SIMD-based Exact Parallel Fuzzy Dilation Operator for Fast Computing of Fuzzy Spatial Relations

Régis Pierrard, Laurent Cabaret, Jean-Philippe Poli, Céline Hudelot

SIMD-based Exact Parallel Fuzzy Dilation Operator for Fast Computing of Fuzzy Spatial Relations

Régis Pierrard[1,2], Laurent Cabaret[2], Jean-Philippe Poli[1], and Céline Hudelot[2]

[1]CEA, LIST, 91191 Gif-sur-Yvette cedex, France.
[2]Université Paris-Saclay, CentraleSupélec, Mathématiques et Informatique pour la Complexité et les Systèmes, 91190, Gif-sur-Yvette, France.
{*regis.pierrard, jean-philippe.poli*}*@cea.fr,* {*laurent.cabaret, celine.hudelot*}*@centralesupelec.fr*

Abstract

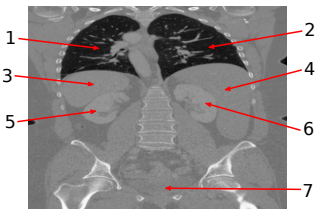
For decades, fuzzy spatial relations have demonstrated their utility and effectiveness for visual reasoning, including semantic annotation and object recognition. However, a major issue is that they often involve fuzzy morphological operators that are compute-intensive leading to long latency in the relation evaluation. As a result, approximate methods have been proposed to compute some relations in an acceptable time, but they are not as generic as the fuzzy dilation or do not make the most of modern computing architectures. In this paper, we introduce the *Reverse* and the *Parallel Reverse* (PR) algorithms. *Reverse* is an exact and efficient algorithm for the fuzzy dilation operator and *PR* combines the *Reverse* algorithm exactness with efficient usage of modern-processor multiple cores using OpenMP. Using SIMD extensions to enhance *Parallel Reverse*, PR_{128} (AVX), PR_{256} (AVX2), and PR_{512} (AVX512) are faster than the state-of-the-art approximate methods while remaining generic and exact. To demonstrate the performance of *PR* and highlight the contribution of the SIMD instructions, an extensive benchmark was carried out on two datasets of natural and artificial images.

1 Introduction

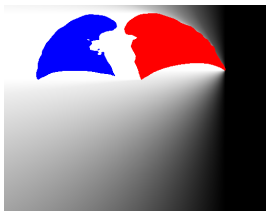
Spatial knowledge plays an important role in many computer vision systems since it is essential to scene understanding [11, 2].With the explainable artificial intelligence (XAI) advent, the need for interpretability and explainability [14, 13, 10] in such systems has been reinforced. The goal of these XAIs is to solve a task and provide an explanation to the result, as shown in figure 1a where the left lung detection is explained by its relations with the other known anatomical objects.

The fuzzy logic framework [24] provides efficient tools to represent and process such spatial information[5, 22]. Indeed, fuzzy logic enables to represent in a unified way a large variety of spatial relations [5] by taking into account their vagueness and their double nature (quantitative and qualitative). Several of these relations rely on a fuzzy morphological dilation (directional relations [3], distances [4] and more complex relationships like parallelism or alignment [21]). However, the standard fuzzy morphological dilation is computationally expensive and thus only approximate methods have been proposed [3, 23] to compute it efficiently. Besides, force-fields methods have been proposed for directional relations [16, 12] but they do not have the same properties and are not as generic as fuzzy-dilation-based methods.

In this paper, we focus on efficient fuzzy dilation computation for assessing fuzzy spatial relations based on



(a) Example of an explained annotation for the annotated organs in the figure: organ 1 is the left lung **because** it is *to the left of* the right lung (organ 2), it is *symmetrical* to the right lung and it is *above* the spleen (organ 3).



(b) Evaluation of the relation *left lung to the left of right lung*. It is assessed by computing the fuzzy degree of intersection between the left lung (in blue) and the fuzzy landscape (gray levels) generated as the dilation of the right lung (reference object in red).

Figure 1: Example of explainable organ annotation [18]. Given a set of objects (7 in figure 1a) and a set of spatial relations, annotation generation is based on the evaluation of a set of relations between all the objects in the image. The caption of Fig. 1a gives an example of annotation explanation and Fig. 1b represents how a specific relation between two objects is evaluated. The original image comes from [9].

multi-core and SIMD properties of modern processors. The fuzzy dilation generates a *fuzzy landscape* [3] (also known as *directional map* [16] or *spatial template* [23]). In a *fuzzy landscape*, the value of each pixel represents to what extent it verifies the relation under study, as shown in Fig. 1b for the relation *to the left of the right lung*. For a given reference object, this fuzzy landscape is generated once and then used to evaluate all the relations of the type *x to the left of the right lung* with all other objects. To generate explanations as in Fig. 1a [18] on a set of images, the most relevant relations between objects are extracted from a training set of images by computing one landscape per image, per object and per investigated relation. For reference, with 7 objects like in Fig. 1a and considering 5 relations (*left*, *right*, *above*, *under*, *close to*), 35 fuzzy landscapes are necessary for one image. With the complexity of the scene and the size of the training set, the number of landscapes to compute can then easily escalate hence the importance of computing them faster.

Unlike Bloch's algorithm [3] that does not make the most of modern CPU architectures and Wang's algorithm [23] whose main parameter value depends on the size of images to approximate the fuzzy dilation well, our proposition returns an exact and optimized implementation of the fuzzy dilation operator. To reduce the computation time, our implementation relies on two ideas: first, only the pixels belonging to the reference object in the input image (for example the red object in Fig. 1b) should be taken into account (*Reverse*) and, second, the algorithms have to be build with modern CPU architecture features in mind like multiple core programming and vector extensions (AVX/AVX2/AVX512) (*Parallel Reverse*, PR_{128}, PR_{256}, and PR_{512}). The remainder of the paper is organized as follows. In Section 2, we present the fuzzy dilation operator. In Section 3 we present the related algorithms. Our propositions are detailed in Section 4 followed by the benchmark description and the results discussion in Section 5. We conclude in Section 6.

2 Fuzzy Dilation Operator

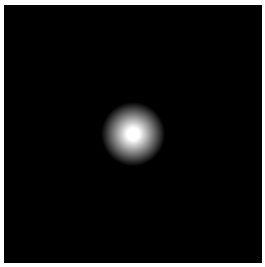
Like the dilation operator in mathematical morphology [19], the fuzzy dilation operator is the result of set-theoretic operations between an input image (representing the reference object on which the dilation is performed) and a structuring element (specifying the nature of the dilation). The result of the fuzzy dilation is represented by a fuzzy landscape, which is a fuzzy set whose membership function represents to which extent each pixel belongs to the dilation. Objects can also be represented as fuzzy landscapes. For instance, Fig. 2c displays the fuzzy landscape corresponding to the dilation of membership function $D_\nu(\mu)$ (dilation of μ by ν). The membership functions μ and ν associated respectively to the reference object and the structuring element are displayed in Fig. 2a and Fig. 2b respectively.

Let S be the space of the image. The fuzzy dilation of μ by ν, called $D_\nu(\mu)$, can then be defined as [3]

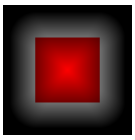
$$\forall x \in S, D_\nu(\mu)(x) = \sup_{y \in S} \Big[t\big(\nu(x-y), \mu(y)\big) \Big] \qquad (1)$$

with μ and ν crisp or fuzzy objects and t a t-norm. Several t-norms are defined in the fuzzy logic literature. In this paper, we use the most common one, the *Zadeh* t-norm, which is the *minimum*. Besides, since we work on

(a) Fuzzy landscape corresponding to the membership function μ associated to the reference object.



(b) Fuzzy landscape corresponding to the structuring element associated to the relation *close to* and represented by the membership function ν.



(c) Fuzzy landscape corresponding to the dilation of μ by ν associated to the membership function $D_\nu(\mu)$ (μ in red).

Figure 2: Fuzzy dilation $D_\nu(\mu)$ (Fig. 2c) of the reference object μ (Fig. 2a) by the structuring element ν (Fig. 2b). The spatial relation represented here is *close to*. The intensity of each pixel of $D_\nu(\mu)$ represents to which extent it satisfies the relation. The structuring element needs to be 4 times as big as other images.

images, S is a finite set so the *supremum* is equivalent to the *maximum*. Thus, the expression we actually implemented is the following

$$\forall x \in S, D_\nu(\mu)(x) = \max_{y \in S} \Big[\min \big(\nu(x-y), \mu(y) \big) \Big] \quad (2)$$

The main advantage of the fuzzy dilation is that various spatial relations, such as distances and directional relative positions between objects, can be computed with the same dilation operator by using different structuring elements. For example, in the case of directional relations, a commonly used structuring element is

$$\forall x \in S, \nu(x) = \max \left(0, 1 - \frac{2}{\pi} \arccos \frac{\overrightarrow{x} \times \overrightarrow{u_\alpha}}{||\overrightarrow{x}||} \right) \quad (3)$$

with $\overrightarrow{u_\alpha}$ a unit vector in the direction α and $\overrightarrow{x}$ the vector from the origin (the center of the structuring element) to x. In Fig. 2, another type of relation is expressed: *close to* μ. To assess if another object of membership function λ is close to μ, a fuzzy pattern matching approach is performed [7]. For instance, in Fig. 1b, for a reference object μ (in red) and another object λ (in blue), the relation λ *to the left of* μ can be evaluated as the fuzzy degree of intersection between λ and $D_\nu(\mu)$. This degree of intersection is defined in [5] such as

$$\mu_{int}(D_\nu(\mu), \lambda) = \frac{\sum_{x \in S} t\Big(D_\nu(\mu)(x), \lambda(x) \Big)}{\min \Big(\sum_{x \in S} D_\nu(\mu)(x), \sum_{x \in S} \lambda(x) \Big)} \quad (4)$$

The two main advantages of this fuzzy-landscape-based approach are its ability to manage any relation that can be generated with a fuzzy dilation and the fact that only one landscape has to be generated for a given relation and a given reference object. Thus, in Fig. 1, the relation *x to the left of the right lung* (reference object in red) can be computed for all x in the set of 7 objects in Fig. 1a with a single landscape generation (corresponding to *to the left of the right lung*).

3 Related Algorithms

3.1 Forward Algorithm

The *Forward* algorithm (Alg.1) is the direct application of Eq. 2 as described in the original paper on fuzzy directional dilations [3]. Three data structures representing 2D images (Fig. 2) are involved: D(N×M) which holds the resulting dilated image (fuzzy landscape), μ(N×M) the input image containing the reference object and ν(2N×2M) the structuring element which is 4 times larger than D and μ to generate all configurations of dilated pixels and input pixels. The computation of each pixel of D leads to applying the structuring element to all pixels of μ, regardless of whether they belong to the contributing object or not. Thus, $(NM)^2$ max/min operations have to be performed leading to a high computation time. Furthermore, while D is scanned forward (from up-left to down-right), μ is

Algorithm 1 Forward algorithm

Require: μ, D, ν
Ensure: D

```
for i ← 0 to N − 1 do
    for j ← 0 to M − 1 do
        for k ← 0 to N − 1 do
            for m ← 0 to M − 1 do
                val ← μ[k][m]
                se ← ν[N + i − k][M + j − m]
                D[i][j] ← max(min(val, se), D[i][j])
```

also scanned forward but ν is scanned backward leading to an inefficient CPU cache usage and an inefficient vectorization.

3.2 Bloch's Algorithm

Bloch proposed an algorithm [3] based on a propagation technique inspired by the *chamfer* method [6]. It returns an approximation of the fuzzy dilation. This algorithm is displayed in Alg. 2 and consists in three main steps:

1. Initializing an array O which stores for each pixel the position of the pixel that led to its update,
2. Performing a forward scan on the image to update $D_\nu(\mu)$ by looping over each pixel's neighbourhood,
3. Performing a backward scan on the image to update $D_\nu(\mu)$ by looping over each pixel's neighbourhood.

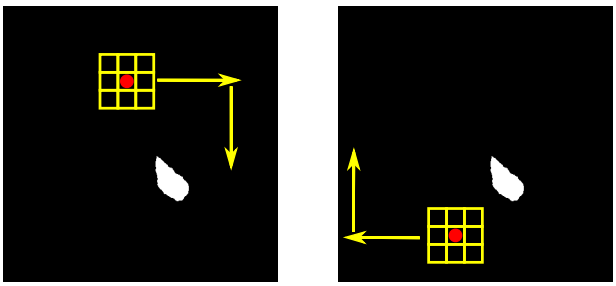
The details of the forward and backward scans are represented in Fig. 3. We can see in this figure that we need the neighbourhood of the pixel under study (i-th row and j-th column) to update the value of $D[i][j]$. This is actually the advantage of this method, since looping over a 8-connected neighbourhood ($V(i,j)$ at lines 10-11-15-16) is much faster than looping over the structuring element in the *Forward* algorithm. To get a better approximation of D, the part composed by step 2 (forward scan) and step 3 (backward scan) can be repeated k-times with a proportional cost in execution time. As describe in the original article, the approximation given by $k = 1$ provides the best trade-off between quality and execution time. As we want to fairly compare algorithms, $k = 1$ is used in all *Bloch*'s benchmarks.

Here, f (lines 10-11-15-16) has the same role as the structuring element in the fuzzy dilation and enables to specify the nature of the relation. The time complexity of this method is $O\big((1+2V)NM\big)$ with V the size of the neighbourhood ($V = 8$ for a 8-connected neighbourhood in 2D, $V = 26$ in 3D). However, this algorithm cannot be efficiently parallelized since it relies on a propagation method. Indeed, the propagation is gradually spread over the arrays D and O to update each pixel, which makes it unsuited to a multithreaded or a direct SIMD-based approach.

Algorithm 2 Bloch's propagation algorithm

Require: μ, D, ν
Ensure: D

1: $O \leftarrow$ empty2DArray() ▷ same size as μ
2: **for** $i \leftarrow 0$ to $N-1$ **do**
3: **for** $j \leftarrow 0$ to $M-1$ **do**
4: **if** $\mu[i][j] > 0$ **then**
5: $O[i][j] = \text{pair}(i,j)$
6: **else**
7: $O[i][j] = \texttt{null}$
8: **for** $i \leftarrow 0$ to $N-1$ **do** ▷ forward pass
9: **for** $j \leftarrow 0$ to $M-1$ **do**
10: $D[i][j] \leftarrow \max_{(i_v,j_v) \text{ in } V(i,j)} \min(D[O[i_v][j_v]], f(i,j,O[i_v][j_v]))$
11: $(i_*, j_*) = \arg\max_{(i_v,j_v) \text{ in } V(i,j)} \min(D[O[i_v][j_v]], f(i,j,O[i_v][j_v]))$
12: $O[i][j] \leftarrow O[i_*][j_*]$
13: **for** $i \leftarrow N-1$ to 0 **do** ▷ backward pass
14: **for** $j \leftarrow M-1$ to 0 **do**
15: $D[i][j] \leftarrow \max_{(i_v,j_v) \text{ in } V(i,j)} \min(D[O[i_v][j_v]], f(i,j,O[i_v][j_v]))$
16: $(i_*, j_*) = \arg\max_{(i_v,j_v) \text{ in } V(i,j)} \min(D[O[i_v][j_v]], f(i,j,O[i_v][j_v]))$
17: $O[i][j] \leftarrow O[i_*][j_*]$



(a) Forward pass (b) Backward pass

Figure 3: Bloch's algorithm relies on the *chamfer* method [6]. Each pixel of the input image is looped over twice: one forward pass (Fig. 3a) from left to right and top to bottom, and one backward pass (Fig. 3b) from right to left and bottom to top. Updating a pixel (in red on the two images) requires to perform operations on its neighbourhood (8-connected neighbourhood), which is represented by the yellow grid.

3.3 Wang's Algorithm

Wang proposed an algorithm based on F-templates [23, 17] that provides an approximation of the fuzzy dilation. The main idea is to only take into account the contribution of pixels in a set of k reference directions. Thus, for each pixel p, only the pixels on straight lines going through p in one of the reference directions contribute to the final result.

While this method is faster than Bloch's algorithm for small images (100×100) and when k is low (90), Gondra and Cabria showed that k must actually be proportional to $\sqrt{MN}$ to keep a good approximation [12]. So k must be greater for bigger images to have a correct approximation (for $k = 90$, the approximation is far from the exact result for a 400×400 image [12]). This tradeoff between accuracy and speed quickly turns to the advantage of *Bloch*. Besides, maintaining the same accuracy requires to set a new value of k each time the size of the input image changes, which regularly happens in the medical images we worked on (cf. Section 5.1). Thus, we discarded this algorithm from our experiments.

4 Proposed Algorithms

Our contribution consists in 2 algorithms: the *Reverse* algorithm that reduces the amount of computations, and PR that enhances *Reverse* using OpenMP and that is declined in PR_{128}, PR_{256}, and PR_{512}, based respectively on AVX, AVX2 and AVX512 extensions.

4.1 Reverse Algorithm

The idea of the *Reverse* algorithm (Alg. 3) is to reorder the operations to eliminate unnecessary processing: a pixel with a zero value in the input image ($val = \mu(y) = 0$) does not contribute (cf. Eq. 2) to $D_\nu(\mu)$ as $min\big(\nu(y-x), \mu(y)\big) = 0$, unlike in the *Forward* algorithm where those pixels cannot be separated from the contributing (active pixels) ones. By processing the computation based on the input image (μ) rather than the dilated image D, one can detect and drop all computations related to these non-contributing pixels (lines 3 and 4 in Alg. 3). So, based on the same equation (Eq. 2), *val* is evaluated once per pixel. Then, its contribution to the dilation is evaluated over D with the structuring element centered around the input pixel position. Due to the associative properties of the *min* and *max* operators, the final result is exactly the same as with the *Forward* algorithm, as shown in Fig 4. We can also note that the *Reverse* algorithm processes all the active pixels of the input image (the non-zero pixels of the reference object) equally regardless of their fuzziness and position. Consequently, the processing time of the fuzzy landscape shall not depends on the object fuzziness, position and shape. However, the number of pixels of the reference object directly affects the number of max/min operations. For an object of size p pixels belonging to a $N \times M$ input image, only $p \times N \times M$ max/min operations are executed, providing an acceleration factor based on the object size. Furthermore, ν and D are both scanned forward, which induces a better cache usage and a direct SIMD alignment.

Algorithm 3 Reverse algorithm

```
Require: μ, D, ν
Ensure: D
1:  for i ← 0 to N − 1 do
2:      for j ← 0 to M − 1 do
3:          val ← μ[i][j]
4:          if val > 0 then
5:              posx ← N − i
6:              posy ← M − j
7:              for k ← 0 to N − 1 do
8:                  for m ← 0 to M − 1 do
9:                      se ← ν[posx + k][posy + m]
10:                     D[k][m] ← max(min(val, se), D[k][m])
```

4.2 PR : Parallel Reverse algorithm

High-level code transformations (like *Reverse*) conjugated with the full usage of the multiple cores and the vector instructions (SIMD) offered by modern CPU architectures have proved their efficiency [15]. While *Reverse* processes only the active pixels, it still computes their contribution pixel by pixel on one core. As seen in Alg. 3 line 10, with the reverse modification, the contribution of one pixel of value $val = \mu(i, j)$ to the output D consists in applying only separable and aligned operations based on D (the output fuzzy landscape) and the structuring element centered on the current active pixel at

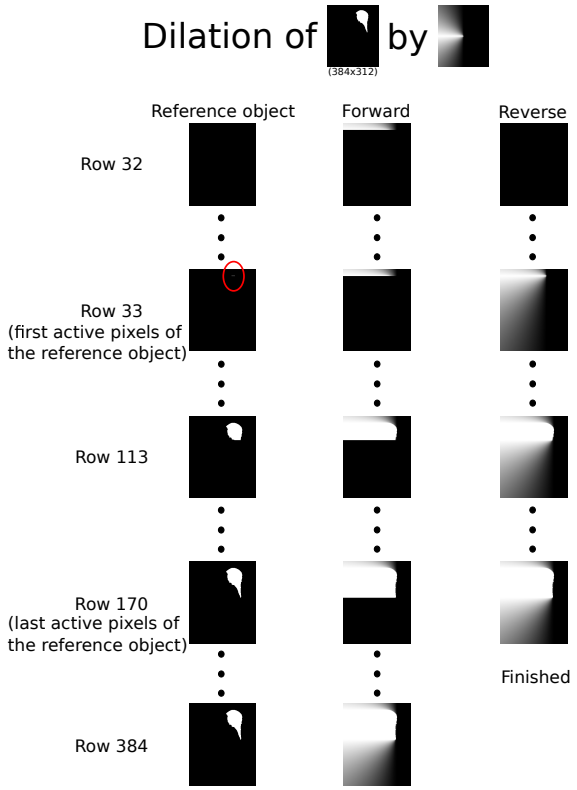


Figure 4: Comparison of the *Forward* and *Reverse* algorithms for computing the fuzzy dilation *to the left of the reference object*. The input image has 384 rows. While *Forward* computes the dilation pixel by pixel, *Reverse* only computes the contributions of non-zero pixels in the reference object. On row 33, the first non-zero pixels of the reference object are barely perceptible and have been surrounded by a red ellipse. Once the reference object has been completely looped over (row 170), the *Reverse* algorithm finishes and returns the same result as the *Forward* algorithm after row 384.

the i-th row and j-th column. Then, for each pixel where $\mu(i,j) > 0$ (active pixels), PR uses the OpenMP parallelization framework [8] to dispatch the computations of D over each core using a strip based spatial decomposition as shown in Fig 5. The internal loop (from line 7 to line 10 in Alg. 3) is processed in parallel using the `#pragma omp for`. By construction, the *Reverse* algorithm prevents data races as each strip of D is processed by a different thread and v is only accessed in read mode. To avoid recreating the threads for each new active pixel, which would harm the overall performance, parallel regions are created before the actual parallelization using the `#pragma omp parallel` directive over the external loop (line 1 in Alg. 3).

4.3 SIMD Optimizations: PR_{128}, PR_{256}, PR_{512}

In addition to the *PR* algorithm, we propose 3 SIMD implementations using explicit SIMD instructions. PR_{128}, PR_{256}, and PR_{512}, respectively use the `_mm_...`, `_mm256_...` and `_mm512_...` based intrinsics operations: broadcast of *val* (`_set1_epi8`) to the pack of integer, unaligned load of *se* and D (`_loadu_epi8`), unaligned store (`_storeu_epi8`) of D, unsigned min (`_min_epu8`) and unsigned max (`_max_epu8`). As the image width is not necessarily a multiple of the SIMD vector size, the image is padded with 0 before execution if needed. Then we propose 4 Parallel Reverse algorithms (Fig. 5): *PR* that is an OpenMP-based parallel version of the *Reverse* algorithm without explicit SIMD instructions, PR_{128} is

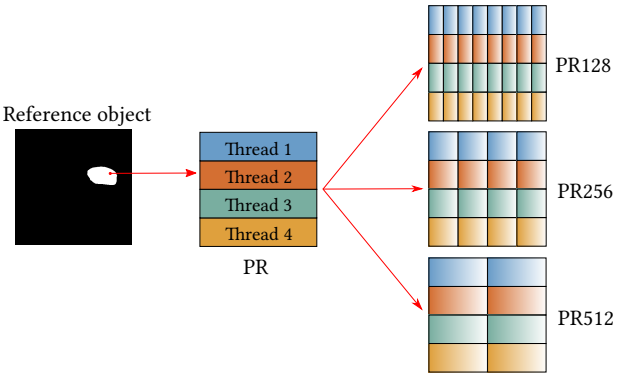


Figure 5: Contribution to the fuzzy dilation of one active pixel (in red) from the reference object (on the left). For the sake of this illustration, only 4 threads and 128 columns are represented. *PR* is the multi-threaded version of *Reverse* where each thread is responsible for a strip of rows of the fuzzy dilation $D_v(\mu)$. With PR_{128}, PR_{256} and PR_{512}, for each thread, columns of $D_v(\mu)$ are distributed using AVX, AVX2 and AVX512 respectively.

the same as *PR* but it uses explicit AVX SIMD instructions (128-bits wide), PR_{256} is specifically designed for the *de facto* standard version AVX2 (256-bits wide instructions), and PR_{512} uses the latest AVX512 extension (512-bits wide instructions). *PR*, PR_{128}, PR_{256}, and PR_{512} can be easily modified to use either 8-bits, 16-bits, 32-bits integers, float or double. As we work on 8-bits words, we process respectively 1, 16, 32 and 64 data by instruction with *PR*, PR_{128}, PR_{256}, and PR_{512}.

With the combined pressure of the SIMD extensions and multiple threads on memory bandwidth, it is not possible to make a direct assumption about the most efficient algorithm. It is therefore necessary to have a comprehensive benchmark to assess the contribution of the SIMD extensions.

5 Benchmark, Results and Analysis

5.1 Dataset

To provide reproducible results [20] and analyze performance with respect to specific image parameters like image and object size, object shape, fuzziness and position, we provide a dedicated structured artificial dataset of 282 images (Fig. 6). This dataset contains: crisp and fuzzy images, round, rectangle and ellipsoidal shapes, a regular distribution of squares, different sizes of images (256×256, 512×512, and 1024×1024) and different sizes of reference object (from 1 pixel to 65536 pixels).

The artificial dataset enable us to evaluate algorithms on specific criteria, but is not representative of real-world applications. Therefore, we completed the artificial dataset with a dataset of medical images [9] (Fig. 7) representative of visual reasoning applications. This natural image dataset contains 10 312×384 images (from 1234 active pixels to 6138 that is 1.0% to 5.1% of the image) and 8 407×1515 images (from 1096 active pixels to 9112 that is 0.2% to 1.4%). Those images correspond to segmented organs like the ones presented in Fig. 1b. The resulting dataset is available online [1].

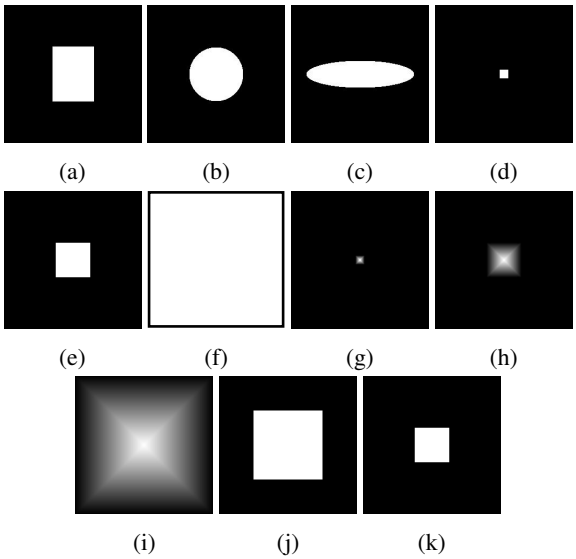


Figure 6: Artificial dataset samples: a) Rectangle, b) Disk, c) Ellipse, d) 256×256 crisp square with 256 active pixels, e) 256×256 crisp square with 4096 active pixels, f) 256×256 crisp square with 65536 active pixels, g) 256×256 fuzzy square with 256 active pixels, h) 256×256 fuzzy square with 4096 active pixels, i) 256×256 fuzzy square with 65536 active pixels, j) 512×512 crisp square with 65536 active pixels, k) 1024×1024 crisp square with 65536 active pixels

(a) 5829 pixels (b) 1234 pixels (c) 2200 pixels (d) 1962 pixels

Figure 7: Natural dataset samples: 312×384 images from a medical dataset of segmented organs. The number of active pixels varies due to the variable sizes of organs.

5.2 Benchmark Configuration

Seven algorithms were evaluated: the *Forward* algorithm, the *Reverse* algorithm, *PR* the parallel version and its three SIMD implementations PR_{128}, PR_{256}, and PR_{512}, and *Bloch*, which is Bloch's algorithm presented in [3] as the fastest acceptable approximate solution (1 forward scan followed by 1 backward scan). As we saw in Section 3.2, *Bloch* is not parallelizable since the updated val-

ues of pixels are progressively propagated through the image. So the implementation of Bloch's algorithm that we tested is monothreaded.

We evaluated the algorithms efficiency using an extensive benchmark on the proposed datasets. Using the structured artificial dataset, we evaluate the algorithm dependency on the shape, the fuzziness, the object size and position in the image, and the image size, while the natural images dataset provides real-world comparison between algorithms.

All computations were performed on an Intel Xeon CPU 6148 (20 cores - fixed 2.4GHz frequency) using executables generated by Intel ICC compiler 2019.3 (with $-O3$ flag).

5.3 Results

The benchmark confirms that the fuzziness, the shape and the position of the reference object in the input image has no effect on the execution time ($< 1\%$ of execution time variation). Consequently, the detailed results are not produced as they are identical to the results presented for fuzzy centered squares. As expected, image size, reference object size and algorithm are the three relevant parameters. Tab. 1 and Tab. 2 present the execution time of the seven algorithms for three image sizes and two reference object sizes: 4096 (referred as *small objects*) and 65536 (referred as *large objects*) respectively. Tab. 3 presents the acceleration ratio of each algorithm using *Reverse* as a reference. Indeed, *Forward* is too slow and not usable in real-world applications and thus the comparison of the meaningful algorithms is easier.

Sequential algorithms

In this part we focus on the *Forward*, *Reverse*, and *Bloch* algorithms. In every test configuration, the *Forward* algorithm is the slowest one. The computation time of one landscape is $\approx$12s for 256×256 images to $\approx$51 minutes for 1024×1024 images. This algorithm is not suitable for time-realistic object relationship evaluation. As expected, due to its construction, the execution time is mainly related to the image size and not to the reference object size. The *Reverse* algorithm is faster than the *Forward* algorithm in every configuration. When the reference object size increases, the execution time of *Reverse* increases

Table 1: Execution time in ms for one fuzzy landscape computation with variable image sizes for a 4096-pixel centered fuzzy square.

Algorithm	256×256	512×512	1024×1024
Forward	11.9×10^3	191×10^3	3075×10^3
Reverse	354.1	1.4×10^3	5.6×10^3
PR	55.2	82.4	297.7
PR_{128}	9.8	12.6	27.1
PR_{256}	9.0	10.7	22.2
PR_{512}	**8.4**	**9.8**	**19.5**
Bloch	33.0	130.9	523.1

Table 2: Execution time in ms for one fuzzy landscape computation with variable image sizes for a 65536-pixel centered fuzzy square.

Algorithm	256×256	512×512	1024×1024
Forward	11.8×10^3	189×10^3	3040×10^3
Reverse	5.6×10^3	22.5×10^3	89.7×10^3
PR	454.1	1307.3	4720.7
PR_{128}	144.8	183.9	407.9
PR_{256}	137.8	162.4	319.5
PR_{512}	124.5	147.4	**277.5**
Bloch	**35.4**	**134.6**	530.9

Table 3: Acceleration ratio with the *Reverse* algorithm as reference with variable image and reference object sizes.

Algorithm	256×256	512×512	1024×1024
4096-pixels centered object			
Forward	3.0×10^{-2}	7.4×10^{-3}	1.8×10^{-3}
Reverse	1	1	1
PR	6.4×10^0	1.7×10^1	1.9×10^1
PR_{128}	3.6×10^1	1.1×10^2	2.1×10^2
PR_{256}	3.9×10^1	1.3×10^2	2.5×10^2
PR_{512}	4.2×10^1	1.4×10^2	2.9×10^2
Bloch	1.1×10^1	1.1×10^1	1.1×10^1
65536-pixels centered object			
Forward	4.8×10^{-1}	1.2×10^{-1}	3.0×10^{-2}
Reverse	1	1	1
PR	1.2×10^1	1.7×10^1	1.9×10^1
PR_{128}	3.9×10^1	1.2×10^2	2.2×10^2
PR_{256}	4.1×10^1	1.4×10^2	2.8×10^2
PR_{512}	4.5×10^1	1.5×10^2	3.2×10^2
Bloch	1.6×10^2	1.7×10^2	1.7×10^2

but still remains faster than *Forward* as it provides an additional acceleration factor (*xs*) compared to the expected *active vs total* pixel ratio. This additional factor is underscored in Tab. 2 for a 256×256 image where $xs = 2.1$. Indeed, in this configuration, all pixels are active and both *Forward* and *Reverse* compute the exact same number of pixels. The fact that *Reverse* uses both D and the structuring element in a cache-friendly way is beneficial.

In every configuration, *Bloch*'s algorithm is faster than *Reverse* with a speedup of $\times 11$ for small objects and $\times 170$ for large objects.

Unlike *Reverse*, *Bloch*'s algorithm computation time does not depend on the reference object size but like *Reverse* linearly increase along the image size. This is due to their complexity: $O\big((1+2V)NM\big)$ for *Bloch* and $O\big(pNM\big)$ for *Reverse*.

SIMD multi-threads algorithms

For small objects (Tab. 1), PR_{512} is the fastest in every configuration. For large objects (Tab. 2) *Bloch* is the fastest for smaller images, while PR_{512} is the fastest for 1024×1024 images. This is explained by the fact that the execution time of *PR* and its derivatives do not follow the image size progression like *Reverse* and *Bloch* as they are significantly more efficient for large images than small ones (Tab. 3). PR_{512} is then competitive with the approximate *Bloch* algorithm despite the fact it is an exact dilation algorithm.

Using all 20 cores of our benchmark processor, PR_{512} is faster than *Reverse* by a factor $f_R \in [42;320]$ and by $f_{PR} \in [6;18]$ compared to *PR*. But using the maximum number of core is not always the optimal solution in our context. As shown in Tab. 4, for a sufficient amount of data (large object and large image), PR_{512} with all cores is the best solution. However, for smaller images and objects, the best option is to use less cores (8 for a 4096-pixels object in a 256×256 image and 16 for a 4096-pixels object in a 512×512 image). In all cases, despite the combined pressure of the SIMD extensions and multiple threads on memory bandwidth, the SIMD natural order is respected as $t_{PR_{512}} < t_{PR_{256}} < t_{PR_{128}} < t_{PR}$. PR_{512} is the best implementation of the *PR* algorithm.

Table 4: Execution time in ms for one fuzzy landscape computation with variable number of active cores and reference object sizes.

Active cores	2	4	8	16	20
65536-pixels object in a 1024×1024 image					
PR_{128}	3933.3	1587.2	833.3	461.2	**407.9**
PR_{256}	3044.4	1140.4	608.4	354.4	**319.5**
PR_{512}	2746.9	978.8	509.3	306.0	**277.5**
4096-pixels object in a 512×512 image					
PR_{128}	50.1	27.0	16.4	**11.8**	12.6
PR_{256}	36.6	20.3	12.9	**9.8**	10.7
PR_{512}	30.5	17.2	11.8	**9.1**	9.8
4096-pixels object in a 256×256 image					
PR_{128}	14.8	9.9	**7.3**	**7.3**	9.8
PR_{256}	10.7	7.6	**6.4**	6.7	9.0
PR_{512}	9.6	7.2	**5.9**	6.7	8.4

Table 5: Acceleration ratio with *Bloch*'s algorithm as reference with variable image and reference object sizes.

	312×384			407×1515		
Alg.	min	mean	max	min	mean	max
PR	$\times 1.0$	$\times 2.2$	$\times 4.8$	$\times 0.8$	$\times 3.3$	$\times 6.7$
PR_{128}	$\times 4.1$	$\times 9.4$	$\times 21.5$	$\times 7.7$	$\times 26.6$	$\times 51.0$
PR_{256}	$\times 4.4$	$\times 9.8$	$\times 20.8$	$\times 9.1$	$\times 30.6$	$\times 57.3$
PR_{512}	$\times 5.0$	$\times$**10.6**	$\times 21.5$	$\times 10.1$	$\times$**33.5**	$\times 61.6$

Natural images

Real-world images confirms PR_{512} as the fastest dilation operator algorithm. In Fig. 8, the low object size dependency of *Bloch* is highlighted by close results for each image category. On these images, even the *PR* algorithm is competitive with *Bloch* as the number of active pixels with respect to the image size is low (Sec. 5.1). Both the accelerating effect of AVX extensions and their limitations when data are not sufficient are well illustrated. Indeed as shown in Tab. 5, for the 407×1515 images, the algorithms keeps accelerating along AVX, AVX2, and AVX512 modifications. This effect is lower or non-existent for 312×384 images.

6 Conclusion

In this paper, we proposed *PR*, a new fast and exact algorithm for the fuzzy dilation operator computation based

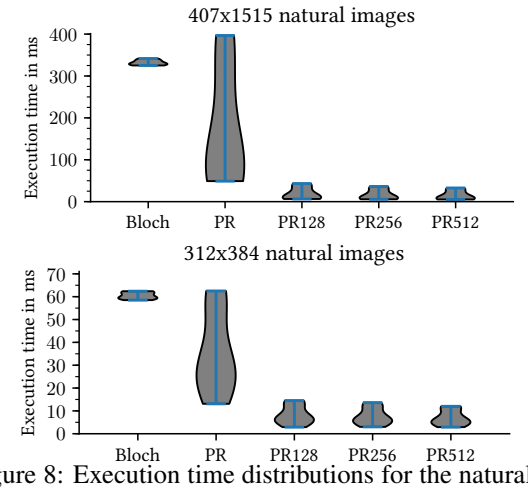


Figure 8: Execution time distributions for the natural image dataset.

on SIMD multi-core architectures. We found out that the new optimized SIMD algorithm is performing better on real-world images than the state-of-the-art approximate algorithm and remains competitive in each situation of our artificial images benchmark. PR_{512}, the AVX512 declination of *PR* is in average $\times 10.6$ faster than *Bloch* on the proposed real-world image dataset for smaller images and $\times 33.5$ faster than *Bloch* for larger images. The use of SIMD instructions combined with a high-level optimization is the key feature that enables us to assess various fuzzy spatial relations in a short time in order to perform spatial reasoning. Although this article focuses on the dilation operator, other morphological operators such as the erosion, the opening and the closing can be optimized the same way. In future work, we plan to develop a GPU-based dilation algorithm.

Acknowledgements

This work was performed using HPC resources from the "Mésocentre" computing center of CentraleSupélec and École Normale Supérieure Paris-Saclay supported by CNRS and Région Île-de-France.

References

[1] FuzzySpatialRelationsDataset. `https://bit.ly/2QZHqIX`, 2019.

[2] I. Biederman. On the semantics of a glance at a scene. In *Perceptual organization*, pages 213–253. Routledge, 2017.

[3] I. Bloch. Fuzzy relative position between objects in image processing: a morphological approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(7):657–664, July 1999.

[4] I. Bloch. On fuzzy distances and their use in image processing under imprecision. *Pattern Recognition*, 32(11):1873 – 1895, 1999.

[5] I. Bloch. Fuzzy spatial relationships for image processing and interpretation: a review. *Image and Vision Computing*, 23(2):89 – 110, 2005.

[6] G. Borgefors. Distance transformations in digital images. *Computer Vision, Graphics, and Image Processing*, 34(3):344 – 371, 1986.

[7] M. Cayrol, H. Farreny, and H. Prade. Fuzzy pattern matching. *Kybernetes*, 11(2):103–116, 1982.

[8] L. Dagum and R. Menon. Openmp: an industry standard api for shared-memory programming. *Computational Science & Engineering, IEEE*, 5(1):46–55, 1998.

[9] O. Jimenez del Toro, H. Müller, M. Krenn, K. Gruenberg, A. A. Taha, M. Winterstein, I. Eggel, A. Foncubierta-Rodríguez, O. Goksel, A. Jakab, G. Kontokotsios, G. Langs, B. H. Menze, T. Salas Fernandez, R. Schaer, A. Walleyo, M. Weber, Y. Dicente Cid, T. Gass, M. Heinrich, F. Jia, F. Kahl, R. Kechichian, D. Mai, A. B. Spanier, G. Vincent, C. Wang, D. Wyeth, and A. Hanbury. Cloud-based evaluation of anatomical structure segmentation and landmark detection algorithms: Visceral anatomy benchmarks. *IEEE Transactions on Medical Imaging*, 35(11):2459–2475, Nov 2016.

[10] F. Doshi-Velez and B. Kim. Towards A Rigorous Science of Interpretable Machine Learning. *arXiv e-prints*, February 2017.

[11] J. Freeman. The modelling of spatial relations. *Computer Graphics and Image Processing*, 4(2):156 – 171, 1975.

[12] I. Gondra and I. Cabria. Computing force field-based directional maps in subquadratic time. *Knowledge-Based Systems*, 95:58 – 70, 2016.

[13] B. Goodman and S.R. Flaxman. European union regulations on algorithmic decision-making and a "right to explanation". *AI Magazine*, 38(3):50–57, 2017.

[14] D. Gunning. Explainable artificial intelligence (xai), 2017.

[15] L. Lacassagne, D. Etiemble, A. Hassan-Zahraee, A. Dominguez, and P. Vezolle. High level transforms for simd and low-level computer vision algorithms. In *ACM Workshop on Programming Models for SIMD/Vector Processing (PPoPP)*, pages 49–56, 2014.

[16] P. Matsakis, J. Ni, and M. Veltman. Directional relationships to a reference object: A quantitative approach based on force fields. In *2009 16th IEEE International Conference on Image Processing (ICIP)*, pages 321–324, Nov 2009.

[17] P. Matsakis, J. Ni, and X. Wang. Object localization based on directional information: Case of 2d raster data. In *18th International Conference on Pattern Recognition (ICPR'06)*, volume 2, pages 142–146, Aug 2006.

[18] R. Pierrard, J-P. Poli, and C. Hudelot. A new approach for explainable multiple organ annotation with few data. In *IJCAI Workshop on Explainable Artificial Intelligence (XAI) 2019*, 2019.

[19] J. Serra. *Image analysis and mathematical morphology*. Academic Press, Inc., 1983.

[20] P. Vandewalle, J. Kovacevic, and M. Vetterli. Reproducible research in signal processing. *Signal Processing Magazine*, 26,3:37–47, 2009.

[21] M.C. Vanegas, I. Bloch, and J. Inglada. Alignment and parallelism for the description of high-resolution remote sensing images. *IEEE Transactions on Geoscience and Remote Sensing*, 51(6):3542–3557, June 2013.

[22] M.C. Vanegas, I. Bloch, and J. Inglada. Fuzzy constraint satisfaction problem for model-based image interpretation. *Fuzzy Sets and Systems*, 286:1 – 29, 2016.

[23] X. Wang, J. Ni, and P. Matsakis. Fuzzy object localization based on directional (and distance) information. In *2006 IEEE International Conference on Fuzzy Systems*, pages 256–263, July 2006.

[24] L.A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338 – 353, 1965.