ALGORITHMS FOR OPENINGS OF BINARY AND LABEL IMAGES WITH RECTANGULAR STRUCTURING ELEMENTS

MARC VAN DROOGENBROECK
Department of Electricity, Electronics and Computer Science
Montefiore B-28, Sart Tilman, B-4000 Liège, Belgium
URL: http://www.ulg.ac.be/telecom

Abstract. Two new families of algorithms for computing openings of binary and label images are presented in this paper. The first family of algorithms is based on an horizontal scan, and a vertical scan that takes the result of the horizontal scan as input. With the results of these two scans it is possible to compute an opening with respect to a rectangle of any size. The second family of algorithms is based on the construction of an exhaustive list of rectangles included in an object. Rectangles of this list all have a maximal extension, i.e. no larger rectangle included in a region contains them. The opening then results from filling the output image with rectangles larger than the structuring element. After a description of the algorithms we provide a comparison of several algorithms in terms of computation time efficiency. The comparison shows that some of the new algorithms advantageously compete with existing algorithms.

Key words: Algorithm, binary opening, opening of a label image

1. Introduction

While most of the research efforts on efficient algorithms were focused on gray level openings, binary openings still play an important role in industrial applications. As often in mathematical morphology, a direct transcription of the definition is an inefficient way to implement operators. Most efficient algorithms result either from the operator properties or from a geometrical interpretation.

In this paper we propose alternatives to implement binary openings with respect to rectangles. The algorithms are inspired by the geometrical interpretation of a binary opening. After a brief reminder we review existing algorithms, describe new algorithms and compare the algorithms.

1.1. Definitions

We first recall the definitions and notations used in this paper. Let X, B be discrete sets defined on Z^2. The erosion $X \ominus B$ and dilation $X \oplus B$ of a set X are defined as:

$$\begin{aligned} X \ominus B &= \bigcap_{b \in B} X_{-b} \\ X \oplus B &= \bigcup_{b \in B} X_b \end{aligned}$$

The opening $X \circ B$ results from an erosion followed by a dilation: $X \circ B = (X \ominus B) \oplus B$. A set B of size n, denoted nB, is usually defined as

$$nB = \underbrace{B \oplus B \oplus \ldots \oplus B}_{n-1 \text{ dilations}} \tag{1}$$

In this paper we adopt a different convention for convenience and assume that, for linear structuring elements, nH is a n-pixels wide segment. We also assume the structuring element B to be a rectangle.

1.2. Review of existing algorithms

Suppose $B = nH \oplus mV$ where H, V respectively are horizontal and vertical segments, and we need to compute $X \circ (nH \oplus mV)$. A direct application of the definition leads to an algorithm with a complexity proportional to $n \times m$, i.e. to the area of the structuring element. A better approach, called *linear decomposition*, is based on the chain rule and results in $(((X \ominus nH) \ominus mV) \oplus nH) \oplus mV$. The complexity of the linear decomposition is proportional to $n + m$. Although better algorithms exist, some commercial softwares implement binary openings with the linear decomposition algorithm because of its simplicity. PECHT [6] and VAN DEN BOOMGAARD [8] have established a *logarithmic decomposition* which further reduces the computation time. The logarithmic decomposition suppresses the redundancy inherent to recursive dilations with a unique structuring element. For example, if $\partial(B)$ denotes the border of B, it can be shown that $9B = B \oplus \partial(B) \oplus \partial(B) \oplus \partial(2B) \oplus \partial(4B)$. According to the property that states that $B \oplus B = B \oplus \partial(B)$ (see [10]), other decompositions are possible. $9B$ could as well has been written as $B \oplus \partial(B) \oplus \partial(B) \oplus \partial(3B) \oplus \partial(3B)$. This formulation offers the small advantage that only two different types of dilation are needed instead of three.

A completely different approach to the implementation of openings was provided by CHAUDHURI [1] who extended the work of HUANG *et al.* [3]. Both authors based their algorithms on a local histogram computed in a sliding window. Contrary to other methods the computation time of histogram based algorithms does not depend on the size of the structuring element, but it depends on the image content.

In [11] VAN HERCK proposed a 1-D algorithm based on separability and a combination of block recursive series which are evaluated forwards and backwards. Similar ideas led GIL and KIMMEL [2] to the definition of efficient algorithms. In their scheme an opening is not seen as an erosion followed by a dilation, but rather as the supremum of translated versions of a function f.

Although all these algorithms were developed for a function f they are applicable to binary openings. Algorithms dedicated to binary openings have also been proposed. VAN VLIET [12] *et al.*, SCHMITT [7], and later VINCENT [13] provided methods which analyze the border of both X and B. LAY [5] developed an algorithm based on the distance function. The geometric interpretation, that an opening is the union of translated B_p included in X, led VAN DROOGENBROECK [9] to propose a propagation algorithm for binary openings with a rectangle and a method for the direct computation of linear openings (also applicable to functions). Although $X \circ (nH \oplus mV) \neq (X \circ nH) \circ mV$, direct linear opening algorithms can still help reducing the computation times as $X \circ (nH \oplus mV) = (((X \ominus nH) \ominus mV) \oplus nH) \oplus mV = (((X \ominus nH) \ominus$

$mV) \oplus mV) \oplus nH = ((X \ominus nH) \circ mV) \oplus nH$. Algorithms based on the geometric interpretation of openings have the drawback that they depend on the content of the image which makes a hardware implementation harder. But implemented in software they appear to be competitive in terms of computing time.

2. First family of algorithms: 2D scan-based algorithms

The algorithms of the first family can be summarized as follows. In a first phase we scan the input image and build two matrices. These matrices contain all the information needed to perform an opening with a rectangle of any size. In the second phase the output image is filled with thresholded values of a matrix computed during the first scan.

Our algorithms, called "*2-D scan*" hereafter, are similar to fast erosion and dilation algorithms that use thresholded distance maps except that our techniques directly provide the opening and work with two matrices.

2.1. Description of the first phase

The first matrix, denoted $HOR[i,j]$ or HOR, is filled with the distance of each pixel contained in an object to the right border of that object. An horizontal scan from right to left provides the result. To compute the second matrix, denoted VER, the algorithm compares the values in the columns of HOR. $VER[i,j]$ is the length of the vertical segment of HOR that has all its value larger or equal to $HOR[i,j]$. Fig. 1 gives a simplified version of the corresponding program fragment. A worked out example is provided in Fig. 2.

2.2. Description of the second phase

The first phase does not depend on the size of the structuring element; it is a generic phase that provides all the rectangles included in X. Because VER was built on HOR, it appears that HOR and VER suffice to compute the opening with a rectangle of any size. In the second phase, each pair of elements $HOR[i,j]$, $VER[i,j]$ is analyzed in order to find appropriate values. Suppose we want to compute $X \circ (nH \oplus mV)$. If $HOR[i,j] \geq n$ and $VER[i,j] \geq m$ then the value $HOR[i,j]$ is copied in the output image, denoted OUT, not only at (i,j) but also in the column at neighboring positions where $nH \oplus mV$ fits. These neighboring positions (i,k) are such that $HOR[i,k] \geq HOR[i,j]$. Because of possible multiple accesses to $OUT[i,j]$, it is important that a new value is put in $OUT[i,j]$ only if it is larger than the previous value.

After all pairs have been analyzed OUT contains the run lengths of all the horizontal segments of $X \circ (nH \oplus mV)$. It then remains to fill the output image OUT to the right according to the values of $OUT[i,j]$. The second phase, when performed on the object of Fig. 2, is illustrated in Fig. 3.

2.3. Implementation issues

There are several ways to optimize the algorithm. The major optimization techniques are listed hereafter:

– *Better use of hardware and software characteristics*. Due to the linear structure of

```
**for all** $i, j$ **do**
    **if** $HOR[i,j] \neq 0$ **then**
        $length \leftarrow 1$
        /* Scan to the top */
        $k \leftarrow j - 1$
        **while** $HOR[i,k] \geq HOR[i,j]$ **do**
                $length \leftarrow length + 1$
                $k \leftarrow k - 1$
        **end while**
        /* Scan to the bottom */
        $k \leftarrow j + 1$
        **while** $HOR[i,k] \geq HOR[i,j]$ **do**
                $length \leftarrow length + 1$
                $k \leftarrow k + 1$
        **end while**
        $VER[i,j] \leftarrow length$
    **else**
        $VER[i,j] \leftarrow 0$
    **end if**
**end for**
```

Fig. 1. Program fragment for building VER.

```
  Original image            HOR                   VER
1 1 1 1 1 0 0 0 1 0   5 4 3 2 1 0 0 0 1 0   2 5 5 5 7 0 0 0 1 0
1 1 1 1 1 1 0 1 0 0   6 5 4 3 2 1 0 1 0 0   1 4 4 4 6 7 0 7 0 0
0 1 1 1 1 1 1 1 0 0   0 7 6 5 4 3 2 1 0 0   0 3 3 3 5 6 6 7 0 0
1 1 1 1 1 1 1 1 1 0   9 8 7 6 5 4 3 2 1 0   1 2 2 2 4 5 5 5 5 0
0 1 1 1 1 1 1 1 1 0   0 8 7 6 5 4 3 2 1 0   0 2 2 2 4 5 5 5 5 0
0 1 0 0 1 1 1 1 1 1   0 1 0 0 6 5 4 3 2 1   0 8 0 0 2 3 3 3 3 3
1 1 1 0 1 1 1 1 1 1   3 2 1 0 6 5 4 3 2 1   2 2 2 0 2 3 3 3 3 3
1 1 1 1 0 1 1 1 1 1   4 3 2 1 0 5 4 3 2 1   1 1 1 1 0 3 3 3 3 3
```

Fig. 2. A binary object and its corresponding two matrices HOR and VER.

memory blocks, accesses along a row in a 2-D array are faster than accesses along a column. In order to accelerate the computation of VER, HOR is transposed before being stored.

– *Avoid some computation redundancy*. If $HOR[i,j] = HOR[i,j-1]$ then (i,j) and $(i,j-1)$ belong to the same rectangle, and therefore have the same vertical extension. This means that $VER[i,j] = VER[i,j-1]$.
– *Avoid rewriting pixels in the output image*. As can be seen in Fig. 2 and Fig. 3,

$$
\begin{array}{ccc}
\texttt{Original image} & \texttt{INTERMEDIATE RESULT} & \texttt{OUT} \\
\texttt{1 1 1 1 1 0 0 0 1 0} & \texttt{0 }\underline{\texttt{4}}\texttt{ 0 0 0 0 0 0 0 0} & \texttt{0 1 1 1 1 0 0 0 0 0} \\
\texttt{1 1 1 1 1 1 0 1 0 0} & \texttt{0 }\underline{\texttt{5}}\texttt{ 0 0 0 0 0 0 0 0} & \texttt{0 1 1 1 1 1 0 0 0 0} \\
\texttt{0 1 1 1 1 1 1 1 0 0} & \texttt{0 5 0 0 }\underline{\texttt{4}}\texttt{ 0 0 0 0 0} & \texttt{0 1 1 1 1 1 1 1 0 0} \\
\texttt{1 1 1 1 1 1 1 1 1 0} & \texttt{0 5 0 0 }\underline{\texttt{5}}\texttt{ }\underline{\texttt{4}}\texttt{ 0 0 0 0} & \texttt{0 1 1 1 1 1 1 1 1 0} \\
\texttt{0 1 1 1 1 1 1 1 1 0} & \texttt{0 5 0 0 5 4 0 0 0 0} & \texttt{0 1 1 1 1 1 1 1 1 0} \\
\texttt{0 1 0 0 1 1 1 1 1 1} & \texttt{0 0 0 0 5 4 0 0 0 0} & \texttt{0 0 0 0 1 1 1 1 1 0} \\
\texttt{1 1 1 0 1 1 1 1 1 1} & \texttt{0 0 0 0 5 4 0 0 0 0} & \texttt{0 0 0 0 1 1 1 1 1 0} \\
\texttt{1 1 1 1 0 1 1 1 1 1} & \texttt{0 0 0 0 0 4 0 0 0 0} & \texttt{0 0 0 0 0 1 1 1 1 0}
\end{array}
$$

Fig. 3. Illustration of the second phase: the last image is the opening for $B = 4H \oplus 4V$. Positions where $HOR[i, j] \geq 4$ and $VER[i, j] \geq 4$ have been underlined.

some pixels (i, j) are part of several run lengths. To avoid that a pixel of OUT is addressed several times, the algorithm starts filling OUT, and compares the remaining length to the value of the intermediate matrix for each location. Then it adapts the remaining length if the value contained in the intermediate matrix is the largest.

All these optimizations were included in the implementations.

2.4. Generalization to label images

Algorithms based on the geometrical interpretation of an opening are easily extended to label images; in practice it is sufficient to check the value of each label. How the 2D scan-base algorithm applies to label images is illustrated in Fig. 4. In this example, the image is made of 4 regions labelled 1 to 4. The label 0 indicates that a pixel does not belong to any region.

3. Second family of algorithms: list-based algorithms

3.1. Principle

If all the rectangles included in X were known, it would be straightforward to keep the rectangles larger than B to perform $X \circ B$. However we do not need all the rectangles as some of them might be included in other rectangles. The second family of algorithms we propose builds a list of rectangles $\mathcal{L} = R_0, \ldots, R_L$ such that

– $\bigcup R_i = X$
– all rectangles of $\mathcal{L}$ are maximum in size: if $\exists R_k$ such that for one of the rectangles R_i included in $\mathcal{L}$, $R_i \subseteq R_k \subseteq X$ then R_k is included in $\mathcal{L}$
– the list does not contain any redundant rectangle: $\forall R_i$, R_j contained in $\mathcal{L}$, $R_i \nsubseteq R_j$ and $R_j \nsubseteq R_i$.

3.2. Summarized description

A complete description of the algorithm is beyond the scope of this paper because it would be necessary to discuss many pathological configurations and the program is about 1000 lines of `C` code long. We therefore limit the description to the major steps.

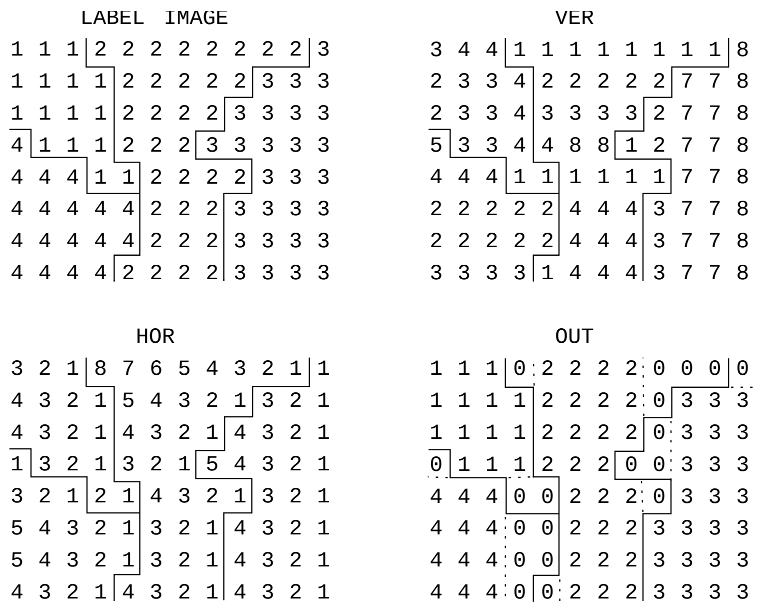
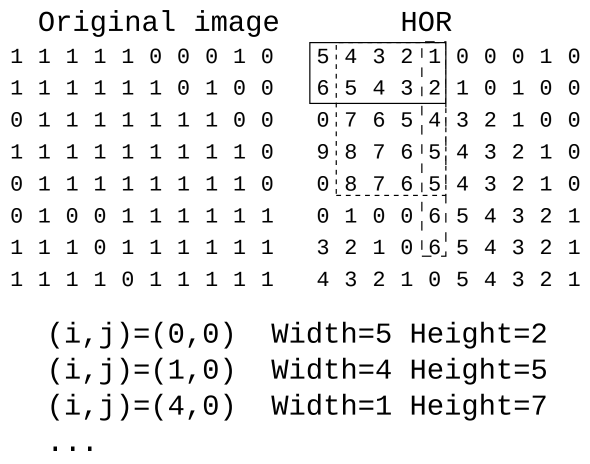

```
LABEL IMAGE                  VER
1 1 1 2 2 2 2 2 2 2 2 3      3 4 4 1 1 1 1 1 1 1 1 8
1 1 1 1 2 2 2 2 2 3 3 3      2 3 3 4 2 2 2 2 2 7 7 8
1 1 1 1 2 2 2 2 3 3 3 3      2 3 3 4 3 3 3 3 2 7 7 8
4 1 1 1 2 2 2 3 3 3 3 3      5 3 3 4 4 8 8 1 2 7 7 8
4 4 4 1 1 2 2 2 2 3 3 3      4 4 4 1 1 1 1 1 1 7 7 8
4 4 4 4 4 2 2 2 3 3 3 3      2 2 2 2 2 4 4 4 3 7 7 8
4 4 4 4 4 2 2 2 3 3 3 3      2 2 2 2 2 4 4 4 3 7 7 8
4 4 4 4 2 2 2 2 3 3 3 3      3 3 3 3 1 4 4 4 3 7 7 8

HOR                          OUT
3 2 1 8 7 6 5 4 3 2 1 1      1 1 1 0 2 2 2 2 0 0 0 0
4 3 2 1 5 4 3 2 1 3 2 1      1 1 1 1 2 2 2 2 0 3 3 3
4 3 2 1 4 3 2 1 4 3 2 1      1 1 1 1 2 2 2 2 0 3 3 3
1 3 2 1 3 2 1 5 4 3 2 1      0 1 1 1 2 2 2 0 0 3 3 3
3 2 1 2 1 4 3 2 1 3 2 1      4 4 4 0 0 2 2 2 0 3 3 3
5 4 3 2 1 3 2 1 4 3 2 1      4 4 4 0 0 2 2 2 3 3 3 3
5 4 3 2 1 3 2 1 4 3 2 1      4 4 4 0 0 2 2 2 3 3 3 3
4 3 2 1 4 3 2 1 4 3 2 1      4 4 4 0 0 2 2 2 3 3 3 3
```

Fig. 4. The 2D scan-based algorithm applied to a label image ($B = 3H \oplus 3V$).

The list-based algorithm first calculates HOR. Then HOR is scanned line per line starting from the upper left-hand corner. When a value $HOR[i,j]$ is encountered the algorithm looks for the vertical extension of the newly detected rectangle. The position of the upper left corner is stored in a C structure as well as other informations, like the width and the height of the rectangle. After having put the rectangle (which has a size of $HOR[i,j] \times VER[i,j]$) in a list, the algorithm moves to the next pixel. It then tries to detect a rectangle that would be smaller in width but larger in height. Such a new rectangle is wide of $HOR[i,j]-1 = HOR[i+1,j]$ pixels and $VER[i+1,j] > VER[i,j]$. If such a rectangle exists it is added to the list $\mathcal{L}$. Fig. 5 shows detected rectangles and the corresponding list. The detection process is repeated as long as $HOR \neq 0$.

In order to counter the detection of redundant rectangles the algorithm uses an additional matrix, denoted D. A critical case, where R3 could be detected several times, is shown in Fig. 6. Locations (i,j) that led to the detection of a rectangle R have to be marked as follows. Suppose (i,j) is a pixel belonging to the first column of R such that $HOR[i,j]$ equals the width of R. Let K be the set vertical coordinates of the first column of R. $\forall k \in K$, if $HOR[i,k] = HOR[i,j]$ then $D[i,k] \leftarrow HOR[i,j]$. As an example, with the image of Fig. 5, the algorithm has detected 21 non-redundant rectangles.

The final result, a list of rectangles, may be seen as a particular data structure for a skeleton or medial axis transform. This is not surprising as it is well known that some skeletons can be expressed as erosions and openings. In his thesis [4], KRESCH



```
    Original image          HOR
1 1 1 1 1 0 0 0 1 0   5 4 3 2 1 0 0 0 1 0
1 1 1 1 1 1 0 1 0 0   6 5 4 3 2 1 0 1 0 0
0 1 1 1 1 1 1 1 0 0   0 7 6 5 4 3 2 1 0 0
1 1 1 1 1 1 1 1 1 0   9 8 7 6 5 4 3 2 1 0
0 1 1 1 1 1 1 1 1 0   0 8 7 6 5 4 3 2 1 0
0 1 0 0 1 1 1 1 1 1   0 1 0 0 6 5 4 3 2 1
1 1 1 0 1 1 1 1 1 1   3 2 1 0 6 5 4 3 2 1
1 1 1 1 0 1 1 1 1 1   4 3 2 1 0 5 4 3 2 1

 (i,j)=(0,0)  Width=5 Height=2
 (i,j)=(1,0)  Width=4 Height=5
 (i,j)=(4,0)  Width=1 Height=7
 ...
```

Fig. 5. An image, the HOR matrix and the first elements of the list of rectangles $\mathcal{L}$.

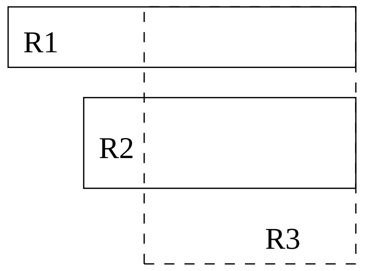


Fig. 6. Multiple detections of rectangle R3 have to be avoided.

developed several variants of skeleton transforms.

There is an other issue with list-based algorithms: intersections between rectangles. As a direct consequence of the detection process, rectangles have a non-empty intersection with all the rectangles detected during the scan of the same run-length. For a large object X it is not rare to have a ratio between the sum of areas of the non-redundant rectangles and the area of X larger than 10. The problem of multiple intersections has to be solved as otherwise the computation time would dramatically increase due to multiple accesses to OUT. Therefore we developed two solutions:

1. Avoid rewriting pixels in OUT owing to the technique described in section 2.3.
2. Stores all the intersections between elements of $\mathcal{L}$. For each rectangle R of $\mathcal{L}$, intersections between R and other rectangles are stored as a list $l(R)$ joined to the `C` structure of R. A thorough examination of all the intersections leads to a variant of the algorithm, called *area-specific* hereafter, for which no output pixel is written twice. The rule is that, when processing a list member R of $\mathcal{L}$, OUT is filled with pixels of the set difference $R \backslash R_i$ where R_i has a larger height than R and a width $\geq nH$. This even allows to compute the area of $X \circ B$ without having to fill the output image. The area-specific algorithm combined with a progressive

cleaning of the list is particularly suited for the computation of a granulometric curve.

4. Discussion and comparison

We have implemented different algorithms for binary openings. All the algorithms have been optimized and algorithms applicable to functions, like the histogram based algorithm, have been rewritten for the case of binary images. The computation times were obtained with the `gprof` profiler.

4.1. Computation times of our algorithms

Figure 7 presents the computation times of 2D scan and listed-based algorithms. Curve 1 (*Generic 2D scan-based*) was obtained with an implementation that includes all the optimization techniques we discussed in section 2.3.

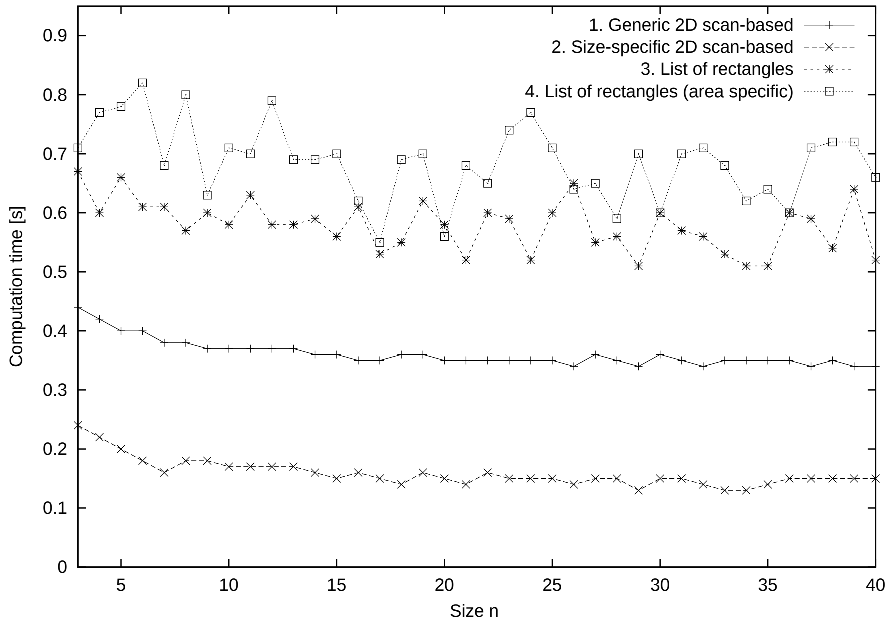


Fig. 7. Computation times of 2D scan and list-based algorithms with respect to a $n \times n$ square.

Curve 2 (*Size-specific 2D scan-based*) corresponds to a particular implementation where we used the size of the structuring element during the building of HOR and VER. All values strictly lower than the rectangle size were replaced by zeros. As a consequence the first phase is not generic anymore but this implementation cuts the computation times by half. The respective contributions of the first and second phase to the total amount of time are about 80%-20%.

In the case of list-based algorithms the respective contributions of the first and second phase to the total amount of time are about 90%-10%. List-based algorithms are mainly justified when openings with multiple sizes are needed, for example when computing a granulometric curve.

As can be seen from the curves, the computation time slightly decreases with the size. This unusual behavior is specific to the algorithms. It can be explained by a decreasing number of rectangles included in X when rectangles get larger.

4.2. Comparison with other algorithms

Fig. 8 compares the computation times of several algorithms on a large binary image. As mentioned previously we have reimplemented algorithms applicable to gray level images to include several optimizations. For the particular case of VAN HERK's algorithm (often used as a benchmark), we transposed the intermediate matrix twice, after $X \ominus nH$ and $((X \ominus nH) \ominus mV) \oplus mV$, to achieve a fair comparison with our algorithms.

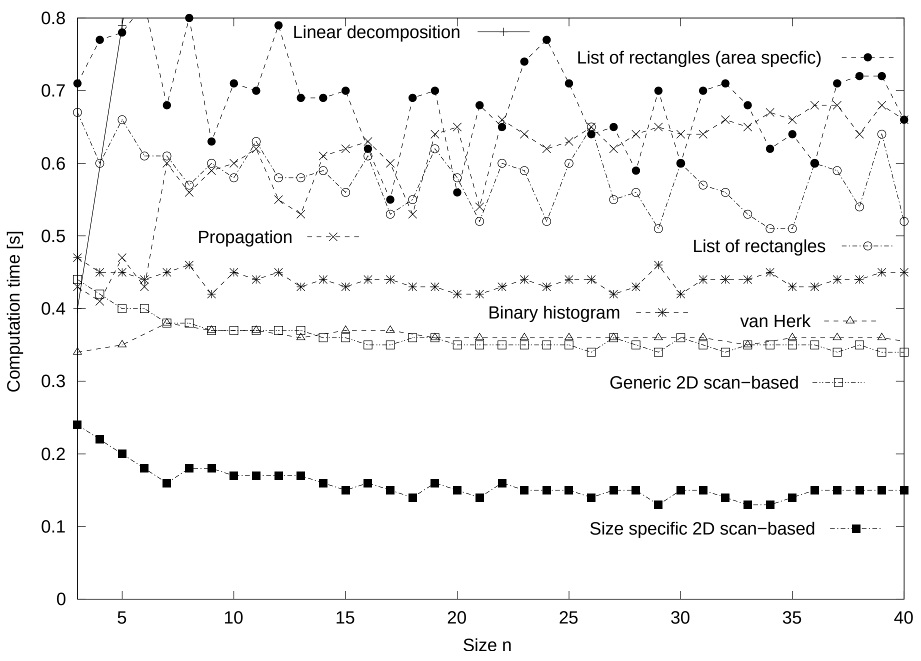


Fig. 8. Computation times of several algorithms for binary openings with respect to a $n \times n$ square.

As can be seen on the graph, the size specific 2D scan-based algorithm is the fastest algorithm. It is twice as fast as VAN HERK's algorithm that needs 3 comparisons per pixel for an erosion or a dilation.

Computation speed is only one criteria for comparison. Other criteria are provided in table I. Algorithms that allow multiple simultaneously computations permit efficient computations of granulometric curves. It should be mentioned that algorithms that process openings in a single step can be used for the computation of binary erosions as well, as $(X \circ B) \ominus B = X \ominus B$.

An extension to gray level images is more difficult and until now we did not find a way to extend our algorithms to the general case of gray level openings.

	Histogram	van Herk	Scan-based	List-based
Program complexity	Low	Low	Medium	High
Relative computation speed	Fast	Fast	Very fast	Slow
Applicable to label images	Yes	No	Yes	Yes
Allow multiple computations	No	No	Yes	Yes
Dependent on the content	Slightly	No	Yes	Yes

TABLE I
Comparison of algorithms that do not depend on the size of the structuring element.

References

1. B. Chaudhuri. An efficient algorithm for running window pel gray level ranking in 2-D images. *Pattern Recognition Letters*, 11(2):77–80, February 1990.
2. J. Gil and R. Kimmel. Efficient dilation, erosion, opening and closing algorithms. In J. Goutsias, L. Vincent, and D. Bloomberg, editors, *Mathematical Morphology and its Applications to Image and Signal Processing V*, pages 301–310, Palo-Alto, USA, June 2000. Kluwer Academic Publishers.
3. T. Huang, G. Yang, and G. Tang. A fast two-dimensional median filtering algorithm. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 27(1):13–18, February 1979.
4. R. Kresch. Morphological image representation for coding applications. PhD thesis, Technion Israel Institute of Technology, Haifa, 1995.
5. B. Laÿ. Recursive algorithms in mathematical morphology. In *Acta Stereologica, Vol. 6*, pages 691–696, Caen, 1987. ICS.
6. J. Pecht. Speeding up successive Minkowski operations. *Pattern Recognition Letters*, 3(2):113–117, 1985.
7. M. Schmitt. *Des algorithmes morphologiques à l'intelligence artificielle*. PhD thesis, École nationale supérieure des mines de Paris, February 1989.
8. R. van den Boomgaard. *Mathematical morphology: Extensions towards computer vision*. PhD thesis, Amsterdam University, March 1992.
9. M. Van Droogenbroeck. On the implementation of morphological operations. In J. Serra and P. Soille, editors, *Mathematical morphology and its applications to image processing*, pages 241–248. Kluwer Academic Publishers, Dordrecht, 1994.
10. M. Van Droogenbroeck. *Traitement d'images numériques au moyen d'algorithmes utilisant la morphologie mathématique et la notion d'objet : application au codage*. PhD thesis, Catholic University of Louvain, May 1994.
11. M. van Herk. A fast algorithm for local minimum and maximum filters on rectangular and octogonal kernels. *Pattern Recognition Letters*, 13(7):517–521, July 1992.
12. L. van Vliet and B. Verwer. A contour processing method for fast binary neighbourhood operations. *Pattern Recognition Letters*, 7:27–36, 1988.
13. L. Vincent. Morphological transformations of binary images with arbitrary structuring elements. *Signal Processing*, 22(1):3–23, January 1991.