

Fast Filling Operations Used in the Reconstruction of Convex Lattice Sets

Sara Brunetti¹, Alain Daurat², and Attila Kuba³

¹ Dipartimento di Scienze Matematiche e Informatiche, Università di Siena,
Pian dei Mantellini 44, 53100, Siena, Italy

`sara.brunetti@unisi.it`

² LSIT CNRS UMR 7005, Université Louis Pasteur (Strasbourg 1),
Pôle API, Boulevard Sébastien Brant, 67400 Illkirch-Graffenstaden, France

`daurat@dpt-info.u-strasbg.fr`

³ Dept. of Image Processing and Computer Graphics,
University of Szeged, Árpád tér 2. H-6720 Szeged, Hungary

`kuba@inf.u-szeged.hu`

Abstract. Filling operations are procedures which are used in Discrete Tomography for the reconstruction of lattice sets having some convexity constraints. In [1], an algorithm which performs four of these filling operations has a time complexity of $O(N^2 \log N)$, where N is the size of projections, and leads to a reconstruction algorithm for convex polyominoes running in $O(N^6 \log N)$ -time. In this paper we first improve the implementation of these four filling operations to a time complexity of $O(N^2)$, and additionally we provide an implementation of a fifth filling operation (introduced in [2]) in $O(N^2 \log N)$ that permits to decrease the overall time-complexity of the reconstruction algorithm to $O(N^4 \log N)$. More generally, the reconstruction of \mathbb{Q} -convex sets and convex lattice sets (intersection of a convex polygon with \mathbb{Z}^2) can be done in $O(N^4 \log N)$ -time.

keywords: Discrete Tomography, Convexity, Filling Operations.

1 Introduction

One of the most intensively studied fields of discrete tomography is the reconstruction of lattice sets or, specially, binary matrices. Several algorithms have been published for reconstructing such sets. It is well-known that a binary matrix from its row and column sums can be reconstructed in polynomial time [3]. The interesting question is which sub-class of binary matrices can be reconstructed in polynomial time. In most cases some kind of (discrete) convexity is supposed on the sets. For example, Kuba published an algorithm [4] to reconstruct so-called hv -convex lattice sets from two projections. As it turned out later the reconstruction problem in this class is NP-complete [5]. Barucci et al. showed [6] that a sub-class of hv -convex lattice sets, namely, the class of hv -convex polyominoes can be reconstructed in polynomial time. This result was extended also to a bigger class, that of hv -convex 8-connected lattice sets [2]. A new bigger

class of convex sets, the so-called Q-convex sets was studied by Brunetti and Daurat [7] and proved that even in this class the reconstruction can be solved in polynomial time.

Most of the algorithms reconstructing sets presenting some convexity properties use special procedures called filling operations. These operations can be applied in iterative procedures to approach the final solutions with two sequences of sets. The first sequence is a sequence of decreasing upper bounds and the second one is a sequence of increasing lower bounds of the solutions.

Originally in [6], four filling operations were defined. In [1], an efficient algorithm was given to apply the filling operations. In [2] a fifth filling operation was introduced to decrease the overall complexity of the reconstruction algorithm. Unfortunately, the algorithm for the filling operations of [1] cannot be generalized with this fifth operation (in [2] this point was not treated). In this paper we provide an implementation of all the five filling operations in the same complexity as the algorithm of [1]. As a result, we get an improvement in the time-complexity of the reconstruction algorithm.

The structure of this paper is the following. Section 2 contains the necessary definitions and notations. The filling operations, the new reconstruction algorithm, its analysis, and a possible generalization are described in Section 3. Section 4 shows the application of the new operation in the case of Q-convex and convex sets. Finally, in Section 5 we show some statistical results connected with the application of the filling operations in computer experiments.

2 Definitions

A lattice set is a finite subset of \mathbb{Z}^2 . A lattice direction is given by an integer vector $\mathbf{p} = (p_x, p_y)$, and it can also be represented by a linear form $p(x, y) = p_y x - p_x y$. The horizontal direction (resp. vertical direction) denoted by h (resp. v) is determined by the vector $(1, 0)$ (resp. $(0, 1)$).

A lattice set is line-convex with respect to a direction p if its intersection with each line in the direction p is made of consecutive points. A set which is line-convex w.r.t. to the horizontal and vertical directions is called hv -convex.

The projection of a lattice set E along a direction p , denoted by $X_p E$, is the function which gives the number of points on any line of direction p , more precisely:

$$X_p E(k) = |\{M \in E : p(M) = k\}| \text{ for any } k \in \mathbb{Z}$$

where p is the linear form associated to \mathbf{p} .

In this article we are interested in the reconstruction of set E which satisfies some convexity constraints from its projections. More precisely if \mathcal{M} is a class of lattice sets, and \mathcal{D} is a finite set of lattice directions, the reconstruction problem for the class \mathcal{M} and the directions \mathcal{D} is the following.

RECONSTRUCTION(\mathcal{M}, \mathcal{D})

Data: A function $f : \mathcal{D} \times \mathbb{Z} \rightarrow \mathbb{Z}_+$ which gives a non-negative integer $f(p, k)$ for any line $p = k$ with $p \in \mathcal{D}$, and such that $\{(p, k) : f(p, k) > 0\}$ is finite.

Task: Reconstructing a lattice set $E \in \mathcal{M}$ such that $X_p E(k) = f(p, k)$ for any $(p, k) \in \mathcal{D} \times \mathbb{Z}$
 In the whole paper $[a, b]$ denotes the discrete interval $\{k \in \mathbb{Z} : a \leq k \leq b\}$.

3 Filling Operations

3.1 Preliminaries

A filling operation is a procedure which has been used in many reconstruction algorithms [2, 4, 6–8]. Formally, a filling operation takes function f of $\text{RECONSTRUCTION}(\mathcal{M}, \mathcal{D})$, and a pair of sets (α, β) such that $\alpha \subset \beta$ and returns a new pair of sets (α', β') with $\alpha \subseteq \alpha' \subseteq \beta' \subseteq \beta$.

We now present classical filling operations which can be used for any class contained in that of line-convex sets w.r.t. \mathcal{D} .

To simplify the description of these operations, we first describe them for the set $\mathcal{D} = \{h, v\}$ consisting of the horizontal and vertical directions. We denote $h_i = f(h, i)$ and $v_j = f(v, j)$. We also suppose without loss of generality that there exist $m, n \in \mathbb{Z}_+$ such that $h_i = 0$ for $i \notin [1, m]$ and $v_j = 0$ for $j \notin [1, n]$.

For any $i \in [1, m]$ we denote the set $\{(i, j) : j \in \mathbb{Z}, (i, j) \in \alpha\}$ by α_i^h . On an analogous way we define $\alpha_j^v, \beta_i^h, \beta_j^v$ for $i \in [1, m]$ and $j \in [1, n]$.

The following notations are used for the extremities of each α_i^h and β_i^h .

$$\begin{aligned} l(\alpha_i^h) &= \min(\{j : (i, j) \in \alpha_i^h\}), & r(\alpha_i^h) &= \max(\{j : (i, j) \in \alpha_i^h\}) \\ l(\beta_i^h) &= \min(\{j : (i, j) \in \beta_i^h\}), & r(\beta_i^h) &= \max(\{j : (i, j) \in \beta_i^h\}) \end{aligned}$$

with the conventions $\min(\emptyset) = +\infty$, $\max(\emptyset) = -\infty$.

With this notation, the four filling operations of [6] on horizontal lines can be defined as:

- If $\alpha_i^h \neq \emptyset$ then $\oplus \alpha_i^h = \{(i, j) \in \beta_i^h : l(\alpha_i^h) \leq j \leq r(\alpha_i^h)\}$.
- $\otimes \alpha_i^h = \{(i, j) \in \beta_i^h : r(\beta_i^h) - h_i < j < l(\beta_i^h) + h_i\}$.
- If $\alpha_i^h \neq \emptyset$, $j' = \max(\{j : (i, j) \notin \beta_i^h \text{ and } j < l(\alpha_i^h)\})$, $j'' = \min(\{j : (i, j) \notin \beta_i^h \text{ and } j > r(\alpha_i^h)\})$ then $\ominus \beta_i^h = \{(i, j) \in \beta_i^h : j' < j < j''\}$.
- If $\alpha_i^h \neq \emptyset$, then $\odot \beta_i^h = \{(i, j) \in \beta_i^h : r(\alpha_i^h) - h_i < j < l(\alpha_i^h) + h_i\}$.

A fifth filling operation \odot' has been introduced in [2, 7, 8]. It permits to reduce the overall complexity of the reconstruction algorithm: In all the reconstruction algorithms the first step of the algorithm is fixing arbitrarily some points on the border of the reconstructing sets (these points are in general called bases or feet). Without the operation \odot' , at least four fixed points were necessary, but with it, only two are necessary. (See [7, p.43-44] for more details.) The operation \odot' simply removes the components of β_i^h (maximum sequences of consecutive elements of β_i^h) which are smaller than the corresponding projection.

To define it formally we need a notation for the extremities of each component. So the sequence $(c_k)_{1 \leq k \leq 2r} = c(\beta_i^h)$ is defined by:

$$c_k < c_{k+1} \text{ and } \{j : (i, j) \in \beta_i^h\} = \bigcup_{k=1}^r [c_{2k-1}, c_{2k} - 1]. \quad (1)$$

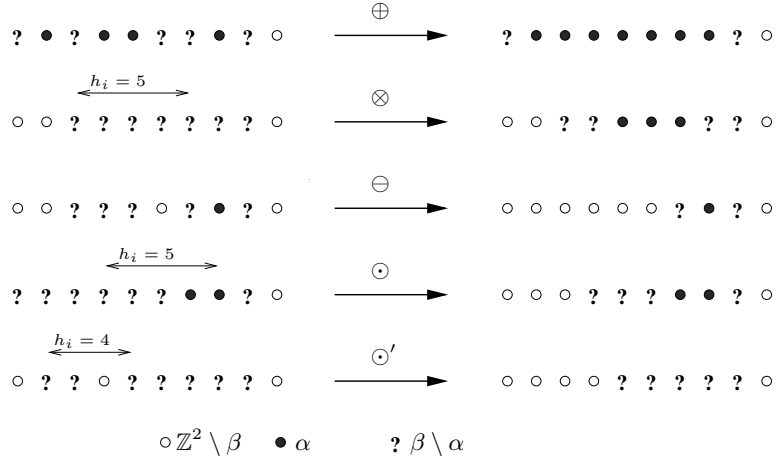


Fig. 1. The filling operations

Then the operation \odot' is defined by:

$$\odot' \beta_i^h = \bigcup_{\substack{1 \leq k \leq r \\ c_{2k} - c_{2k-1} \geq h_i}} [c_{2k-1}, c_{2k} - 1].$$

We can also define these five operations on the vertical lines.

The reconstruction algorithms described in [2, 6, 7] iteratively apply the filling operations in a fixed order on all the lines of $[1, m] \times [1, n]$ (iterative step). The k th iteration gives rise to a new couple (α_k, β_k) from $(\alpha_{k-1}, \beta_{k-1})$, and the iterative process ends when an invariant couple is obtained, that is, $(\alpha_{k'}, \beta_{k'}) = (\alpha_{k'-1}, \beta_{k'-1})$. There are several methods to construct the initial couple of sets (α_0, β_0) . For example in [6], β_0 is the complete rectangle $[1, m] \times [1, n]$ and α_0 consists of a set of points (called feet) located on the four edges of β . More generally, we assume that $\alpha_0 \subseteq \beta_0 \subseteq [1, m] \times [1, n]$.

In [1, 6] only the first four filling operations are considered. If $N = \max(m, n)$, the whole iterative process runs in $O(N^4)$ -time in [6]. In [1] the author proves that this process can be executed in $O(N^2 \log N)$ -time. The best time-complexity with the five filling operations is $O(N^3)$ [7]. Now we will describe a procedure which performs the five filling operations in $O(N^2 \log N)$ -time.

3.2 The New Algorithm

At first we describe the data structures we use in the algorithm.

For each horizontal line of index i , we use the following data.

- The scalar variables $l_1(\alpha_i^h)$ and $l_2(\alpha_i^h)$ for $l(\alpha_i^h)$. The first one is only updated when the operations are performed on the i th horizontal line. The second one is updated for any change of any point on this line.

- The scalar variables $r_1(\alpha_i^h)$ and $r_2(\alpha_i^h)$ which are defined in a similar way.
- The scalar variable $l_1(\beta_i^h)$ and $l_2(\beta_i^h)$ for $l(\beta_i^h)$. The first one is only updated when the operations are made on the i th horizontal line. The second one is updated for any change of any point on this line and if $\alpha_i^h \neq \emptyset$, then $l_2(\beta_i^h) = \max(\{j : (i, j) \notin \beta_i^h \text{ and } j < l(\alpha_i^h)\}) + 1$ (so it anticipates the operation \ominus on the line).
- The scalar variables $r_1(\beta_i^h)$ and $r_2(\beta_i^h)$ which are defined in a similar way.
- The integer array `next_in_betaih` such that `next_in_betaih[j]` gives the nearest point of β_i^h on the right of $(i, j) \in \beta_i^h$. Moreover this array also indicates the leftmost point of β_i^h , indexed by $-\infty$. Formally for any $(i, j) \in \beta_i^h \cup \{(i, -\infty)\}$, `next_in_betaih[j]` = $\min(\{k > j : (i, k) \in \beta_i^h\})$.
- The array `prev_in_betaih` which is analogous to `next_in_betaih` but gives the nearest point on the left: For any $(i, j) \in \beta_i^h \cup \{(i, +\infty)\}$ `prev_in_betaih[j]` is $\max(\{k < j : (i, k) \in \beta_i^h\})$.
- An optimized structure denoted by $c(\beta_i^h)$ to represent the ordered sequence of intervals $[c_{2k-1}, c_{2k} - 1]$ defined by (1).

We suppose that we have the following operations on this structure:

- `insert($c(\beta_i^h), u, v$)`: insertion of a new interval $[u, v - 1]$ which is disjoint with the intervals $[c_{2k-1}, c_{2k} - 1]$.
- `delete($c(\beta_i^h), u, v$)`: deletion of an interval $[u, v - 1]$.
- `search($c(\beta_i^h), x$)`: returns (c_{2k-1}, c_{2k}) such that $c_{2k-1} \leq x < c_{2k}$.

We use a structure such that these operations are made in $O(\log(r))$ -time.

For example, an implementation with AVL trees permits this (see [9]).

- Another ordered sequence $d(\beta_i^h)$ of intervals $[d_{2k-1}, d_{2k} - 1]$ which represents the components which are to be deleted by the operation \odot' . This sequence is represented by the same structure as $c(\beta_i^h)$.

There are also data not associated directly to a line.

- The sets α and β are simply implemented by a boolean two-dimensional array.
- A set `lines_to_treat` stores the lines which are to be treated by the filling operations. The only operations used on this structure are the vacancy test, the extraction of one arbitrary element and the insertion of one element (if not already present in the structure). These operations can be executed in constant time if the set is implemented as an array B of booleans coupled with an array A (implementing a stack) of the elements both indexed in $[1, m + n]$ and an integer variable cA for the cardinality. Precisely the implementation is the following.

```
isempty(lines_to_treat)
```

```
return(cA = 0)
```

```
extract(lines_to_treat)
```

```
x ← A[cA]; B[x] ← false; cA ← cA - 1
```

```
if x ≤ m then
```

```
    return(h = x)
```

```
else
```

```
    return(v = x - m)
```

```

    end if
  add_line(lines_to_treat, h = i)
  if not(B[i]) then
    B[i] ← true; cA ← cA + 1; A[cA] ← i
  end if
  add_line(lines_to_treat, v = j)
  if not(B[m + j]) then
    B[m + j] ← true; cA ← cA + 1; A[cA] ← m + j
  end if

```

Now we can describe precisely the algorithm for the filling operations.

The two first procedures put a point in α or remove a point from β . They update the data for the horizontal and vertical lines passing through the point.

```

put_in_alphah(i, j)
  if (i, j) ∉ β then
    exit(no solution)
  end if
  if (i, j) ∈ α then
    return // Nothing to do !
  end if
  α ← α ∪ {(i, j)}
  for (p, i', j') ∈ {(h, i, j), (v, j, i)} do
    l2(αi'p) ← min(l2(αi'p), j'); r2(αi'p) ← max(r2(αi'p), j')
  end for
  add_line(lines_to_treat, v = j)
remove_from_betah(i, j)
  if (i, j) ∈ α then
    exit(no solution)
  end if
  if (i, j) ∉ β then
    return // Nothing to do !
  end if
  β ← β \ {(i, j)}
  for (p, i', j', x) ∈ {(h, i, j, hi), (v, j, i, vj)} do
    if j' = l2(βi'p) or j' < l2(αi'p) then
      l2(βi'p) ← next_in_betai'p[j']
    end if
    if j' = r2(βi'p) or j' > r2(αi'p) then
      r2(βi'p) ← prev_in_betai'p[j']
    end if
    next_in_betai'p[prev_in_betai'p[j']] ← next_in_betai'p[j']
    prev_in_betai'p[next_in_betai'p[j']] ← prev_in_betai'p[j']
    (u, u') ← search(c(βi'p), j'); delete(c(βi'p), u, u')
    if u < j' then
      insert(c(βi'p), u, j')
    end if
    if j' + 1 < u' then
      insert(c(βi'p), j' + 1, u')
    end if
    if u' - u < x then

```

```

    delete( $d(\beta_{i'}^p), u, u'$ )
  end if
  if  $0 < j' - u < x$  then
    insert( $d(\beta_{i'}^p), u, j'$ )
  end if
  if  $0 < u' - (j' + 1) < x$  then
    insert( $d(\beta_{i'}^p), j' + 1, u'$ )
  end if
end for
add_line(lines_to_treat,  $v = j$ )

```

This procedure applies the filling operations on a horizontal line.

```

treat_line( $h = i$ )


---


( $d_k$ ) $_{1 \leq k \leq 2r} \leftarrow d(\alpha_i^h)$  // Operation  $\odot'$ 
for all  $k \in [1, r]$  and  $j \in [d_{2k-1}, d_{2k} - 1]$  do
  remove_from_beta_h( $i, j$ )
end for
if  $l_1(\alpha_i^h) = +\infty$  then // Operation  $\oplus$ 
  for all  $j \in [l_2(\alpha_i^h) + 1, r_2(\alpha_i^h) - 1]$  do
    put_in_alpha_h( $i, j$ )
  end for
else
  for all  $j \in [l_2(\alpha_i^h) + 1, l_1(\alpha_i^h) - 1] \cup [r_1(\alpha_i^h) + 1, r_2(\alpha_i^h) - 1]$  do
    put_in_alpha_h( $i, j$ )
  end for
end if
for all  $j \in [l_1(\beta_i^h), l_2(\beta_i^h) - 1] \cup [r_2(\beta_i^h) + 1, r_1(\beta_i^h)]$  do // Operation  $\ominus$ 
  remove_from_beta_h( $i, j$ )
end for
if  $r_2(\beta_i^h) - h_i + 1 \leq l_2(\beta_i^h) + h_i - 1$  then // Operation  $\otimes$ 
  if  $l_2(\alpha_i^h) = +\infty$  then
    for all  $j \in [r_2(\beta_i^h) - h_i + 1, l_2(\beta_i^h) + h_i - 1]$  do
      put_in_alpha_h( $i, j$ )
    end for
  else
    for all  $j \in [r_2(\beta_i^h) - h_i + 1, l_2(\alpha_i^h) - 1] \cup [r_2(\alpha_i^h) + 1, l_2(\beta_i^h) + h_i - 1]$  do
      put_in_alpha_h( $i, j$ )
    end for
  end if
end if
if  $l_2(\alpha_i^h) \neq +\infty$  then // Operation  $\odot$ 
  for all  $j \in [l_2(\beta_i^h), r_2(\alpha_i^h) - h_i] \cup [l_2(\alpha_i^h) + h_i, r_2(\beta_i^h)]$  do
    remove_from_beta_h( $i, j$ )
  end for
end if


---


 $l_1(\alpha_i^h) \leftarrow l_2(\alpha_i^h); r_1(\alpha_i^h) \leftarrow r_2(\alpha_i^h); l_1(\beta_i^h) \leftarrow l_2(\beta_i^h); r_1(\beta_i^h) \leftarrow r_2(\beta_i^h)$ 

```

The procedures $\text{put_in_alpha}_v(i, j)$, $\text{remove_from_beta}_v(i, j)$, $\text{treat_line}(v = j)$ are similar. This is the main procedure for the filling operations.

filling_operations(α_0, β_0)

$\alpha \leftarrow \alpha_0; \beta \leftarrow \beta_0$

```

 $\beta \leftarrow \beta \setminus \{(i, j) : h_i = 0 \text{ or } v_j = 0\}$ 
for all  $l \in \{h = i : 1 \leq i \leq m \text{ and } h_i > 0\} \cup \{v = j : 1 \leq j \leq n \text{ and } v_j > 0\}$  do
  add_line(lines_to_treat,  $l$ )
end for
initialize  $l_1, l_2, r_1, r_2, \text{next\_in\_beta}, \text{prev\_in\_beta}, c, d$  for all the lines of lines_to_treat
while not(isempty(lines_to_treat)) do
   $l \leftarrow \text{extract}(\text{lines\_to\_treat})$ 
  treat_line( $l$ )
end while
return( $\alpha, \beta$ )

```

3.3 Correctness of the Algorithm

- At the end of the executions of the procedure `put_in_alpha` and `remove_from_beta`, all the variables $l_2, r_2, \text{next_in_beta}, \text{prev_in_beta}, c, d$ are updated according to the actual α and β .
- The modifications of α and β done by `treat_line` correspond exactly to the five filling operations $\odot', \oplus, \ominus, \otimes, \odot$ executed in this order. In particular, if an instruction “`exit(no solution)`” is executed, the filling operations lead to a situation where $\alpha \not\subseteq \beta$.
- At the end of the procedure `treat_line`($h = i$), the line $h = i$ is invariant w.r.t. the five filling operations, so during the execution of the algorithm all the lines which are *not* in `lines_to_treat` are invariant w.r.t. to the filling operations: when `lines_to_treat` is empty, (α, β) is invariant w.r.t. to the filling operations.
- The algorithm stops after a finite number of steps because $(|\beta \setminus \alpha|, |\text{lines_to_treat}|)$ decreases lexicographically at each iteration of `filling_operations`.

3.4 Analysis of Complexity

Let $N = \max(\{m, n\})$.

- The procedures `put_in_alpha` and `remove_from_beta` are executed in $O(1)$ and $O(\log N)$ -time respectively.
- The procedure `treat_line` has a time complexity $O(1 + p \log N)$ where p is the number of times the procedure `put_in_alpha`, or `remove_from_beta` run.
- The procedure `put_in_alpha` ^{h} is never done more than once on a point and `remove_from_beta` ^{h} is never done more than twice: once for the first four filling operations and a second time for the fifth operation \odot' . So these two procedures are executed less than $2N^2$ times.
- Similarly the procedures `put_in_alpha` ^{v} and `remove_from_beta` ^{v} are executed less than $2N^2$ times.
- The procedure `treat_line` is repeated less than $2N + 8N^2$ times because `lines_to_treat` is filled first with less than $2N$ lines and then a line is added to it only from `put_in_alpha` or `remove_from_beta`. So the global time-complexity of the algorithm is $O(N^2 \log N)$.

3.5 Differences with Gebala's Algorithm

The procedures performing the filling operations which are described by Gebala in [1] have the same structure than the ones presented here. However our algorithm presents several improvements:

- Gebala's algorithm does not apply the fifth filling operation.
- Gebala's algorithm uses a tree (free_0) to store the points on each line which are not in β . In our algorithm this structure is not needed because we use the arrays `next_in_beta` and `prev_in_beta`. Thanks to this, there is no loop in the procedures `put_in_alpha` and `remove_from_beta` which simplifies the analysis of the complexity of these procedures. Moreover these arrays need only $O(1)$ -time operations.
- Gebala uses two trees (tree_{row} and tree_{col}) in the place of `lines_to_treat`.

In fact if we restrict our algorithm to work with the first four filling operations, the ordered sequences c and d are not necessary and so our algorithm runs in $O(N^2)$ -time, that is better compared to the complexity $O(N^2 \log N)$ of Gebala's algorithm. Unfortunately, the additional fifth filling operation increases the time-complexity of our algorithm to $O(N^2 \log N)$.

3.6 Extension to Any Finite Set of Lattice Directions

Let \mathcal{D} be a finite set of lattice directions, \mathcal{M} be a class of lattice sets containing the line-convex sets w.r.t. \mathcal{D} . We suppose that f is a function as in $\text{RECONSTRUCTION}(\mathcal{M}, \mathcal{D})$. The size of f will be measured by $N = \max_{p \in \mathcal{D}} (\max(\{k : f(p, k) > 0\}) - \min(\{k : f(p, k) > 0\}) + 1)$.

The filling operations described above can be easily generalized to any set of directions:

- The procedures `put_in_alpha` and `remove_from_beta` must update the data for all the lines parallel to one of the directions of \mathcal{D} .
- The procedure `treat_line` is unchanged.
- The initial β_0 is always included in $\mathcal{G} = \{M \in \mathbb{Z}^2 : \forall p \in \mathcal{D} \min(\{k : f(p, k) > 0\}) \leq p(M) \leq \max(\{k : f(p, k) > 0\})\}$ which contains less than N^2 points.
- The time-complexity of the whole algorithm is still $O(N^2 \log N)$ as the procedures `put_in_alpha` and `remove_from_beta` are done at most two times on each point and each direction.

4 Consequence on the Reconstruction of Convex Sets

We now consider two special classes of lattice sets for which the new implementation of the filling operations improves the complexity of the algorithm solving the reconstruction problem.

4.1 Reconstruction of Q-convex Sets

Let $p = bx - ay$ and $q = dx - cy$ define two lattice directions, and M be a point of \mathbb{Z}^2 ; the four quadrants around M are the four regions delimited by the lines of directions p and q and passing through M . More precisely;

$$\begin{aligned} Z_0^{pq}(M) &= \{M' \in \mathbb{Z}^2 : p(M') \leq p(M) \text{ and } q(M') \leq q(M)\}, \\ Z_1^{pq}(M) &= \{M' \in \mathbb{Z}^2 : p(M') \geq p(M) \text{ and } q(M') \leq q(M)\}, \\ Z_2^{pq}(M) &= \{M' \in \mathbb{Z}^2 : p(M') \geq p(M) \text{ and } q(M') \geq q(M)\}, \\ Z_3^{pq}(M) &= \{M' \in \mathbb{Z}^2 : p(M') \leq p(M) \text{ and } q(M') \geq q(M)\}. \end{aligned}$$

Definition 1. A lattice set E is Q-convex w.r.t. $\mathcal{D} = \{p, q\}$ if $Z_k^{pq}(M) \cap E \neq \emptyset$ for all $k \in \{0, 1, 2, 3\}$ implies $M \in E$.

Definition 2. A lattice set is Q-convex w.r.t. a set \mathcal{D} of directions if it is Q-convex w.r.t. every pair of directions included in \mathcal{D} .

We denote the class of the Q-convex sets w.r.t. \mathcal{D} by $\mathcal{Q}(\mathcal{D})$. In [10] it is proved that there is an algorithm for RECONSTRUCTION($\mathcal{Q}(\mathcal{D}), \mathcal{D}$) which runs in time $O(N^2(N^2 + F(N)))$, where $F(N)$ is the complexity of the filling operations. We can deduce:

Theorem 1. RECONSTRUCTION($\mathcal{Q}(\mathcal{D}), \mathcal{D}$) can be solved in $O(N^4 \log N)$ -time where $N = \max_{p \in \mathcal{D}}(\max(\{k : f(p, k) > 0\}) - \min(\{k : f(p, k) > 0\}) + 1)$.

4.2 Reconstruction of convex lattice sets

Definition 3. A lattice set is convex if it is the intersection of a convex polygon and \mathbb{Z}^2 . We denote the class of convex lattice sets by \mathcal{C} .

If $(p_i)_{i=1..4}$ are four lattice directions determined by the vectors $(\mathbf{p}_i)_{i=1..4} = (a_i, b_i)_i$ and with the slopes $(\lambda_i)_i = (-b_i/a_i)_i$ then the cross-ratio of the four directions $(p_i)_{i=1..4}$ denoted by $[p_1, p_2, p_3, p_4]$ is the element of $\mathbb{R} \cup \{\infty\}$ defined by:

$$[p_1, p_2, p_3, p_4] = \frac{(\lambda_3 - \lambda_1)(\lambda_4 - \lambda_2)}{(\lambda_3 - \lambda_2)(\lambda_4 - \lambda_1)}.$$

The ordered cross-ratio of $(p_i)_{i=1..4}$ is $[p_{\sigma(1)}, p_{\sigma(2)}, p_{\sigma(3)}, p_{\sigma(4)}]$, where σ is the permutation such that $\lambda_{\sigma(i)} < \lambda_{\sigma(i+1)}$. The ordered cross-ratio of four lattice directions is always a rational number which is greater than 1.

It is known that if \mathcal{D} is a set of directions containing four directions whose ordered cross-ratio is not in $\{4/3, 3/2, 2, 3, 4\}$, then the convex lattice sets and Q-convex lattice sets w.r.t \mathcal{D} are uniquely determined by their projections along \mathcal{D} (see [11, 12]). From the same scheme as in [7, 10] we can deduce:

Theorem 2. If \mathcal{D} is a set of directions containing four directions whose ordered cross-ratio is not in $\{4/3, 3/2, 2, 3, 4\}$, then RECONSTRUCTION(\mathcal{C}, \mathcal{D}) can be solved in $O(N^4 \log N)$ -time, where $N = \max_{p \in \mathcal{D}}(\max(\{k : f(p, k) > 0\}) - \min(\{k : f(p, k) > 0\}) + 1)$.

5 The Filling Operations in Practice

In this paper we have proved that the five filling operations can be made until the invariance of α and β in $O(N^2 \log N)$ time. However if we do not apply the fifth filling operation \odot' this complexity decreases to $O(N^2)$. Let us consider the following algorithm which does contain the fifth filling operation.

filling_operations2(α_0, β_0)

$\alpha \leftarrow \alpha_0$

$\beta \leftarrow \beta_0$

repeat

 Apply the four operations $\oplus, \ominus, \odot, \otimes$ to (α, β) until invariance of α and β

 Apply the operation \odot' to (α, β)

until the last operation \odot' leaves (α, β) invariant

return(α, β)

The time-complexity of this algorithm is $O(lN^2)$, where l is the number of iterations of the **repeat** loop. The only theoretical upper bound we have found for l is N^2 . To have a better estimation of l we have conducted the following experiment:

- We have considered the set of directions $\mathcal{D} = \{h, v\}$ and the class of lattice sets $\mathcal{Q}(\mathcal{D})$.
- We have generated 10^6 sets $\mathcal{Q}(\mathcal{D})$ having a fixed sum of $m + n$ by the algorithm described in [13].
- For each set, we have computed its projections, the initial sets α_0, β_0 given by the algorithm described in [7], and then the algorithm **filling_operations2** is applied.

Table 1. The number of iterations in the algorithm **filling_operations2** applied to the reconstruction to Q-convex sets w.r.t. the horizontal and vertical directions

$l \backslash m + n$	10	30	50	70	90	110
1	996977	994865	996970	997909	998468	998764
2	3023	5134	3030	2091	1532	1236
3	0	1	0	0	0	0

Table 1 gives the frequencies of the number l of iterations. In this experiment we have always $l \leq 3$. So it seems reasonable to make the conjecture that l is bounded by a constant. With it, the time-complexity of **filling_operations2** is $O(N^2)$.

6 Conclusion and Perspectives

In this paper, we presented an implementation of the five filling-operations in $O(N^2 \log N)$ -time, where N is the size of the projections. The new implementa-

tion permitted to reconstruct Q -convex sets in $O(N^4 \log(N))$ -time from projections in the same directions as the ones used for Q -convexity. This represented an improvement of the previous fastest algorithm which run in $O(N^5)$ -time.

The introduction of the fifth operation has permitted to reduce the complexity of the reconstruction because it allowed to fix two points instead of four. Additional considerations could perhaps induce a faster algorithm. In particular, the phase which fixes some points (bases) could be faster in the case of three directions and more, because in this case experiments show that these bases are very rarely needed (see [14, Annexe B]). This could lead to an algorithm with a complexity of $O(N^2)$ -time, but at the moment we have only experimental hints.

Acknowledgments

This work was partially supported by the NSF Grant DMS 0306215, the OTKA Grant T048476 and the CNRS program IRMC.

References

1. Gebala, M.: The reconstruction of convex polyominoes from horizontal and vertical projections. In: Proc. of SOFSEM '98. Volume 1521 of LNCS. (1998) 350–359
2. Brunetti, S., Del Lungo, A., Del Ristoro, F., Kuba, A., Nivat, M.: Reconstruction of 4- and 8-connected convex discrete sets from row and column projections. *Linear Algebra Appl.* **339** (2001) 37–57
3. Ryser, H.J.: Combinatorial properties of matrices of zeroes and ones. *Canad. J. Math.* **9** (1957) 371–377
4. Kuba, A.: Reconstruction of two-directionally connected binary patterns from their two orthogonal projections. *Comp. Vis. Graph. Image process.* **27** (1984) 249–265
5. Woeginger, G.H.: The reconstruction of polyominoes from their horizontal and vertical projections. *Inform. Process. Lett.* **77**(5-6) (2001) 225–229
6. Barcucci, E., Del Lungo, A., Nivat, M., Pinzani, R.: Reconstructing convex polyominoes from horizontal and vertical projections. *Theoret. Comput. Sci.* **155** (1996) 321–347
7. Brunetti, S., Daurat, A.: An algorithm reconstructing convex lattice sets. *Theoret. Comput. Sci.* **304** (2003) 35–57
8. Brunetti, S., Daurat, A.: Reconstruction of discrete sets from two or more X-rays in any direction. In: Proc. of IW CIA 2000, Université de Caen (2000) 241–258
9. Knuth, D.E.: Balanced Trees (section 6.2.3). In: *Sorting and Searching*. Volume 3 of *The Art of Computer Programming*. Addison-Wesley (1998) 458–475
10. Brunetti, S., Daurat, A.: Reconstruction of Q -convex sets. In Herman, G.T., Kuba, A., eds.: *Advances in Discrete Tomography and its Applications*. Appl. Numer. Harmon. Anal. Birkhäuser (To Appear)
11. Gardner, R.J., Gritzmann, P.: Discrete tomography: Determination of finite sets by X-rays. *Trans. Amer. Math. Soc.* **349** (1997) 2271–2295
12. Daurat, A.: Determination of Q -convex sets by X-rays. *Theoret. Comput. Sci.* **332** (2005) 19–45
13. Brunetti, S., Daurat, A.: Random generation of Q -convex sets. *Theoret. Comput. Sci.* **347** (2005) 393–414
14. Daurat, A.: Convexité dans le plan discret. Application la tomographie. PhD thesis, LLAIC1, and LIAFA Université Paris 7 (2000)