Image Anal Stereol 2004;23:177-183
Original Research Paper

A TIME-OPTIMAL ALGORITHM FOR THE ESTIMATION OF CONTACT DISTRIBUTION FUNCTIONS OF RANDOM SETS

JOHANNES MAYER

Department of Applied Information Processing and Department of Stochastics, University of Ulm, D-89069 Ulm, Germany
e-mail: jmayer@mathematik.uni-ulm.de
(Accepted June 18, 2004)

ABSTRACT

This paper presents a linear-time and therefore time-optimal algorithm for the estimation of distance distribution functions and contact distribution functions of random sets. The distance distribution function is the area fraction of a dilated set, where this function depends on the size of the structuring element used for the dilation. Furthermore, contact distribution functions are related to distance distribution functions. Minus-sampling estimators are used for the estimation.

Keywords: algorithm, contact distribution, distance distribution, estimation, estimator, minus-sampling.

INTRODUCTION

Random sets, *i.e.*, random variables whose values are sets, have successfully been applied to model patterns in various fields, such as materials science (Ohser and Mücklich, 2000), medicine (Mattfeldt *et al.*, 1996), physics (Mecke, 1998), and astrophysics (Mecke *et al.*, 1994). In their statistical analysis, so-called contact distribution functions (cdf's) are very important, *cf.* Serra (1982); Stoyan *et al.* (1995); Ohser and Mücklich (2000). For a stationary random closed set Ξ, the spherical contact distribution function is the distribution function of the random (minimum Euclidean) distance of an arbitrary point x outside of Ξ to (the closest point at the boundary $\partial\Xi$ of) Ξ. The Euclidean metric yields the spherical contact distribution function. Using other metrics, such as the city-block metric, yields further important contact distribution functions. There are various estimators for contact distribution functions, *cf.* Stoyan *et al.* (1995). The classical estimators are minus-sampling estimators. Stoyan *et al.* (2001) point out that these estimators for cdf's are as good as the more sophisticated estimators concerning the mean squared error. Therefore, the classical (minus-sampling) estimators are used in the following.

The cdf can be expressed in terms of the area fraction of the stationary random closed set dilated by a structuring element with variable size. In case of the spherical cdf, the structuring element is the unit-sphere.

When working with real data, binary images, *i.e.*, digital images with two phases, are the usual basis for the estimation. They can be interpreted as discretized samples of random sets. Thus, the minus-sampling estimators have to be formulated for this case. This is not difficult. However, it turns out, that the running time of algorithms computing these estimators is usually not linear in the number of pixels of the image, but somewhat in between linear and quadratic – even when a distance transform is used and computed in linear-time. The presented novel algorithm additionally uses an efficient data structure to compute the minus-sampling estimator in linear time. In the following, the two-dimensional case is covered. However, the results can be generalized analogously to higher dimensions (with not much difficulty).

PRELIMINARIES

Let A and B be two arbitrary subsets of $\mathbb{R}^2$. By $A^c := \{x \in \mathbb{R}^2 : x \notin A\}$ the *complement* of the set A is denoted. Let c be a real constant. Then $cA := \{cx : x \in A\}$ is the *scalar multiplication* of c and A. The set $A \oplus B := \{x+y : x \in A, y \in B\}$ is called the *Minkowski addition* of A and B. The *Minkowski subtraction* of B from A is defined as $A \ominus B := (A^c \oplus B)^c$. The set B is called the *structuring element* in the Minkowski addition and subtraction above. A structuring element B_m can be defined using a norm m as $B_m := \{x \in \mathbb{R}^2 : m(x) \leq 1\}$. The disc $b(o,r)$ with radius r around the origin is, thus, equal to the set $rB_{m_{\mathscr{E}}}$, where $m_{\mathscr{E}}$ denotes the Euclidean norm. (Note that each norm $m(\cdot)$ has an associated metric $d(x,y) := m(x-y)$.)

Let X be a set from the extended convex ring over $\mathbb{R}^2$. Then $A(X)$ denotes the *area*. Let Ξ be a stationary random closed set over the extended convex ring. Let

W be a non-empty, bounded convex set, the so-called *sampling window*. $\Xi \cap W$ is the random set Ξ observed within the sampling window W. The *specific area* or *area fraction* of Ξ is defined as

$$A_A(\Xi) := \frac{\mathbb{E}[A(\Xi \cap W)]}{A(W)}. \tag{1}$$

Let x be a (fixed) arbitrary point of $\mathbb{R}^2$. Then $A_A(\Xi) = \mathbb{P}(o \in \Xi) = \mathbb{P}(x \in \Xi)$ holds. Thus, $\widehat{A_A}(\Xi) = \mathbb{1}_\Xi(x)$ is a simple and unbiased estimator for $A_A(\Xi)$. Given a finite set of points G of $\mathbb{R}^2$, an improved estimator for the area fraction is given by $\widehat{A_A}(\Xi) = \frac{1}{\#G}\sum_{x\in G}\mathbb{1}_\Xi(x)$ which is also unbiased and known as the *point count method*. (#G denotes the cardinality of the set G.) G can be seen as the vertices of a grid – the locations of the pixels of a binary image – and $\mathbb{1}_\Xi(x)$ can be seen as the values of the pixels of a binary image. Thus, the previous estimator can easily be used for samples of Ξ given as a binary image.

The area fraction can be studied for so-called *parallel sets* $\Xi \oplus rB$ of Ξ, where B is a compact and convex subset of $\mathbb{R}^2$. This yields the following morphological function, where this function depends on the size r of the structuring element B:

$$A_A(r) := A_A(\Xi \oplus rB) \tag{2}$$

$A_A(r)$ is sometimes called the *distance distribution function*. A normalized version of $A_A(r)$ is the so-called *contact distribution function* $H^B(r)$ which can be defined as

$$H^B(r) := 1 - \frac{1 - A_A(r)}{1 - A_A(0)} \tag{3}$$

for $r \geq 0$. For example, choosing $B = b(o,1)$ yields the well-known *spherical contact distribution function*, denoted $H^s(r)$.

Let $B = B_m$ for some norm m. Then, $A_A(r)$ is the distribution function of the random (minimal) distance between an arbitrarily chosen point $x \in \mathbb{R}^2$ and Ξ. Furthermore, $H^B(r)$ is the distribution function of the random (minimal) distance between an arbitrarily chosen point $x \in \Xi^c$ and Ξ.

Let G be a finite grid over $\mathbb{R}^2$. Given a metric d, the *distance transform* of the set X on G is defined as

$$D_d^X(x) := \min\{d(x,y) : y \in G \cap X^c\} \tag{4}$$

for each $x \in G$. It associates each point with its minimal distance to X^c (on the grid) – a different distance in comparison to the distance distribution and the contact distribution. For a binary image being a discretization of the set X on a rectangular grid G, this transform can be computed in linear time (in the number of pixels) for various metrics, such as the Euclidean, the city-block, the maximum, etc., *cf.* Rosenfeld and Pfaltz (1966; 1968); Danielsson (1980); Soille (1991); Vincent (1991); Ragnemalm (1993); Breu (1995).

SOME ESTIMATION ALGORITHMS FOR DISTANCE DISTRIBUTION FUNCTIONS

As before, let Ξ be a stationary random closed set over the extended convex ring, B a compact and convex subset of $\mathbb{R}^2$, and W a non-empty, bounded convex set, namely the sampling window. A minus-sampling estimator for the distance distribution function is the following:

$$\widehat{A_A}(r) = \frac{A([(\Xi \cap W) \oplus rB] \cap (W \ominus rB))}{A(W \ominus rB)} \tag{5}$$

for $r \geq 0$ such that $A(W \ominus rB) > 0$. *Edge-effects* are removed through the reduction of the sampling window from W to $W \ominus rB$.

Let G be a finite rectangular grid over $\mathbb{R}^2$ for the discretization of Ξ and W. The area of the unit cell of G is called A_0. Furthermore, let $B = B_m$ for some norm m and let d be the metric associated with m. Then, the above estimator can be discretized as

$$\begin{aligned}\widetilde{A_A}(r) &= \frac{A_0 \sum_{x\in G} \mathbb{1}_{[0,r]}(D_d^{\Xi^c}(x)) \cdot \mathbb{1}_{(r,\infty]}(D_d^W(x))}{A_0 \sum_{x\in G} \mathbb{1}_{(r,\infty]}(D_d^W(x))} \\ &= \frac{\sum_{x\in G} \mathbb{1}_{[0,r]}(D_d^{\Xi^c}(x)) \cdot \mathbb{1}_{(r,\infty]}(D_d^W(x))}{\sum_{x\in G} \mathbb{1}_{(r,\infty]}(D_d^W(x))}\end{aligned} \tag{6}$$

for $r \geq 0$ such that the denominator does not vanish. Informally speaking, this estimator gives the fraction of all pixels with (minimal) distance greater than r to the boundary of the sampling window W that have (minimal) distance at most r to Ξ.

The direct algorithm

The direct algorithm to compute the above discretized estimator first computes the distance transforms of Ξ^c and W (in linear time). (Note that the distance transform of W can easily be computed directly if W is a rectangle.) Thereafter, it computes the above fraction for each value of r.

To make the description of the direct algorithm precise, the following listing is an implementation of this algorithm in Java. However, an implementation in C, C++, or C# would look very similar. For simplicity considerations, the images are represented by two-dimensional arrays. The complementation of a binary image is done by the method `C()` and the distance transform is computed by the method `DT()`. These methods are used as black boxes.

```
1 float[] estimateDirect(boolean[][] im,
2     boolean[][] sw, int metric)
3 {
4     // width and height of im and sw
5     int width = im.length;
6     int height = im[0].length;
7
8     // distance transform of sw
9     // and the complement of im
10    float[][] dim = DT(C(im), metric);
11    float[][] dsw = DT(sw, metric);
12
13    // arrays for numerator and denominator
14    int min = Math.min((width+1)/2,
15                       (height+1)/2);
16    int[] num = new int[min];
17    int[] denom = new int[min];
18    for (int r = 0; r < min; r++)
19        num[r] = denom[r] = 0;
20
21    // process each pixel for all values of r
22    for (int r = 0; r < min; r++)
23        for (int y = 0; y < height; y++)
24            for (int x = 0; x < width; x++) {
25                // increment denominator
26                if (r < dsw[x][y])
27                    denom[r]++;
28
29                // increment numerator
30                if (r >= dim[x][y]
31                    && r < dsw[x][y])
32                    num[r]++;
33            }
34
35    // compute the estimator for all r
36    // such that denom[r] != 0
37    for (int r = 0; r < min; r++)
38        num[r] = (denom[r] == 0)
39                    ? Float.NaN
40                    : (float) num[r]
41                      / (float) denom[r];
42
43    return num;
44 }
```

The parameter `im` is a discretization of a sample of Ξ and the parameter `sw` is a discretization of W, *i.e.*, `im` and `sw` are binary images. It is assumed that both have the same size (computed in lines 5 and 6). The parameter `metric` is passed to the method that computes the distance transform. It determines the structuring element B_m. The numerator of the estimator is stored in the array `num`; the denominator is stored in the array `denom`. For values of r such that the denominator of the estimator vanishes, the estimator is set to "not a number" (`Float.NaN`), since it cannot be determined in this case.

The computation of the fraction for one value of r takes $\Theta(\#G)$ time and there are $\Theta(\sqrt{\#G})$ values of r (if the image is quadratic). Thus, the direct algorithm has in each case time-complexity $\Theta(\#G\sqrt{\#G})$. This is much more than linear time, *i.e.*, $\Theta(\#G)$. Consider, for example, a 1000×1000 pixels image, *i.e.*, $\#G = 10^6$. Then, $\#G\sqrt{\#G} = 10^9$ is significantly more than just $\#G$.

An improved algorithm

The direct algorithm processed the image once for each value of r. An obvious improvement is, thus, to process the image only once and to compute for each pixel the range within which it is counted in the numerator resp. the denominator. Using an array for the numerator and the denominator, the elements of this array have to be initialized with zero at the beginning and incremented by one in the computed range for each pixel.

The following Java implementation makes the informal description of the improved algorithm precise:

```
1 float[] estimateImproved(boolean[][] im,
2     boolean[][] sw, int metric)
3 {
4     // width and height of im and sw
5     int width = im.length;
6     int height = im[0].length;
7
8     // distance transform of sw
9     // and the complement of im
10    float[][] dim = DT(C(im), metric);
11    float[][] dsw = DT(sw, metric);
12
13    // arrays for numerator and denominator
14    int min = Math.min((width+1)/2,
15                       (height+1)/2);
16    int[] num = new int[min];
17    int[] denom = new int[min];
18    for (int r = 0; r < min; r++)
19        num[r] = denom[r] = 0;
20
21    // process all pixels
22    for (int y = 0; y < height; y++)
23        for (int x = 0; x < width; x++) {
24            float d_xi = dim[x][y];
25            float d_wc = dsw[x][y];
26            d_xi = Math.min(d_xi, d_wc);
27
28            // increment denominator
29            for (int r = 0; r < d_xi; r++)
30                denom[r]++;
31
32            // increment numerator
33            // and denominator
34            for (int r = d_xi; r < d_wc; r++) {
35                num[r]++;
36                denom[r]++;
37            }
```

```
38          }
39
40      // compute the estimator for all r
41      // such that denom[r] != 0
42      for (int r = 0; r < min; r++)
43          num[r] = (denom[r] == 0)
44                       ? Float.NaN
45                       : (float) num[r]
46                         / (float) denom[r];
47
48      return num;
49  }
```

It is interesting to study, how many increments are necessary for all pixels together. This yields the amortised time complexity of this algorithm. The number of all increments is obviously between once and twice the number of increments of the denominator. Thus, only the number of increments of the denominator is studied. For simplicity considerations let the sampling window be quadratic with even edge length n. If the distance of a pixel to W^c is d, this leads to d increments of the denominator. It is easily verified that there are $4(2k)-4$ pixels with distance $\frac{n}{2}-k+1$ to W^c for $k=1,\ldots,\frac{n}{2}$. Thus, the total number of increments of the denominator is in each case

$$\begin{aligned}&\sum_{k=1}^{n/2}(4(2k)-4)\left(\frac{n}{2}-k+1\right)\\ =\;&\frac{n^3+3n^2+2n}{6}=\Theta(n^3)=\Theta(\#G\sqrt{\#G}),\end{aligned}\quad(7)$$

with $n=\sqrt{\#G}$. Thus, this algorithm also has time-complexity $\Theta(\#G\sqrt{\#G})$ in each case.

The novel algorithm

The problem of the improved algorithm is that in each case $\Theta(\sqrt{\#G})$ increments are necessary for a huge subset of the pixels. However, these increments are done within a continuous range of array elements. Now, the novel algorithm uses the improved algorithm, but it has a more sophisticated data structure to represent the numerator and the denominator than just a simple array. The basic requirement for this data structure is that only a constant number of increments (and decrements) is necessary to increment a whole (continuous) range (with the same value). Figure 1 shows two different representations of the same integer sequence $(1,2,1,3,3,2)$.

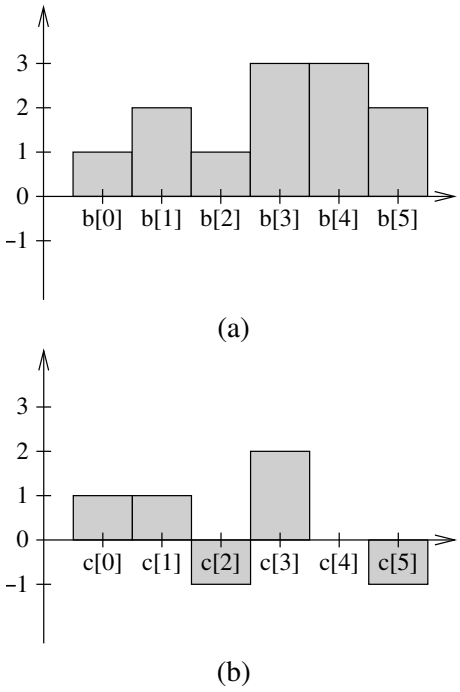

Fig. 1. *Two array representations of the same integer sequence.*

(a) shows the direct representation of the sequence. In (b), the first element of the sequence and the differences to the previous element are stored, *i.e.*, $c[i]=b[i]-b[i-1]$ for $i>0$ and $c[0]=b[0]$. The array b can, thus, easily be restored from c as follows:

$$b[0]=c[0]\quad\text{and}\quad b[i]=b[i-1]+c[i],\qquad(8)$$

for $i>0$. To increment the elements of the sequence in the range $[n,n+m-1]$ it takes m increments in representation (a) and only two increments resp. decrements in representation (b), namely an increment of $c[n]$ and a decrement of $c[n+m]$. Therefore, during the processing of the pixels, the numerator and the denominator should be stored using representation (b). At the end, this representation is converted to the representation (a).

The following Java implementation gives a precise description of the novel algorithm:

```
1 float[] estimateNovel(boolean[][] im,
2     boolean[][] sw, int metric)
3 {
4     // width and height of im and sw
5     int width = im.length;
6     int height = im[0].length;
7
8     // distance transform of sw
9     // and the complement of im
10    float[][] dim = DT(C(im), metric);
11    float[][] dsw = DT(sw, metric);
12
```

```
13      // arrays for numerator and denominator
14      // in representation (b)
15      int min = Math.min((width+1)/2,
16                         (height+1)/2);
17      int[] num = new int[min];
18      int[] denom = new int[min];
19      for (int r = 0; r < min; r++)
20          num[r] = denom[r] = 0;
21
22      // process all pixels
23      for (int y = 0; y < height; y++)
24          for (int x = 0; x < width; x++) {
25              float d_xi = dim[x][y];
26              float d_wc = dsw[x][y];
27
28              // increment denominator
29              if (d_wc > 0)
30                  denom[d_wc-1]++;
31
32              // increment numerator
33              if (d_xi < d_wc) {
34                  num[d_xi]++;
35                  num[d_wc]--;
36              }
37          }
38
39      // reconstruct representation (a)
40      // in num and denom
41      for (int r = 1; r < min; r++)
42          num[r] += num[r-1];
43      for (int r = min-1; r > 0; r--)
44          denom[r-1] += denom[r];
45
46      // compute the estimator for all r
47      // such that denom[r] != 0
48      for (int r = 0; r < min; r++)
49          num[r] = (denom[r] == 0)
50                       ? Float.NaN
51                       : (float) num[r]
52                         / (float) denom[r];
53
54      return num;
55  }
```

The numerator of the estimator is stored in the array `num` according to representation (b). The denominator is stored in the array `denom` slightly different; it contains the changes from right to left and not from left to right. All pixels are processed in the lines 23–37. For each pixel, `d_xi` is its (minimal) distance to Ξ and `d_wc` is its (minimal) distance to W^c; *cf.* lines 25–26. The numerator has to be incremented in the range $[$`d_xi`$,\ldots,$`d_wc`$-1]$, which requires one increment and one decrement; *cf.* lines 33–36. Analogously, the denominator has to be incremented in the range $[0,\ldots,$`d_wc`$-1]$. Thus, at most one increment is necessary, since the changes are stored from right to left; *cf.* lines 29–30. After the integer sequences for the numerator and the denominator have been reconstructed in the lines 41–44, the estimator can be computed in the lines 48–52.

Concluding, the time-complexity of the novel algorithm is in each case $\Theta(\#G)$, where $\#G$ is the number of pixels of the image. Therefore, this algorithm is time-optimal, because linear time is the best possible case.

EMPIRICAL STUDY

So far, "only" asymptotical results have been presented. They show that the novel algorithm is at least of theoretical importance. But the constants hidden in the Θ-notation may be so huge that the novel algorithm could be of no practical importance.

For this reason, the (stationary) Boolean model with discs of radius 10 pixels and given hypothetical area fraction $p \in \{0.1, 0.2, \ldots, 0.9\}$ has been simulated for quadratic sampling windows of sizes (*i.e.*, edge lengths) 512, 1024, 2048, and 4096 pixels. Samples of such Boolean models are shown in Figure 2.

The experiments have been conducted on an AMD Athlon 900 with 1.5 GB of main memory, SuSE Linux 7.3, and IBM JDK 1.3.0. The command `java` was called with the option `-mx1024m`.

For each sampling window size and each hypothetical area fraction, ten samples of the Boolean model have been simulated. For each algorithm (direct, improved, and novel), the mean of the execution times was taken over the ten executions (with the different samples). Since the Euclidean distance transform, which has been used for the experiments – the algorithm of Danielsson (1980) has been implemented – is very time consuming, the execution times have been determined excluding (*cf.* Table 1) and including (*cf.* Table 2) the time necessary for the distance transforms.

Both tables show that the novel algorithm is also in practice much faster than the direct and the improved algorithm. The novel algorithm is for the given image sizes at best 68 times as fast as the improved algorithm and 1887 times as fast as the direct algorithm (excluding the time necessary for the distance transform). This is a significant improvement.

RELATED WORK

Most publications about estimators for contact distribution functions make no statement on the implementation of these estimators. The only publication (known to the author) that contains explicit descriptions of algorithms is that of Bhattacharya (2003). These descriptions are quite informal and indicate that the direct (or, at best, the improved) algorithm has been used.

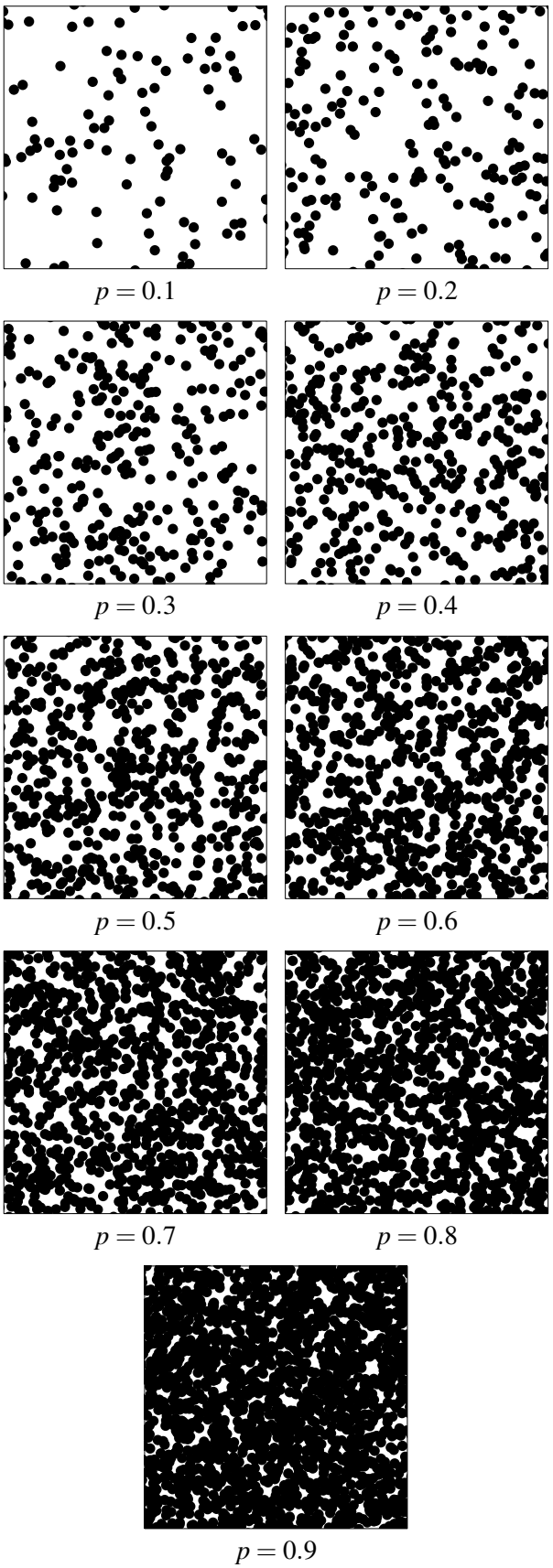

Fig. 2. *Samples of the Boolean model with discs of radius 10 pixels and hypothetical area fraction p within a sampling window sized* 512×512 *pixels.*

Klaus Mecke confirmed that he has not yet seen the presented "novel" algorithm been published.

CONCLUSION

To compute the minus-sampling estimator for distance distribution functions, it has been shown that a straightforward approach is not sufficient. Therefore, a novel efficient algorithm has been presented. This algorithm has in each case linear time-complexity, which is optimal.

Although, the algorithms are only given for the two-dimensional case, they can easily be adapted to higher dimensions.

Table 1. *Execution times of the novel (above), the improved (middle), and the direct (below) algorithm (excluding the execution time for the distance transforms).*

Hypothetical Area Fraction p	Size n (in pixels)			
	512	1024	2048	4096
0.1	0.020 s	0.077 s	0.31 s	1.25 s
	0.13 s	0.97 s	8.00 s	70.1 s
	4.48 s	34.6 s	289 s	2338 s
0.2	0.021 s	0.080 s	0.31 s	1.25 s
	0.13 s	1.07 s	8.34 s	64.9 s
	4.37 s	36.1 s	284 s	2279 s
0.3	0.019 s	0.076 s	0.32 s	1.27 s
	0.14 s	1.05 s	9.17 s	70.8 s
	4.43 s	34.5 s	277 s	2287 s
0.4	0.020 s	0.078 s	0.31 s	1.20 s
	0.13 s	1.05 s	10.3 s	72.2 s
	4.39 s	35.2 s	282 s	2226 s
0.5	0.017 s	0.075 s	0.31 s	1.18 s
	0.13 s	1.11 s	11.0 s	75.2 s
	4.35 s	34.6 s	282 s	2227 s
0.6	0.018 s	0.076 s	0.28 s	1.19 s
	0.13 s	1.23 s	9.66 s	76.9 s
	4.26 s	34.3 s	276 s	2158 s
0.7	0.018 s	0.077 s	0.29 s	1.15 s
	0.14 s	1.12 s	10.1 s	76.6 s
	4.12 s	34.0 s	257 s	2127 s
0.8	0.018 s	0.074 s	0.29 s	1.18 s
	0.13 s	1.10 s	9.86 s	79.7 s
	4.15 s	33.3 s	262 s	2102 s
0.9	0.018 s	0.071 s	0.29 s	1.14 s
	0.14 s	1.08 s	9.43 s	76.2 s
	4.07 s	31.8 s	261 s	2069 s

Table 2. *Execution times of the novel (above), the improved (middle), and the direct (below) algorithm (including the execution time for the distance transforms).*

Hypothetical Area Fraction p	Size n (in pixels)			
	512	1024	2048	4096
0.1	0.044 s	1.87 s	7.08 s	28.2 s
	0.56 s	2.65 s	14.7 s	96.8 s
	4.91 s	36.3 s	296 s	2365 s
0.2	0.043 s	1.70 s	6.81 s	27.2 s
	0.54 s	2.68 s	14.8 s	90.8 s
	4.79 s	37.7 s	290 s	2305 s
0.3	0.042 s	1.68 s	6.69 s	26.7 s
	0.53 s	2.62 s	15.5 s	96.0 s
	4.83 s	36.1 s	283 s	2313 s
0.4	0.041 s	1.62 s	6.48 s	25.9 s
	0.51 s	2.57 s	16.5 s	96.6 s
	4.78 s	36.7 s	288 s	2251 s
0.5	0.039 s	1.62 s	6.41 s	24.9 s
	0.51 s	2.62 s	16.5 s	98.7 s
	4.73 s	36.1 s	288 s	2251 s
0.6	0.038 s	1.53 s	6.06 s	24.5 s
	0.49 s	2.68 s	15.4 s	99.5 s
	4.62 s	35.8 s	282 s	2181 s
0.7	0.038 s	1.49 s	5.93 s	23.2 s
	0.48 s	2.51 s	15.6 s	98.5 s
	4.47 s	35.4 s	263 s	2149 s
0.8	0.037 s	1.41 s	5.59 s	22.6 s
	0.47 s	2.43 s	15.1 s	101 s
	4.49 s	34.7 s	268 s	2124 s
0.9	0.035 s	1.39 s	5.49 s	22.1 s
	0.46 s	2.36 s	14.6 s	96.5 s
	4.40 s	33.1 s	266 s	2090 s

Experimental results show that the novel algorithm is also in practice much better than simpler algorithms. The Boolean model has been chosen for the experiments. Since there is no major dependency between the running-time of the algorithms and the image data, this is representative.

The presented algorithms can also be used to compute estimators of contact distribution functions, since distance distribution functions and contact distribution functions are related to each other as shown.

ACKNOWLEDGMENTS

The author is grateful to Volker Schmidt for valuable comments. He also wants to thank Klaus Mecke for helpful remarks which initiated the experimental part of this work.

REFERENCES

Bhattacharya A (2003). Measurement of Contact Distribution Functions on 2d Binary Images Using Image Processing Techniques. Diploma Thesis, University of Kaiserslautern.

Breu H, Gil J, Kirkpatrick D, Werman M (1995). Linear Time Euclidean Distance Algorithms. IEEE Trans Pattern Anal 17:529-33.

Danielsson PE (1980). Euclidean Distance Mapping. Comput Vision Graph 14:227-48.

Lang C, Ohser J, Hilfer R (2001). On the Analysis of Spatial Binary Images. J Microsc 202:1-12.

Mattfeldt T, Schmidt V, Reepschläger D, Rose C, Frey H (1996). Centered Contact Density Functions for the Statistical Analysis of Random Sets. J Microsc 183:158-69.

Mecke K (1998). Integral Geometry and Statistical Physics. Int J Mod Phys B 12:861-99.

Mecke K, Buchert T, Wagner H (1994). Robust Morphological Measures for Large-Scale Structure in the Universe. Astron Astrophys 288:697-704.

Ohser J, Mücklich F (2000). Statistical Analysis of Microstructures in Materials Science. Chichester: John Wiley & Sons.

Ohser J, Steinbach B, Lang C (1998). Efficient Texture Analysis of Binary Images. J Microsc 188:20-8.

Ragnemalm I (1993). The Euclidean Distance Transform. Ph.D. Thesis, University of Linköping.

Rosenfeld A, Pfaltz J (1966). Sequential Operations in Digital Picture Processing. J ACM 13:471-94.

Rosenfeld A, Pfaltz J (1968). Distance Functions in Digital Pictures, Patt Recogn 1:33-61.

Serra J (1982). Image Analysis and Mathematical Morphology. London: Academic Press.

Soille P (1991). Spatial Distributions from Contour Lines: An Efficient Methodology Based on Distance Transformations. J Vis Commun Image R 2:138-50.

Stoyan D, Kendall WS, Mecke J (1995). Stochastic Geometry and its Applications. Chichester: John Wiley & Sons.

Stoyan D, Stoyan H, Tscheschel A, Mattfeldt T (2001). On the Estimation of Distance Distribution Functions for Point Processes and Random Sets. Image Anal Stereol 20:65-9.

Vincent L (1991). Exact Euclidean Distance Function by Chain Propagations. Proc IEEE Comput Vision Patt R 520-5.