Probability, Networks and Algorithms

Reconstructing binary images from discrete X-rays

K.J. Batenburg

REPORT PNA-E0418 NOVEMBER 2004

ISSN 1386-3711

Reconstructing binary images from discrete X-rays

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 reconstructions which resemble the original images very closely 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 with great accuracy.

1998 ACM Computing Classification System: I.4.5
Keywords and Phrases: Discrete tomography, image reconstruction, projections, network problems

Reconstructing binary images from discrete X-rays

Kees Joost Batenburg
Leiden University, P.O. Box 9512, 2300 RA Leiden, The Netherlands and
CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

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 reconstructions which resemble the original images very closely 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 with great accuracy.

Keywords: Discrete tomography, Image reconstruction (I.4.5), Projections (I.4.7.d), Network problems (G.2.2.d)

1. Introduction

Discrete tomography (DT) concerns 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 (white) and 1 (black). 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, 16, 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 have been proved to be 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, 5]. 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 [6, 10, 13]. In [6] the authors propose a relaxation of the original problem that can be solved efficiently by linear programming. In [10] several greedy heuristic algorithms are described for reconstructing binary images from more than two projections. The authors of [13] propose an iterative algorithm which starts at a real-valued solution and gradually moves toward a binary solution. Several enhancements on this approach are presented in [4].

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 neighborhoods 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 favor of the smoothness constraints.

In this paper 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 large class 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 present reconstruction results for 256×256 pixel 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 2 we introduce some notation and describe our reconstruction problem in a formal context. Section 3 forms the main part of this paper. We introduce our algorithm and describe each of its parts in detail. In Section 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 brief performance comparison between our algorithm and two alternative approaches.

2. Preliminaries

In this paper 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

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

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. For the sake of convenience 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 colors, *black* (1) and *white* (0).
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 $X_vF : \mathcal{S}_v \to \mathbb{N}_0$ 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 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 lines $v_1, v_2, \ldots, v_k$. More formally:

Problem 1 (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. Find a function $F \in \mathcal{F}$ such that $X_{v_i}F = \phi_i$
for $i = 1, \ldots, k$.

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. The experimental results in Section 4 provide valuable insights regarding the feasibility of computing a reconstruction that closely resembles the original image when the original image is relatively smooth.
Our algorithm is based on the fact that the problem of reconstructing binary images from two projections can be solved efficiently. Problem 1 is trivially equivalent with the following problem:

Problem 2 (Reformulated reconstruction problem).
Let $v_1, \ldots, v_k$ and $\phi_1, \ldots, \phi_k$ be as in Problem 1.
Find a function $F \in \mathcal{F}$ such that for $1 \leq i < j \leq k$: $X_{v_i}F = \phi_i$ and $X_{v_j}F = \phi_j$.

The formulation of Problem 2 emphasizes the fact the if F is a solution of the reconstruction problem, F must satisfy all pairs of prescribed projections simultaneously. In each iteration our algorithm takes one such 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. We use this distance for defining termination conditions for our algorithm, as it does not always converge to a perfect reconstruction.

3. Algorithm description

3.1 Overview

Before we delve into the details, we will first give a global description of our algorithm. The purpose of the algorithm is to solve the Reconstruction Problem 1 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.

Every iteration consists of choosing two lattice 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 lattice 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 quite efficiently. The weights are chosen in such a way that the new reconstruction is not very different from the previous reconstruction. In this way, the information gathered from previous reconstructions, using different pairs of projections, 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 4, this does not restrict the algorithm to the reconstruction of completely smooth images. It is merely used for guiding the algorithm into the right direction in the search space.

At the start of the algorithm there is no prior reconstruction. Instead, we solve a real-valued relaxation of the binary reconstruction problem, using all lattice directions, which provides a first approximate solution.

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 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.

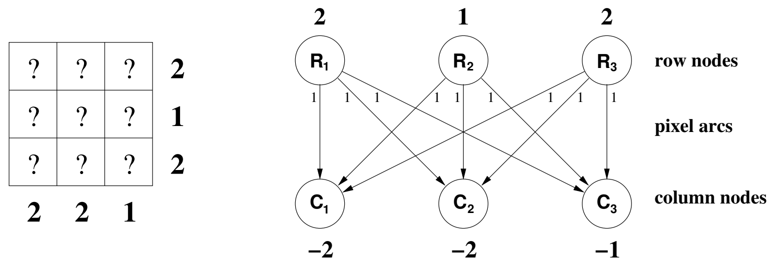


Figure 1: A 3×3 reconstruction problem and the corresponding network.

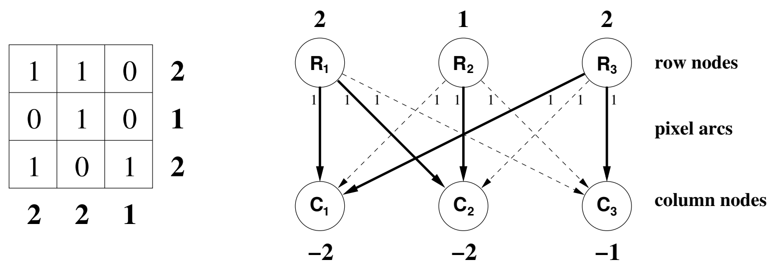


Figure 2: A solution of the transportation problem in G corresponds to a solution of the reconstruction problem.

We construct a capacitated network G, as shown in Figure 1. The *row nodes* R_1, R_2, R_3 correspond to the image rows, the *column nodes* C_1, C_2, C_3 correspond to the image columns. There is an arc between every pair of row and column nodes. Each arc (R_i, C_j) corresponds to a single pixel of the image (the cell that intersects with row i and column j). Therefore we will refer to these arcs as *pixel arcs.* All pixel arcs are assigned a capacity of 1. The row nodes act as *sources.* The linesum for each row determines the (nonnegative) excess at the corresponding row node. The column nodes act as *sinks.* Every column node has a nonpositive excess, which is the negated linesum of the corresponding column. We denote the flow through arc (R_i, C_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

$$\sum_{j=1}^{3} x_{ij} = r_i \quad \text{for } i = 1, 2, 3,$$

$$\sum_{i=1}^{3} x_{ij} = s_j \quad \text{for } j = 1, 2, 3,$$

$$0 \leq x_{ij} \leq 1 \quad \text{for } 1 \leq i, j \leq 3.$$

This transportation problem is a special case of the more general max-flow problem. It is a well known fact that there exists a solution for which all the arc flows x_{ij} are integral, i.e., 0 or 1. Efficient methods for finding such a 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 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. The resulting bipartite graph is not necessarily complete: the nodes that correspond to two lattice lines are connected by an arc if and only if they 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 (R_i, C_j). We will refer to these weights as *pixel weights.* Instead of finding any solution of the transportation problem in G, we now try to find a solution that maximizes the total weight:

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

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

In the same way as the transportation problem is a special case of the general max-flow problem, 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.

We will now illustrate the idea by an example. Suppose that we have an image $Y \in \mathcal{F}$ and we want the resulting reconstruction to have the same value as $Y = (y_{ij})$ in as many lattice points as possible, while completely satisfying the prescribed horizontal and vertical projections. Put

$$w_{ij} = y_{ij} \qquad 1 \le i \le m, 1 \le j \le n.$$

In this way, we express the preference for sending positive flow along pixel-arcs for which the corresponding pixel-value in Y is 1. It is not difficult to see that solving the corresponding min-cost max-flow problem results in the two-projection solution which has the least pixel-differences with Y.

3.3 An iterative network flow algorithm

Unfortunately, the problem of reconstructing binary images from more than two projections cannot be described as a single network flow problem. Every pair of projection directions has a corresponding network. As was emphasized by Problem 2, a binary image is a solution to the reconstruction problem if and only if the corresponding flow in each of those networks is feasible and maximal. Because the reconstruction problem is NP-hard for more than two projections the equivalent problem of simultaneously solving the network flow problems is also intractable. In [20] a push-relabel algorithm is described for finding an approximate solution, using the network flow model.

We propose a different, iterative approach. 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 that corresponds to the first two directions. By solving the minimum cost network flow problem 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. The procedure is repeated until some termination condition is satisfied. Figure 3 shows the basic steps of the algorithm.

At a very basic level, a similar approach was suggested in [11], but as far as we know it has never been further explored. The authors of [11] suggested an iterative algorithm in which only the value of a pixel in the previous reconstruction is used to determine the pixel-weight for that pixel in the new iteration. When only information about the pixel itself is used, the algorithm usually does not converge and does not yield good reconstructions. When neighboring pixels are also taken into account, however, the performance of the algorithm turns out to improve dramatically. We use the neighboring pixels in combination with a smoothness assumption, which favors neighborhoods of pixels having the same value over random neighborhoods.

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.

3.4 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. A different approach is to use another reconstruction algorithm, for example one of the algorithms from [10] and use the resulting image for determining the weight distribution. We

```
Compute the real-valued start solution $X^* = (x^*_{ij})$ (see Section 3.4);
Compute the binary start solution $X^0$ by solving a min-cost
max-flow problem for directions $d_1$ and $d_2$, 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 3.6;
    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 3.5;
end
```

Figure 3: Basic steps of the algorithm.

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

$$Bx = b \tag{1}$$

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 (1) has a binary solution, $\tilde{x}$. Let $\hat{x}$ be any binary solution of (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 (1) has a binary solution $\tilde{x}$. Let $\bar{x}$ be an integral solution of (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 the only integers that are not less than their squares. Observation 2 shows that the binary solutions of (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 (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 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.

3.5 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 neighborhood. The basic idea is that a black pixel that is surrounded mainly by black pixels is more likely to remain black in the next reconstruction than a black pixel that is surrounded by white pixels.

Let $F \in \mathfrak{F}$ be the reconstructed image from the previous iteration. As a neighborhood of the pixel $p = (x_p, y_p)$ we choose a square centered in (x_p, y_p). The reason for preferring a square neighborhood 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 *neighborhood radius*. Put

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

In case p is near the boundary of the image, the neighborhood N may contain fewer than $(2r+1)^2$ pixels. Let s be the number of pixels in N that does not have the same color as p, and let f be the relative fraction of such pixels:

$$s = |\, \{\, (x, y) \in N : \quad F(x, y) \neq F(x_p, y_p) \,\} \,|,$$
$$f = \frac{s}{|N|}.$$

Put $d = F(x_p, y_p) - \frac{1}{2}$. We now choose the pixel-weight w_{x_p,y_p} of p:

$$w_{x_p,y_p} = \begin{cases} F(x_p, y_p) + 8d & (f = 0) \\ F(x_p, y_p) + (3 - 4f)d & (0 < f < 0.35) \\ F(x_p, y_p) & (f \ge 0.35). \end{cases}$$

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 neighborhood of p has the same color as p, this preference is very strong. If the neighborhood contains a few pixels having a different color, the preference is smaller. If there are many pixels in the neighborhood that have a different color, the preference is even smaller. The chosen pixel weights are somewhat arbitrary. Several alternative expressions performed equally well, but we used this weight function for all our experimental results. The neighborhood 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 neighborhood radius leads to a strong smoothing effect. After 50 iterations, the neighborhood radius is set to 1. From this point on, the fine details in the image are gradually reconstructed.

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. For example, in the case of four directions – horizontal, vertical, diagonal and anti-diagonal – the algorithm performs much better when all 6 pairs of directions are used than when only the pairs horizontal/vertical and diagonal/anti-diagonal are used. Tables 1 and 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.

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

Table 1: Projection-pair order for four projections

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 2: Projection-pair order for 5 projections

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 4 projections, for example, using the heuristic would result in using only 2 direction pairs, alternating between those pairs. Such cycling is much less likely to occur when the number of projections is larger.

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 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. Termination after the maximum number of iterations usually means that the algorithm did not find an accurate reconstruction.

4. Results

4.1 Experimental setup

We have implemented our algorithm in C++. We use the *CS2* network flow library by Andrew Goldberg for solving the minimum cost 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 true physical object. Although in practical situations the "original" image is unknown, we can compare the original image to 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. In this paper, we do not focus on which set of directions is best for our algorithm. We use a fixed set of directions instead. Put

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

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

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 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 4b shows a much more complicated example. The object contains many cavities of various sizes and has a very complex boundary. Some of the white holes inside the black 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 4c is even more complex. It requires 8 projections to be reconstructed perfectly.

When the image contains no structure at all, our algorithm performs very badly. Figure 4d 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.

The image in Figure 4e has more structure, but it contains no large black areas. Still, the image is smooth enough to be reconstructed perfectly from only 5 projections. This example also illustrates

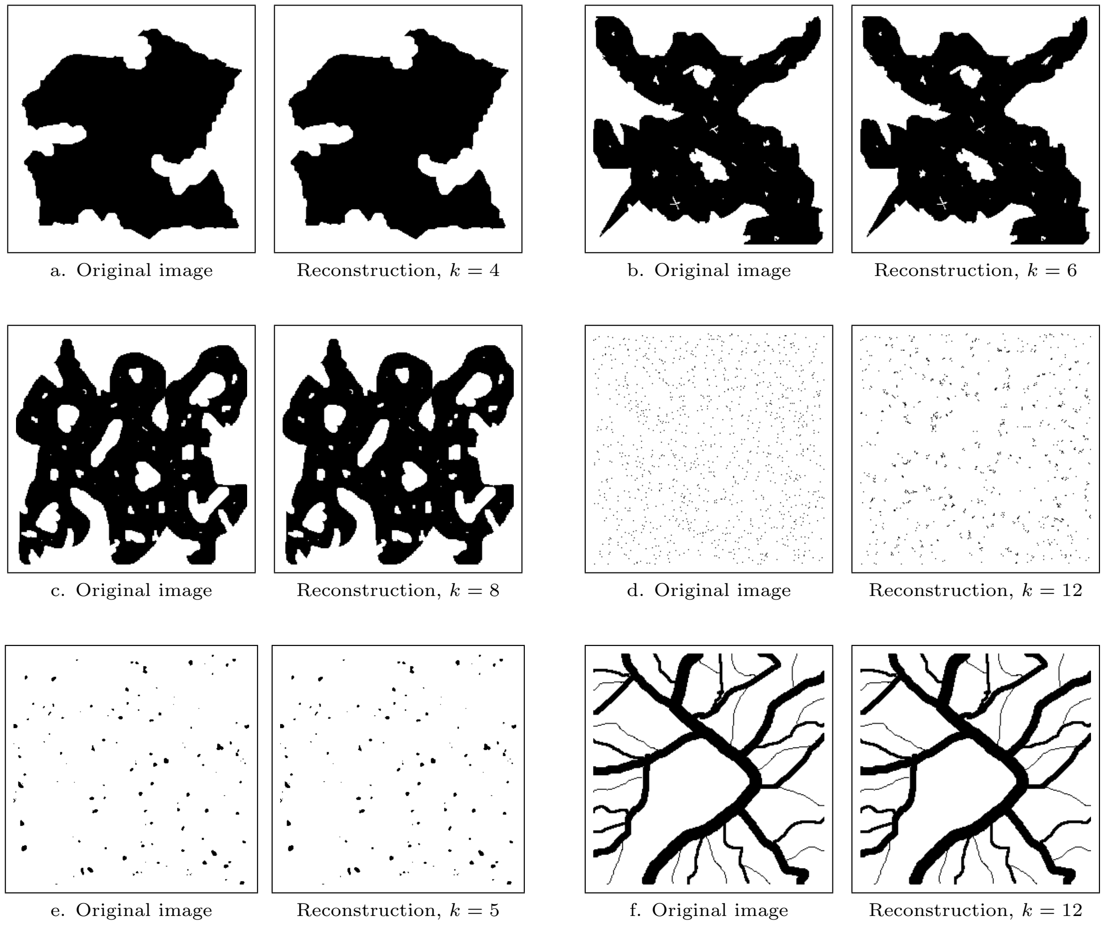


Figure 4: Six original images and their reconstructions.

that very fine, non-smooth details can be reconstructed by our algorithm, as long as the entire image is sufficiently smooth.

Figure 4f shows a vascular system, containing several very thin vessels. Our algorithm can reconstruct the original image perfectly from 12 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.

Although the examples in Figure 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 converges faster to an accurate solution. Figure 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.

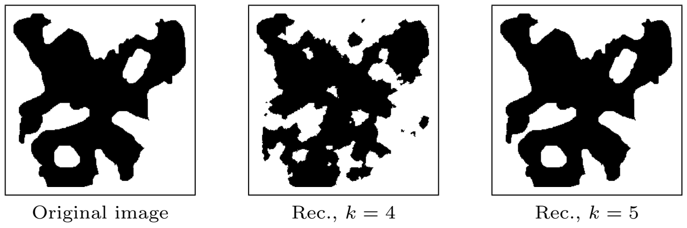


Figure 5: Using too few projections results in very bad reconstructions.

4.3 Random polygons

For our first class of test images, we implemented a program which generates images that consist of a number of black polygons on a white 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 6 shows several example images that were generated by the program, using two different pairs of parameters (n, p).

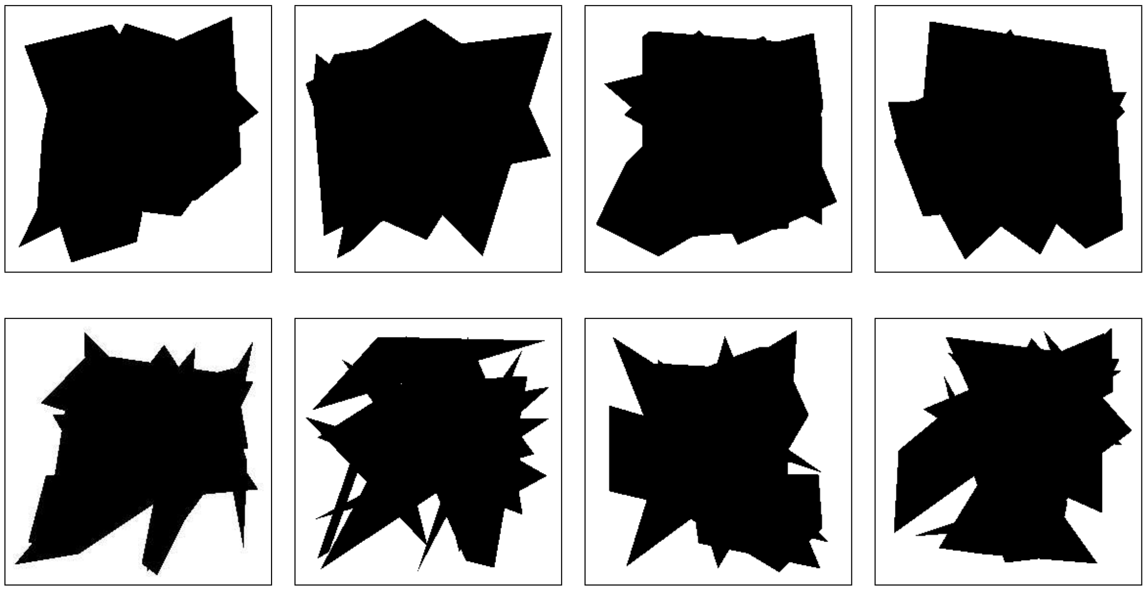


Figure 6: Test images generated by the polygon program. Top row: $n = 5, p = 8$. Bottom row: $n = 12, p = 4$.

For both these pairs (n, p) we performed 200 test runs, using 3, 4 and 5 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 succesful. By a *succesful 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.

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
12	4	3	195	0	51	2428	119	15
		3*			50	2408	118	15
		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: Experimental results for the “random polygons” test cases

Table 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 succesful reconstructions. Our algorithm typically requires far fewer iterations for successful reconstructions compared to unsuccessful reconstructions.

When using our definition of a succesful 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 succesful 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.

4.4 Random ellipses

For the second class of test images we implemented a program which generates images that consist of a number of black ellipses on a white 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. The program generates a random image containing the desired number of ellipses. For each ellips, 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 ellips is rotated. Put $a = \frac{1}{r_x^2}, b = \frac{1}{r_y^2}$. The equation describing the interior of the ellips becomes

$$\begin{aligned}&(a\cos^2\phi + b\sin^2\phi)(x - x_c)^2 + \\ &(a\sin^2\phi + b\cos^2\phi)(y - y_c)^2 + \\ &2(b - a)\cos\phi\sin\phi(x - x_c)(y - y_c) \leq 1\end{aligned}$$

All pixels (x, y) which satisfy this equation are colored black. While generating ellipses, the ellipses that have already been drawn are not taken into account in any way. Therefore it may happen that a previously drawn ellips is completely covered by a new one. Figure 7 shows five examples of images that were generated by this program, each using different parameter settings.

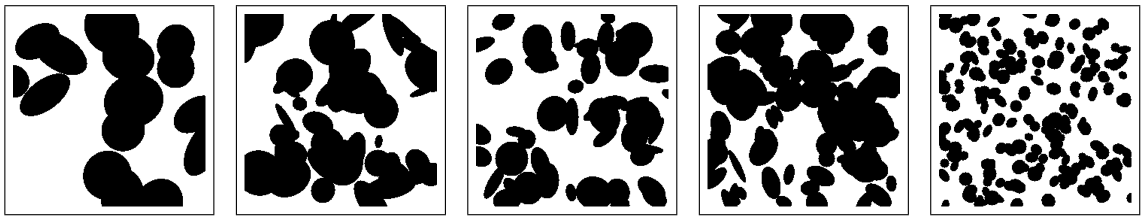

a. $(15, 20, 40)$ b. $(50, 5, 35)$ c. $(50, 5, 25)$ d. $(100, 5, 25)$ e. $(200, 5, 10)$

Figure 7: Five images that were generated by the ellips program, each using different parameters (n, r_{min}, r_{max}).

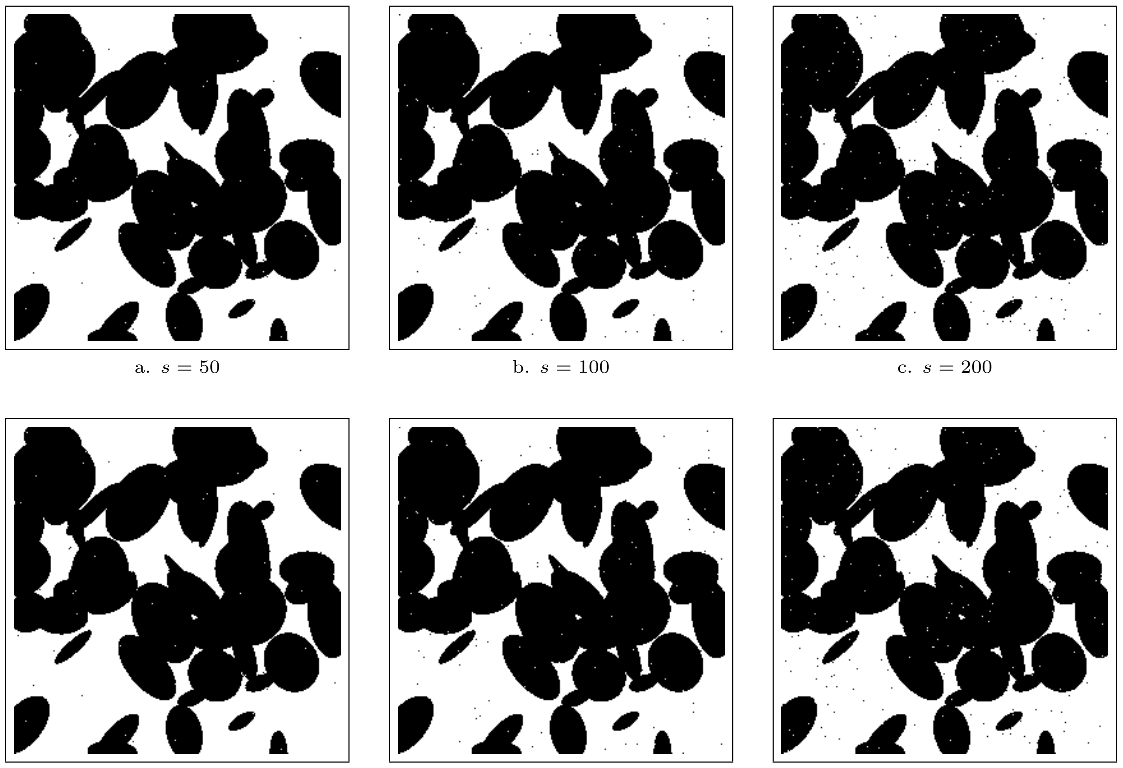
The images generated by the program are all very smooth: they contain large white and black areas. As Figure 7 shows, by adjusting the program parameters we can sample many classes of images, each having their own characteristics. For each of the parameter settings shown in Figure 7 we performed 200 test runs. All test images are of size 256×256. The results are shown in Table 4.

N	r_{min}	r_{max}	k	#suc.	#perf.	proj.err.	pixel err.	#iter.	time
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

Table 4: Experimental results for the “ellips” test cases

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 of size 256 × 256 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 subsequently inverting a set of s random pixels, 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 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.



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

Reconstruction, $k = 9$ Reconstruction, $k = 11$ Reconstruction, $k = 12$

Figure 8: Top: polluted versions of an image generated by the ellips program. Bottom: corresponding reconstructions of the polluted images.

4.6 Noisy projection data

In the previous section we considered noisy images for which the projection data is known exactly. We now turn to the case where the projection data itself is noisy, which is practically more realistic.

The network flow approach from Section 3.2 cannot be used directly when the projection data is not perfect, as the transportation problem that corresponds to any two-projection subproblem may not have an exact solution. Instead of the transportation problem, we need to use a slightly modified network flow problem, see Figure 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 row node R_i, there are two arcs from S to R_i. The first arc has capacity r_i (the prescribed row projection) and cost 0. The

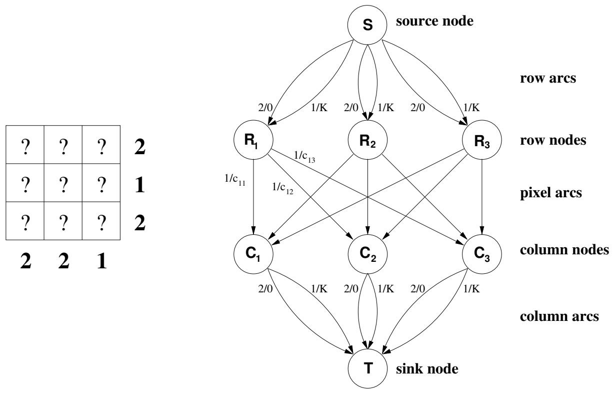


Figure 9: Network for the case of noisy projection data.

second arc, which we call the *overflow arc*, has capacity $n - r_i$ and cost K, where K is a very large integer. The set of arcs from the column nodes to the sink node has a similar structure, where the overflow arc for column j has capacity $m - s_j$. The pixel arcs all have capacity one and arc (R_i, C_j) has cost $c_{ij} = -w_{ij}$, where w_{ij} is the pixel-weight of the pixel in row i and column j.

The amount F of flow through the network, which equals the number of ones (black 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 black pixels is the same for all pairs of projections.

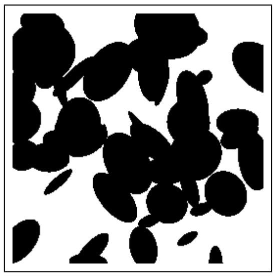
In each iteration of the algorithm the network that corresponds to the selected pair of directions is constructed and a flow of F units from S to T is computed that has minimal cost. Exceeding a prescribed linesum results in a high penalty cost of K per flow unit, but is allowed. Clearly, if $F \leq mn$, 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.

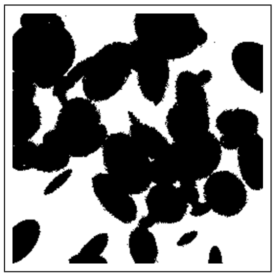
To generate noisy projection data, we start with perfect projection data and generate for each linesum v a random sample r from a normal distribution with average $\mu = 1$ and variance σ^2. The original linesum is replaced by rv. This model seems reasonable, since in many practical measurements the noise level increases with the amount of material (black pixels) that an X-ray passes. Figure 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 from 12 projections reveals almost all features of the original image.

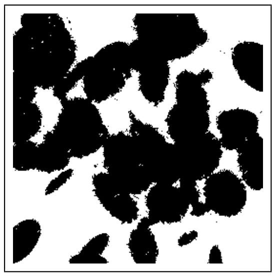
4.7 Comparison with alternative approaches

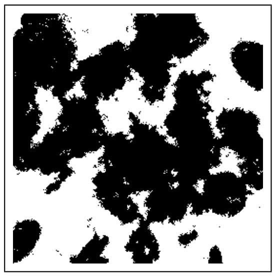
As there are hardly any test images for discrete tomography available from the literature, it is very difficult to compare our results to those of other authors. In addition, there are few papers which describe the reconstruction of images as large as 256×256 pixels. In this section we compare the reconstruction results of our algorithm to the results of two alternative algorithms for a small set of test images.

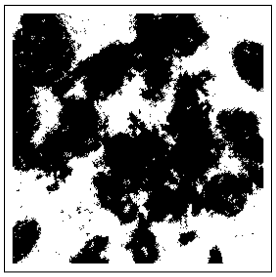
In their paper "The discrete Radon transform and its approximate inversion via linear programming" [6], Fishburn et al. describe a relaxation of the Reconstruction Problem (Problem 1) that can be solved by linear programming. The authors present reconstruction results for three small test images (around 40×40 pixels). The resulting reconstruction is real-valued, all pixel values are between 0 and 1. We implemented this approach using the GNU Linear Programming Kit. As a post-processing step, all pixel values are rounded to obtain a binary solution.

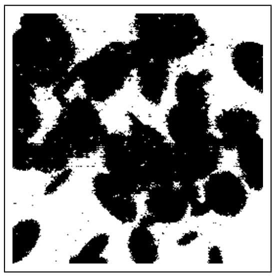

a. Original image

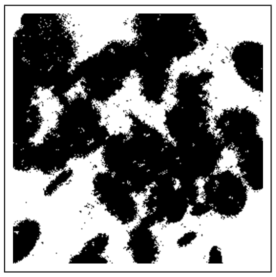

b. Rec., $k = 8, \sigma = 0.01$


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., $k = 12, \sigma = 0.05$

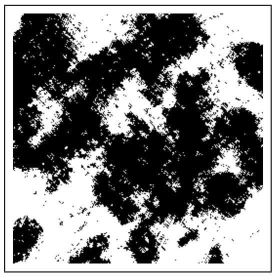

h. Rec., $k = 12, \sigma = 0.1$

Figure 10: Reconstruction results for noisy projection data.

In the paper “Approximating binary images by discrete X-rays” [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 the two alternative approaches do not incorporate smoothness assumptions. Figure 11 shows the reconstruction results for four test images. The test images all have a size of 128×128 pixels, because the running time of the linear programming approach becomes prohibitively large for larger images. The figure indicates the number of projections that was used for each test image and the running times for each of the algorithms. Note that the running times depend on the actual implementation. Our implementation of the two alternative approaches could probably be optimized more.

The reconstruction results indicate that the reconstruction quality of our algorithm is far better than for the alternative approaches when the image is relatively smooth. Even for the vessel-like structure, which contains many thin lines our algorithm computes a very accurate reconstruction. The algorithm from Gritzmann et al. is the fastest, but it results in a poor reconstruction quality. Our algorithm is not much slower and is even very fast compared to the approach of Fishburn et al.

5. Discussion

The experimental results show clearly that for a broad spectrum of images, our algorithm can compute high quality reconstructions from a small number of projections. This gives rise to great 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 pessimistic. We have shown that rather soft smoothness assumptions are already sufficient to change the reconstructibility properties dramatically.

Although discrete tomography has a substantial number of potential applications, it has never been

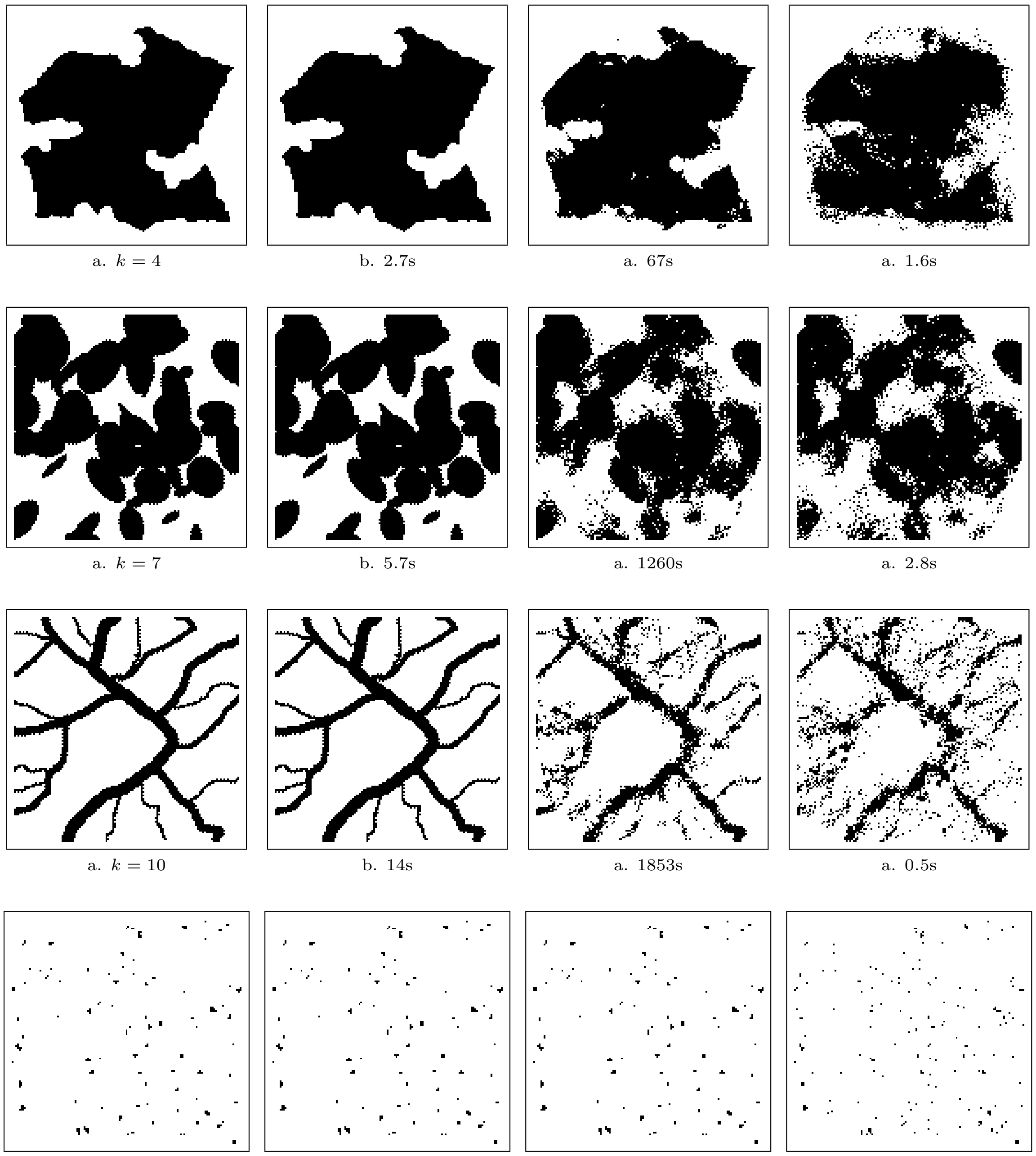


Figure 11: Reconstruction results for four test images. a. Original image; b. result of our algorithm; c. result of Fishburn et al. [6]; d. result of Gritzmann et al. [10].

used in large practical areas. We think that our soft smoothness assumptions are satisfied in many cases that occur in practice. 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, so that tiny, non-smooth details can also be reconstructed.

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, the modelling of images as plane sets is more appropriate (see, e.g., [18], [17]). In medical imaging, for example, the measured linesums are actually line integrals of a certain density function on the plane. The basic principles of our algorithm can be generalized to cover a much wider range of discrete tomography models. We intend to report on several such generalizations in future publications.

For each of the image classes that we used for testing, the results showed a clear and often also sharp 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. Although their worst case running time is not linear in the input size, we observed almost linear behavior when increasing the image size. 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 4.6 our algorithm can also be used in the case of noisy projection data. It is hard to make a good comparison between our algorithm and alternative approaches, because few good alternatives are available. The results in Section 4.7 indicate that for smooth images the reconstruction quality of our algorithm is much better than for the two alternative approaches that we tested.

6. Conclusion

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 utilizes 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 brief comparison between our algorithm and two alternative approaches showed that for images which are relatively smooth the reconstruction quality of our algorithm is far better than for the other two algorithms. Future work will include the exploration of several possible generalizations and methods to lower the running time of the algorithm.

References

1. R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network flows: theory, algorithms, and applications.* Prentice-Hall, Inc., 1993.
2. R.P. Anstee. The network flows approach for matrices with given row and column sums. *Discrete Mathematics*, 44:125–138, 1983.
3. E. Barcucci, A. Del Lungo, M. Nivat, and R. Pinzani. Reconstructing convex polyominoes from horizontal and vertical projections. *Theoretical Computer Science*, 155:321–347, 1996.
4. K.J. Batenburg. Analysis and optimization of an algorithm for discrete tomography. In V. Di Ges A. Del Lungo and A. Kuba, editors, *Electronic Notes in Discrete Mathematics*, volume 12. Elsevier, 2003.
5. S. Brunetti, A. Del Lungo, F. Del Ristoro, A. Kuba, and M. Nivat. Reconstruction of 4- and 8-connected convex discrete sets from row and column projections. *Linear Algebra and its Applications*, 339:37–57, 2001.
6. P. Fishburn, P. Schwander, L. Shepp, and R. Vanderbei. The discrete Radon transform and its approximate inversion via linear programming. *Discrete Applied Mathematics*, 75:39–61, 1997.
7. D. Gale. A Theorem on Flows in Networks. *Pacific Journal of Mathematics*, 7:1073–1082, 1957.
8. R.J. Gardner, P. Gritzmann, and D. Prangenberg. On the computational complexity of reconstructing lattice sets from their X-rays. *Discrete Mathematics*, 202:45–71, 1999.
9. A.V. Goldberg. An Efficient Implementation of a Scaling Minimum-Cost Flow Algorithm. *Journal of Algorithms*, 22:1–29, 1997.
10. P. Gritzmann, S. de Vries, and M. Wiegelmann. Approximating binary images from discrete X-rays. *SIAM Journal on Optimization*, 11:522–546, 2000.
11. P. Gritzmann, D. Prangenberg, S. de Vries, and M. Wiegelmann. Success and Failure of Certain Reconstruction and Uniqueness Algorithms in Discrete Tomography. *International Journal of Imaging Systems and Technology*, 9:101–109, 1998.
12. L. Hajdu and R. Tijdeman. Algebraic aspects of discrete tomography. *Journal für die reine und angewandte Mathematik*, 534:119–128, 2001.
13. L. Hajdu and R. Tijdeman. An algorithm for discrete tomography. *Linear Algebra and its Applications*, 339:147–169, 2001.

14. G.T. Herman and A. Kuba, editors. *Discrete Tomography: Foundations, Algorithms and Applications*. Birkhäuser, Boston, 1999.
15. J.R. Jinschek, K.J. Batenburg, H. Calderon, D. Van Dyck, F.-R. Chen, and Ch. Kisielowski. Prospects for bright field and dark field electron tomography on a discrete grid. Microscopy and Microanalysis, Vol. 10 Suppl. 3, Cambridge Journals Online, 2004.
16. C. Kisielowski, P. Schwander, F. Baumann, M. Seibt, Y. Kim, and A. Ourmazd. An approach to quantitative high-resolution transmission electron microscopy of crystalline materials. *Ultramicroscopy*, 58:131–155, 1995.
17. A. Kuba. Reconstruction of measurable plane sets from their two projections taken in arbitrary directions. *Inverse Problems*, 4:513–527, 1988.
18. J.L. Prince and A.S. Willsky. Reconstructing Convex Sets from Support Line Measurements. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(4), 1990.
19. H.J. Ryser. Combinatorial properties of matrices of zeros and ones. *Canadian Journal of Mathematics*, 9:371–377, 1957.
20. P.M. Salzberg, P.I. Rivera-Vega, and A. Rodríguez. Network Flow Model for Binary Tomography on Lattices. *International Journal of Imaging Systems and Technology*, 9:147–154, 1998.
21. P. Schwander, C. Kisielowski, F. Baumann, Y. Kim, and A. Ourmazd. Mapping projected potential, interfacial roughness, and composition in general crystalline solids by quantitative transmission electron microscopy. *Physical Review Letters*, 71:4150–4153, 1993.
22. C.H. Slump and J.J. Gerbrands. A Network Flow Approach to Reconstruction of the Left Ventricle from Two Projections. *Computer Graphics and Image Processing*, 18:18–36, 1982.
23. K. Tanabe. Projection method for solving a singular system. *Numerical Mathematics*, 17:203–214, 1971.
24. B. Wang and F. Zhang. On the precise number of $(0,1)$-matrices in $\mathcal{A}(R,S)$. *Discrete Mathematics*, 187:211–220, 1998.