

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

Kees Joost Batenburg

Leiden University and CWI, The Netherlands

`kbatenbu@math.leidenuniv.nl`

**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.

**Keywords:** Discrete tomography, Image reconstruction, Network flow problems

## 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 [22] and Gale [9]

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 [19–21, 24]. 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 [25]) 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 [12] 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 [27]. 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, 6]. 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 [7, 14, 17, 28]. In [7] the authors propose a relaxation of the original problem that can be solved efficiently by linear programming. In [28] the approach from [7] is extended to include a smoothness prior. Several greedy heuristic algorithms for reconstructing binary images from more than two projections are described in [14]. The authors of [17] 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 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 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 4.6. 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 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 performance comparison between our algorithm and two alternative approaches.

## 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. For the case  $d = 3$  we refer to [4], which describes how the approach from the present paper can be generalized.

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  $A \rightarrow \{0, 1\}$ . 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* (1) and *white* (0).

We call a vector  $v = (a, b) \in \mathbb{Z}^2 \setminus \{(0, 0)\}$  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  $X_v F : \mathcal{S}_v \rightarrow \mathbb{N}_0$  defined by

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

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 1** (Reconstruction problem).

Let  $k \geq 1$  and  $v_1, \dots, v_k$  be given distinct lattice directions. Let  $\phi_i : \mathcal{S}_{v_i} \rightarrow \mathbb{N}_0$  ( $i = 1, \dots, k$ ) 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  $\phi_i$  all have finite domains, we sometimes consider  $\phi_i$  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 [16]. 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  $|\cdot|$  denotes the Euclidean norm.

### 3 Algorithm description

#### 3.1 Overview

The purpose of our algorithm is to solve the Reconstruction Problem (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.

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 4, this does not restrict the algorithm to the reconstruction of completely smooth images.

### 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, 25]. 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  $3 \times 3$  binary image  $F$  are given.

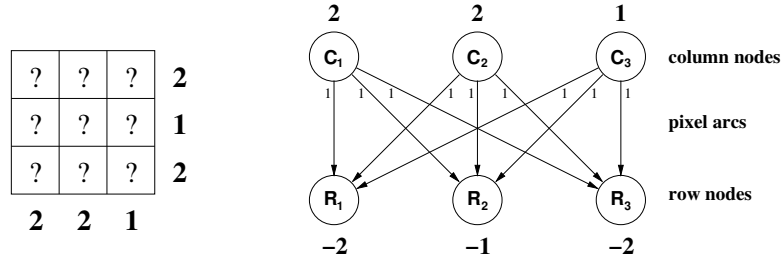


Figure 1: A  $3 \times 3$  reconstruction problem and the corresponding network. Arc capacities are indicated along the arcs of the network.

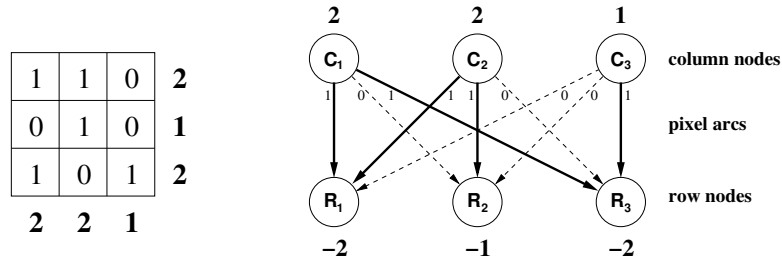


Figure 2: A solution of the transportation problem in  $G$  corresponds to a solution of the reconstruction problem. The flow through each arc is indicated in the network.

We construct a capacitated network  $G$ , as shown in Figure 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  $(C_i, R_j)$  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, which is the negated linesum of the corresponding row. 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 the transportation problem has a solution, then there exists a solution for which all the arc flows  $x_{ij}$  are integral, i.e., 0 or 1 (see, e.g., Theorem 6.5 of [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 flow through each arc (0 or 1) is indicated along the arc.

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 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

```

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  $v_1$  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 3.6;
    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.

to find a solution of the transportation problem that maximizes the total weight:

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

*such that  $X = (x_{ij})$  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 [25] for reconstructing an image of the left ventricle using a priori information.

### 3.3 An iterative network flow algorithm

Figure 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 [15], but as far as we know it has never been

further explored. It does not use a smoothness assumption, which plays an 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.

### 3.4 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

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

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 [16]:

**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  $b_1, \dots, b_t$  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  $b_1, \dots, b_t$  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 [26] 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 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 as  $p$ , and let  $f_p$  be the relative

fraction of such pixels:

$$s_p = |\{(x, y) \in N_p : F(x, y) = F(x_p, y_p)\}|,$$

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

Let  $g : [0, 1] \rightarrow (0, \infty)$  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  $g(f) = 1$  for all  $f \in [0, 1]$ , 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 4 that incorporating a preference for local smoothness in the local weight function results in (empirically) 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 4 we used the following local weight function:

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

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 4 we use  $L = 10000$ .

### 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.

Table 1: Projection-pair order for four projections

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

Table 2: Projection-pair order for five projections

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

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.

### 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 [23] provides a complexity survey for 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 [8] 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 3.5, the computation of pixel weights can be performed in  $O(mn)$  time.

Our algorithm is always terminated after at most  $U$  iterations (see Section 3.7). 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 at least necessary to consider only a very narrow class of images, instead of the wide range that we consider in this paper.

## 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 min-cost max-flow problems (see [13]). 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 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. We use a fixed set of directions and do not focus here on which set of directions is best for our algorithm. Put

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

When reconstructing images from  $k \leq 16$  projections, we use the set  $D_k = \{v_1, \dots, v_k\}$  of lattice directions. In Section 4.3 we make one exception to this rule, as we consider two different sets of four lattice 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

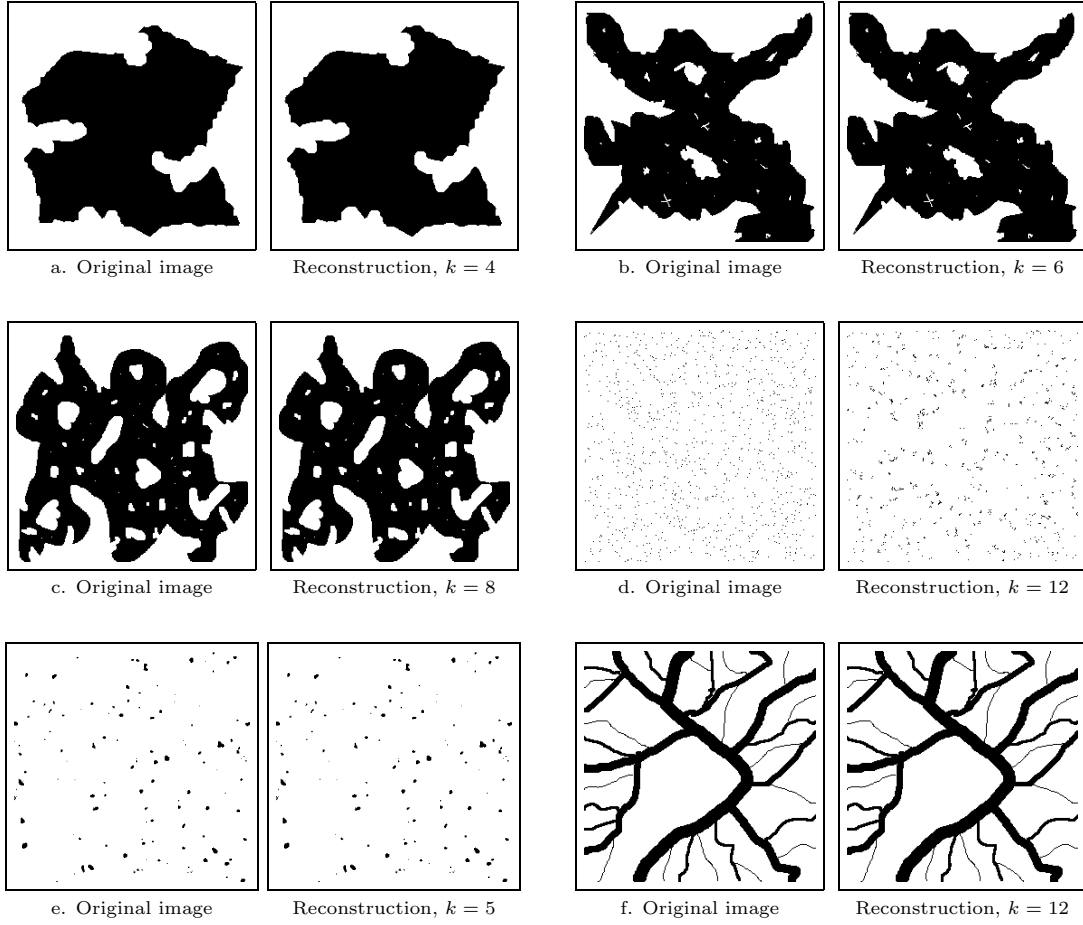


Figure 4: Six original images and their reconstructions.

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 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 usually 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.

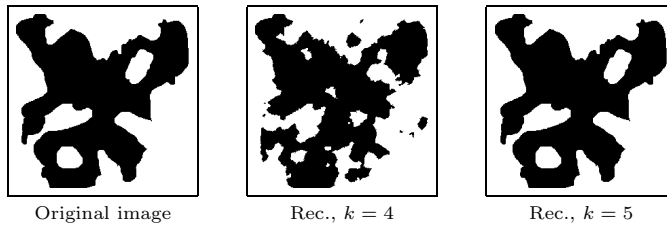


Figure 5: Using too few projections results in very bad reconstructions.

### 4.3 Union of random convex polygons

For any  $F \in \mathcal{F}$ , put  $I_F = \{(x, y) \in A : F(x, y) = 1\}$ . We say that  $F$  is an image of a *convex object* if  $I_F = \text{conv } I_F \cap \mathbb{Z}^2$ , where  $\text{conv } I_F$  denotes the convex hull (in  $\mathbb{R}^2$ ) of  $I_F$ . Images of convex objects are particularly important in discrete tomography as several strong theoretical results are available for such images. In [11], necessary and sufficient conditions on the set of directions are given for which convex objects are uniquely determined by their X-rays. In particular, it is proved that any set of seven prescribed mutually nonparallel lattice directions have the property that all images of convex objects are uniquely determined by their X-rays in these directions (among all images of convex objects). It is also proved in [11] that certain sets of four lattice directions, such as the set  $D' = \{(1, 0), (0, 1), (1, 2), (2, -1)\}$  have this property. These results hold even if we impose no bound on the size of  $A$ . In [6], a polynomial time algorithm is described for reconstructing images of convex objects from their projections in suitable sets of lattice directions, such as the set  $D'$  given above.

For our first class of test images, we implemented a program which generates images that consist of a number of convex 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 from  $A$  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 sample images that were generated by the program, using three different pairs of parameters  $(n, p)$ . All test images described in this section are of size  $256 \times 256$ .

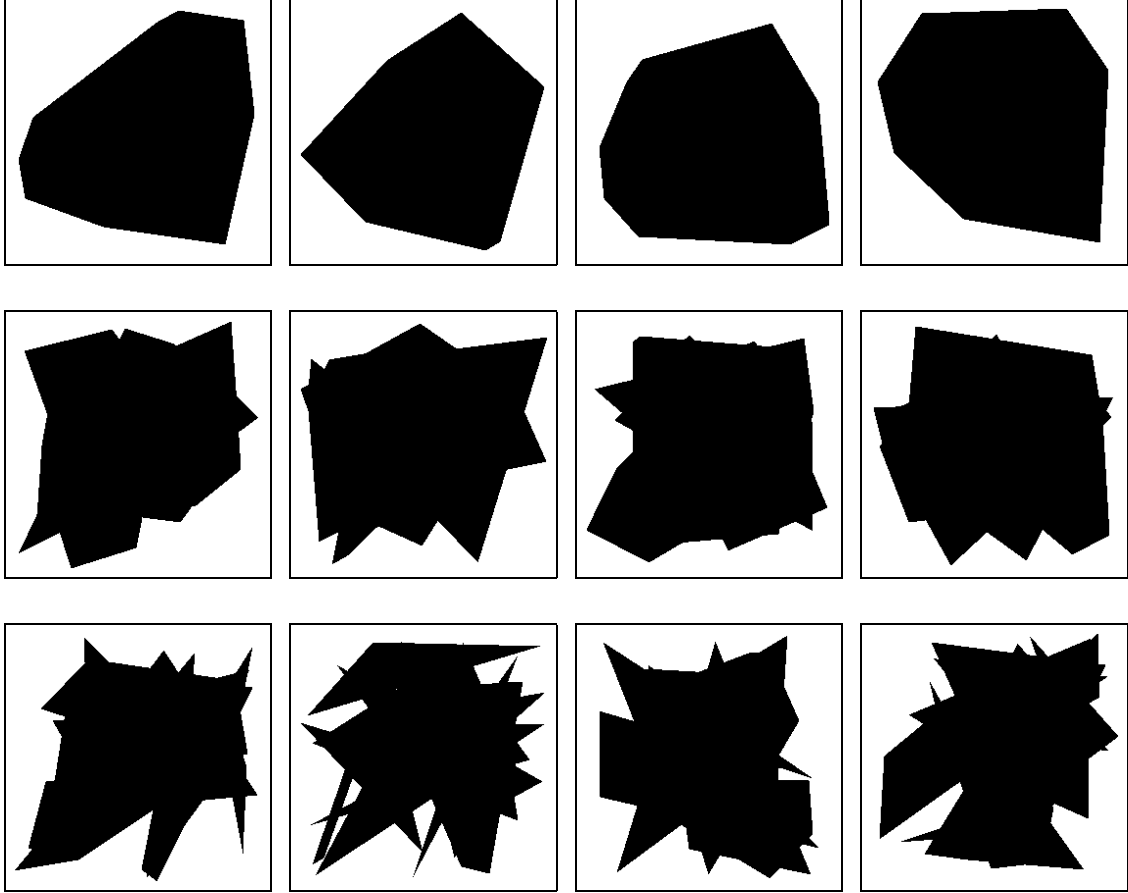


Figure 6: Test images generated by the polygon program. Top row:  $n = 1, p = 25$ . Middle row:  $n = 5, p = 8$ . Bottom row:  $n = 12, p = 4$ .

For all three pairs  $(n, p) \in \{(1, 25), (5, 8), (12, 4)\}$  we performed 200 test runs, using a varying set of three to five lattice directions.



Figure 7: Two images of convex objects that have the same projections in all lattice directions from the set  $D_4$ .

The set  $D_4 = \{(1, 0), (0, 1), (1, 1), (1, -1)\}$  is not sufficient to determine all images of convex objects uniquely by their X-rays, which is illustrated in Figure 7. Besides experiments using the

sets  $D_3$ ,  $D_4$  and  $D_5$ , we also performed experiments for the set  $\tilde{D}_4 = \{(1,0), (0,1), (1,2), (2,-1)\}$  which is sufficient to uniquely determine all images of convex objects. It may be expected that using the set  $\tilde{D}_4$  yields better reconstruction results than the set  $D_4$  for convex objects (the case  $(n,p) = (1,25)$ ) and perhaps also for the composite objects from the other image classes. It should be noted however, that the number of prescribed linesums is  $3n + 3m - 2$  for the set  $D_4$ , while there are  $4m + 4n - 4$  prescribed linesums for the set  $\tilde{D}_4$ .

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.

By a *successful reconstruction* we mean that the distance (as a sum of absolute values) 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: Experimental results for the “random convex polygons” test cases

$n$	$p$	$k$	#success	#perfect	proj.error	pixel error	#iter.	time(s)
1	25	3	200	187	0.5	24	111	13
		4	200	200	0.0	0.0	67	11
		4b	200	200	0.0	0.0	60	10
5	8	3	200	59	32	538	109	14
		4	200	200	0.0	0.0	66	11
		4b	200	200	0.0	0.0	83	15
		5	200	200	0.0	0.0	61	10
12	4	4	190	190	12	62	123	18
		4b	194	187	18	107	267	25
		5	200	200	0.0	0.0	108	18
		6	200	200	0.0	0.0	63	14

Table 3 shows the test results. The first three columns contain the test parameters  $n$ ,  $p$  and  $k$ . In the third column, the set  $D_4$  is referred to as “4” and the set  $\tilde{D}_4$  is referred to as “4b”. 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.

There is no set of three lattice directions such that the projections along these directions uniquely determine all images of convex objects among the set of such images (see [11]). Still, using only three projections the algorithm obtained a perfect reconstruction for 187 out of 200 test images from the class  $(n, p) = (25, 1)$ . 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.

Surprisingly, using the lattice directions from  $\tilde{D}_4$  instead of  $D_4$  does not seem to improve the experimental results for any of the three test cases, even though the number of linesums is substantially larger for the set  $\tilde{D}_4$ . Using the set  $D_4$ , the algorithm reconstructed the original image for all 200 test images in the class  $(n, p) = (1, 25)$ . We tested our algorithm using the projections of a convex polygon that is not uniquely determined by its X-rays in the directions of  $D_4$ . The resulting reconstruction was convex and had the exact prescribed projections.

#### 4.4 Union of random ellipses

For the second class of test images we implemented a program which generates a superposition 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. 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 black. Figure 8 shows five examples of images that were generated by this program, each using different parameter settings.

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 8 we performed 200 test runs. All test images are of size  $256 \times 256$ . The results are shown in Table 4. For those sets of test

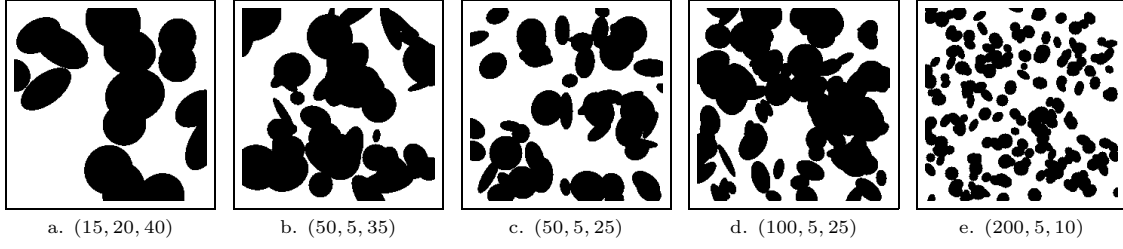


Figure 8: Five images that were generated by the ellipse program, each using different parameters  $(n, r_{min}, r_{max})$ .

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.

Table 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

#### 4.5 Noisy projection data

So far we have assumed that the prescribed projections are perfect, without any noise. The network flow approach from Section 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 9. Each arc in the network has an associated capacity  $u$  and cost  $c$ , which are shown in the figure as  $u/c$ .

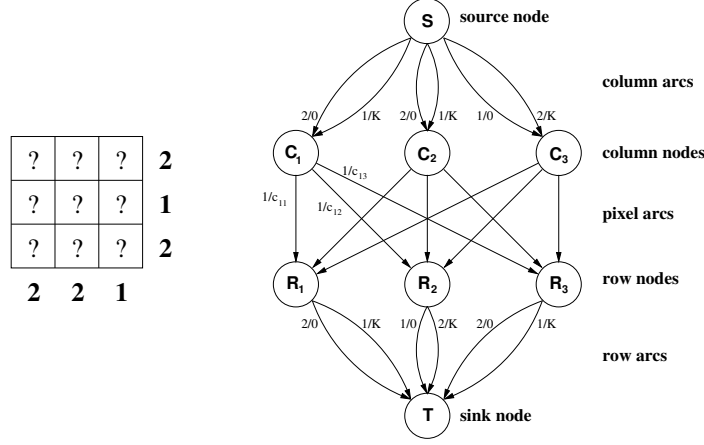


Figure 9: Network for the case of noisy projection data.

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  $n - r_j$ . 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 (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 Gaussian distribution with average  $\mu = 1$  and variance  $\sigma^2$ . 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 (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.

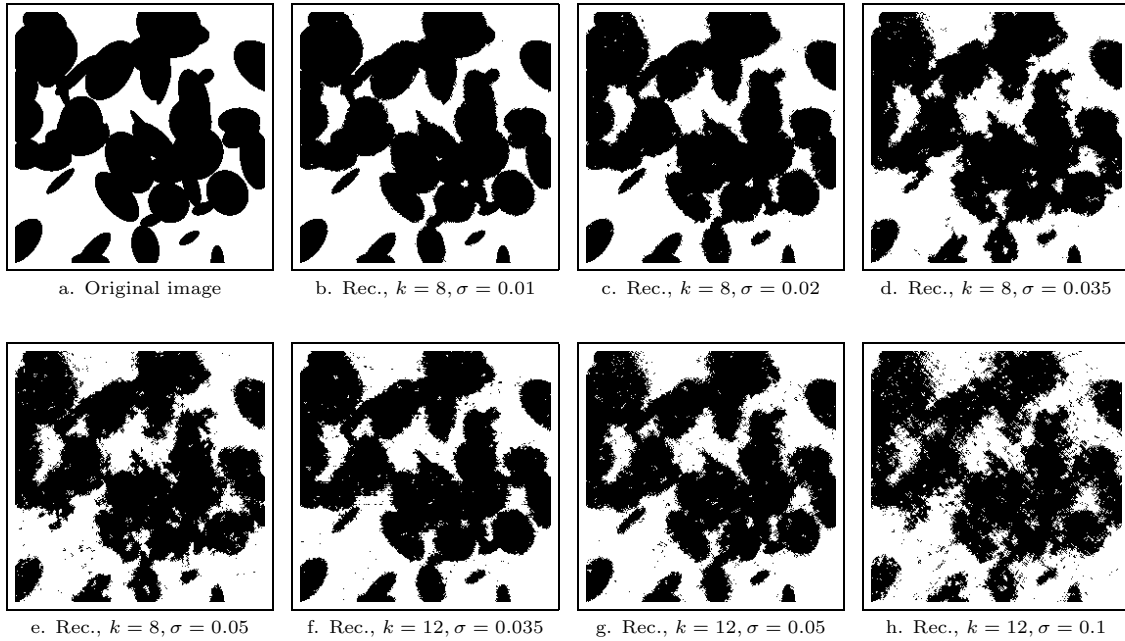


Figure 10: Reconstruction results for noisy projection data.

#### 4.6 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 literature of other reconstruction algorithms that are suitable for large images (i.e.,  $256 \times 256$  pixels or more).

In [7], 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  $40 \times 40$  pixels). The resulting reconstruction is real valued; all pixel values are between 0 and 1. In [28], Weber et al. describe an extension of the linear program from [7] which incorporates a smoothness prior. We implemented the *FSSV2* approach described in their paper

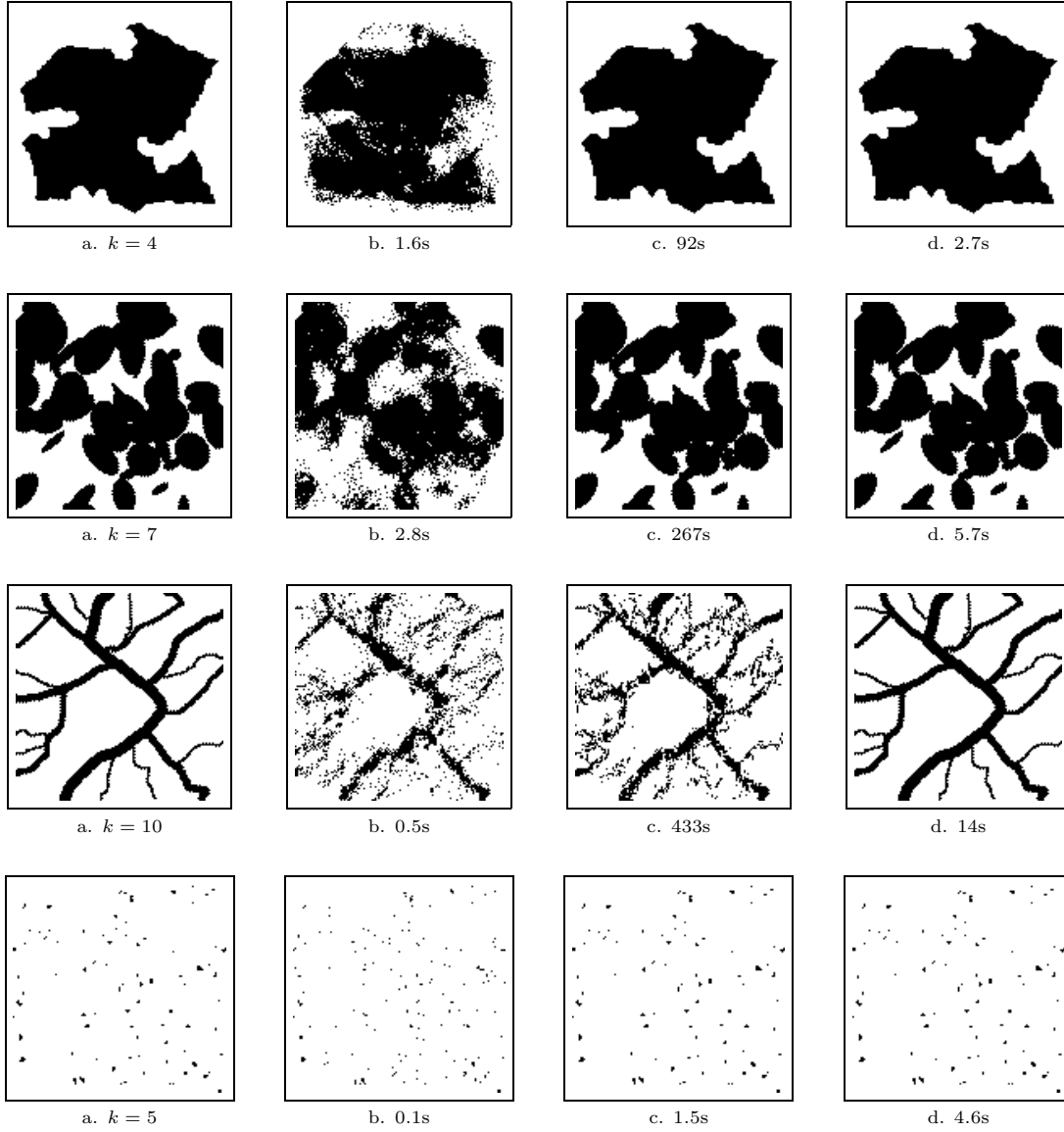


Figure 11: Reconstruction results for four test images. a. Original image; b. result of Gritzmann et al. [14]; c. result of Weber et al. [28]; d. result of our algorithm.

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 [28].

In [14], Gritzmann et al. describe a greedy algorithm for reconstructing binary images from their projections. The paper reports reconstruction results for images up to  $512 \times 512$  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 11 shows the reconstruction results for four test images. The test images all have a size of

$128 \times 128$  pixels, because the running time of the linear programming approach becomes prohibitively large for larger images. The figure indicates the number of projections 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 [28]. The greedy approach from [14] 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 very accurate 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.

## 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 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., [18]). 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 the book [10] on *Geometric Tomography*). 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 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 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  $1000 \times 1000$  pixels, although that may require several hours of computation time.

By adapting our algorithm as described in Section 4.5, 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 few good alternatives are available. The results in Section 4.6 indicate that for relatively smooth images our approach is much faster than the linear programming approach from [28], which also uses a smoothness assumption, and yet yields comparable or better reconstruction quality. Our algorithm is slower than the greedy approach from [14], but the greedy approach yields inferior reconstruction quality.

## 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. Future work will include the exploration of several possible generalizations to other discrete tomography problems.

## References

- [1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin, *Network flows: theory, algorithms, and applications*, Prentice-Hall, 1993.
- [2] R.P. Anstee, “The network flows approach for matrices with given row and column sums,” *Discrete Math.*, Vol. 44, pp. 125–138, 1983.
- [3] E. Barcucci, A. Del Lungo, M. Nivat, and R. Pinzani, “Reconstructing convex polyominoes from horizontal and vertical projections,” *Theoret. Comput. Sci.*, Vol. 155, pp. 321–347, 1996.
- [4] K.J. Batenburg, “A new algorithm for 3D binary tomography,” Proceedings of the Workshop on Discrete Tomography and its Applications, New York, *Electron. Notes Discrete Math.*, Vol. 20, pp. 247–261, 2005.
- [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 Appl.*, Vol. 339, pp. 37–57, 2001.
- [6] S. Brunetti and A. Daurat, “An algorithm for reconstructing lattice convex sets,” *Theoret. Comput. Sci.*, Vol. 304, pp. 35–57, 2003.
- [7] P. Fishburn, P. Schwander, L. Shepp, and R. Vanderbei, “The discrete Radon transform and its approximate inversion via linear programming,” *Discrete Appl. Math.*, Vol. 75, pp. 39–61, 1997.
- [8] H.N. Gabow and R.E. Tarjan, “Faster scaling algorithms for network problems,” *SIAM J. Comput.*, Vol. 18, pp. 1013–1036, 1989.
- [9] D. Gale, “A theorem on flows in networks,” *Pacific J. Math.*, Vol. 7, pp. 1073–1082, 1957.
- [10] R.J. Gardner, *Geometric Tomography*, Cambridge University Press, New York, 1995.
- [11] R.J. Gardner and P. Gritzmann, “Discrete tomography: determination of finite sets by X-rays,” *Trans. Amer. Math. Soc.*, Vol. 349, pp. 2271–95, 1997.
- [12] R.J. Gardner, P. Gritzmann, and D. Prangenberg, “On the computational complexity of reconstructing lattice sets from their X-rays,” *Discrete Math.*, Vol. 202, pp. 45–71, 1999.
- [13] A.V. Goldberg, “An efficient implementation of a scaling minimum-cost flow algorithm,” *J. Algorithms*, Vol. 22, pp. 1–29, 1997.

- [14] P. Gritzmann, S. de Vries, and M. Wiegmann, "Approximating binary images from discrete X-rays," *SIAM J. Optim.*, Vol. 11, pp. 522–546, 2000.
- [15] P. Gritzmann, D. Prangenberg, S. de Vries, and M. Wiegmann, "Success and failure of certain reconstruction and uniqueness algorithms in discrete tomography," *Int. J. Imag. Syst. Tech.*, Vol. 9, pp. 101–109, 1998.
- [16] L. Hajdu and R. Tijdeman, "Algebraic aspects of discrete tomography," *J. Reine Angew. Math.*, Vol. 534, pp. 119–128, 2001.
- [17] L. Hajdu and R. Tijdeman, "An algorithm for discrete tomography," *Linear Algebra Appl.*, Vol. 339, pp. 147–169, 2001.
- [18] G.T. Herman and A. Kuba, editors, *Discrete Tomography: Foundations, Algorithms and Applications*, Birkhäuser, Boston, 1999.
- [19] J.R. Jinschek, K.J. Batenburg, H. Calderon, D. Van Dyck, F.-R. Chen, and C. Kisielowski, "Prospects for bright field and dark field electron tomography on a discrete grid," *Microsc. Microanal.*, Vol. 10 Suppl. 3, Cambridge Journals Online, 2004.
- [20] J.R. Jinschek, H.A. Calderon, K.J. Batenburg, V. Radmilovic and C. Kisielowski, "Discrete Tomography of Ga and InGa Particles from HREM Image Simulation and Exit Wave Reconstruction," *MRS Proc.*, Vol. 839, pp. 4.5.1–4.5.6, 2004.
- [21] 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*, Vol. 58, pp. 131–155, 1995.
- [22] H.J. Ryser, "Combinatorial properties of matrices of zeros and ones," *Canad. J. Math.*, Vol. 9, pp. 371–377, 1957.
- [23] A. Schrijver, *Combinatorial optimization. Polyhedra and efficiency*, Springer-Verlag, Berlin, 2003.
- [24] 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," *Phys. Rev. Lett.*, Vol. 71, pp. 4150–4153, 1993.
- [25] C.H. Slump and J.J. Gerbrands, "A network flow approach to reconstruction of the left ventricle from two projections," *Comput. Gr. Im. Proc.*, Vol. 18, pp. 18–36, 1982.

- [26] K. Tanabe, "Projection method for solving a singular system," *Numer. Math.*, Vol. 17, pp. 203–214, 1971.
- [27] B. Wang and F. Zhang, "On the precise number of  $(0, 1)$ -matrices in  $\mathcal{A}(R, S)$ ," *Discrete Math.*, Vol. 187, pp. 211–220, 1998.
- [28] S. Weber, C. Schnörr and J. Hornegger, "A linear programming relaxation for binary tomography with smoothness priors," *Electron. Notes Discrete Math.*, Vol. 12, 2003.