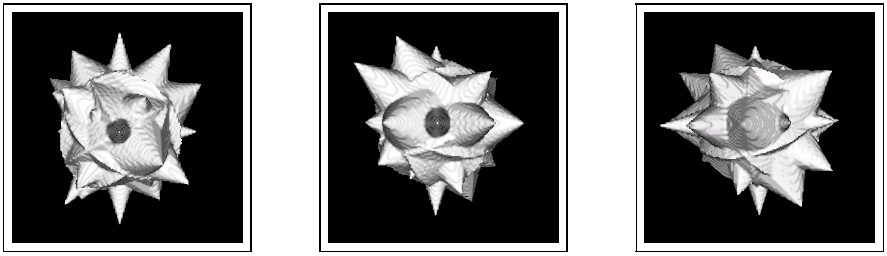
The image sizes and the reconstruction times on a 2.4GHz Pentium IV are shown in the figure.

4.6. Discussion and conclusions

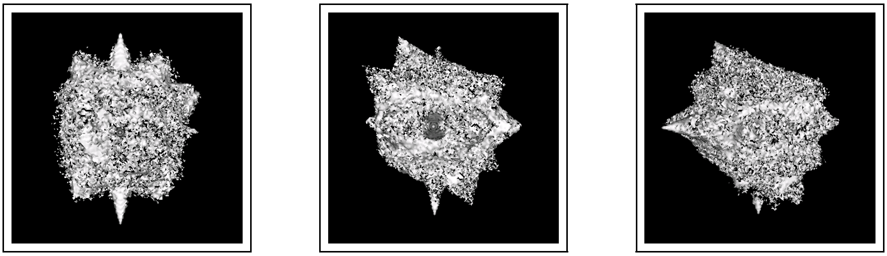
The results suggest that the algorithm is capable of computing high quality reconstructions from a small number of projections. To further assess the performance of our algorithm it is necessary to perform tests on a large number of 3D volumes, sampled from different classes of random volumes. For the 2D version, results of such tests have already been described in Chapter 3. The high reconstruction quality for 2D images gives rise to confidence in the capabilities of the 3D generalization.

All 3D volumes that we used for testing the algorithm are relatively smooth. As local smoothness is assumed when choosing the voxel weights, our algorithm can only be expected to work well for such images. We believe that a large fraction of images that occur in practice are relatively smooth.

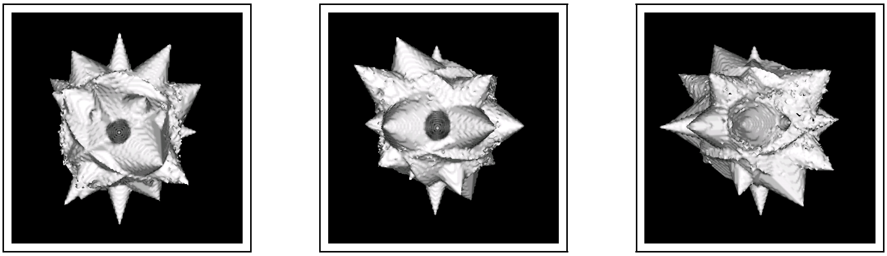
Even without the use of parallel computation the running time of our algorithm is not excessively large. We expect that when a high performance parallel computer is available, the reconstruction time can be reduced to less than half a minute for a $128 \times 128 \times 128$ volume.



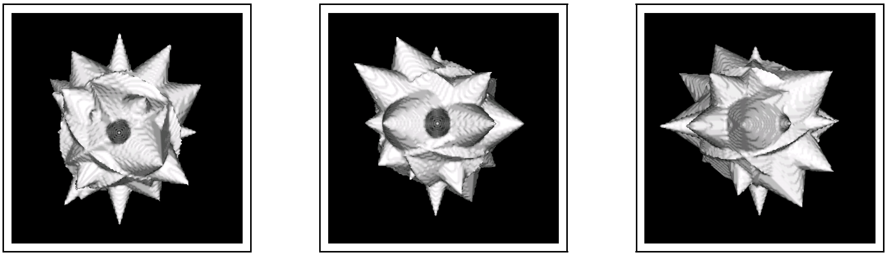
Original volume of size 128 × 128 × 128.



Reconstruction from two projections: $(1,0,0)$ and $(0,1,0)$.

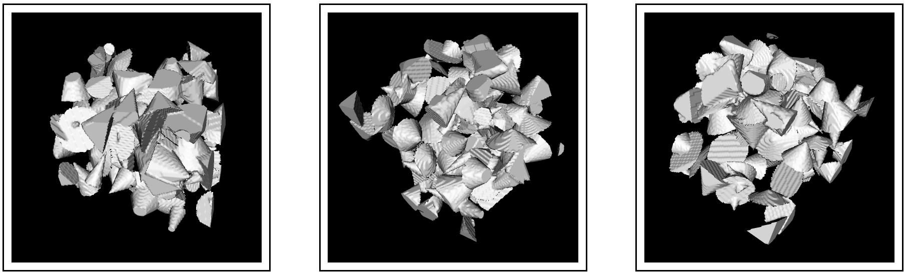


Reconstruction from three projections: $(1,0,0)$, $(0,1,0)$ and $(0,0,1)$.

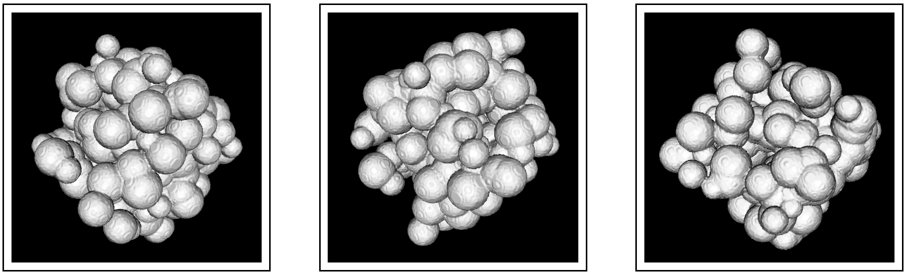


Reconstruction from four projections: $(1,0,0)$, $(0,1,0)$, $(0,0,1)$ and $(1,1,0)$.

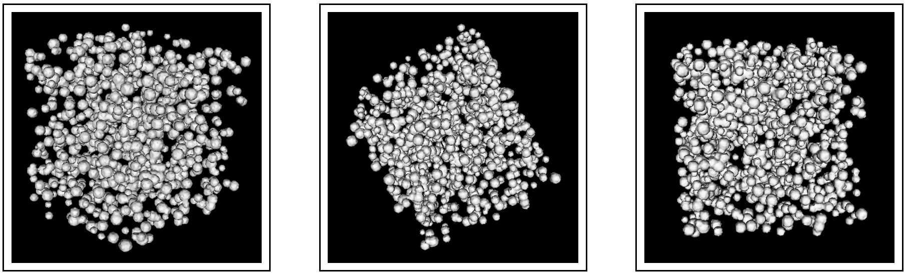
Figure 4.4: *Reconstruction results for a volume consisting of cones pointing outward.*



Random cones, 128 × 128 × 128: perfect reconstruction in 359s.



100 spheres, 169 × 169 × 169: perfect reconstruction in 293s.



1000 small spheres, 139 × 139 × 139: perfect reconstruction in 304s.

Figure 4.5: *Three volumes that were perfectly reconstructed from six projections.*

Bibliography

[1] Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: *Network flows: theory, algorithms and applications*, Prentice-Hall (1993).

[2] Bertsekas, D., Frangioni, A., Gentile, C.: RelaxIV, The MCFClass Project, `www.di.unipi.it/di/groups/optimize/Software/MCF.html` (2004).

[3] Gardner, R.J., Gritzmann, P., Prangenberg, D.: On the computational complexity of reconstructing lattice sets from their X-rays. *Discrete Math.*, **202**, 45–71 (1999).

[4] Herman, G.T., Kuba, A., eds.: *Discrete tomography: foundations, algorithms and applications*. Birkhäuser, Boston (1999).

[5] Jinschek, J.R., Batenburg, K.J., Calderon, H.A., Van Dyck, D., Chen, F.-R., Kisielowski, C.: Prospects for bright field and dark field electron tomography on a discrete grid. *Microsc. Microanal.*, **10, suppl. 3** (2004).

[6] Jinschek, J.R., Calderon, H.A., Batenburg, K.J., Radmilovic, V., Kisielowski, C.: Discrete Tomography of Ga and InGa Particles from HREM Image Simulation and Exit Wave Reconstruction. *MRS Proceedings*, **839**, 4.5.1–4.5.6 (2004).

[7] Kisielowski, C., Schwander, P., Baumann, F., Seibt, M. Kim, Y., Ourmazd, A.: An approach to quantitative high-resolution transmission electron microscopy of crystalline materials. *Ultramicroscopy* **58**, 131–155 (1995).

[8] Schroeder, W., Martin, K., Lorensen, B.: *The visualization toolkit: an object-oriented approach to 3D graphics*, 3rd Edition, Kitware, Inc., (2003).

[9] Schwander, P., Kisielowski, C., Baumann, F., Kim, Y., Ourmazd, A.: Mapping projected potential, interfacial roughness, and composition in general crystalline solids by quantitative transmission electron microscopy. *Phys. Rev. Lett.*, **71**, 4150–4153 (1993).

Chapter 5

An algorithm for the reconstruction of binary images without an intrinsic lattice

A compressed version of this chapter has been accepted for publication in the Proceedings of the 13th International Conference on Discrete Geometry for Computer Imagery (DGCI), Lecture Notes in Computer Science, Springer (2006).

Abstract. Tomography is a powerful technique to obtain accurate images of the inside of an object in a nondestructive way. With conventional reconstruction algorithms, such as filtered backprojection, many projections are required to obtain high quality reconstructions. If the object of interest is known in advance to be *discrete*, i.e., it consists of only a few different materials, the use of this prior knowledge in the reconstruction procedure can dramatically reduce the number of required projections.

In this chapter we propose a new algorithm for the reconstruction of binary images from a small number of their projections. Our algorithms relies on the fact that the problem of reconstructing an image from only two projections can be formulated as a *network flow problem* in a graph. We derive this formulation for parallel beam and fan beam tomography and show how it can be used to compute binary reconstructions from more than two projections.

Our algorithm is capable of computing high quality reconstructions from very few projections. We evaluate its performance on both simulated and real experimental projection data and compare it to other reconstruction algorithms.

5.1. Introduction

Tomography deals with the reconstruction of the density distribution inside an object from a number of its projections [10, 11]. In many applications, such as the reconstruction of medical CT images, a large number of different density values will occur. Typically, the number of projections that is required to obtain sufficiently accurate reconstructions is large in such cases (more than a hundred).

For certain applications, however, it is known in advance that only a few possible density values can occur. Many objects scanned in industry for nondestructive testing or reverse engineering purposes are made of one homogeneous material, resulting in two possible density values: the material and the surrounding air. Another example is medical digital subtraction angiography, where one obtains projections of the distribution of a contrast in the vascular system.

The field of *discrete tomography* deals with the reconstruction of images from a small number of projections, when the set of pixel values is known to have only a few discrete values [7]. By using this prior information about the set of possible values, it may be possible to dramatically reduce the amount of projection data that is required to obtain accurate reconstructions. For some applications, such as the atomic scale reconstruction of crystals from electron tomographic projections, using few projections is a necessity as the scanning process damages the sample [9]. In industrial imaging a reduction of the scanning time may result in cost savings. In medical imaging, reducing the number of projections reduces the amount of radiation used.

Since the 1990s discrete tomography has received considerable interest in the mathematics and computer science communities. Both theoretical and practical aspects have been studied; see [7] for an overview. Most of the theory was developed for reconstructing binary images that are defined on a lattice, i.e., a subset of $\mathbb{Z}^2$. In [6] it was shown that the binary reconstruction problem is NP-hard for more than two projections.

In Chapter 3 we proposed an algorithm for reconstructing binary images that are defined on a lattice, using some smoothness assumptions. This algorithm exploits the fact that the reconstruction problem for only two projections *can* be solved in polynomial time. The proposed reconstruction procedure is iterative: in each iteration a new reconstruction is computed using only two projections *and* the reconstruction from the previous iteration. The new reconstruction simultaneously resembles the image from the previous iteration and adheres to the two selected projections.

In this chapter we describe a new iterative algorithm for reconstructing binary images that do *not* have an intrinsic lattice structure (i.e., subsets of the plane), which is based on ideas similar to those used in Chapter 3. To solve the two-projection subproblems efficiently, a different pixel grid has to be used in each iteration, corresponding to the selected pair of projections. The reconstruction problem can then be solved as a special case of the *minimum cost network flow* problem in graphs, for which efficient polynomial time algorithms are available [1]. Special care has to be taken to handle noisy projection data. We mainly focus on parallel beam tomography, as the network flow approach is particularly well suited for the parallel beam geometry. Our algorithm can also be applied to fan beam data.

Alternative approaches to the binary reconstruction problem that have been considered in the literature are, among others, adaptation of continuous algebraic reconstruction algorithms to the binary reconstruction problem [3], stochastic reconstruction using Gibbs priors (Chapter 8 of [7]), linear programming [4, 15, 16], and D.C. programming [14]. To the best of our knowledge, no reconstruction results have been reported in the literature for images of size larger than 256×256 pixels (the paper [14] reports on some results for images of size 256×256). Our reconstruction results show that discrete tomography can be used effectively on images of this size.

Section 5.2 contains the basic definitions and concepts that are necessary to define our algorithm. In Section 5.3 we consider the problem of reconstructing a binary image from two projections and formulate this problem as a minimum cost flow problem. The algorithm for reconstructing images from more than two projections, which is described in Section 5.4, builds upon the techniques from Section 5.3. In Section 5.5, results of several experiments with simulated projection data are presented, followed by reconstruction results based on real-world data. The results are compared to two alternative algorithms. Section 5.6 concludes.

5.2. Preliminaries

In this section the tomographic scanning geometry is described and notation is introduced to describe the imaging process mathematically.

5.2.1. Data collection geometries

This chapter deals with *transmission tomography*: a beam passes through an unknown object, which attenuates the beam. The intensity of the attenuated beam is measured at the other side of the object by a detector array. The measured intensity of a ray depends on the length of the intersection between the ray and the unknown object, as well as on the materials the object is made of.

We consider two different scanning geometries: parallel beam and fan beam. In parallel beam tomography, a parallel beam passes through the object. The resulting intensities are measured by a parallel array of detectors. The basic setup is shown in Figure 5.1a. The source and detector array rotate around the object, acquiring a number of projections from different angles. Parallel beams are often used in electron tomography, where the projections of a nanosample are acquired by an electron microscope.

Fan beam tomography is often used in medical imaging. Figure 5.1b shows the imaging setup. Rays from a single source pass through the object of interest. The detector array covers a range of angles. The rays from the source cover the entire cross-section of the object. Both the source and the detector array rotate around the object, acquiring projections from several angles. Parallel beam tomography can be considered as a special case of fan beam tomography, where the point source is located at an infinite distance from the object.

In the remainder of this chapter we use the term *X-ray source* to denote the source of any beam (i.e., X-rays, electrons) that is used to measure projections of the unknown object.

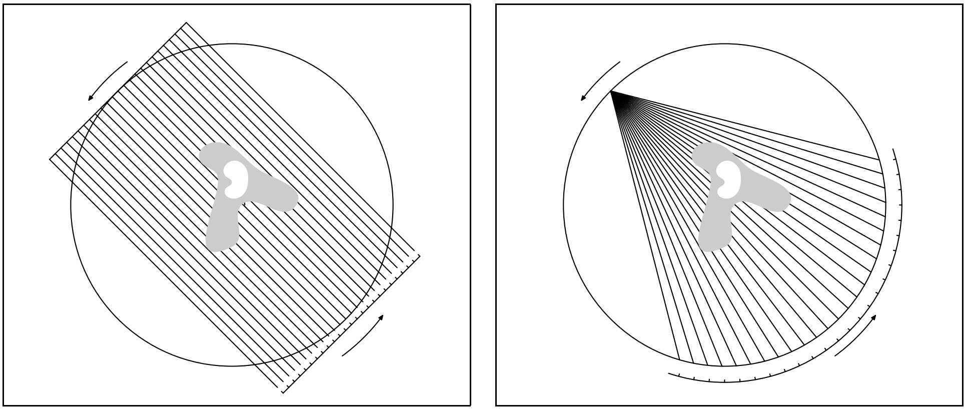


Figure 5.1: *(a) Left: Parallel beam tomography setting (b) Right: Fan beam tomography setting*

5.2.2. Definitions and notation

Let $R \in \mathbb{R}_{>0}$ be the *scanner radius*. Let $D = \{\theta_1, \ldots, \theta_d\}$ be a set of disjoint real numbers in the interval $[0, 2\pi)$, the *projection angles*. This is the set of angles at which the X-ray source is located during the measurements.

Let $T = \{t_0, \ldots, t_n\}$ be a set of real numbers in the interval $(-\frac{\pi R}{2}, \frac{\pi R}{2})$ satisfying $t_0 < t_1 < \ldots < t_n$. We call T the set of *fan parameters*. The fan parameters determine the set $F = \{\gamma_0, \ldots, \gamma_n\}$ of *fan angles* by the relation $\gamma_i = t_i/R$ for $0 \leq i \leq n$.
The scanner radius, the projection angles and the fan parameters jointly define the *data collection geometry* $S = (R, D, T)$. From this point on we assume that the data collection geometry is fixed.
For $\tilde{t} \in (-\frac{\pi R}{2}, \frac{\pi R}{2})$, put $\gamma(\tilde{t}) = \tilde{t}/R$. Let $\theta \in [0, 2\pi)$, $t, t' \in (-\frac{\pi R}{2}, \frac{\pi R}{2})$. Define

$$l_\theta(x, y, t) = x\cos(\theta + \gamma(t)) + y\sin(\theta + \gamma(t)) - R\sin\gamma(t) \qquad x, y \in \mathbb{R}.$$

Define the *fan half line* $L_\theta(t)$ for projection angle θ and fan parameter t as

$$L_\theta(t) = \{(x, y) \in \mathbb{R}^2 : \quad l_\theta(x, y, t) = 0 \quad \text{and} \quad y\cos\theta - x\sin\theta < R\}$$

and the *fan strip* $S_\theta(t, t')$ as

$$S_\theta(t, t') = \left\{ (x, y) \in \mathbb{R}^2 : \quad \begin{array}{ll} l_\theta(x, y, t) & \geq 0 \quad \text{and} \\ l_\theta(x, y, t') & \leq 0 \end{array} \right\}.$$

Figure 5.2b shows the geometric meaning of the last definitions. The half line $L_\theta(0)$ has angle θ with the y-axis, ends at $p = (-R\sin\theta, R\cos\theta)$ and passes through the origin O. Any half line $L_\theta(t)$ also ends in p and has angle $\gamma(t)$ with $L_\theta(0)$. The strip $S_\theta(t, t')$ covers the area in between two such half lines.

The reason for using the *fan parameter* t instead of directly using the *fan angle* γ is that we can now consider the parallel beam geometry as a special case of the fan beam geometry. If we let $R \to \infty$, we obtain (using $\lim_{\gamma \to 0} \frac{\sin\gamma}{\gamma} = 1$):

$$S_\theta(t,t') = \{(x,y) \in \mathbb{R}^2 : \quad t \le x\cos\theta + y\sin\theta \le t'\}$$

This is illustrated in Figure 5.2a. The line $x\cos\theta + y\sin\theta = t$ has an angle θ with the y-axis and has distance $|t|$ to the origin. By specifying the fan parameters instead of the fan angles, we can use the same model for both fan beam and parallel beam tomography.

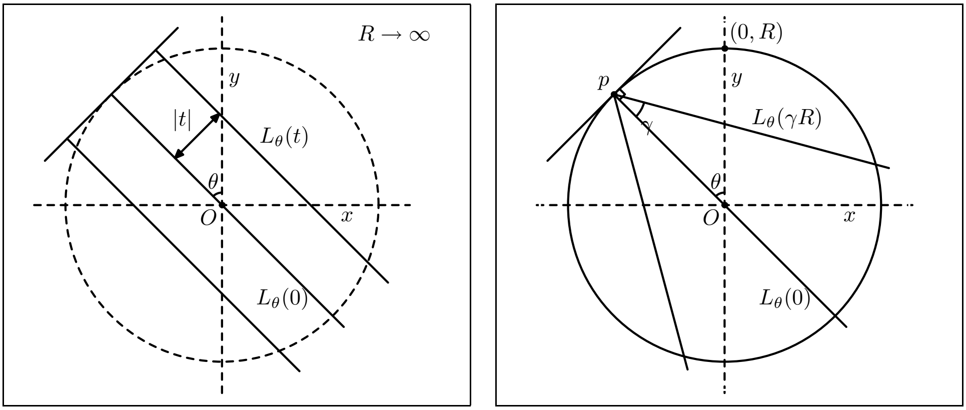


Figure 5.2: *(a) Left: Parallel beam geometry (b) Right: Fan beam geometry*

We call the set $S_{\theta_i}(t_0,t_n)$ the *field of view* for projection angle θ_i. Define the *imaging area* I as

$$I = \bigcap_{k=1}^{d} S_{\theta_k}(t_0,t_n).$$

The imaging area is the set of all points in $\mathbb{R}^2$ that are within the field of view for all projection angles.

5.3. Two projections

We first deal with the problem of reconstructing a gray level image or a binary image (i.e., black-and-white) from only two projections. We consider the unknown image as a mapping $f : I \to B$, where B can be either the closed interval $[0,1]$ (gray value reconstruction) or the set $\{0,1\}$ (binary reconstruction).

Problem 13 *Let* $\bar{\theta}_1, \bar{\theta}_2 \in D$ *be two different projection angles. Let* $p_1 = (p_{11} \dots p_{1n})^T \in \mathbb{R}^n$, $p_2 = (p_{21} \dots p_{2n})^T \in \mathbb{R}^n$ *be two vectors of nonnegative real numbers (the measured strip projections for projection angles* $\bar{\theta}_1$ *and* $\bar{\theta}_2$*, respectively). Construct a function* $f : I \to B$

such that

$$\begin{aligned}\iint_{S_{\bar{\theta}_1}(t_{i-1},t_i)} f(x,y)\,\mathrm{d}y\,\mathrm{d}x &= p_{1i} && \text{for } i=1,\dots,n \quad \text{and}\\ \iint_{S_{\bar{\theta}_2}(t_{i-1},t_i)} f(x,y)\,\mathrm{d}y\,\mathrm{d}x &= p_{2i} && \text{for } i=1,\dots,n.\end{aligned}$$

Depending on whether B is the unit interval or the set $\{0,1\}$, we refer to the *gray value variant* and the *binary variant* of Problem 13 respectively. We call an integral of the form $\iint_{S_{\bar{\theta}_k}(t_{i-1},t_i)} f(x,y)\,\mathrm{d}y\,\mathrm{d}x$ a *strip projection*.

To represent a mapping $f : I \to B$ in a computer we have to resort to an approximate representation. An image f is represented as a 2D array of pixels. Every measured strip projection then gives rise to a linear equation on the pixel values of f. Combining the linear equations for all measured strip projections yields a large system of equations. A real-valued solution of this system can be computed by methods from linear algebra. However, it is likely that such a solution will not be *binary*. Note that the reconstruction problem from two projections is severely underdetermined, so it may have an infinite number of real-valued solutions.

Two projections jointly define a *two-projection grid*. The rows and columns of this grid correspond to the fan strips of the two projections. Define *grid cell* C_{ij} ($1 \le i,j \le n$) as follows:

$$C_{ij} = S_{\bar{\theta}_1}(t_{i-1},t_i) \cap S_{\bar{\theta}_2}(t_{j-1},t_j)$$

Figure 5.3 shows examples of two-projection grids for the parallel and fan beam case. Grid cells that are outside the imaging area are not drawn in the figure.

For any polygon-shaped set $V \subset \mathbb{R}^2$, denote its area by $A(V)$. As a shorthand notation we denote the area of grid cell C_{ij} by a_{ij}. In the parallel beam case all grid cells are parallelogram-shaped and have the same area.

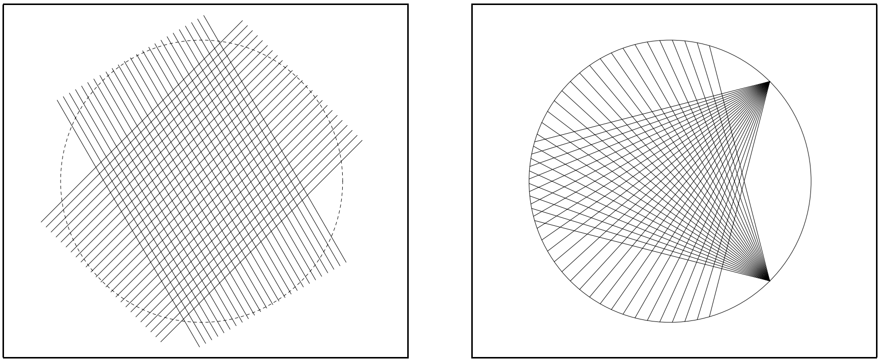


Figure 5.3: *(a) (Left) Two parallel beam projections jointly define a two-projection grid. (b) (Right) Two-projection grid defined by two fan-beam projections.*

For fan beam projections we have to make some additional assumptions on the pair of projections, to ensure that a two-projection grid can be constructed. We assume that

$$\begin{aligned}(-R\sin\bar{\theta}_1, R\cos\bar{\theta}_1) &\notin S_{\bar{\theta}_2}(t_0, t_n)\\ (-R\sin\bar{\theta}_2, R\cos\bar{\theta}_2) &\notin S_{\bar{\theta}_1}(t_0, t_n)\\ L_{\bar{\theta}_1}(t_i)\cap L_{\bar{\theta}_2}(t_j) &\neq \emptyset \qquad \text{for all} \quad 0\le i,j\le n.\end{aligned} \tag{5.1}$$

The first two properties ensure that the source of the first projection is not within the fan range of the second projection and vice versa. The third property states that each fan half line of the first projection intersects each fan half line of the second projection.

A *two-projection image* is a mapping $\{1,\dots,n\}\times\{1,\dots,n\}\to[0,1]$ which assigns a real value — the *gray value* — in the interval $[0,1]$ to each grid cell of a two-projection grid. A *binary two-projection image* is a two-projection image for which all gray values are either 0 (black) or 1 (white). Figure 5.4 shows two examples of a binary two-projection image on the grids from Figure 5.3.

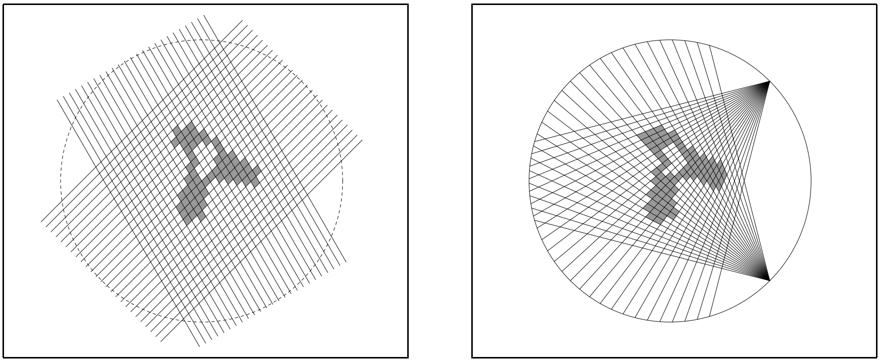


Figure 5.4: *(a) Left: Binary two-projection image on the grid from Figure 5.3a (parallel beam case). (b) Right: Binary two-projection image on the grid from Figure 5.3b (fan beam case).*

It is often convenient to consider a two-projection image X as a *matrix* (x_{ij}), where x_{ij} denotes the gray value of cell C_{ij}. It is straightforward to compute the strip projections of a two-projection image X for the projection angles $\bar{\theta}_1$ and $\bar{\theta}_2$, by summation of all entries in a row or column of X, weighted according to the cell areas. Define $P_1, P_2 : [0,1]^{n\times n}\to\mathbb{R}^n$ by

$$P_1(X) = \begin{pmatrix}\sum_{i=1}^{n} a_{i1}x_{i1}\\ \cdots\\ \sum_{i=1}^{n} a_{in}x_{in}\end{pmatrix} \tag{5.2}$$

and

$$P_2(X) = \begin{pmatrix}\sum_{j=1}^{n} a_{1j}x_{1j}\\ \cdots\\ \sum_{j=1}^{n} a_{nj}x_{nj}\end{pmatrix}. \tag{5.3}$$

We call P_1 and P_2 the *projections* of X for angles $\bar{\theta}_1$ and $\bar{\theta}_2$, respectively. Define the *total projection* $S : [0,1]^{n\times n} \to \mathbb{R}$ by

$$S(X) = \sum_{1\leq i,j\leq n} a_{ij}x_{ij}.$$

The representation of a binary image as a two-projection image is only an approximate representation. Possibly there does not exist a two-projection image for which the projections are exactly equal to the measured projections of Problem 13. In addition, if the measured projections have been obtained from a physical experiment they will usually contain errors. If no perfect solution of the reconstruction problem exists, we would like to find a reconstruction that adheres to the measured projections as well as possible. We now define a reconstruction problem for two-projection images that allows for such errors.

Problem 14 *Let* $\bar{\theta}_1, \bar{\theta}_2 \in D$ *be two different projection angles. Let* $p_1 = (p_{11}\dots p_{1n})^T \in \mathbb{R}^n_{\geq 0}$, $p_2 = (p_{21}\dots p_{2n})^T \in \mathbb{R}^n_{\geq 0}$ *(the measured strip projections for projection angles* $\bar{\theta}_1$ *and* $\bar{\theta}_2$ *respectively). Let* $\bar{T} \in \mathbb{R}$ *(the* prescribed total projection*). Construct a matrix* $X \in B^{n\times n}$ *such that* $S(X) = \bar{T}$ *and*

$$|P_1(X) - p_1|_1 + |P_2(X) - p_2|_1$$

is minimal (where $|\cdot|_1$ *denotes the sum-norm).*

In any instance of Problem 14, $S(X)$ is considered to be a fixed parameter. A good value for the parameter $\bar{T}$ can be computed from the measured projection data. For a binary image, $S(X)$ represents the total area of the white pixels in X. For any image X we have $S(X) = |P_1(X)|_1 = |P_2(X)|_1$, so in order to minimize $|P_1(X) - p_1|_1 + |P_2(X) - p_2|_1$, $S(X)$ has to be close to $|p_1|_1$ and $|p_2|_1$. A sensible value for $\bar{T}$ is given by $(|p_1|_1 + |p_2|_1)/2$. For the parallel beam case, Problem 14 can only have a binary solution if $\bar{T}$ is an integral multiple of the pixel area a. A good value for $\bar{T}$ can be found by rounding $(|p_1|_1 + |p_2|_1)/2$ to the nearest multiple of a. For fan beam projections, such a rounding step cannot be applied, as the cell area is not constant. In the fan beam case we only deal with the gray value variant of Problem 14, as the binary variant cannot be solved efficiently within our proposed model.

We will now show that Problem 14 can be formulated as a *min cost flow problem* in a particular graph. In fact, this network flow approach can be used to solve a generalization of Problem 14. This generalization offers the possibility of incorporating prior knowledge about the unknown image in the reconstruction, by specifying a *weight* w_{ij} for each pixel (i,j):

Problem 15 *Let* $\bar{\theta}_1, \bar{\theta}_2, p_1, p_2, \bar{T}$ *be as in Problem 14. Let* $W = (w_{ij}) \in \mathbb{R}^{n\times n}$ *and* $\alpha \in \mathbb{R}$. *Construct a matrix* $X \in B^{n\times n}$ *such that* $S(X) = \bar{T}$ *and*

$$\alpha(|P_1(X) - p_1|_1 + |P_2(X) - p_2|_1) - \sum_{1\leq i,j\leq n} w_{ij}a_{ij}x_{ij}$$

is minimal.

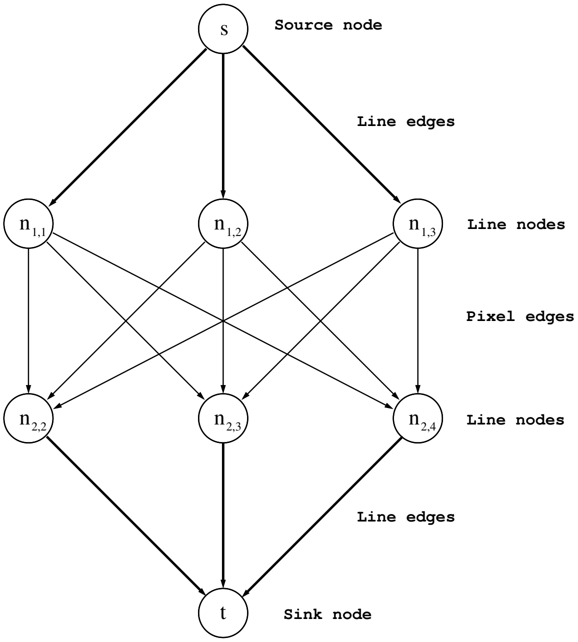


Figure 5.5: *Basic structure of the associated graph*

Problem 15 is a generalization of Problem 14. Setting $\alpha = 1$ and $w_{ij} = 0$ for $1 \leq i, j \leq n$ yields Problem 14. We call the matrix W the *weight map*. The weight map is used extensively in the algorithm for reconstructing a binary images from more than two projections that we describe in the next section.

The basic idea of using network flow methods for the reconstruction of binary images from two projections was first described by Gale in 1957 [5], in the context of reconstructing binary *matrices* from their row and column sums.

With a pair of projection angles $(\bar{\theta}_1, \bar{\theta}_2)$ and their respective measured projections (p_1, p_2), we associate a *directed* graph $G = (V, E)$, where V is the set of nodes and E is the set of edges. We call G the *associated graph*, see Figure 5.5. The set V contains a node s (the *source*), a node t (the *sink*), one node for each strip of projection angle $\bar{\theta}_1$ and one node for each strip of projection angle $\bar{\theta}_2$. The node that corresponds to $S_{\bar{\theta}_k}(t_{i-1}, t_i)$ has label $n_{k,i}$. We call the nodes $n_{k,i}$ *line nodes*. Every pair $(n_{1,i},\ n_{2,j})$ of nodes is connected by a (directed) edge. We call these edges *pixel edges* and denote the set of all pixel edges by $E_p \subset E$. Besides the point edges the set E contains the subsets $E_1 = \{(s, n_{1,i}) : i = 1, \ldots, n\}$ and $E_2 = \{(n_{2,j}, t) : j = 1, \ldots, n\}$ of directed edges. We call the elements of E_1 and E_2 the *line edges of* G. The complete set of edges of G is given by $E = E_p \cup E_1 \cup E_2$. In the remainder of this section we assume that the reader is familiar with the basic concepts of *network flows*. The book [1] provides an excellent introduction to this subject.

To each edge $e \in E$ we assign a real-valued capacity, according to the following *capacity function* $U : E \to \mathbb{R}_{\geq 0}$:

$$\begin{aligned} U((n_{1,i},n_{2,j})) &= a_{ij} && \text{for} \quad 1 \le i,j \le n \\ U((s,n_{1,i})) &= \sum_{j=1}^{n} a_{ij} && \text{for} \quad 1 \le i \le n \\ U((n_{2,j},t)) &= \sum_{i=1}^{n} a_{ij} && \text{for} \quad 1 \le j \le n \end{aligned}$$

A *flow* in G is a mapping $Y : E \longrightarrow \mathbb{R}_{\ge 0}$ such that $Y(e) \le U(e)$ for all $e \in E$ and such that for all $v \in V \backslash \{s,t\}$:

$$\sum_{w:\,(w,v)\in E} Y((w,v)) = \sum_{w:\,(v,w)\in E} Y((v,w)).$$

Define the *size* S(Y) of a flow Y as

$$S(Y) = \sum_{i=1}^{n} Y((s,n_{1,i})) = \sum_{j=1}^{n} Y((n_{2,j},t)) = \sum_{1\le i,j\le n} Y((n_{1,i},n_{2,j})).$$

Note that if a flow Y is specified on the pixel edges, its value for all line edges can be computed using the flow conservation constraint. A flow Y is called an *integral flow* if $Y(e) \in \mathbb{N}_0$ for all $e \in E$.

For any matrix $X \in [0,1]^{n\times n}$ we define its *corresponding flow* Y_X by

$$Y_X((n_{1,i},n_{2,j})) = a_{ij}x_{ij} \qquad \text{for} \quad 1 \le i,j \le n.$$

Note that Y_X is a mapping $E \longrightarrow \mathbb{R}_{\ge 0}$, which we define by specifying its values on a subset of E. The value of Y_X on the line edges of G then follows from the flow conservation constraints:

$$Y_X((s,n_{1,i})) \;=\; [P_1(X)]_i \quad \text{for} \quad 1 \le i \le n \tag{5.4}$$

and

$$Y_X((n_{2,j},t)) \;=\; [P_2(X)]_j \quad \text{for} \quad 1 \le j \le n. \tag{5.5}$$

With each edge $e \in E$ we assign a *cost function*, which determines the cost of sending a certain amount x of flow through that edge. The cost function C_e depends on the weight map W, the constant α and the measured projections p_1 and p_2 and is defined by

$$\begin{aligned} C_{(n_{1,i},n_{2,j})}(x) &= -w_{ij}x && \text{for} \quad i,j = 1,\dots,n \\ C_{(s,n_{1,i})}(x) &= 2\alpha \max(x - p_{1,i}, 0) && \text{for} \quad i = 1,\dots,n \\ C_{(n_{2,j},t)}(x) &= 2\alpha \max(x - p_{2,j}, 0) && \text{for} \quad j = 1,\dots,n \end{aligned}$$

Define the *total cost* $C(Y)$ of a flow Y as

$$C(Y) = \sum_{e\in E} C_e(Y(e)). \tag{5.6}$$

Lemma 2 *There is a 1-1 correspondence between the solutions of the gray value variant of Problem 15 and the flows Y of size $\bar{T}$ in G for which $C(Y)$ is minimal among all flows of size $\bar{T}$.*

Proof. For any real-valued vector $p \in \mathbb{R}^n$, define $|p|^+ = \sum_{i=1}^{n} \max(p_i, 0)$. Combining Equation (5.4), (5.5) and (5.6), we find that

$$C(Y_X) = 2\alpha(|P_1(X) - p_1|^+ + |P_2(X) - p_2|^+) - \sum_{1 \le i,j \le n} w_{ij} a_{ij} x_{ij}.$$

Using the identities

$$S(X) = |P_k(X)|_1 = |p_k|_1 + |P_k(X) - p_k|^+ - |p_k - P_k(X)|^+ \quad \text{for} \quad k = 1, 2,$$

and

$$|P_k(X) - p_k|_1 = |P_k(X) - p_k|^+ + |p_k - P_k(X)|^+,$$

we find that

$$\begin{aligned} C(Y_X) = {} & \alpha(|P_1(X) - p_1|_1 + |P_2(X) - p_2|_1) - \sum_{1 \le i,j \le n} w_{ij} a_{ij} x_{ij} \; + \\ & \alpha(2S(X) - |p_1|_1 - |p_2|_1). \end{aligned} \tag{5.7}$$

The mapping $X \rightarrow Y_X$ yields a 1-1 correspondence between the flows in G of size $\bar{T}$ and the two-projection images X for which $S(X) = \bar{T}$. If we fix $S(X)$, as in Problem 15, the term $\alpha(2S(X) - |p_1|_1 - |p_2|_1)$ in Equation (5.7) is constant. We conclude that the problem of finding a flow Y of size $\bar{T}$ in G for which $C(Y)$ is minimal is equivalent to finding a solution of the gray value variant of Problem 15. □

The minimum cost flow problem in graphs is a well-known problem in combinatorial optimization for which efficient, polynomial time algorithms exist (see [1] for a thorough description or [13] for an overview). However, most algorithms for this problem deal with *linear cost functions* for the edges, i.e., cost functions of the form $C_e(x) = cx$, where c is a constant. The cost function for a line edge can be formulated as a linear cost function by replacing the line edge with *two* parallel edges. For a line edge (s, n_{1i}) in the original graph G, the first new edge has a capacity of p_{1i} and a cost function of 0. The second parallel edge has a capacity of $(\sum_{j=1}^{n} a_{ij}) - p_{1i}$ and a cost function $C(x) = 2\alpha x$, see Figure 5.6a. For any minimum cost flow, the second parallel edge will only carry a nonzero flow if the first edge is completely filled, because of the cost of using the second edge.

For the parallel beam case, a solution of the *binary* version of Problem 15 can be computed efficiently (i.e., in polynomial time) using the min cost flow formulation. It is a well-known fact that if a network has only *integral* edge capacities and edge costs, there is always an integral flow among all flows for which the total cost is minimal (see, e.g., [1]). Such an integral flow can be computed in polynomial time. Note that for the parallel beam case, the pixel area is constant. We first divide all edge capacities in the associated graph by a, the pixel area. The total flow $\bar{T}$ and the coefficients of the linear cost functions for the edges

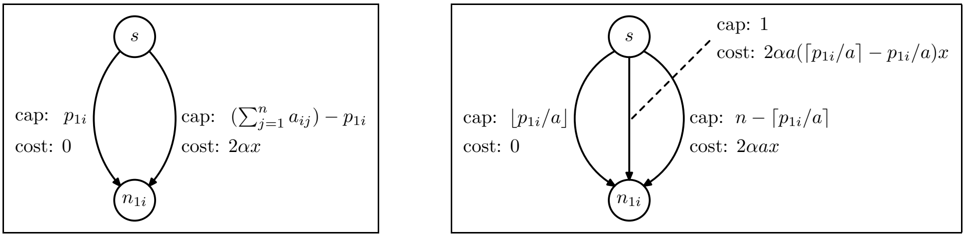

s
cap: p_{1i}
cost: 0
cap: $(\sum_{j=1}^{n} a_{ij}) - p_{1i}$
cost: $2\alpha x$
n_{1i}
s
cap: 1
cost: $2\alpha a(\lceil p_{1i}/a \rceil - p_{1i}/a)x$
cap: $\lfloor p_{1i}/a \rfloor$
cost: 0
cap: $n - \lceil p_{1i}/a \rceil$
cost: $2\alpha a x$
n_{1i}

Figure 5.6: *(a) (Left) To obtain linear cost functions for the edges, each line edge is replaced with two parallel edges that have linear cost functions. (b) (right) In the parallel beam case, the line edges are replaced by three parallel edges that each have integral capacity.*

have to be scaled accordingly. If $\bar{T}/a$ is not an integral value, it is rounded to the nearest integer; otherwise it would clearly be impossible to find an integral flow of size $\bar{T}/a$. We denote the graph that is obtained by this scaling step by G'. All pixel edges in G' have a capacity of 1, but the capacities of the line edges (one pair for each projected strip, as in Figure 5.6a) may still not be integral. If a pair of parallel edges that represents a line edge does not have integral capacities, we add a third parallel edge, as in Figure 5.6b. The figure also indicates the capacity and the cost function for each of the three parallel edges. Note that the derivatives of the cost functions are increasing constants for the three parallel edges, from left to right. Therefore, in any minimum cost flow the second edge will only be used if the first one is completely filled and the third edge will only be used if the second one is full. It is easy to verify that for any *integral* amount x of flow from s to n_{1i} the total cost for the three parallel edges equals $2\alpha \max(x - p_{1i}/a, 0)$ and all three edge capacities are integral. To obtain integral coefficients in all cost functions, note that the solution of the minimum cost flow problem in a graph does not change if we multiply all costs by the same constant (only the *total cost* of the solution changes). For each (linear) cost function $C_e = cx$, we multiply c by a constant K and round the result. As $\text{round}(Kcx)/Kcx \rightarrow 1$ for large values of K, the effect of this rounding step can be made negligible by choosing K to be large.

In the case of fan beam projections it is also possible to use the network flow approach. However, finding a *binary* solution of the reconstruction problem is now much more difficult. Figure 5.3b shows a two-projection grid for the fan beam case. In the parallel beam case, all pixel edges have a capacity of 1, thus any integral flow corresponds to a binary image. As the cell area $A(C_{ij})$ is no longer constant in the fan beam case (see Figure 5.3b), the capacity of the pixel edges will no longer be constant. Although there is no *guarantee* that the minimum cost flow corresponds to a binary solution in this case, min cost flow algorithms can still be used to compute a gray value image. As we will show in Section 5.5, this gray value image can still be very useful for computing a binary reconstruction, as most pixel values will typically be very close to 0 or 1.

If we set α to a very large value in Problem 15, the solution(s) of the reconstruction Problem will correspond to the measured projections as well as possible, according to the sum-norm. For the experiments in Section 5.5 we used $\alpha = 10000$. In those experiments the pixel weights w_{ij} are always between -2 and 2.

5.4. More than two projections

In the previous section we showed that the two-projection reconstruction problem can be formulated as a network flow problem. For the case of parallel beam projections, a binary solution can be found efficiently. For the fan beam case the network flow approach can also be used, but it does not guarantee a binary solution. Unfortunately, there is no straightforward generalization of the network flow approach to the case of more than two projections. In this section we study the following reconstruction problem:

Problem 16 *Let* $p_1 = (p_{11} \dots p_{1n})^T, \dots, p_d = (p_{d1} \dots p_{dn})^T \in \mathbb{R}^n$ *be vectors of nonnegative real numbers (the measured strip projections for projection angles* $\theta_1, \dots, \theta_d \in D$ *respectively). Construct a function* $f : I \to \{0,1\}$ *such that*

$$\iint_{S_{\bar{\theta}_k}(t_{i-1},t_i)} f(x,y)\,\mathrm{d}A \quad = \quad p_{ki} \qquad \text{for } i = 1,\dots,n,\ k = 1,\dots,d.$$

We propose an iterative algorithm, which makes use of the fact that the two-projection problem can be solved efficiently. The algorithm computes a reconstruction from more than two projections by solving a series of two-projection subproblems, each using two projection angles from the set D. The algorithm uses the concept of a weight map, as defined in Section 5.3. In each iteration a new pair of projection angles is selected. An instance of Problem 15 is then solved on the two-projection grid that corresponds to those two angles. The weight map is computed using the reconstruction from the previous iteration, in such a way that solutions are preferred which resemble the reconstruction from the previous iteration. The previous reconstruction was computed using a different pair of projections, which are thus incorporated into the new reconstruction. Repeating this argument, projections from earlier iterations are also incorporated.

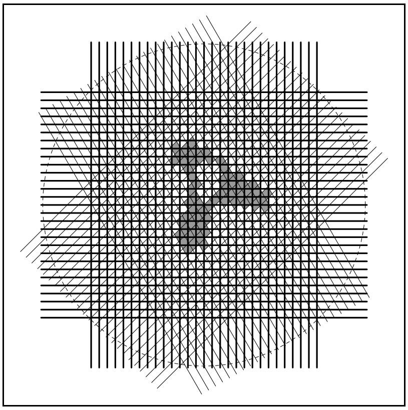


Figure 5.7: *To display a reconstruction on a two-projection grid, it has to be converted to the standard pixel grid.*

To display a reconstruction that has been computed on a two-projection grid, it has to be converted to an image on the standard square pixel grid, aligned to the horizontal and vertical axes. Figure 5.7 shows the standard pixel grid, superimposed on a two-projection image. The value of each pixel in the standard pixel grid is computed by averaging the gray values of the overlapping pixels in the two-projection image, weighted by the overlap area. We will describe the details of this computation in Section 5.4.2.

Figure 5.8 shows the basic steps of the algorithm. First, a start solution is computed, using all projections simultaneously. The start solution should provide a good first approximation of the unknown image, while being easy to compute. The start solution can be computed on the standard $n \times n$ square pixel grid. For the experiments in Section 5.5 we used the SIRT (Simultaneous Iterative Reconstruction Technique, see Chapter 7 of [10]) to compute the start solution, which yields a gray value reconstruction.

Subsequently, the "total area of the white pixels in the unknown object" $\bar{T}$ is computed as $\bar{T} := (\sum_{k=1}^{d} |p_k|)/dn$. The value of $\bar{T}$ is used during the main loop of the algorithm, to determine the total flow in each of the network flow problems, corresponding to two-projection subproblems. Note that in the case of parallel beam tomography, $\bar{T}$ has to be rounded to the nearest integral multiple of the cell area, to be able to find *binary* solutions for the two-projection problems.

Next, the algorithm enters the main loop. In each iteration τ of the main loop a new pair $(\bar{\theta}_1^{\tau}, \bar{\theta}_2^{\tau})$ of projection angles is first selected, which determines the two-projection grid for iteration τ. We refer to the cell (i, j) of this grid as C_{ij}^{τ}. Section 5.4.1 describes several possible angle selection schemes.

Subsequently the weight map $W^{\tau} = (w_{ij}^{\tau})$ is computed from the previous reconstruction $X^{\tau-1}$. We describe the details of this computation in Subsection 5.4.3. The weight map W, the total flow $\bar{T}$ and the projections for angles $(\bar{\theta}_1^{\tau}, \bar{\theta}_2^{\tau})$ define an instance of Problem 15. Solving this problem by the network flow approach yields the new reconstruction X^{τ}.

The criterion for termination of the main loop is described in Section 5.4.4, which also describes how the final reconstruction X^* is computed.

5.4.1. Choosing the pair of directions

Not every pair of projection angles is suitable to be used for solving a two-projection problem. At least such a pair should satisfy the conditions in Equation 5.1, but it is also important that both projection angles are sufficiently orthogonal. Figure 5.9 shows the grid cell shape for two projection angles that are almost orthogonal, versus the shape for two projection angles that have a small angular difference, resulting in a flat, stretched pixel shape. Such pixels are not well suited for accurate image representation, as the pixel diameter (i.e., the farthest distance between any two points in the pixel) is too large. Therefore the angular difference between $\bar{\theta}_1^{\tau}$ and $\bar{\theta}_2^{\tau}$ should always be larger than a certain minimum value, $\delta_{\min}$. We call a pair of projection angles *valid* if it satisfies all requirements to be used for solving a two-projection problem. We used $\delta_{\min} = \pi/4$ for all experiments (including the fan-beam experiments) in Section 5.5.

One can think of several criteria for choosing the projection pair at the start of a new iteration. Let X be a two-projection image for projection angles $\bar{\theta}_1, \bar{\theta}_2$. Define the *projection*

```
Compute the start solution $X^0$ on the standard pixel grid;
$\bar{T} := (\sum_{k=1}^{d} |p_k|)/dn$;
$\tau := 0$;
while (stop criterion is not met) do
begin
    $\tau := \tau + 1$;
    Select a new pair of projection angles $\bar{\theta}_1^\tau, \bar{\theta}_2^\tau \in D$;
    Compute the weight map $W^\tau = (w_{ij}^\tau)$ for the two-projection grid
    corresponding to $(\bar{\theta}_1^\tau, \bar{\theta}_2^\tau)$, using the previous reconstruction $X^{\tau-1}$;
    Compute a solution $X^\tau$ with $S(X^\tau) = \bar{T}$ of Problem 15
    on the two-projection grid for angles $(\bar{\theta}_1^\tau, \bar{\theta}_2^\tau)$, using the
    weight map $W^\tau$;
end
Return the final reconstruction $X^*$ (see Section 5.4.4);
```

Figure 5.8: *Basic steps of the algorithm for plane sets.*

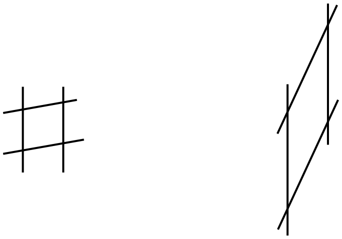


Figure 5.9: *Pixel shape for two different two-projection grids in the parallel beam case. (a) Left: Two projection angles that are almost orthogonal. (b) Right: Small angular difference between both projection angles.*

$P_k(X)$ for projection angle $\theta_k \in D$ as follows:

$$P_k(X) = \begin{pmatrix} \sum_{1 \le i,j \le n} A(C_{ij} \cap S_{\theta_k}(t_0, t_1)) x_{ij} \\ \cdots \\ \sum_{1 \le i,j \le n} A(C_{ij} \cap S_{\theta_k}(t_{n-1}, t_n)) x_{ij} \end{pmatrix}$$

Recall that $A(V)$ denotes the area of a subset $V \in \mathbb{R}^2$. Note that for the two angles $\bar{\theta}_1$ and $\bar{\theta}_2$ this definition is equivalent to Equations (5.2) and (5.3).

A good choice for the new pair of projection angles $\theta_i, \theta_j \in D$ in each iteration of the main loop is to choose the pair of valid directions for which the summed projection distance

$$|P_i(X) - p_i|_2 + |P_j(X) - p_j|_2$$

is largest. We used this criterion for the experiments in Section 5.5.

Under certain conditions it may be better to use a scheme that guarantees that all projections are used equally often. If the number of valid pairs of projection angles is small (e.g., $d = 4,5$), it is likely that *cycling* occurs between two or three projection pairs when the total projection distance is used to select a new pair of projections. The other pairs will not be used, which may degrade the reconstruction quality. Also, when one projection contains significantly more noise than other projections, it will be selected over and over again, as it is more likely to have a higher projection error than the other projections. Possible selection schemes that avoid these problems are to select a random valid pair in each iteration, to select all pairs sequentially in a fixed order, or to select one random angle first and then choose the second angle such that the summed projection distance is largest.

5.4.2. Converting between different grids

Let $X = (x_{ij})$ be a two-projection image and let C_{ij} denote grid cell (i, j) in the corresponding two-projection grid. To display the image X, we need to convert it to an image $X' = (x'_{i'j'})$ on the square pixel grid, aligned with the horizontal and vertical axes. Denote the grid cells of this new grid by $C'_{i'j'}$ $(1 \leq i', j' \leq n)$. As X cannot be represented perfectly on the grid of X', we have to resort to an approximation. For each cell $C'_{i'j'}$ the intersection is computed with all cells of the "old" grid and the gray-value is averaged, weighted by the intersection area:

$$x'_{i'j'} = \sum_{1 \leq i,j \leq n} \frac{A(C_{ij} \cap C'_{i'j'})}{A(C'_{i'j'})} x_{ij}$$

The coefficients $A(C_{ij} \cap C'_{i'j'})/A(C'_{i'j'})$ can be precomputed for each pair of two-projection grids. For a fixed pixel (i', j') in the new grid, the number of pixels (i, j) in the old grid for which $A(C_{ij} \cap C'_{i'j'})$ is nonzero is typically very small, because only valid projection pairs are used (see Section 5.4.1). By using a sparse representation that stores only the nonzero coefficients, the conversion of an image from one two-projection grid to another two-projection grid can be performed in linear time.

Another operation that is used by our algorithm, to obtain the weight map, is the computation of the *average gray value* in a small *neighbourhood* of a point $(x, y) \in \mathbb{R}^2$. Most common pixel neighbourhood definitions that are used in image processing, such as the 4-neighbourhood and 8-neighbourhood, are very suitable for square pixel grids. However, as our algorithm deals with many different pixel grids for which the pixel sizes and shapes may vary, we have to use a more general neighbourhood definition.

Define the *r-neighbourhood* $N(x, y, r)$ of (x, y), as

$$N(x,y,r) = \{(\tilde{x},\tilde{y}) \in \mathbb{R}^2 : \sqrt{(x-\tilde{x})^2 + (y-\tilde{y})^2} \leq r\}$$

Let X be a gray value image and denote the corresponding grid cells by C_{ij}. The *r-neighbourhood gray value* $\Gamma_X(x, y, r)$ of a point (x, y) can now be computed as

$$\Gamma_X(x,y,r) = \sum_{1 \leq i,j \leq n} \frac{A(N(x,y,r) \cap C_{ij})}{A(N(x,y,r))} x_{ij}. \tag{5.8}$$

Although an intersection area of the form $A(N(x,y,r)\cap C_{ij})$ can in principle be computed exactly (up to floating point errors), a sufficiently accurate approximation can be obtained by representing $N(x,y,r)$ as a regular polygon with $\tilde{n}$ vertices, where $\tilde{n}$ is a large integer, e.g., $\tilde{n}=100$. The intersection area of two convex polygons of $\tilde{m}$ and $\tilde{n}$ vertices respectively can be computed in $O(\tilde{m}+\tilde{n})$ time, see Section 7.6 of [12].

5.4.3. Computing the weight map

The weight map W^{τ} is computed from the previous reconstruction $X^{\tau-1}$ and the measured projections for angles $(\bar{\theta}_1^{\tau},\bar{\theta}_2^{\tau})$. We denote the grid cells of the two-projection grid from the previous iteration by $C'(i'j')$ and the grid cells of the new two-projection grid by $C(i,j)$.

Define the *cell center* m_{ij} of cell C_{ij} as the intersection point of two fan half lines as

$$m_{ij}=(x_{m_{ij}},y_{m_{ij}}):=L_{\bar{\theta}_1^{\tau}}((t_{i-1}+t_i)/2)\cap L_{\bar{\theta}_2^{\tau}}((t_{j-1}+t_j)/2)$$

The pixel weight w_{ij}^{τ} of pixel (i,j) depends directly on the r-neighbourhood gray value of the cell center m_{ij}, computed using the previous reconstruction X'. In all experiments of Section 5.5 the parameter r is a constant. For each cell center m_{ij}, the r-neighbourhood gray value $\Gamma_{X^{\tau-1}}(x_{m_{ij}},y_{m_{ij}},r)$ of m_{ij} in the image $X^{\tau-1}$ is computed according to Equation (5.8).

Let $g:[-1,1]\to\mathbb{R}_{>0}$ be an odd function (i.e., $g(-v)=-g(v)$), which is increasing on the interval $[0,1]$. We call g the *local weight function*. Together with the neighbourhood radius r, the local weight function determines the preference for locally smooth regions.

The pixel weight w_{ij}^{τ} is computed as

$$w_{ij}^{\tau}=g(2(\Gamma_{X^{\tau-1}}(x_{m_{ij}},y_{m_{ij}},r)-\frac{1}{2}))$$

Note that $0\leq\Gamma_{X^{\tau-1}}(x_{m_{ij}},y_{m_{ij}},r)\leq 1$.

The basic motivation of the local weight function is that, as the neighbourhood of a pixel becomes more white, the preference to make this pixel white in the next iteration increases. The same holds for black neighbourhoods. If a pixel neighbourhood consists of 50% black and 50% white pixels, no preference is expressed as the pixel weight is zero. Which local weight function works best for a certain set of projection data depends on local properties of the (unknown) original image. If the original image contains large regions that are either completely black or completely white, it makes sense to use $g(v)=\text{sign}(v)v^2$: the pixel weight increases strongly near $v=1$ (and decreases strongly near $v=-1$), which results in a strong preference for local neighbourhoods that are either completely white or completely black. On the other hand, if the image contains many fine details (i.e., local neighbourhoods containing both black and white pixels), it will be better to use $g(x)=\text{sign}(v)\sqrt{|v|}$, which results in a weaker preference for completely homogeneous neighbourhoods. for the experiments in Section 5.5 we used the function

$$g(v)=\begin{cases}v & \text{if} \quad |v|\neq 1\\ 2v & \text{if} \quad |v|=1.\end{cases} \tag{5.9}$$

This function expresses a strong preference for pixels that are surrounded solely by pixels of the same value to retain their value in the next iteration. For pixel neighbourhoods that are not homogeneous, the pixel weight function is linear. The results in Section 5.5 show that this local weight function works well for a varied set of test images.

The other parameter that determines the preference for smooth reconstructions is the neighbourhood radius r. If the neighbourhood radius is large, the local weight function expresses a preference for pixels to be assigned the same value as the majority of their neighbouring pixels in a large surrounding area, which corresponds to some form of "blurring". Fine details cannot be reconstructed in this way, but on the other hand the influence of noise is reduced. If the neighbourhood radius is small, the local weight function expresses a preference for pixels to retain their value from the previous reconstruction.

The interplay of the local weight function and the neighbourhood radius r is quite complex and difficult to analyze, due to the wide range of possible local weight functions. Therefore, the results in Section 5.5, which use only a single local weight function, are not necessarily the best possible results for our algorithm. A different choice for the local weight function and neighbourhood radius may lead to better results.

5.4.4. Termination and final output

Define the *total projection distance* $\Delta(X)$ as

$$\Delta(X) = \sum_{k=1}^{d} |P_k(X) - p_k|_2$$

The total projection distance provides an indication of the quality of the reconstruction X without having knowledge of the unknown original image. Therefore it can be used to define a termination criterion for the main loop of the algorithm. The algorithm terminates if no improvement has been made in the total projection distance (i.e., by finding a new reconstruction that has a smaller projection distance than the best one so far) during T iterations, where T is a constant. We used $T = 30$ for all experiments in Section 5.5.

The result of one iteration of the algorithm is always a reconstruction that adheres well to the two projection angles that were used in that iteration. If the projection data contain noise, errors caused by the noise in those two projections will always be present in the last reconstructions. To reduce this effect, the final reconstruction that is returned by the reconstruction algorithm is computed as an *average* image over the last T' iterations, where T' is a constant. The reconstructions from the last T' iterations, each computed on a two-projection grid, are converted to the standard square pixel grid. On this grid, the pixel values are averaged and thresholded at 0.5 to obtain a binary reconstruction. We used $T' = 15$ for all experiments in Section 5.5.

5.5. Experimental results

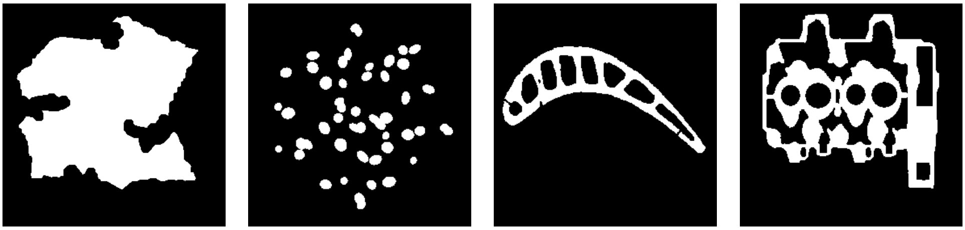
In this section we present reconstruction results of our algorithm. First, we present reconstruction results for a set of four phantom images (i.e., synthetic images). For each of the phantoms we computed the measured projection data by simulation. In this way the noise level can be controlled, which is difficult for datasets obtained by real measurements. Subsequently, we provide reconstruction results for a real measured dataset, obtained by scanning a raw diamond using a micro-CT scanner.

We implemented the iterative network flow algorithm in C++, using the *RelaxIV* library [2] to solve the min cost flow problems. All experiments were performed on a 2.4GHz PentiumIV PC.

As noted in Section 5.4.2, a large part of the computations that are involved in converting an image from one grid to a different grid, or computing the neighbourhood gray value, can be precomputed. In the running times that are reported in this section we assume that these computations have already been carried out, as the precomputing step is independent from the measured projection data and has to be performed only once for each data collection geometry.

5.5.1. Parallel beam projections

The four phantom images that we use to evaluate the reconstruction time and the reconstruction quality of our algorithm are shown in Figure 5.10.



a. single object 300×300

b. 50 ellipses 256×256

c. turbine blade 276×276

d. cylinder head 276×276

Figure 5.10: *Four phantom images*

The phantom in Figure 5.10a represents a single object, which is not convex. In contrast with the first phantom, the phantom in Figure 5.10b represents many small separate objects. The image contains 50 ellipses of various sizes. The phantom in Figure 5.10c represents a cross section of a turbine blade from a jet engine. Checking turbine blades for cracks and other defects is crucial, as defects in one of the blades may lead to malfunction of the engine. Note that the turbine blade phantom contains several small gaps and cracks, which we would like to see in the reconstruction. The fourth phantom, in Figure 5.10d represents a cross section of a cylinder head in a motorcycle engine. Cylinder heads, which are often made of aluminium, are occasionally scanned in industry for reverse engineering, using X-rays.

First, we compare our reconstruction results from perfect parallel beam projections to two alternative approaches. It is well known that continuous tomography algorithms such as Filtered Backprojection require a large number of projections. Algebraic reconstruction methods, such as ART and SIRT (see Chapter 7 of [10]) typically perform much better than backprojection algorithms if the number of projections is very small. Our algorithm uses the SIRT algorithm to compute a start solution. As a first comparison, we compare the final reconstruction computed by our algorithm to the continuous SIRT reconstruction. In [15, 16], Weber et al. describe a linear programming approach to binary tomography which incorporates a smoothness prior. We implemented the *R-BIF* approach which is described in [16] using the ILOG CPLEX interior point solver [8]. The real-valued pixel values are rounded to binary values as a postprocessing step, which was also done in [15]. Besides the projection data, the linear program depends on a parameter α, which determines the preference for smoothness. For our set of four phantoms we found that $\alpha = 0.2$ works well.

For our network flow algorithm we use the local weight function of Equation (5.9). as the local weight function. The neighbourhood radius is set to 1.5 times the diameter of a pixel (in the phantom image).

Figure 5.11 shows the reconstruction results for the four phantoms from parallel beam projection data. For each of the phantoms, the number of (equally spaced) strips in a projection equals the width, in pixels, of the phantom. Quantitative results for the four phantoms are shown in Table 5.1.

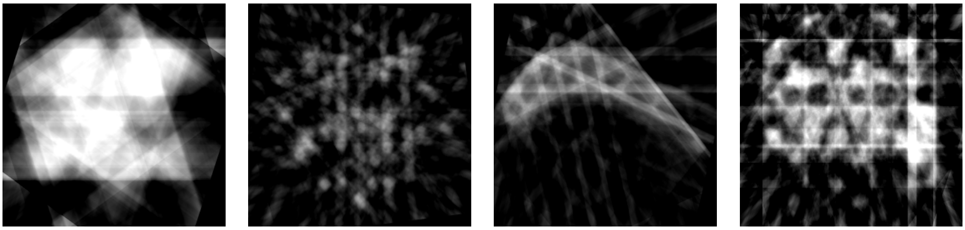
phantom	size	#proj.	R-BIF		Network Flow	
			#errors	time(min)	#errors	time(min)
single object	300×300	5	220	9.2	58	2.0
50 ellipses	256×256	8	216	3.5	152	1.5
turbine blade	276×276	7	214	3.2	108	1.8
cylinder head	276×276	10	375	10	665	1.9

Table 5.1: *Quantitative comparison between the R-BIF reconstruction and the reconstruction computed by our network flow algorithm. The table shows the reconstruction time in minutes and the total number of pixel differences between the reconstruction and the original phantom.*

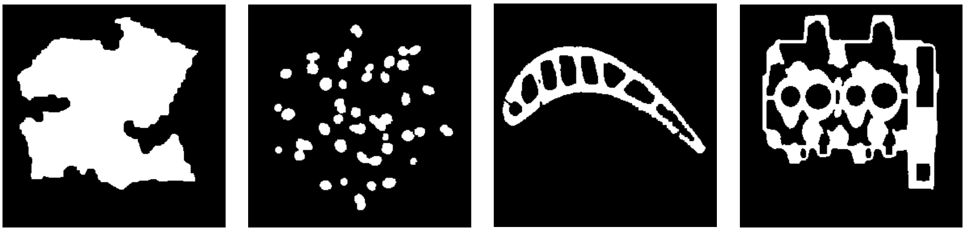
The number d of projections is chosen for each phantom as the minimum number for which our algorithm computes an accurate reconstruction from the projection data. The projection angles are equally spaced between 0 and 180 degrees. Every projection consists of n strip projections, that each have a width equal to the phantom image pixel width.

Our algorithm is capable of computing an accurate reconstruction of each of the four phantom images from a small number of projections. The reconstruction quality of the R-BIF algorithm appears to be similar, although there are some small differences between the reconstructions by both methods. The reconstruction quality for both these algorithms is much better than for the continuous SIRT algorithm, which is immediate by visual inspection of the resulting images. We observed that the number of projections that is required to compute an accurate reconstruction is the same for the R-BIF algorithm as for our algorithm, for each of the four phantoms. As there are major differences between both algorithms, this suggests that this minimal number of projections is an intrinsic property of the images themselves. It may be very difficult or even impossible to compute accurate reconstructions from fewer projections by using a different algorithm.

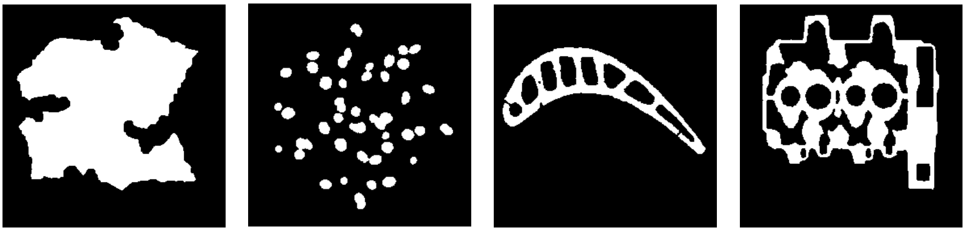
The reconstructions that result from using too few projections may be quite different



a. SIRT, $d = 5$ b. SIRT, $d = 8$ c. SIRT, $d = 7$ d. SIRT, $d = 10$



e. R-BIF, $d = 5$ f. R-BIF, $d = 8$ g. R-BIF, $d = 7$ h. R-BIF, $d = 10$



i. NF, $d = 5$ j. NF, $d = 8$ k. NF, $d = 7$ l. NF, $d = 10$

Figure 5.11: *Reconstruction results for the four phantoms from parallel beam projections, using SIRT (top), R-BIF from [16] (middle) and our network flow algorithm (NF, bottom). The figure captions show the number d of projections that was used.*

between both algorithms. Figure 5.12 shows a reconstruction of the turbine blade phantom from 6 projections for both algorithms.

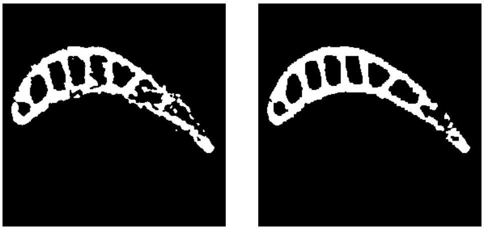


Figure 5.12: *(a) (Left) Network flow reconstruction from 6 projections. (b) (Right) R-FIB reconstruction from 6 projections.*

The minimal number of projections that is required to reconstruct a given (unknown) image depends strongly on characteristics of the unknown image, as can be seen by the varying number of projections that is required for the four phantom images.

If more projections are available than the minimum number that is required, using a few additional projections reduces the number of iterations of our algorithm, resulting in a shorter running time.

Our algorithm has several advantages compared to the R-BIF linear programming approach. First, the network flow algorithm is faster than general linear programming as used in [16], even though we used a high performance interior point solver for solving the linear program. It was demonstrated in Chapter 4 for the case of lattice images that the network flow approach for 2D reconstruction can be extended to a highly efficient algorithm for 3D reconstruction. Within each iteration a series of 2D reconstruction problems is solved, instead of one big 3D problem. The 2D reconstructions can be computed fast (because each subproblem is relatively small) and in parallel. A similar extension is possible for our new algorithm. Dealing with large volumes seems to be very difficult when using general linear programming, as the number of variables becomes huge for large 3D volumes.

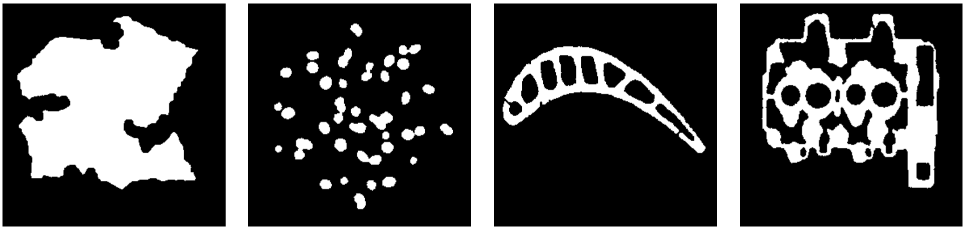
Another advantage of our algorithm is that it can deal with noise effectively, as we will show in the next subsections. The linear programming approach from [15, 16] is not capable of handling noisy data. It can be extended to the case of noisy data, as described in [17]. However, the presented algorithm for dealing with noisy data solves a *series* of linear programs, which results in far longer reconstruction times.

More recently, Schüle et al. developed a different reconstruction algorithm based on *D.C. programming* (Difference of Convex functions) [14]. The results of this algorithm seem to be very promising and it does not have some of the drawbacks of the linear programming approach. We intend to perform a comparison between a wider set of reconstruction algorithms in the future.

5.5.2. Fan beam projections

The main focus of our reconstruction results is on parallel beam projection data. For parallel beam data the network flow methods guarantees a *binary* solution of each two-projection reconstruction problem. For fan beam data, the solution of the two-projection problems is not necessarily binary. Figure 5.13 shows reconstruction results from fan beam projections. For all four phantoms, the center of rotation of the beam source is the center of the image and the distance from the source to the center (the detector radius R) equals the width (and height) of the image. The fan parameters $t_0, \dots, t_n$ are equally spaced.

The results show that even though the solutions of the two-projection subproblems are not necessarily binary, the algorithm is still capable of computing a binary reconstruction. The reconstructions are somewhat less accurate than for the parallel beam case. In particular, there is a small gap visible in the left part of the turbine blade phantom, which is not visible in the reconstruction. The cylinder head reconstruction contains a small crack at the top, which is not present in the original phantom. Adding two more projections for these phantoms results in reconstructions without these errors. As the pixel sizes and shapes vary significantly between pixel grids in different iterations of the algorithm, we expect the



a. $d = 5$, 2.8min. b. $d = 8$, 3.3min. c. $d = 7$, 3.2min. d. $d = 10$, 8.1min.

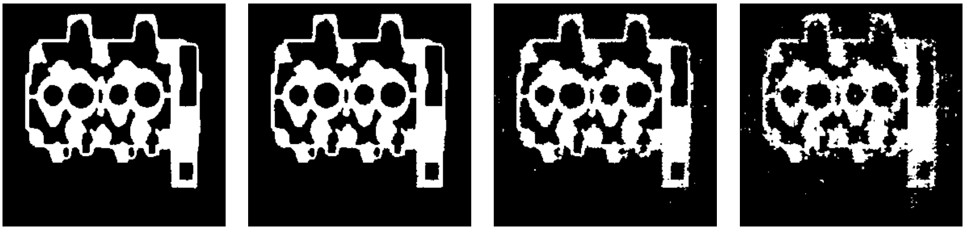
Figure 5.13: *Reconstruction results of the network flow algorithm from fan beam projections. For each phantom, the distance of the X-ray source to the center of the image equals the width (and height) of the image. The figure captions show the number d of projections and the reconstruction time in minutes.*

loss of accuracy caused by conversions between different grids to be larger for fan beam tomography than for parallel beam tomography.

5.5.3. Noisy projection data

So far all experiments were carried out with perfect projection data. We now focus on the reconstruction of images from noisy projection data, which is practically more realistic. We also assume that the noise is independent for each projected strip. To be more precise, we assume that the noise is additive and that it follows a Gaussian distribution with $\mu = 0$ and σ constant over all projected strips. The standard deviation σ is expressed as $\sigma = vy$, where y denotes the average measured strip projection over all projected strips. For example, taking $v = 0.1$ results in independently distributed additive noise for each strip projection, following the $\mathcal{N}(0, 0.1y)$ distribution.

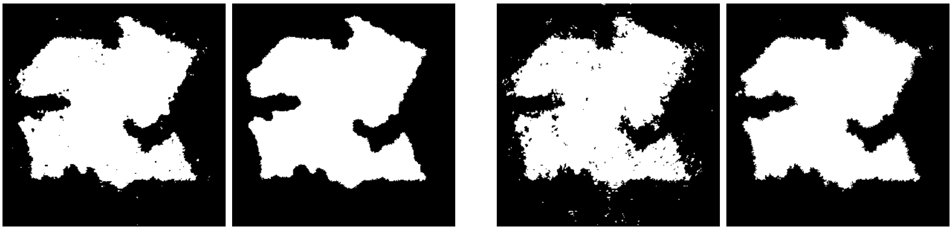
Figure 5.14 shows reconstruction results for the cylinder head phantom, using 10 parallel beam projections and varying noise levels. The reconstruction quality decreases gradually as the noise level is increased. Up to $v = 0.02$, the noise hardly has any visible influence on the reconstruction quality.



a. $v = 0.01$ b. $v = 0.02$ c. $v = 0.05$ d. $v = 0.10$

Figure 5.14: *Reconstruction results of the network flow algorithm from 10 parallel beam projections for increasing noise levels.*

It is clear from Figure 5.14d that for high noise levels the reconstruction is not very smooth at all. The smoothness of the reconstruction can be increased by increasing the size r of the local neighbourhood used in the computation of the weight map. However, increasing the neighbourhood size reduces the ability to reconstruct fine details. If such details are not present, such as in the “single object” phantom, or if they are not important, increasing the neighbourhood size results in better reconstructions for high noise levels. Figure 5.15 compares reconstructions of the single object phantom for neighbourhood sizes of 1.5 and 6 times the diameter of a pixel in the phantom image. Increasing the neighbourhood size results in far better reconstructions. The ability to compute reconstructions under noisy conditions is very important in practical applications.



a. $v = 0.02, r = 1.5$ b. $v = 0.02, r = 6$ c. $v = 0.05, r = 1.5$ d. $v = 0.05, r = 6$

Figure 5.15: *Reconstruction results of the network flow algorithm for the single object phantom with two noise levels ($v = 0.02$ and $v = 0.05$) and two local neighbourhood sizes ($r = 1.5$ and $r = 6$).*

5.5.4. Real-world data

Besides the experiments with simulated projection data, we also performed a reconstruction experiment with real X-ray CT data. Figure 5.16a shows a *sinogram* obtained by X-ray tomography of a slice of raw diamond (courtesy of SkyScan, Antwerp, Belgium). The sinogram shows the measured parallel beam projections of the slice for 500 consecutive projection angles, equally spaced between 0 and 180 degrees. Each horizontal line of the sinogram corresponds to a measured projection. If the diamond is not polluted by other materials, it can be considered as a homogeneous object, which leads to a binary reconstruction problem.

Figure 5.16b shows a gray value reconstruction computed by the SIRT algorithm from the sinogram in Figure 5.16a, using 125 of the available 500 projections. The continuous reconstruction was thresholded to obtain a binary reconstruction, which is shown in Figure 5.16c. The sinogram in Figure 5.16a contains several artifacts, caused by some problems with the scanning device at the time of the experiment. In particular there are several very faint projections (dark horizontal lines) visible. When using continuous tomography, the influence of such errors becomes smaller as the number of projections is increased. A second source of errors is that the sinogram is an 8-bit gray scale image, which limits the accuracy of the measured projection values. The errors make the task of reconstructing the diamond slice from few projections by discrete tomography more challenging.

Figure 5.17 shows reconstruction results from 5 and 10 projections for the SIRT algorithm and our network flow algorithm. Although the basic shape of the diamond slice can be distinguished in the SIRT reconstruction from 10 projections, this reconstruction is of very

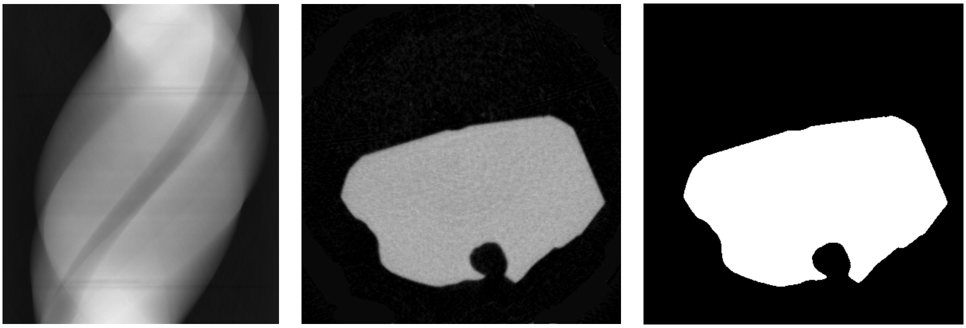
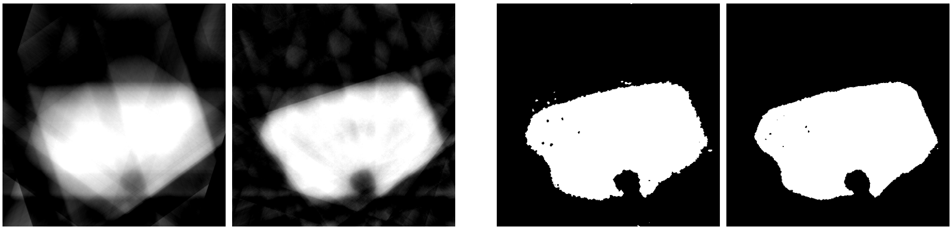


Figure 5.16: *(a) (Left) Parallel beam sinogram of a raw diamond slice (courtesy of SkyScan, Antwerp, Belgium). (b) (Middle) SIRT reconstruction from 125 projections. (c) (Right) Thresholded SIRT reconstruction: a binary image.*

poor quality. The reconstruction computed by our algorithm from 5 projections approximates the real shape quite well. Due to noise and other errors in the projection data there are still some errors in the reconstruction. The reconstruction from 10 projections is much more accurate. In particular, the boundary of the diamond slice is reconstructed well.

This example shows that our algorithm is capable of handling real-world projection data, even if the data contains significant noise or other errors. The number of projections that is required to compute an accurate reconstruction is far lower than for continuous tomography, which is currently used in most applications.



a. SIRT, $d = 5$ b. SIRT, $d = 10$ a. NF, $d = 5$ b. NF, $d = 10$

Figure 5.17: *Reconstruction results for the SIRT algorithm and our network flow algorithm (NF) from 5 and 10 equally spaced projections.*

5.6. Conclusions

We have described a novel algorithm for the reconstruction of binary images from a small number of their projections. Our algorithm is iterative. In each iteration a reconstruction problem is solved that depends on two of the projections and the reconstruction from the previous iteration. We showed that the two-projection reconstruction problem is equivalent to the problem of finding a flow of minimal cost in the associated graph. This equivalence allows us to use network flow algorithms for solving the two-projection subproblems. The network flow approach is most effective for parallel beam projections, but it can also deal with fan beam projection data.

The reconstruction results show that the reconstruction quality of our algorithm is far better than for the SIRT algorithm from continuous tomography. A comparison with the R-BIF linear programming approach from [16], which uses a smoothness prior, shows that both algorithms yield comparable reconstruction quality. Our algorithm runs faster than the linear programming approach and can be easily extended to 3D reconstruction or reconstruction from noisy projections. These extensions are difficult to accomplish efficiently for the linear programming approach.

Our results for noisy projection data shows that the algorithm is capable of dealing with significant noise levels. The reconstruction quality decreases gradually as the noise level is increased. A smoother reconstruction can be obtained by increasing the local neighbourhood size that is used to compute the weight map. For images that do not contain fine details it is likely that increasing the smoothness results in better reconstructions.

By providing reconstruction results of a raw diamond slice from real X-ray scanner data, we showed that our algorithm is capable of computing high quality reconstructions from real-world data.

In future research we intend to perform more extensive comparisons between different discrete tomography algorithms. Such a comparison is often difficult, as each algorithm makes different assumptions on the class of images, the detector setting, etc. Generalization of our algorithm to 3D reconstruction is straightforward, following the same approach as in Chapter 4. Even for 3D objects that could be reconstructed as a series of 2D slices, an algorithm that takes the 3D connectivity of the object into account (using a 3D local neighbourhood) could possibly improve the reconstruction quality.

Bibliography

[1] Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: *Network flows: theory, algorithms and applications*, Prentice-Hall (1993).

[2] Bertsekas, D.P., Tseng, P.: RELAX-IV: a faster version of the RELAX code for solving minimum cost flow problems. *LIDS Technical Report LIDS-P-2276*, MIT, (1994).

[3] Censor, Y.: Binary steering in discrete tomography reconstruction with sequential and simultaneous iterative algorithms. *Linear Algebra Appl.*, **339**, 111–124 (2001).

[4] Fishburn, P., Schwander, P., Shepp, L., Vanderbei, R.: The discrete Radon transform and its approximate inversion via linear programming, *Discrete Appl. Math.* **75**, 39–61 (1997).

[5] Gale, D.: A theorem on flows in networks. *Pacific J. Math.*, **7**, 1073–1082 (1957).

[6] Gardner, R.J., Gritzmann, P., Prangenberg, D.: On the computational complexity of reconstructing lattice sets from their X-rays. *Discrete Math.*, **202**, 45–71 (1999).

[7] Herman, G.T., Kuba, A., eds.: *Discrete tomography: foundations, algorithms and applications*. Birkhäuser, Boston (1999).

[8] *ILOG CPLEX*, `http://www.ilog.com/products/cplex/`

[9] Jinschek, J.R., Calderon, H.A., Batenburg, K.J., Radmilovic, V., Kisielowski, C.: Discrete tomography of Ga and InGa particles from HREM image simulation and exit wave reconstruction. *MRS Proc.*, **839**, 4.5.1–4.5.6 (2004).

[10] Kak, A.C., Slaney, M.: *Principles of computerized tomographic imaging*. SIAM (2001).

[11] Natterer, F.: *The mathematics of computerized tomography*, SIAM (2001).

[12] O'Rourke, J.: *Computational geometry in C*, Cambridge University Press (2001).

[13] Schrijver, A.: *Combinatorial optimization: polyhedra and efficiency*. Algorithms and Combinatorics series, **24**, Springer, Heidelberg (2003).

[14] Schüle, T., Schnörr, C., Weber, S., Hornegger, J.: Discrete Tomography by Convex-Concave Regularization and D.C. Programming. *Discrete Appl. Math.*, **151**, 229–243 (2005).

[15] Weber, S., Schnörr, C., Hornegger, J.: A linear programming relaxation for binary tomography with smoothness priors. *Electron. Notes Discrete Math.*, **12** (2003).

[16] Weber, S., Schüle, T., Schnörr, C., Hornegger, J.: A Linear Programming Approach to Limited Angle 3D Reconstruction from DSA Projections. *Methods Inf. Medicine*, **4**, Schattauer Verlag, 320-326 (2004).

[17] Weber, S., Schüle, T., Hornegger, J., Schnörr, C.: Binary Tomography by Iterating Linear Programs from Noisy Projections. *Proc. of IWCIA 2004*, Lecture Notes in Computer Science **3322**, 38–51 (2004).

Chapter 6

On the reconstruction of crystals by discrete tomography

This chapter has been published as: K.J. Batenburg and W.J. Palenstijn, "On the reconstruction of crystals through discrete tomography", Proceedings of the Workshop on Combinatorial Image Analysis, Lecture Notes in Computer Science, Vol. 3322, 23–37, Springer (2004).

Abstract. We consider the application of discrete tomography to the reconstruction of crystal lattices from electron microscopy images. The model that is commonly used in the literature to describe this problem assumes that the atoms lie in a strictly regular grid. In practice, this is often not the case. We propose a model that allows for nonregular atom positions. We describe a two-step method for reconstructing the atom positions and types. For the first step, we give an algorithm and evaluate its performance.

6.1. Introduction

Over the past ten years, the research field of discrete tomography has received considerable attention [2]. Among the principal motivations for studying the tomographic reconstruction of images which have a small discrete set of pixel values is a demand from materials science. Advances in the field of electron microscopy have made it possible to count the number of atoms in each column of a crystal lattice along a small number of directions [3] (see Figure 6.1). For this application the reconstruction problem consists of retrieving the individual atom positions from a small number of projections.

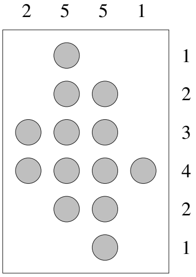


Figure 6.1: *Atom grid with horizontal and vertical projections.*

In the mathematical model that is commonly used in the literature to describe the reconstruction problem, it is assumed that the atoms lie in a strictly regular grid. Each column of atoms corresponds to a single column of pixels in the resulting image.

The number of atoms in each column is not measured directly by the detector array of the microscope. Processing of the raw measured data results in a *reconstructed exit wave* of the crystal. The phase of the exit wave (see Figure 6.2a) can be regarded as a projected image of the crystal. Figure 6.2b shows the measured phase along a line of projected atom columns. In the exit wave phase image each atom column has a width of several pixels, typically around 10.

The atom columns appear as spikes in the measured projection. For crystals that consist of only one type of atom, the height of a peak depends (almost) linearly on the number of atoms in the corresponding column. Therefore, the number of atoms in a column can be found by dividing the peak height by the projection height of a single atom and rounding to the nearest integer.

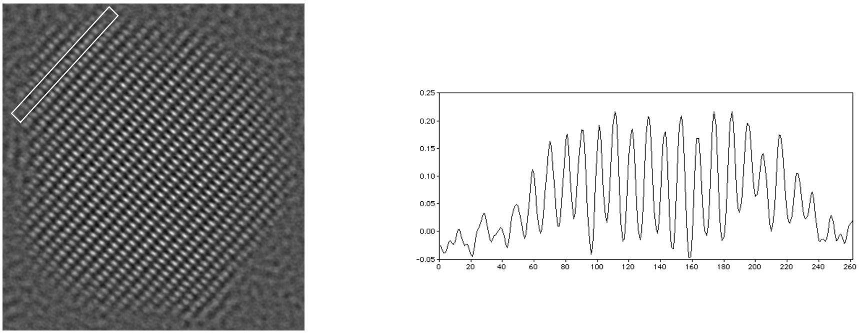


Figure 6.2a. Figure 6.2b.

Figure 6.2: *a) Exit wave reconstruction of a CdSe nanocrystal b) Measured phase along a line in the crystal projection (marked in the left figure); courtesy of dr. Ch. Kisielowski, NCEM, Lawrence Berkeley Lab.*

For a solid crystal without any irregularities the grid model corresponds well to physical reality. Unfortunately, many crystals that are of interest in materials science contain one or more irregularities, known as *defects*, in the regular grid structure. Figure 6.3 shows examples of the various types of defects that may occur in a crystal that contains two different atom types, indicated by light and dark gray circles. The defects marked by a, c and d are examples of *point defects*. Defect b is an example of an *edge dislocation*. Note that in most cases the surrounding atoms have moved away from their grid positions.

If a crystal contains defects, the atoms do not lie in straight columns, which makes the method of counting atoms by looking at the peak heights inapplicable.

Figure 6.4a shows a single atom and its projection as measured by the detector array. Figure 6.4b shows the same atom shifted slightly to the right. Although the displacement is less than the width of a detector cell, we can tell that the atom has moved from the observed

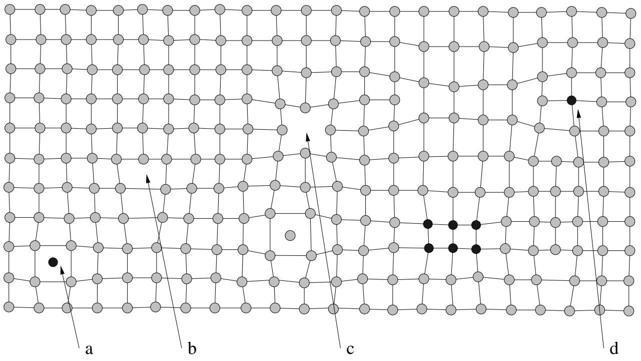


Figure 6.3: *Overview of various crystal defects; courtesy of prof.dr. H. Föll.*

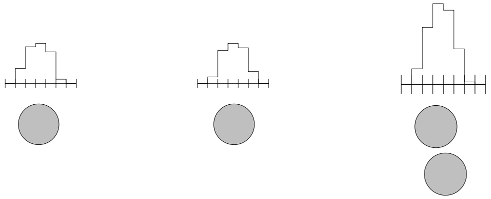


Figure 6.4a. Figure 6.4b. Figure 6.4c.

Figure 6.4: *a) A single atom and its measured projection. b) The same atom shifted slightly to the right. c) Two atoms for which the projection shows only one peak.*

pattern. Clearly, when measuring a single atom, assuming perfect, noiseless measurements, it is possible to determine its position with greater accuracy than the actual detector width. Figure 6.4c shows two atoms which are not vertically aligned. The measured data shows only a single peak. Yet, by using the a priori knowledge that the peak is a superposition of two atom projections, it is possible to recover the horizontal positions of both atoms exactly by solving a least squares problem (see Section 6.3.2).

This raises the question if recovering the positions of the atoms is also possible for larger atom configurations, consisting of many more atoms. If the configuration contains several different atom types, it is not even clear if the number of atoms of each type can be computed from the measured projection data.

When performing a tomographic reconstruction, projection data from several directions is available. We want to reconstruct the measured set of atoms (i.e., the number of atoms, their horizontal and vertical coordinates and their types) from the projections.

We propose a two-step method, which first performs a reconstruction on each projection separately. The first step consists of determining the atom positions and types in the direction orthogonal to the projection direction, for all separate projections. This results in a discrete tomography problem which is much simpler than the original reconstruction problem. The second step consists of solving the discrete tomography problem. In this chapter we focus on the first step. Clearly, the ability to accurately perform this first step is a necessary condition for the two-step procedure to be feasible. The second step will be considered in future research.

Figure 6.5 shows the basic idea of the two-step approach. First the x-coordinates of the atoms and their types are computed from the vertical projection. Then the y-coordinates of the atoms, along with their types, are computed from the horizontal projections. The process may be repeated if more projections are available. Subsequently the resulting data is combined into a discrete tomography problem. As the computation for each separate projection also yields the types of the projected atoms, it may be possible to solve the tomography problem for each atom type independently if the sample contains several atom types.

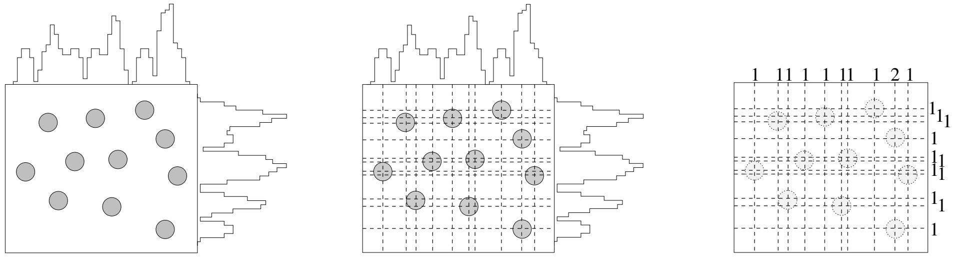


Figure 6.5a. Figure 6.5b. Figure 6.5c.

Figure 6.5: *a) A set of atoms and two of its projections. b) Coordinates of the atoms, reconstructed from the projections. c) The resulting discrete tomography problem.*

In this chapter we explore to what extent the atom positions and their types can be recovered from a single projection. We restrict ourselves to the processing of 1-dimensional projections of 2-dimensional images. We present an algorithm that computes the atom positions and their types. Our experimental results demonstrate that even when many atoms are projected on top of each other it is still possible to recover the positions and types of the individual atoms that constitute the projection. The algorithm does not perform an exhaustive search and is not guaranteed to find a solution. We evaluate its performance and show that it is capable of finding accurate reconstructions for a varied set of test data. The results show that even in the presence of a limited amount of noise the algorithm performs well.

6.2. Preliminaries

Although the model that we study in this chapter is quite flexible, we have to make several simplifications in comparison with physical reality. In this section we define our model and describe in what ways the model may deviate from actual physical experiments.

We restrict ourselves to 2-dimensional images. Although generalization of our methods to 3-dimensional images is not straightforward, we consider the development of algorithms for 2D images as a necessary first step towards the development of more complicated algorithms for the 3D case.

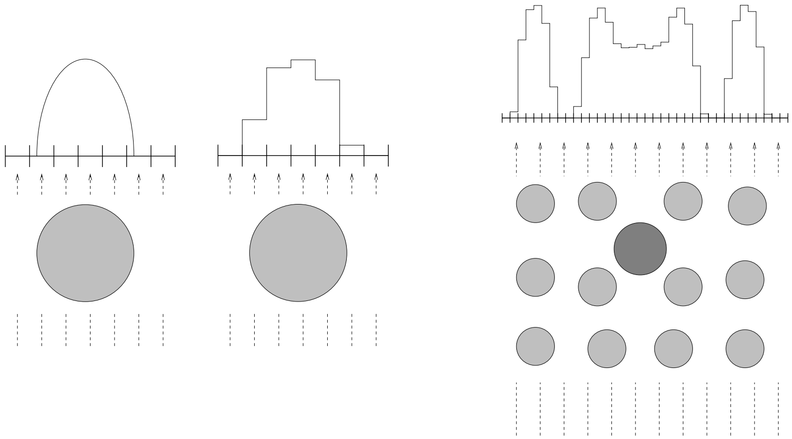


Figure 6.6a. Figure 6.6b. Figure 6.6c.

Figure 6.6: *a) A single atom and its projection function. b) The projection of an atom as measured by the detector array. c) A set of atoms and its measured projection.*

Figure 6.6a shows a schematic representation of the experimental setup that we will study. Using an electron microscope, one can measure the effect of an atom on a passing electron beam. The figure shows a single atom a, centered in (x_a, y_a), and its effect on a vertical electron beam. The effect of the atom can be described by the *projection function* $f_a(x)$, which gives the magnitude of the effect for any given x. We assume the following properties of f_a:

- f_a is independent of the y-coordinate of the atom;
- f_a is not the zero-function;
- there exists an r such that $f_a(x) = 0$ if $|x - x_a| > r$. The smallest such r is the *atom radius* r_a;
- f_a is continuous;
- $f_a(x_a + x)$ is independent of the atom position x_a. In other words: when we move the atom horizontally, its projection function shifts along with it;
- f_a is nondecreasing for $x \leq x_a$;
- f_a is symmetric, i.e., $f_a(x_a - x) = f_a(x_a + x)$ for all x.

The first five properties are essential to the algorithm that we propose. The remaining properties are assumed for computational convenience and efficiency and are physically realistic. It is possible to remove these assumptions by adapting our algorithm if necessary.

We use the convention of representing atoms by circles in our figures. We assume that the projection function of an atom is independent of the orientation of the atom. In other words, an atom looks the same from all directions.

The detector array does not measure the projection function directly: each detector cell measures the average value of the projection function on a certain horizontal interval (see Figure 6.6b).

In practice, it is not possible to isolate a single atom. When the atom is part of a larger crystal (see Figure 6.6c), it is only possible to measure the projection function f_{tot} of the whole crystal. We assume that f_{tot} is the sum of the projection functions of all individual atoms that make up the crystal:

$$f_{\text{tot}}(x) = \sum_{\text{atoms } a} f_a(x).$$

This assumption is not completely valid in practice. The projection functions of the individual atoms do not add up linearly although the nonlinear effects are small. The method that we describe in this chapter can also be used in the nonlinear case, as long as adding an atom a will result in a sufficiently large increase of $f_{\text{tot}}(x)$. We assume that the measurable effect of the atoms on the electron beam is limited to the interval $[0,w]$ measured by the microscope where w is the width of the detector array. In other words, $f_{\text{tot}}(x) = 0$ for $x < 0$ or $x > w$. It is possible to drop this assumption, i.e., to allow for a sample which stretches beyond the detector array, but this will result in a severe loss of accuracy near the edges.

The detector that measures the projection consists of n_d consecutive detector cells that have a fixed width w_{det}. We denote the measurement of detector cell $i = 0, 1, \ldots, n_d - 1$ by m_i. For noiseless measurements, m_i is the integral of the projection function over the interval $[x_i, x_i + w_{\text{det}}]$:

$$m_i = \int_{x_i}^{x_i + w_{\text{det}}} f_{\text{tot}}(x)\,\mathrm{d}x$$

Note that the detector cells are equally spaced and $x_i = i w_{\text{det}}$.

We assume that the crystal consists of a small number of *atom types*, e.g., Cd, Se, Au, etc. Atoms of the same type are indistinguishable. Therefore, each atom type has a single projection function modulo shifts. The atom types that make up the crystal and their projection functions are known in advance. The main problem that we study in this chapter concerns the recovery of the individual atoms from the collective projection:

Problem 17 *(Reconstruction Problem) Given the detector measurements and a set of atom types with their projection functions, determine a set A of atoms (their types and x-coordinates), such that the Euclidean distance between the projection of A and the measured projection data is minimal.*

It is possible that a solution to the reconstruction problem exists which is quite different from the actual atom configuration. The simplest example of this would occur if two atom types share the same projection function. If certain projection functions are integral linear combinations of different projection functions, similar problems occur.

The detector measurements always contain a certain amount of noise. A high noise level may also result in a solution to the reconstruction problem that is different from the actual measured atom configuration.

We now choose a particular set of atom projection functions for demonstrating the basic concepts of our model and our algorithm. We assume that the atom projection functions are projections of circles, multiplied by a *density*. Each atom type t has an associated pair (r_t, ρ_t), the radius and density of the atom. The projection function of an atom a of type t, centered in x_a, is given by

$$f_a(x) = \begin{cases} 2\rho_t \sqrt{r_t^2 - (x - x_a)^2} & \text{if } x \in [x_a - r_t, x_a + r_t] \\ 0 & \text{otherwise.} \end{cases}$$

Put $u = \sqrt{r_t^2 - (x - x_a)^2}$. Then the function F_a, defined by:

$$F_a(x) = \begin{cases} 0 & \text{if } x \leq x_a - r_t \\ \rho_t((x - x_a)u + r_t^2(\arctan(\frac{x - x_a}{u}) + \frac{\pi}{2})) & \text{if } x \in (x_a - r_t, x_a + r_t) \\ \rho_t \pi r_t^2 & \text{if } x \geq x_a + r_t \end{cases}$$

is a primitive function of the projection function. Hence, the contribution $m_i(a)$ of a to the value measured by detector cell i is given by

$$m_i(a) = \int_{x_i}^{x_i + w_{\text{det}}} f_a(x)\, \mathrm{d}x \quad = \quad F_a(x_i + w_{\text{det}}) - F_a(x_i).$$

Our algorithm for solving the reconstruction problem searches for the atom types and their x-coordinates from left to right. The interval $[0, w]$ is split into grid cells and each of the cells is assigned a number of atoms, possibly of different types. We call such an assignment of atoms to grid cells a *configuration*. An important step of the algorithm is to determine the minimum and maximum contribution of an atom a to the measured value m_i in detector i if a is assigned to grid cell $c = [l_c, r_c]$. Figure 6.7 shows the maximum (bold) and minimum (dashed) values that could be measured in each detector cell if a single atom lies somewhere within the indicated interval from the left circle to the right circle.

Because of the restrictions on f_a (it is symmetric around x_a and nondecreasing for $x \leq x_a$), these values can be easily determined. We denote the minimal contribution of a to detector cell i by $m_{i,\min}(a, c)$ and the maximal contribution by $m_{i,\max}(a, c)$.

For a given configuration s, let c_a denote the cell in which atom a lies. Then

$$m_{i,\min}(s) = \sum_{\text{atoms } a \in s} m_{i,\min}(a, c_a)$$

is a lower bound on the value measured in detector i and

$$m_{i,\max}(s) = \sum_{\text{atoms } a \in s} m_{i,\max}(a, c_a)$$

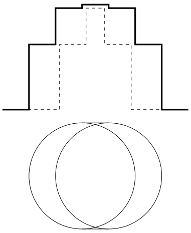


Figure 6.7: *maximal (bold) and minimal (dashed) values measured in each detector cell, for a single atom.*

is an upper bound on the value measured in detector i. We
use these bounds extensively in the *block phase* of our algorithm (see Section 6.3.1). We say that a configuration s is *k-admissible* if

$$\begin{array}{ll} m_{i,\min}(s) \leq m_i \leq m_{i,\max}(s) & \text{for } 0 \leq i \leq k \text{ and} \\ m_{i,\min}(s) \leq m_i & \text{for } i > k. \end{array}$$

Suppose that we try to add atoms to s in order to obtain a configuration which matches the measured data in all detector cells. If we require that the additional atoms only affect the values measured in detector cells to the right of cell k, then k-admissibility is a necessary condition for s to be extendable to a fitting configuration.

In a given configuration s, the position of each atom is determined up to the width of a grid cell. During the fitting phase of our algorithm (see Section 6.3.2), the coordinates of the atoms are fixed at real values that lie within their respective cells. We call the resulting assignment $\tilde{s}$ of coordinates to atoms an *exact configuration* for s. For an exact configuration $\tilde{s}$, the *simulated measurement* (the value that would be measured by the detector array for this configuration) is given by

$$m_i(\tilde{s}) = \int_{x_i}^{x_i + w_{\text{det}}} \sum_{a \in \tilde{s}} f_a(x)\,\mathrm{d}x.$$

We define the *k-distance* of an exact configuration $\tilde{s}$ to the projection data as the square root of $\sum_{i=1}^{k}(m_i - m_i(\tilde{s}))^2$.

6.3. Algorithm

In this section we describe an algorithm for solving the reconstruction problem. Our main goal is to demonstrate that it is indeed possible to recover the atom types and their approximate positions from the projection data in many cases. We use heuristics to limit the search space. It may happen that the algorithm fails to find an optimal solution or even that it fails to find an approximate solution. However, we demonstrate in Section 6.4 that our algorithm is capable of finding high quality solutions for a diverse set of test cases.

The reconstruction problem faces us with the task of recovering both the number of atoms of each type and the x-coordinates of all these atoms. The x-coordinates are real values, so they can take an infinite number of different values. In order to obtain a finite search space, we impose a (1-dimensional) grid on the interval $[0, w]$ and determine for each atom in which grid cell it lies, instead of determining the exact atom positions. The imposed grid is typically finer than the detector grid, e.g., twice as fine. We call this grid the *fine grid*. It is relatively easy to calculate lower and upper bounds on the number of atoms present in each grid cell.

We say that a grid cell c in the fine grid *contains* an atom a if the *left side* of a, defined as $x_a - r_a$ is in c. In this way, the projection of a has no effect on the values measured by detector cells to the left of c. Our algorithm constructs atom configurations incrementally from left to right. Once a configuration has been constructed up to grid cell c, we can check

```
$S_{-1} := \{\textit{empty configuration}\}$;
for $b := 0$ to $n_b$ do
begin
   $S_b := \emptyset$;
   $k_b$ := index of last detector cell covered by blocks $0,\dots,b$;
   foreach $s \in S_{b-1}$ do
   begin
      foreach $b$-extension $t$ of $s$ which is $k_b$-admissible (see Section 6.3.1) do
      begin
         fit $t$ to the measured data (see Section 6.3.2);
         if $t$ fits the data well enough then
            $S_b := S_b \cup \{t\}$;
      end
   end
   cull $S_b$ (see Section 6.3.3);
end
```

Figure 6.8: *Outline of the algorithm.*

if the configuration is k-admissible, where k is the index of the last detector cell that is entirely left of the right boundary of c.

The main loop of our algorithm iterates over the fine grid, from left to right, in steps that comprise a number of fine grid cells. We call a group of fine cells that constitute such a step a *block*. The size of a block is always a power of two (see Section 6.3.1). We denote the number of blocks that cover the interval $[0,w]$ by n_b.

The algorithm maintains a set S of configurations that partially (up to the current block of fine grid cells) fit the measured data well. At the start of iteration b, a configuration $s \in S$ contains atoms up to block $b-1$. New atoms are then added to s in block b, forming a new configuration t, which is called a *b-extension* of s.

When the end of the projection data has been reached, the set S contains configurations that fit the measured data well on the interval $[0,w]$. Each configuration provides an estimate of the atom positions and their types. We remark that the atom coordinates are not determined exactly; they are determined up to grid cells in the fine grid. Figure 6.8 shows an outline of our algorithm. In the next sections the various steps of the algorithm are described in more detail.

6.3.1. Block phase

When searching for b-extensions of a configuration s, we want to determine the possible sets of atoms that the fine grid cells in b can contain such that the corresponding b-extension still fits the measured data. We first determine all such sets for the entire block, where we require the atoms to lie within the left and right boundary of the block without restricting them to a single fine cell. To this end, we extend the admissibility concept to the “coarse” grid of

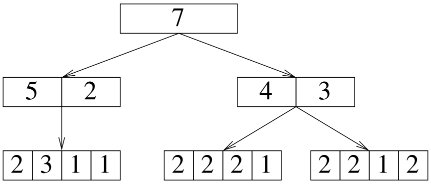


Figure 6.9: *Admissible configurations are determined recursively, forming a tree of refinements.*

blocks, i.e., we determine the minimal and maximal contribution of atoms contained in the current block to the measured values m_i.

We determine all sets of atoms that the current block can contain which satisfy this “extended k_b-admissibility”. Subsequently, the block grid is refined by splitting the block into its left and right halves. As each atom must lie on one of both sides, we form all partitions of the atom set into a left and a right half. For each partition, we adjust the upper and lower bounds on the projection. If these (narrower) bounds still satisfy the extended k_b-admissibility, we recursively split it into a new grid that is again twice as fine. We repeat this procedure until we have reached the fine grid level. Note that we always choose a power of two as the block size.

As an example, we consider a single block of four fine grid cells in the reconstruction of a crystal that consists of only one atom type (see Figure 6.9). The root node of the tree represents the current block b. Suppose that, by using the lower and upper bounds on the contribution of atoms in b to the measured data, it has been determined that block b must contain 7 atoms. These atoms can be split in various ways among the left and right halves of b. For each of the partitions we can compute finer admissibility bounds. In the example tree of Figure 6.9, only two partitions satisfy the finer admissibility bounds: 5 atoms left and 2 atoms right or 4 atoms left and 3 atoms right. By repeating this procedure we end up at the leaf nodes, which represent assignments of the atoms to the four fine grid cells in the block.

6.3.2. Fitting phase

In the block phase, candidate atom configurations are selected which satisfy the admissibility criterion. However, admissibility is not a sufficient condition that a configuration must satisfy to fit the measured data. The main problem with the admissibility condition is that it considers all detector values separately. For example, the measured effect of an atom can never be equal to its lower bound for all detectors simultaneously.

For a given configuration s which has been constructed up to block b, the fitting procedure constructs an exact configuration $\tilde{s}$ for which the atom positions lie in the prescribed cells such that the k_b-distance of $\tilde{s}$ to the measured data is minimal:

$$\textbf{minimize} \quad \sum_{i=1}^{k_b} (m_i - m_i(\tilde{s}))^2.$$

If the k_b-distance of the resulting exact configuration is larger than a constant D (the *fitting* *cutoff*), we can conclude that the configuration s is most likely not a good partial solution to the reconstruction problem and it is discarded.

For solving this least squares problem, we use the Levenberg-Marquardt (LM) algorithm [1, §4.7.3]. The LM algorithm requires an initial approximation to the solution of the least squares problem, from which it iteratively moves to a local optimum. For a configuration s' that is a b-extension of a configuration s, the x-coordinates of atoms that have been added in the current block b are initialized randomly in a small interval around the center of their respective fine cells. Atoms from previous blocks are initialized at the values found by the LM algorithm when applied to s.

As the algorithm progresses, the number of atoms up to the current block — and consequently the number of variables in the least squares problem — will become increasingly large. It is very unlikely that the addition of a new block will significantly affect the positions of atoms that are far to the left of the current block. The positions of such atoms have already been determined by many prior LM steps. In order to limit the number of variables in the least squares problem, we fix the positions of these atoms. To be precise, all atoms in cells that are at least H fine cells to the left of the start of the current block are fixed, where H is a positive integer constant. Consequently, the terms of the least squares problem that correspond to detectors that are solely affected by the fixed atoms are also removed.

The LM algorithm uses the partial derivatives of $m_i(\tilde{s})$ with respect to the atom positions x_a. These derivatives can be expressed easily in terms of the projection function:

$$\frac{\partial}{\partial x_a} m_i(\tilde{s}) = f_a(x_i) - f_a(x_i + w_{\text{det}})$$

where x_i is the left bound of detector cell i. The intuitive explanation of this expression is that if atom a is moved to the right over a distance ∂x_a, the value measured in detector cell i is increased by $f_a(x_i)\partial x_a$ due to the part of the atom projection of a that "slides" into the range of detector cell i at the left boundary, and decreased by $f_a(x_i + w_{\text{det}})\partial x_a$ due to the part of the projection of a that moves out of detector cell i at the right boundary.

Although the LM algorithm will find a locally optimal solution of the least squares problem, it will not necessarily find a global optimum. In its basic form, the LM algorithm does not allow boundary constraints. We implemented the boundary constraints by adding a *penalty function* for each atom to the sum of squared differences. For an atom a that must lie within the interval $[l_a, r_a]$, the penalty is defined as:

$$p_a = \begin{cases} 1000(l_a - x_a)^{1.1} & \text{if } x_a < l_a \\ 0 & \text{if } l_a \le x_a \le r_a \\ 1000(x_a - r_a)^{1.1} & \text{if } x_a > r_a. \end{cases}$$

Note that the penalty function is continuously differentiable, which is a requirement for the LM algorithm. The reason for using the penalty method over other methods that use hard boundary constraints is that it provides more information. When an atom a is slightly outside its bounds in the optimal solution, this is a strong indication that a better configuration exists, for which all variables are within their bounds.

Strictly speaking, the formulation of the minimization problem that is solved by the LM algorithm does not depend on the concept of configurations at all (not taking the penalty functions into account). Without a proper start solution and penalty functions, however, the problem will be extremely difficult to solve analytically, as it has a huge number of local optima.

6.3.3. Culling phase

As the algorithm proceeds from left to right, the number of states in S_b will typically become larger and larger, unless we resort to *culling* the set after each block.

Although we do not make rigid assumptions on the horizontal positions of the atoms, we expect that for crystals the atoms will tend to lie in columns, resulting in separated peaks. It may happen that a single peak in the projection data can be approximated well by several different atom configurations. Suppose that such a peak is followed by one or more zero measurements. If we can extend any of the partial atom configurations (up to the peak) to a configuration that fits the whole projection, we can extend all the other configurations in exactly the same way. Therefore, we delete almost all configurations $s \in S_b$ for which

$$m_{i,\min}(s) = m_{i,\max}(s) = 0 \quad \text{for all detectors } i \text{ to the right of } b.$$

We keep only those configurations s for which the k_b-distance of the corresponding exact configuration $\tilde{s}$ to the measured data is minimal. Note that we can store the deleted configurations so that we can retrieve alternative partial solutions later if desired. This form of culling reduces the number of configurations enormously between consecutive peaks.

Different configurations always result in different sets of boundary constraints for the LM algorithm. This does not mean, however, that the corresponding exact configurations cannot be very similar. For example, suppose that in the measured atom configuration, an atom is near a fine cell boundary. There will be two configurations that approximate the actual configuration very well: one that places the atom in the cell to the left of the boundary and one that places the atom to the right. After the fitting phase, the resulting exact configurations will typically be almost identical.

Note that the penalty approach in the fitting phase allows atoms to move slightly out of their boundaries. We define the *boundary violation* of an atom a as the squared distance to its nearest cell bound if a lies outside its assigned cell and zero otherwise.

To prevent the superfluous processing of nearly identical configurations, we delete configurations for which the corresponding exact configuration is *almost identical* to another one, retaining only the configuration for which the exact configuration adheres best to its cell boundaries, i.e., for which the sum of the boundary violations is minimal. We say that two exact configurations $\tilde{s}$ and $\tilde{s}'$ are *almost identical* if

- the number of atoms of each type is the same in $\tilde{s}$ and $\tilde{s}'$;
- for each pair of corresponding atoms between $\tilde{s}$ and $\tilde{s}'$ (when sorted by x-coordinate) the distance between their respective x-coordinates is smaller than a positive constant C.

This form of culling reduces the number of configurations significantly while processing peaks in the projection data.

6.3.4. Noise

The algorithm that we described does not take noise into account. When working with practical data, however, noise will always be present. We assume that we have a good indication of the noise level in advance. It is not difficult to adapt our algorithm to find solutions in the case of noise. Comparisons of $m_{i,\min}$ or $m_{i,\max}$ with the projection data are made less strict, allowing a margin that depends on the noise level. Additionally, the fitting cutoff D has to be increased since solutions will fit the data less accurately.

6.4. Experimental results

In this section we report reconstruction results of our algorithm for a set of characteristic test images. As the purpose of these experiments is mainly to show the feasibility of the approach, we do not provide large-scale statistical data on the performance.

We implemented the algorithm in C++. For solving the nonlinear least squares problems, we used the MINPACK implementation of the LM-algorithm. We used a 1.4GHz Opteron machine with 2Gb of RAM.

In some of the tests we added noise to the projection data to simulate noisy measurements. For each detector i a random sample $\tilde{r}$ from a normal distribution with average $\mu = 1$ and variance σ^2 is generated. The original measurement m_i for that detector is replaced by $\tilde{r}m_i$. For each test the value of σ is indicated in the table. If $\sigma = 0$ no noise was added.

For all tests, we set $w_{\text{det}} = 1$ and scaled all other quantities accordingly. We used the circle projections, described in Section 6.2. We set the value of the constant H (see Section 6.3.2) to 40 times the number of fine cells per detector. The constant C (see Section 6.3.3) was set to 0.2 times the width of a fine cell. The fitting cutoff D was typically set to 0.01 for test cases without noise, to 3 for test cases with noise level $\sigma = 0.01$, to 5 for $\sigma = 0.03$, and to 10 for $\sigma = 0.05$. In cases where this value of D did not result in a solution, we increased the fitting cutoff slightly.

The first test set is the atom configuration in Figure 6.3. Table 6.1 shows the reconstruction results for two choices of atom radii. The atom densities are the same for both tests and the data contains no noise. Each block of fine grid cells has a size of two detector cells. For the first choice of atom radii each detector cell is split into two fine cells. For the second choice the fine cells correspond directly to the actual detector cells. The number of fine grid cells per detector cell is indicated in the table. In the column “atoms(type)” the number of atoms of types 0 and 1 is listed. The column “type errors” indicates the number of atoms that were classified as the wrong atom type in the reconstruction. The column “cell errors” indicates the number of atoms that the algorithm placed in the wrong cell. We call a cell error an “off by one error” if the cell in the reconstruction is directly adjacent to the actual cell and an “off by $>$ 1 error” if this is not the case.

For the next set of tests, we decomposed Figure 6.3 into several slices and modified some slices to create additional test cases (see Figure 6.10). The slices each have their own characteristics. Some contain two atom types, others only one. Yet, we assume for the cases a, b, c, d, e, f and g that the slices contain two atom types, so that the algorithm must find out by itself if only one atom type occurs. The results are shown in Table 6.2. The test case

	atoms(type)	(r_0,ρ_0) (r_1,ρ_1)	fine cells per det.	runtime (min)	type errors	cell errors	
						off by one	off by > 1
Fig. 6.3	241(0) 8(1)	(5, 1) (4.5, 1.41)	2	26	0	6	0
		(7, 1) (6, 1.41)	1	78	0	6	0

Table 6.1: *Reconstruction results for the atom configuration in Figure 6.3, using different pairs of atom radii.*

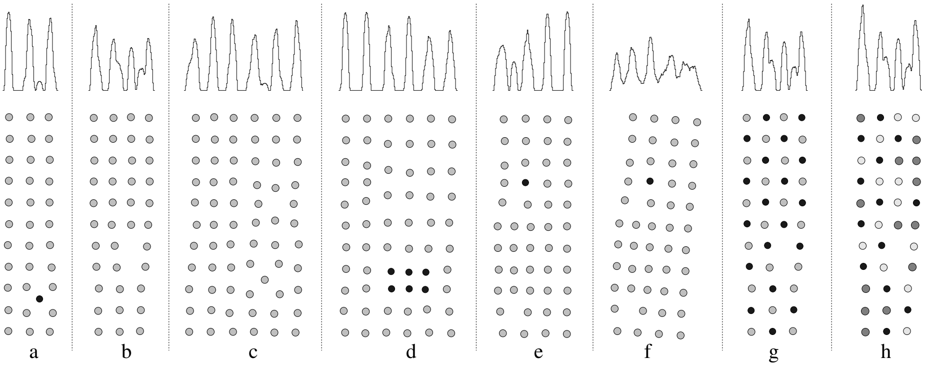


Figure 6.10: *Test configurations with their projections.*

c* is the same as test case c, except that we assume that the slice contains only one atom type. Therefore type errors cannot be made by the algorithm.

For reconstructing the slice in test case h, three atom types are used. For the values (r,ρ) of the three atom types we used $(5,1)$, $(4.5,1.41)$ and $(5.4,1.37)$ respectively.

6.5. Discussion

The experimental results show that our algorithm is able to reconstruct all test sets accurately when there is no noise. Noise is clearly a problem for our algorithm. When reconstructing atom configurations for which we know in advance that they contain only a single atom type, the tolerance for noise is much higher than for the case of multiple atom types. Even for a noise level of $\sigma = 0.05$ the reconstruction is still quite accurate, considering that the size of a fine grid cell is $1/10$th the size of an atom. When there is more than one atom type the runtime becomes prohibitively large for noisy data. For three atom types a noise level of $\sigma = 0.01$ already resulted in a runtime that was unacceptably large (not shown in the table, as we terminated the computation after one day). Our experiments suggest that for two atom types a noise level around $\sigma = 0.01$ still allows accurate reconstruction in reasonable time. We performed some additional experiments with projection data of thicker samples, containing longer atom columns. The results suggest that the runtime increases strongly when the column height increases.

	atoms(type)	σ	fine cells per det.	runtime (min)	type errors	cell errors	
						off by one	off by > 1
Fig. 6.10a	33(0) 1(1)	0	2	8	0	1	0
		0.01	1	15	0	5	0
Fig. 6.10b	39(0) 0(1)	0	2	17s	0	0	0
		0.01	1	13	0	6	0
Fig. 6.10c	66(0) 0(1)	0	2	2	0	0	0
		0.01	1	27	0	9	0
Fig. 6.10c*	66	0.03	1	26s	N/A	17	0
		0.05	1	2		21	2
Fig. 6.10d	56(0) 6(1)	0	2	15	0	4	0
		0.01	1	19	0	7	0
Fig. 6.10e	47(0) 1(1)	0	2	5	0	3	0
		0.01	1	172	5	9	1
Fig. 6.10f	47(0) 1(1)	0	2	17s	0	0	0
		0.01	1	91	0	4	0
Fig. 6.10g	20(0) 19(1)	0	2	1	0	0	0
		0.01	1	35	0	7	0
Fig. 6.10h	13(0) 13(1) 13(2)	0	1	187	0	0	0

Table 6.2: *Reconstruction results for the slices in Figure 6.10 using different noise levels.*

6.6. Conclusions

In this chapter we demonstrated that it is indeed possible to reconstruct the atom types and their approximate x-coordinates from the measured projection data, even in the presence of limited noise. For a noise level of $\sigma = 0.01$ we are able to obtain quite accurate reconstructions even when the sample contains two atom types. Our algorithm is not guaranteed to find the optimal solution of the reconstruction problem, yet it provides good reconstruction results on our set of characteristic test images. In future research we will address the second step in the reconstruction procedure: solving the 2D tomography problem that results after the individual projections have been processed by our algorithm.

Bibliography

[1] Gill, P. E., Murray, W., Wright, M.H.: *Practical Optimization*. Academic Press, London and New York (1981).

[2] Herman, G.T., Kuba, A., eds.: *Discrete tomography: foundations, algorithms and applications*. Birkhäuser, Boston (1999).

[3] Jinschek, J.R., Batenburg, K.J., Calderon, H.A., Van Dyck, D., Chen, F.-R., Kisielowski, C.: Prospects for bright field and dark field electron tomography on a discrete grid. *Microsc. Microanal.*, **10, suppl. 3** (2004).

Nederlandse samenvatting

Sinds het begin van de ontwikkeling van de Röntgenfotografie, aan het eind van de 19e eeuw, is het mogelijk om beelden te maken van het inwendige van mensen zonder te opereren. De mate waarin de Röntgenstraling geabsorbeerd wordt hangt af van het type weefsel. Hoe meer straling door een orgaan wordt geabsorbeerd, des te lichter is de Röntgenfoto op de overeenkomstige positie. Een belangrijk nadeel van Röntgenfoto's is dat het feitelijk *projectiebeelden* zijn. Twee organen die achter elkaar liggen, gezien in de richting van de Röntgenbundel, worden op elkaar "geprojecteerd", waardoor het lastig kan zijn om ze van elkaar te onderscheiden.

De opkomst van de computer in de jaren '70 maakte het mogelijk om op efficiënte wijze beelden te *berekenen* uit Röntgenfoto's die deze nadelen niet hebben, door middel van *tomografie*. Bij tomografie wordt niet één, maar een hele reeks van Röntgenfoto's gemaakt vanuit verschillende hoeken. Op deze manier wordt een groot aantal projectiebeelden verkregen (vaak enkele honderden). Uit al deze twee-dimensionale projectiebeelden wordt vervolgens met de computer een drie-dimensionaal beeld berekend van het inwendige van de patiënt.

Tomografie heeft niet alleen medische toepassingen. In de industrie is het vaak belangrijk om op een niet-destructieve manier in beeld te brengen hoe objecten er van binnen uitzien, bijvoorbeeld voor kwaliteitscontrole, of om kennis op te doen van producten van de concurrentie. Omdat de meeste industriële objecten niet beschadigen onder invloed van Röntgenstraling kunnen hiervoor Röntgenbundels met een hoge intensiteit worden gebruikt, zodat ook materialen die veel straling absorberen in beeld gebracht kunnen worden. Aan de andere kant wil men het aantal projectiebeelden graag zo klein mogelijk houden, om de totale opnametijd te beperken.

Een andere interessante toepassing van tomografie is te vinden in de electronenmicroscopie. Het doel is hier wederom het in beeld brengen van het inwendige van een preparaat, zonder in het preparaat te snijden. In plaats van een Röntgenbundel wordt in dit geval een electronenbundel gebruikt. Door het preparaat in de microscoop te draaien kunen projectiebeelden vanuit verschillende hoeken worden gemaakt. Vaak beschadigt de electronenbundel het preparaat, zodat slechts een klein aantal projectiebeelden kan worden gemaakt.

De meeste bestaande reconstructie-algoritmes voor tomografie geven onbevredigende resultaten als het aantal projectiebeelden zeer gering is (bijvoorbeeld minder dan 10). Een manier om het aantal projectiebeelden te beperken en toch goede resultaten te krijgen is het gebruik van *extra voorkennis* in het reconstructie-algoritme. In de industriële tomografie

0	0	0	0	0	1	1	1	3
0	0	1	0	0	0	1	1	3
0	0	0	1	0	0	1	0	2
0	0	1	1	1	0	1	0	4
0	1	1	1	1	1	1	0	6
0	1	0	1	0	1	0	0	3
0	1	1	0	0	1	1	0	4
1	1	0	0	0	0	1	0	3
1	4	4	4	2	4	7	2	

0	0	0	1	0	1	1	0	3
0	0	1	0	0	0	1	1	3
0	0	1	0	0	1	0	0	2
0	0	1	0	1	0	1	1	4
0	1	1	1	1	1	1	0	6
0	1	0	1	0	0	1	0	3
0	1	0	1	0	1	1	0	4
1	1	0	0	0	0	1	0	3
1	4	4	4	2	4	7	2	

Figure 6.11: *a. (links): Een binair beeld met de bijbehorende horizontale en verticale projecties. b. (rechts): Een tweede beeld met dezelfde projecties als het eerste beeld.*

is het bijvoorbeeld vaak van tevoren bekend uit welke materialen het object bestaat. Een goede reconstructie zal dus slechts enkele grijswaarden bevatten, overeenkomend met de verschillende materialen.

Het reconstrueren van beelden die slechts uit een klein aantal verschillende grijswaarden bevatten is het domein van de *discrete tomografie*, het vakgebied waar dit proefschrift over gaat. Het proefschrift gaat over het reconstrueren van beelden uit een klein aantal projecties. In het bijzonder gaat het over het reconstrueren van *binaire* beelden: beelden waarvoor elke pixel een waarde 0 of 1 heeft, oftewel zwart-wit plaatjes. Figuur 6.11a toont een voorbeeld van een binair beeld, met daarbij de horizontale en verticale *projecties*. De horizontale projectie kan uit het beeld worden berekend door in elke (horizontale) rij het aantal enen te tellen. Op dezelfde manier verkrijgen we de verticale projectie door in elke (verticale) kolom het aantal enen te tellen. Het aantal enen op een bepaalde lijn door het beeld noemen we een *lijnprojectie*. De verzameling van alle lijnprojecties in een bepaalde richting noemen we kortweg een *projectie*.

Eén van de belangrijkste problemen uit de discrete tomografie is het reconstrueren van afbeeldingen uit slechts *twee* projecties, zoals in Figuur 6.11a. Uit een gegeven aantal enen voor elke rij en kolom moet nu een beeld worden berekend dat de gegeven projecties heeft. Er bestaat niet altijd een oplossing van dit reconstructieprobleem. Als bijvoorbeeld de som van de lijnprojecties van de kolommen niet gelijk is aan de som van de lijnprojecties van de rijen, heeft het probleem zeker geen oplossing. De som van de lijnprojecties in een willekeurige richting is immers gelijk aan het totale aantal enen in het (onbekende) beeld.

Al in de jaren '50 publiceerde Ryser een zeer efficient algoritme om een binair beeld te vinden met voorgeschreven horizontale en verticale projecties. In strikt wiskundige zin heeft hij daarmee het reconstructieprobleem voor twee projecties volledig opgelost. In de praktijk kleeft er echter een groot nadeel aan het reconstrueren van beelden uit slechts twee projecties, en zonder het gebruik van extra voorkennis. Een illustratie hiervan is te zien in Figuur 6.11. Het beeld in Figuur 6.11b heeft dezelfde horizontale en verticale projecties als het beeld in Figuur 6.11a en toch zijn de beide beelden behoorlijk verschillend. De oplos-

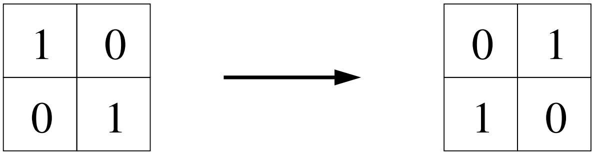


Figure 6.12: *Een switching component kan worden omgevormd in een tweede beeld met dezelfde projecties door de enen en nullen te verwisselen.*

sing van het reconstructieprobleem is in dit geval namelijk niet uniek. Dit hangt samen met het bestaan van zogenaamde “switching components”, zie Figuur 6.12. Als je in de switching component de enen en de nullen verwisselt, ontstaat er een nieuw beeld dat dezelfde projecties heeft. Voor praktische toepassingen van discrete tomografie is het van groot belang dat we er redelijk zeker van kunnen zijn dat de reconstructie daadwerkelijk lijkt op het object waar de projectiedata van afkomstig zijn. Denk bijvoorbeeld maar eens aan medische toepassingen, waar het essentieel is dat de reconstructie een nauwkeurig beeld geeft van de organen van de patiënt. Het is duidelijk dat we hier niet op zoek zijn naar een willekeurige reconstructie, maar naar een reconstructie die lijkt op het origineel.

Het *netwerk-stroom model*, dat de rode draad vormt in dit proefschrift, kan worden gebruikt om een reconstructie te vinden die de voorgeschreven projecties heeft, maar ook nog aan aanvullende voorwaarden voldoet. De theorie van *netwerk-stromen* is afkomstig uit de besliskunde. Bij netwerk-stromen gaat het om het vervoer van een bepaalde grootheid, bijvoorbeeld olie, door een netwerk van pijpleidingen die elk een beperkte capaciteit hebben. Eén van de bekendste problemen op dit gebied is het *maximale stroom* probleem, waarbij het erom gaat zoveel mogelijk olie van een bron naar een nieuwe bestemming te transporteren zonder de capaciteit van de leidingen te overschrijden. Netwerk-stroom problemen komen in allerlei varianten voor in de praktijk en zijn daarom in de afgelopen eeuw uitvoerig bestudeerd. In het bijzonder zijn er efficiënte methoden ontwikkeld om het maximale stroom probleem op te lossen.

Hoewel discrete tomografie en de theorie van netwerk-stromen op het eerste gezicht niet gerelateerd lijken, bestaat er een verassende relatie tussen beide gebieden. Het discrete tomografieprobleem voor twee projecties kan namelijk ook worden beschreven als een maximale stroom probleem in een netwerk, waarbij de capaciteiten van de pijpleidingen afhangen van de voorgeschreven lijnprojecties. Figuur 6.13a toont een (heel klein) binair tomografieprobleem met twee projecties, horizontaal en verticaal. Figuur 6.13b toont het bijbehorende netwerk. De “pijpleidingen” van het netwerk worden in de besliskunde meestal “takken” genoemd. De verbindingspunten tussen takken heten “knooppunten” of kortweg “knopen”. In de figuur is naast elke tak de capaciteit aangegeven. De drie takken van de “bron” (boven) naar de knopen in de laag daaronder hebben als capaciteiten de lijnprojecties van resp. de eerste, tweede en derde rij in het tomografieprobleem. Alle takken in de middelste laag hebben capaciteit 1. De drie takken naar de “bestemming” (onderaan) hebben als capaciteiten de verticale lijnprojecties van het tomografieprobleem. Om het tomografieprobleem op te lossen berekenen we een maximale stroom in dit netwerk. Het blijkt dat we dan uit de hoeveelheid stroom die door elk van de takken in de middelste laag loopt di-

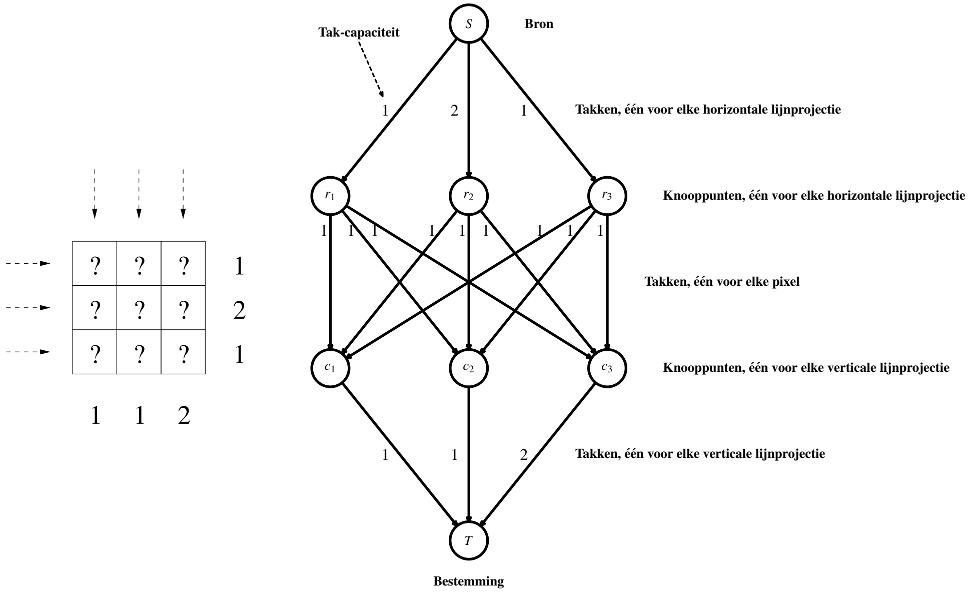

? ? ? 1
? ? ? 2
? ? ? 1
1 1 2
Tak-capaciteit
S
Bron
1
2
1
Takken, één voor elke horizontale lijnprojectie
r_1
r_2
r_3
Knooppunten, één voor elke horizontale lijnprojectie
1 1 1 1 1 1 1 1 1
Takken, één voor elke pixel
c_1
c_2
c_3
Knooppunten, één voor elke verticale lijnprojectie
1
1
2
Takken, één voor elke verticale lijnprojectie
T
Bestemming

Figure 6.13: *a. (links): Een* 3×3 *binair tomografieprobleem. b. (rechts): Het bijbehorende netwerk.*

rect een oplossing van het tomografieprobleem kunnen aflezen. Deze bijzondere eigenschap werd voor het eerst in de jaren ’50 beschreven door Gale.

Het probleem dat de oplossing niet uniek hoeft te zijn kunnen we aanpakken binnen het netwerk-stroom model door een *voorkeur* aan te geven voor bepaalde oplossingen. Aan elk van de takken in de middelste laag, die overeenkomen met de pixels in het tomografieprobleem, kennen we een *gewicht* toe. Hoe hoger dit gewicht voor een pixel, des te sterker is de voorkeur voor een oplossing waarbij die pixel de waarde 1 heeft. Het vinden van een reconstructie waarvoor het totale gewicht maximaal is (van alle pixels opgeteld) kan eveneens efficiënt worden opgelost met bestaande technieken uit de besliskunde. Met behulp van de gewichten kunnen we dus voorkennis over het onbekende beeld gebruiken bij het vinden van een reconstructie.

Dit proefschrift gaat hoofdzakelijk over nieuwe algoritmes voor discrete tomografie die gebaseerd zijn op het netwerk-stroom model. In Hoofdstuk 1 wordt het model geïntroduceerd en wordt bewezen dat met behulp van dit model een aantal discrete tomografieproblemen direct kunnen worden opgelost. Het eerste hoofdstuk blikt tevens vooruit op de onderwerpen die in de resterende hoofdstukken aan de orde komen. Eén van de belangrijkste toepassingen van discrete tomografie is het berekenen van drie-dimensionale reconstructies van *nanokristallen* met *atomaire resolutie*. Nanokristallen zijn zeer kleine kristallen die uit een paar honderd, of soms een paar duizend atomen bestaan. Nanokristallen hebben een *roosterstructuur*: de posities van de atomen zijn gerangschikt in een periodieke structuur. Met een electronen microscoop kunnen twee-dimensionale projectiebeelden worden gemaakt van het kristal. Door de juiste kijkrichtingen uit te kiezen kan men ervoor zorgen dat een aantal atomen, die op een lijn liggen in dezelfde richting als de kijkrichting, alle op hetzelfde punt van het projectiebeeld worden afgebeeld. Vervolgens kan met behulp van het projectiebeeld het aantal atomen in zo'n "geprojecteerde kolom" worden geteld. De lijnprojecties die op deze manier gemeten worden resulteren in een discreet tomografieprobleem. De laatste sectie van Hoofdstuk 1 introduceert deze toepassing.

In het tweede hoofdstuk wordt ingegaan op het reconstructieprobleem uit twee projecties, horizontaal en verticaal. In het bijzonder wordt er gekeken naar het geval dat er extra voorkennis beschikbaar is over het te reconstrueren beeld. Als we bijvoorbeeld al weten dat het te reconstrueren object convex is, d.w.z. dat het object geen "inhammen" of "gaten" bevat, kunnen we deze eigenschap gebruiken in het reconstructie-algoritme. Ook kan het voorkomen dat we voor een bepaalde toepassing van discrete tomografie al een grote verzameling beelden tot onze beschikking hebben die alle enigszins op elkaar lijken. In dat geval kan met behulp van statistische methoden voor elke mogelijke reconstructie bepaald worden hoe groot de *kans* is dat die reconstructie past bij het karakteristieke beeld van die toepassing.

Er zijn veel variaties denkbaar in het soort voorkennis dat gebruikt kan worden. Daarom vraagt het reconstructieprobleem om een flexibele aanpak, die voor al deze variaties bruikbaar is. De aanpak van Hoofdstuk 2 beschouwt het reconstructieprobleem als een *optimaliseringsprobleem.* Aan elk van de beelden die de beide voorgeschreven projecties hebben wordt een "waarde" toegekend, die aangeeft hoe goed het beeld overeenkomt met de beschikbare voorkennis. De beste reconstructie is het beeld waarvoor de waarde het hoogst is. Het vinden van de beste reconstructie is een moeilijk algoritmisch probleem. Daarom moeten we in het algemeen genoegen nemen met een benadering van de beste oplossing: een beeld waarvan de waarde hoger is dan van bijna alle andere mogelijke beelden. In Hoofdstuk 2 wordt een *evolutionair algoritme* beschreven om zo'n benadering te vinden. De term "evolutionair algoritme" is een verzamelnaam voor optimalisatie-algoritmes die gebaseerd zijn op ideeën uit de evolutietheorie. Het algoritme houdt voortdurend een verzameling "kandidaat-beelden" bij, die ook wel de "populatie" wordt genoemd. Nieuwe beelden worden gecreëerd, door beelden uit de huidige populatie licht te wijzigen (mutatie) of met elkaar te combineren (cross-over). Nadat op deze wijze een groep nieuwe beelden is gevormd wordt een aantal beelden die een relatief hoge waarde hebben toegevoegd aan de huidige populatie en beelden met een relatief lage waarde worden uit de populatie verwijderd (selectie). Op deze manier wordt de gemiddelde waarde van beelden in de populatie steeds hoger, tot de waardering afvlakt in de buurt van een maximum. Het evolutionaire algoritme uit Hoofdstuk 2 maakt telkens gebruik van het netwerk-stroom model om nieuwe beelden te berekenen die de beide voorgeschreven projecties hebben. Een nadeel van het evolutionaire algoritme is dat het berekenen van een goede reconstructie veel rekentijd vraagt. Voor beelden van meer dan 50×50 pixels wordt deze rekentijd zo lang dat het algoritme in de praktijk niet bruikbaar is.

In Hoofdstuk 3 wordt een ander algoritme beschreven dat ook gebruik maakt van het netwerk-stroom model. Dit keer worden tenminste drie projecties gebruikt. Het gebruik van meer projecties zorgt er over het algemeen voor dat het reconstructieprobleem minder oplossingen heeft, zodat we kunnen verwachten dat een reconstructie beter lijkt op het "onbekende originele beeld". Toch is ook wanneer bijvoorbeeld 4 of 5 projecties beschikbaar zijn het gebruik van extra voorkennis door het reconstructie-algoritme essentieel. In dit geval is de voorkennis die we gebruiken, dat het te reconstrueren beeld redelijk "glad" is. Dit betekent dat het beeld bestaat uit een betrekkelijk klein aantal samenhangende gebieden, die elk geheel wit of geheel zwart zijn. Het reconstructie-algoritme heeft een voorkeur voor gladde oplossingen, maar dit is geen harde eis: fijne details, zoals een enkele witte pixel die omgeven is door zwarte pixels, kunnen ook worden gereconstrueerd.

Het algoritme uit Hoofdstuk 3 berekent een reconstructie uit meer dan twee projecties door een reeks van eenvoudigere reconstructieproblemen op te lossen, elk met maar twee projecties. Een voorbeeld: als er vier projecties beschikbaar zijn, wordt eerst een beeld berekend dat goed overeenkomt met de eerste twee projecties. Vervolgens wordt een beeld berekend dat goed overeenkomt met de resterende twee projecties. Zoals we reeds eerder zagen heeft het reconstructieprobleem uit twee projecties over het algemeen een meerdere oplossingen. Het algoritme zoekt voor de tweede stap de reconstructie die zo goed mogelijk overeenkomt met het beeld dat in de eerste stap was berekend. Dit proces wordt herhaald met een nieuw tweetal projecties, bijvoorbeeld de eerste en de derde projectie, enzovoort. Als alle paren van projecties aan de beurt zijn geweest wordt weer met het eerste paar verder gegaan. Tijdens elke stap worden dus niet alleen twee projecties gebruikt, maar ook het beeld dat in de vorige stap is berekend, waarbij een ander tweetal projecties werd gebruikt. Zodoende worden uiteindelijk alle projecties verwerkt in de reconstructie. Hoofdstuk 3 bevat naast een beschrijving van dit algoritme ook diverse reconstructieresultaten van testbeelden. Het algoritme blijkt zeer effectief te zijn voor het reconstrueren van beelden die redelijk glad zijn. In tegenstelling tot het evolutionaire algoritme uit Hoofdstuk 2 kan men met dit algoritme ook grotere beelden (bijv. 256×256 pixels) in korte tijd reconstrueren.

De algoritmes uit het tweede en derde hoofdstuk zijn geschikt voor het reconstrueren van *twee-dimensionale* beelden. In de praktijk is het vaak belangrijk om *drie-dimensionale* beelden te reconstrueren, uit een aantal *twee-dimensionale* projecties. In de medische tomografie bijvoorbeeld, wil men vaak een drie-dimensionaal beeld van het inwendige van de patiënt reconstrueren uit een aantal twee-dimensionale Röntgenfotos die gemaakt zijn vanuit verschillende hoeken. Als alle projectierichtingen in één vlak liggen – dus als de beelden zijn opgenomen door de Röntgenbron en de camera stapsgewijs te draaien om een bepaalde as – is het mogelijk om een drie-dimensionale reconstructie te berekenen door het beeld simpelweg op te splitsen in twee-dimensionale plakjes die loodrecht staan op de rotatie-as, en elk van de plakjes apart te reconstrueren. Als de projectierichtingen niet in een vlak liggen is dit niet mogelijk. In Hoofdstuk 4 wordt een uitbreiding van het algoritme uit Hoofdstuk 3 geïntroduceerd die geschikt is voor het reconstrueren van drie-dimensionale beelden. De projectie-richtingen hoeven bij dit algoritme niet in een vlak te liggen.

Het reconstructieprobleem dat in Hoofdstuk 3 en 4 aan de orde komt heeft betrekking op beelden met een *roosterstructuur*. Dit betekent dat het beeld op een natuurlijke wijze kan worden opgedeeld in cellen (de pixels van het beeld) die ofwel waarde 0 ofwel waarde 1 kunnen hebben. Een lijnprojectie van het beeld wordt verkregen door de waarde van alle cellen waarvan het middelpunt precies op een bepaalde lijn ligt bij elkaar op te tellen. In sommige gevallen kan zo'n lijn "gaten" bevatten, zoals te zien is in Figuur 6.14a. Dit roostermodel is geschikt voor tomografie van nanokristallen: de nanokristallen hebben

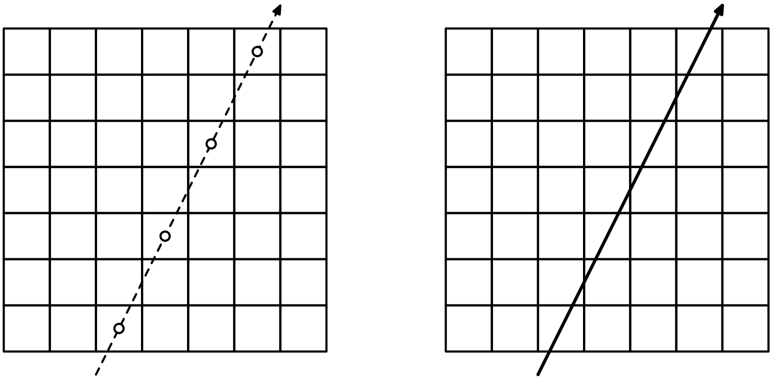


Figure 6.14: *a. (links): Als het beeld op een* rooster *gedefinieerd is, telt een pixel alleen mee in een lijnprojectie als de lijn door het middelpunt van de pixel gaat. b. (rechts): Bij beelden zonder roosterstructuur hangt de bijdrage van een pixel aan de lijnprojectie af van de lengte van het lijnstuk dat door die pixel loopt.*

een natuurlijke roosterstructuur en een lijnprojectie komt overeen met het aantal atomen in een geprojecteerde atoomkolom. Een lijnprojectie is hier dus altijd een geheel getal. Bij de meeste andere toepassingen van tomografie, bijvoorbeeld medische tomografie, is er geen sprake van zo'n natuurlijke roosterstructuur. Het beeld moet nog steeds worden onderverdeeld in pixels, om het in de computer te kunnen representeren, maar deze roosterstructuur is slechts kunstmatig. De bijdrage van een pixel aan een bepaalde lijnprojectie hangt nu af van de lengte van de doorsnede van de lijn met de betreffende pixel, zie Figuur 6.14b. Dit is een duidelijk verschil t.o.v. het roostergeval, waar een cel ofwel in zijn geheel tot een lijn behoort, ofwel helemaal niet. In Hoofdstuk 5 wordt een netwerk-stroom algoritme beschreven dat geschikt is voor het reconstrueren van beelden die geen natuurlijke roosterstructuur hebben. Behalve reconstructieresultaten voor een aantal gesimuleerde testbeelden worden ook resultaten beschreven voor "echte" meetdata. Met behulp van een micro-CT-scanner (die werkt op basis van Röntgenstraling) zijn projectiebeelden gemaakt van een ruwe diamant. Een plaatje van zo'n diamant kan men opvatten als een binair beeld, omdat de dichtheid binnenin de diamant constant is. De resultaten laten zien dat het algoritme voor objecten zonder natuurlijke roosterstructuur ook goede resultaten geeft bij het gebruik van echte meetdata.

Het laatste hoofdstuk van dit proefschrift, Hoofdstuk 6, gaat over het reconstrueren van nanokristallen met discrete tomografie, wat reeds eerder werd geïntroduceerd aan het eind van Hoofdstuk 1. In principe kunnen de algoritmes uit Hoofdstuk 2 t/m 4 direct worden gebruikt voor de reconstructie van nanokristallen met atomaire resolutie. In de praktijk is de roosterstructuur van dergelijke kristallen echter bijna nooit perfect. Verstoringen in de ideale roosterstructuur worden ook wel *defecten* genoemd en het in beeld brengen van deze verstoringen is zeer belangrijk in de materiaalwetenschappen. In Hoofdstuk 6 wordt een methode geïntroduceerd om discrete tomografie te kunnen gebruiken, ook als de roosterstructuur niet perfect is.

Curriculum Vitae

Kees Joost Batenburg werd geboren op 15 augustus 1980 te Rotterdam. In 1998 behaalde hij het VWO-diploma op het Farel College in Ridderkerk. In datzelfde jaar begon hij met de studies Wiskunde, Natuurkunde en Informatica aan de Universiteit Leiden. Dit resulteerde na een jaar in drie propedeuses. Vanaf het tweede jaar richtte hij zich exclusief op de studies Wiskunde en Informatica. Gedurende zijn middelbare schooltijd en de eerste jaren van zijn studie nam hij regelmatig deel aan internationale programmeerwedstrijden. In 2002 schreef hij onder begeleiding van prof. dr. R. Tijdeman zijn afstudeerscriptie in de Wiskunde, “Analysis and optimization of an algorithm for discrete tomography”, waarmee hij cum laude afstudeerde.

In september 2002 begon hij als Onderzoeker in Opleiding bij het Centrum voor Wiskunde en Informatica (CWI) in Amsterdam en bij de Universiteit Leiden. Zijn onderzoek werd gefinancierd door de Nederlandse Organisatie voor Wetenschappelijk Onderzoek (NWO). Op het CWI werd hij begeleid door dr. ir. H. J. J. te Riele en in Leiden door prof. dr. R. Tijdeman. In het eerste jaar van zijn promotieonderzoek voltooide hij zijn studie Informatica. Hij studeerde in 2003 cum laude af met de scriptie “An evolutionary algorithm for discrete tomography”, begeleid door dr. W. A. Kosters.

Naast de artikelen die zijn opgenomen in dit proefschrift schreef hij gedurende zijn promotieonderzoek diverse andere artikelen op het gebied van discrete tomografie. Hij gaf regelmatig voordrachten op internationale conferenties en ontwikkelde samenwerkingsverbanden met enkele buitenlandse onderzoeksgroepen. In het bijzonder richtte hij zich ook op toepassingen van discrete tomografie. In dit kader bezocht hij in 2006 gedurende twee maanden het Lawrence Berkely Lab in Californië, om te werken aan de toepassing van discrete tomografie in de electronen microscopie.

Vanaf 1 september 2006 is hij werkzaam als postdoc bij de Universiteit Antwerpen, in de groep van prof. dr. J. Sijbers.

Other publications

The author has published several articles on discrete tomography that are not part of this thesis. The following papers describe (new) reconstruction algorithms:

Batenburg, K.J.: A learning classifier approach to tomography. *Proc. of the 17th European Conference on Artifical Intelligence* (ECAI'06), IOS Press (2006).

Batenburg, K.J., Kosters, W.A.: A neural network approach to real-time discrete tomography. *Lecture Notes in Comput. Sci.*, **4040**, 389–403 (2006).

Batenburg, K.J., Kosters, W.A.: Neural networks for discrete tomography. *Proc. of BNAIC'05*, 21–27 (2005).

Batenburg, K.J., Kosters, W.A.: A discrete tomography approach to Japanese puzzles. *Proc. of BNAIC'04*, 243–250 (2004).

Batenburg, K.J.: Analysis and optimization of an algorithm for discrete tomography. *Electron. Notes Discrete Math.*, **12** (2003).

The following papers deal with applications of discrete tomography:

Batenburg, K.J., Jinschek, J.R., Calderon, H.A., Kisielowski, C.: Incorporating prior knowledge for atomic resolution discrete tomography. *Proc. of the 16th International Microscopy Congress*, Sapporo, Japan (2006).

Jinschek, J.R., Batenburg, K.J., Calderon, H.A., Kilaas, R., Radmilovic, V., Kisielowski, C.: Atomic resolution electron tomography based on discrete mathematics. *Proc. of the 16th International Microscopy Congress*, Sapporo, Japan (2006).

Batenburg, K.J., Kübel, C., Kaiser, U.: 3D imaging of nanomaterials by discrete tomography. *Proc. of Microsc. & Microanal.*, Chicago, USA (2006).

Batenburg, K.J., Sijbers, J.: Discrete tomography from micro-CT data: application to the mouse trabecular bone structure. *Proc. SPIE*, **6142** (2006).

Batenburg, K.J., Sijbers, J.: Trabecular bone reconstruction from micro-CT data using discrete tomography. *Proc. of SPS-DARTS'06* (2006).

Batenburg, K.J., Jinschek, J.R., Kisielowski, C.: Atomic resolution electron tomography on a discrete grid: atom count errors. *Microsc. Microanal.*, **11, suppl. 2** (2005).

Jinschek, J.R., Calderon, H.A., Batenburg, K.J., Radmilovic, V., Kisielowski, C.: Discrete tomography of Ga and InGa particles from HREM image simulation and exit wave reconstruction. *MRS Proc.*, **839**, 4.5.1–4.5.6 (2004).

Jinschek, J.R., Batenburg, K.J., Calderon, H.A., Van Dyck, D., Chen, F.-R., Kisielowski, C.: Prospects for bright field and dark field electron tomography on a discrete grid. *Microsc. Microanal.*, **10, suppl. 3** (2004).