Parallelization Strategy for Elementary Morphological Operators on Graphs

Imane Youkana, Jean Cousty, Rachida Saouli, Mohamed Akil

Parallelization strategy for elementary morphological operators on graphs

Imane Youkana[1,2], Jean Cousty[1], Rachida Saouli[1,2] and Mohamed Akil[1]

[1] Université Paris-Est, Laboratoire d'Informatique Gaspard-Monge, ESIEE Paris.
[2] Université de Biskra, Département d'Informatique, Biskra, Algérie.
{imane.youkana,jean.cousty,rachida.saouli,mohamed.akil}@esiee.fr

Abstract. This article focuses on the graph-based mathematical morphology operators presented in *[J. Cousty* et al*, "Morphological filtering on graphs", CVIU 2013]*. These operators depend on a size parameter that specifies the number of iterations of elementary dilations/erosions. Thus, the associated running times increase with the size parameter. In this article, we present distance maps that allow us to recover (by thresholding) all considered dilations and erosions. The algorithms based on distance maps allow the operators to be computed with a single linear-time iteration, without any dependence to the size parameter. Then, we investigate a parallelization strategy to compute these distance maps. The idea is to build iteratively the successive level-sets of the distance maps, each level set being traversed in parallel. Under some reasonable assumptions about the graph and sets to be dilated, our parallel algorithm runs in $O(n/p + K \log_2 p)$ where n, p, and K are the size of the graph, the number of available processors, and the number of distinct level-sets of the distance map, respectively.

1 Introduction

Mathematical morphology provides a set of filtering and segmenting tools that are very useful in applications to image analysis. There is a growing interest for considering digital objects not only composed of points but also composed of elements lying between them and carrying structural information about how the points are glued together. The simplest of these representations are the graphs. The domain of an image is considered as a graph (which can be planar or not) whose vertex set is made of the pixels and whose edge set is given by an adjacency relation on these pixels. Note that this adjacency relation can be either spatially invariant or spatially variant leading to operators that are either spatially invariant or spatially variant. Graphs are also useful to process other kinds of discrete structures defined for instance on 3-dimensional meshes. In this context, it becomes relevant to consider morphological transformations acting on the subsets of vertices, the subsets of edges and the subgraphs of a graphs and not only those acting on the set of all subsets of pixels.

Mathematical morphology on graphs was pioneered by Vincent [12] who proposes operators relying on a dilatation (and its adjunct erosion) that act on the vertices of a graph. More recently, [3, 6] introduce basic dilatations and erosions

that map a set of vertices to a set of edges and a set of edges to a set of vertices. It was shown in [3] that these operators can be combined in order to obtain operators acting on the subsets of edges, on the subsets of vertices and on the subgraphs of a given graph. In particular, interesting openings and closings (and then the associated alternate sequential filters) are obtained by iteration of the basic operators. The number of iterations constitutes a filtering parameter related to the size of the features to be preserved or removed. Therefore, based on the straightforward definition, the time-complexity of the associated algorithm increases with the size parameter. More precisely, for a parameter value of λ the algorithm runs in $O(\lambda.n)$ time, where n is the size of the underlying graph. In this article, our main contributions are twofold: we first propose to use distance maps in order to avoid the dependence to the parameter λ when computing the results of the operators of [3]; and then we propose a parallelization strategy leading to fast computation, in particular, for multicore/multithread architectures.

After presenting background notions about morphology and graphs in Section 2, we investigate in Section 3 some distance maps that lead to characterizations of the dilations and erosions presented in [3]. Since we are interested in operators that map sets of edges to sets of vertices and sets of vertices to sets of edges, we introduce edge-vertex and vertex-edge distance maps. Given a set of edges (resp. vertices), the edge-vertex (resp. vertex-edge) distance map provides for each vertex (resp. edge) a geodesic distance to the closest edge (resp. vertex) of the input set. In order to computed these distance maps, we adapt classical linear-time algorithm for distance maps in unweighted graphs. These algorithms derive from breadth first search. Whatever the size parameter, any dilation, erosion, opening and closing of [3] can be obtained by thresholding these distance maps. Therefore, the time complexity of the associated algorithms is linear with respect to the size of the graph, without any dependence to the size parameter.

In Section 4, we propose a parallel algorithm to compute the proposed distance maps, hence the morphological operators of [3]. Parallel and/or separable algorithms for morphological operators and distance maps on images have been widely studied [10, 8, 2, 5, 11, 9, 1, 7]. Based on the regular structure of the space, such computations use a static partitioning of the image into rows, columns or blocks processed in parallel. In order to handle the non-regular structure of a graph, our parallelization strategy is based on dynamic partitioning which depends on the input set and which is iteratively computed during the execution. The time complexity of our parallel algorithm is analyzed. In particular it depends of the complexity of two auxiliary functions to manage the dynamic partitions. These functions are presented in Section 5. Under some reasonable assumptions about the graph and set under consideration, our algorithm runs in $O(n/p + K \log_2 p)$ time, where n, p, and K are the size of the underlying graph, the number of available processors and the number of distinct level sets of the distance map, respectively. In the considered practical cases, this complexity is dominated by the $O(n/p)$ term.

2 Background notions for morphology on graphs

A *(undirected) graph* is a pair $X = (X^\bullet, X^\times)$ where $X^\bullet$ is a set and $X^\times$ is composed of unordered pairs of distinct elements in $X^\bullet$, *i.e.*, $X^\times$ is a subset of $\{\{x, y\} \subseteq X^\bullet \mid x \neq y\}$. Each element of $X^\bullet$ is called a *vertex or a point (of X)*, and each element of $X^\times$ is called an *edge (of X)*.

Important remark. Hereafter, the workspace is a graph $\mathbb{G} = (\mathbb{G}^\bullet, \mathbb{G}^\times)$ and we consider the sets $\mathcal{G}^\bullet$, $\mathcal{G}^\times$ and $\mathcal{G}$ of respectively all subsets of $\mathbb{G}^\bullet$, all subsets of $\mathbb{G}^\times$ and all subgraphs of $\mathbb{G}$.

Mathematical morphology on graphs, as introduced in [3], relies on four basic operators. The operators $\delta^\bullet$ and $\epsilon^\bullet$ are defined from $\mathcal{G}^\times$ to $\mathcal{G}^\bullet$ by:

$$\delta^\bullet(X^\times) = \{x \in \mathbb{G}^\bullet \mid \exists \{x, y\} \in X^\times\}, \text{ for any } X^\times \subseteq \mathbb{G}^\times; \text{ and} \quad (1)$$

$$\epsilon^\bullet(X^\times) = \{x \in \mathbb{G}^\bullet \mid \forall \{x, y\} \in \mathbb{G}^\times, \{x, y\} \in X^\times\}, \text{ for any } X^\times \subseteq \mathbb{G}^\times. \quad (2)$$

The operators $\epsilon^\times$, and $\delta^\times$ are defined from $\mathcal{G}^\bullet$ to $\mathcal{G}^\times$ by:

$$\epsilon^\times(X^\bullet) = \{\{x, y\} \in \mathbb{G}^\times \mid x \in X^\bullet \text{ and } y \in X^\bullet\}, \quad \text{for any } X^\bullet \subseteq \mathbb{G}^\bullet; \text{ and} \quad (3)$$

$$\delta^\times(X^\bullet) = \{\{x, y\} \in \mathbb{G}^\times \mid x \in X^\bullet \text{ or } y \in X^\bullet\}, \quad \text{for any } X^\bullet \subseteq \mathbb{G}^\bullet. \quad (4)$$

In order to obtain efficient filters (opening, closings, and associated alternate sequential filters), which are parametrized by a integer value related to a notion of size of the features to be preserved or removed, one needs to consider iterated versions of the basic building blocks presented above. Let α be an operator acting on $\mathcal{G}^\bullet$ or on $\mathcal{G}^\times$ and let i be a non negative integer. The operator α^i is defined by the identity when $i = 0$ and by $\alpha \circ \alpha^{i-1}$ otherwise.

Since the operators defined above map the elements of $\mathcal{G}^\bullet$ (*i.e.*, subsets of vertices) to those of $\mathcal{G}^\times$ (*i.e.*, subsets of edges) or the elements of $\mathcal{G}^\times$ to those of $\mathcal{G}^\bullet$, they cannot be directly iterated. However, any composition of an operator acting from $\mathcal{G}^\bullet$ to $\mathcal{G}^\times$ (resp. from $\mathcal{G}^\times$ to $\mathcal{G}^\bullet$) with an operator from $\mathcal{G}^\times$ to $\mathcal{G}^\bullet$ (resp.from $\mathcal{G}^\bullet$ to $\mathcal{G}^\times$) leads to an operator on $\mathcal{G}^\bullet$ (resp. on $\mathcal{G}^\times$). Then, such composition can be iterated and eventually followed again by an operator from $\mathcal{G}^\bullet$ to $\mathcal{G}^\times$ (resp. from $\mathcal{G}^\times$ to $\mathcal{G}^\bullet$). Therefore, to define iterated operators on graphs, two cases can be distinguish depending whether a final composition with an operator from $\mathcal{G}^\bullet$ to $\mathcal{G}^\times$ (resp. from $\mathcal{G}^\times$ to $\mathcal{G}^\bullet$) is considered or not.

Definition 1 (Iterated dilations/erosions) *Let λ be a nonnegative integer.*

Case 1 (even values of λ). *If λ is even, the operators $\delta_{\lambda/2}$ and $\epsilon_{\lambda/2}$ are defined on $\mathcal{G}^\bullet$ by $\delta_{\lambda/2} = (\delta^\bullet \circ \delta^\times)^{\lambda/2}$ and $\epsilon_{\lambda/2} = (\epsilon^\bullet \circ \epsilon^\times)^{\lambda/2}$; the operators $\Delta_{\lambda/2}$ and $\varepsilon_{\lambda/2}$ are defined on $\mathcal{G}^\times$ by $\Delta_{\lambda/2} = (\delta^\times \circ \delta^\bullet)^{\lambda/2}$ and $\varepsilon_{\lambda/2} = (\epsilon^\times \circ \epsilon^\bullet)^{\lambda/2}$.*

Case 2 (odd values of λ). *If λ is odd, the operators $\delta_{\lambda/2}$ and $\epsilon_{\lambda/2}$ are defined from $\mathcal{G}^\bullet$ to $\mathcal{G}^\times$ by $\delta_{\lambda/2} = \delta^\times \circ (\delta^\bullet \circ \delta^\times)^{(\lambda-1)/2}$ and $\epsilon_{\lambda/2} = \epsilon^\times \circ (\epsilon^\bullet \circ \epsilon^\times)^{(\lambda-1)/2}$; the operators $\Delta_{\lambda/2}$ and $\varepsilon_{\lambda/2}$ are defined from $\mathcal{G}^\times$ to $\mathcal{G}^\bullet$ by $\Delta_{\lambda/2} = \delta^\bullet \circ (\delta^\times \circ \delta^\bullet)^{(\lambda-1)/2}$ and $\varepsilon_{\lambda/2} = \epsilon^\bullet \circ (\epsilon^\times \circ \epsilon^\bullet)^{(\lambda-1)/2}$.*

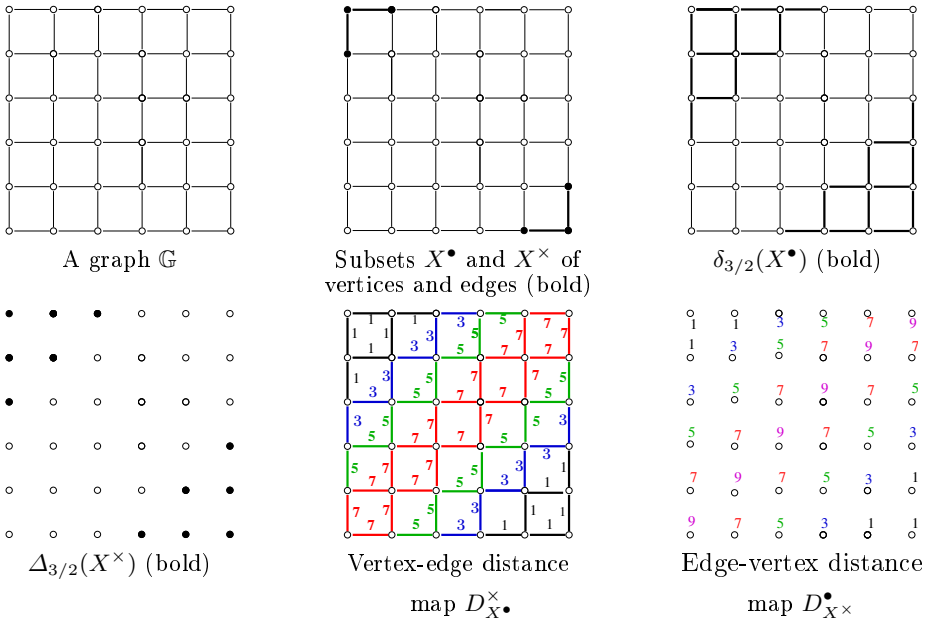

Fig. 1. Illustration of some morphological operators on graphs and of vertex-edge and edge-vertex distance maps.

Illustrations of the operators $\delta_{\lambda/2}$ and $\Delta_{\lambda/2}$ are provided in Figure 1 for $\lambda = 3$.

The operators $\delta^\bullet, \delta^\times, \delta_{\lambda/2}$ and $\Delta_{\lambda/2}$ are all morphological dilations and the operators $\epsilon^\times, \epsilon^\bullet, \epsilon_{\lambda/2}$ and $\varepsilon_{\lambda/2}$ are their adjunct erosions. Thus, any composition of one of these dilations with its adjunct erosion leads to a morphological filter which is either an opening or a closing depending on the composition order. In particular, when the integer parameter λ is even (resp. odd), the compositions of $\delta_{\lambda/2} \circ \epsilon_{\lambda/2}$ and $\epsilon_{\lambda/2} \circ \delta_{\lambda/2}$ (resp. $\Delta_{\lambda/2} \circ \epsilon_{\lambda/2}$ and $\varepsilon_{\lambda/2} \circ \delta_{\lambda/2}$) filters on $G^\bullet$ and the compositions of $\Delta_{\lambda/2} \circ \varepsilon_{\lambda/2}$ and $\varepsilon_{\lambda/2} \circ \Delta_{\lambda/2}$ (resp. $\delta_{\lambda/2} \circ \varepsilon_{\lambda/2}$ and $\epsilon_{\lambda/2} \circ \Delta_{\lambda/2}$) filters on $G^\times$. The simultaneous application of these compositions on the vertices and on the edges of any element in $\mathcal{G}$ (*i.e.*, on any subgraph of $\mathbb{G}$) leads to a subgraph of $\mathbb{G}$, hence morphological filtering on subgraphs.

When the size parameter λ is an even integer, the operators $\delta_{\lambda/2}$ and $\epsilon_{\lambda/2}$ correspond to the dilation and erosion proposed in [12]. It is known [12] that the result of these operators can be obtained by thresholding a (geodesic) distance map instead of iterating the basic dilation or erosion.

Let x and y be two vertices in $\mathbb{G}^\bullet$. A *(vertex-vertex) path from x to y* is a sequence $(x_0, u_0, \ldots, u_{\ell-1}, x_\ell)$ such that $x_0 = x$, $x_\ell = y$, and, for any i in $\{0, \ldots, \ell - 1\}$, we have $u_i = \{x_i, x_{i+1}\}$. The *length of a path* $\pi = (x_0, u_0, \ldots, u_{\ell-1}, x_\ell)$ is the number of its elements minus one, *i.e.*, the integer value 2ℓ. A *shortest path from x to y* is a path of minimal length from x to y. We denote by $L(x, y)$ the length of a shortest path from x to y. The *(vertex-vertex) distance map $D_{X^\bullet}$ to*

a set $X^\bullet \subseteq \mathbb{G}^\bullet$ is the map from $\mathbb{G}^\bullet$ to the set of integers such that:

$$D_{X^\bullet}(x) = \min\{L(x,y) \mid y \in X^\bullet\}, \text{ for any } x \in X^\bullet. \tag{5}$$

Then, when λ is even, the following relation characterizes the dilation $\delta_{\lambda/2}$:

$$\delta_{\lambda/2}(X^\bullet) = \{x \in \mathbb{G}^\bullet \mid D^\bullet_{X^\bullet}(x) \leq \lambda\}, \text{ for any } X^\bullet \in \mathcal{G}^\bullet. \tag{6}$$

Based on Equation 6, to obtain the dilation of a set of vertices, one needs to compute a distance map and to threshold it. An advantage, compared to the computation based on the iterative definition, is to avoid the dependence to the parameter λ in the algorithm time-complexity. More precisely, it is known that the distance map and thresholding computations (see *e.g.* Algorithm 1 for distance map) can be done in linear-time with respect to the size of the graph $\mathbb{G}$. In particular, Algorithm 1 is a variation on breadth-first search, which is a linear-time algorithm with respect to the size of $\mathbb{G}$. Observe that at line 8 of Algorithm 1, the distance value given to y is equal to the one of its predecessor x plus two. This is indeed correct with respect to the above definition of the length of a path, for which, *e.g.*, the distance between two neighbors is equal to two.

It can be deduced from the duality properties stated in [3] that $(\delta_{\lambda/2}, \epsilon_{\lambda/2})$ and $(\Delta_{\lambda/2}, \varepsilon_{\lambda/2})$ are pairs of dual operators, meaning that one operator in the pair can be easily computed from the other due to complementation operations. Hence, in order to provide efficient algorithms for these operators, we only need to focus on the operators $\delta_{\lambda/2}$ and $\Delta_{\lambda/2}$ and deduce the others by duality. For instance, the erosion $\epsilon_{\lambda/2}(X^\bullet)$ can be obtained with the same algorithm as $\delta_{\lambda/2}(X^\bullet)$ provided a complementation on both the input and output of the dilation algorithm. It is also straightforward to obtain a similar linear-time algorithms, based on an *edge-edge distance map*, for the edge dilation $\Delta_{\lambda/2}$ and erosion $\varepsilon_{\lambda/2}$ when λ is even.

Algorithm 1: Sequential vertex-vertex distance map.

Data: a connected graph $\mathbb{G} = (\mathbb{G}^\bullet, \mathbb{G}^\times)$, a subset $X^\bullet$ of $\mathbb{G}^\bullet$.
Result: the distance map $D^\bullet_{X^\bullet}$ to the set $X^\bullet$.

```
1 Q := an empty queue with FIFO property;
2 foreach vertex x in G• do
3     if x ∈ X• then Q.push(x); D×_X•(u) := 0;
4     else D×_X•(x) := ∞;
5 while Q.isNotEmpty() do
6     x := Q.pop();
7     foreach vertex y adjacent to x in G do // i.e., when {x, y} ∈ G×
8         if D•_X•(y) = ∞ then Q.push(y); D•_X•(y) := D•_X•(x) + 2 ;
```

The next section presents an approach based on distance maps to obtain linear time algorithms for $\Delta_{\lambda/2}$ and $\delta_{\lambda/2}$ when λ is odd. Then, Section 4 presents

a parallelization strategy leading to efficient parallel algorithms for all morphological operators on graphs presented in [3].

3 Vertex-edge and edge-vertex distance maps

When considering an odd value of λ, an important difference with the even case is that the results and arguments of the dilations $\delta_{\lambda/2}$ and $\Delta_{\lambda/2}$ are not homogeneous: one of them is a set of edges whereas the other one is a set of vertices. In order to deal with this inhomogeneity, we introduce the edge-vertex and vertex-edge distance maps. Given a set of edges (resp. vertices), the edge-vertex (resp. vertex-edge) distance map provides for each vertex (resp. edge) of $\mathbb{G}$ a distance to the closest edge (resp. vertex) of the input set. These distance maps allow us to characterize (by thresholding) the dilations $\delta_{\lambda/2}$ and $\Delta_{\lambda/2}$ when λ is odd. Finally, Algorithm 1 is adapted to compute these distance maps.

The distance maps considered in this section rely on the lengths of shortest paths from vertices to edges. A *(vertex-edge) path from a vertex x of $\mathbb{G}$ to an edge u* of $\mathbb{G}$ is a sequence $(x_0, u_0, \ldots, x_\ell, u_\ell)$ such that $u_\ell = u$, $x_\ell \in u_\ell$, and $(x_0, u_0, \ldots, x_\ell)$ is a vertex-vertex path from x to x_ℓ. The *length of a path* $(x_0, u_0, \ldots, x_l, u_\ell)$ is the number of its elements minus one, *i.e.*, the integer value $2\ell + 1$. A *shortest path from a vertex x of $\mathbb{G}$ to an edge u of $\mathbb{G}$* is a path of minimal length from x to u. We denote by $L(x, u)$ the length of a shortest path from x to u. Finally, given a subset $X^\bullet$ of vertices of $\mathbb{G}$, we define the *vertex-edge distance map to $X^\bullet$* as the map $D^\times_{X^\bullet}$ from $\mathbb{G}^\times$ to the set of integers such that:

$$D^\times_{X^\bullet}(u) = \min\{L(x, u) \mid x \in X^\bullet\}, \text{for any } u \in X^\times. \tag{7}$$

Dualy, given a subset $X^\times$ of edges, the *edge-vertex distance map to the set* $X^\times$ is the map $D^\bullet_{X^\times}$ from $\mathbb{G}^\bullet$ to the set of integers such that:

$$D^\bullet_{X^\times}(x) = \min\{L(x, u) \mid u \in X^\times\}, \text{ for any } x \in X^\bullet. \tag{8}$$

Edge-vertex and vertex-edge distance maps are illustrated in Figure 1.

The next property states that the dilatations $\delta_{\lambda/2}$ and $\Delta_{\lambda/2}$ can also be characterized with distance maps when λ is odd.

Property 2 *Let λ be any odd positive integer. The following relations hold true:*

$$\delta_{\lambda/2}(X^\bullet) = \{u \in \mathbb{G}^\times \mid D^\times_{X^\bullet}(u) \leq \lambda\}, \text{ for any } X^\bullet \in \mathcal{G}^\bullet; \text{ and}$$
$$\Delta_{\lambda/2}(X^\times) = \{x \in \mathbb{G}^\bullet \mid D^\bullet_{X^\times}(x) \leq \lambda\}, \text{ for any } X^\times \in \mathcal{G}^\times.$$

Algorithms 2 and 3 presented below compute these distance maps in linear time with respect to size $|\mathbb{G}^\bullet| + |\mathbb{G}^\times|$ of $\mathbb{G}$.

4 Parallel algorithm for distance maps on graphs

Contrary to the parallel computation of distance maps on an image, which is often based on a static partitioning of the image into rows, columns or blocks

Algorithm 2: Vertex-edge distance map.

Data: A connected graph $(\mathbb{G}^\bullet, \mathbb{G}^\times)$, A subset $X^\bullet$ of $\mathbb{G}^\bullet$.
Result: The vertex-edge distance map $D^\times_{X^\bullet}$ to the set $X^\bullet$.

$\mathcal{Q}$:= an empty queue with FIFO property;
foreach *edge* $u = \{x, y\}$ *in* $\mathbb{G}^\times$ **do**
  if $x \in X^\bullet$ *or* $y \in X^\bullet$ **then** $\mathcal{Q}$.push(u); $D^\times_{X^\bullet}(u) := 1$;
  else $D^\times_{X^\bullet}(u) := \infty$;
while $\mathcal{Q}$.*isNotEmpty()* **do**
  $u := \mathcal{Q}$.pop() ;
  foreach *edge* v *in* $\mathbb{G}^\times$ *adjacent to* u **do** // *i.e.*, when we have $v \cap u \neq \emptyset$
    if $D^\times_{X^\bullet}(v) = \infty$ **then** $\mathcal{Q}$.push(v); $D^\times_{X^\bullet}(v) := D^\times_{X^\bullet}(u) + 2$;

Algorithm 3: Edge-vertex distance map.

Data: A connected graph $(\mathbb{G}^\bullet, \mathbb{G}^\times)$, a subset $X^\times$ of $\mathbb{G}^\times$.
Result: The edge-vertex distance map $D^\bullet_{X^\times}$ to the set $X^\times$.

$\mathcal{Q}$:= an empty queue with FIFO property;
foreach *vertex* x *in* $\mathbb{G}^\bullet$ **do** $D^\bullet_{X^\times}(x) := \infty$;
foreach *edge* $u = \{x, y\}$ *in* $\mathbb{G}^\times$ **do**
  if $D^\bullet_{X^\times}(x) = \infty$ **then** $\mathcal{Q}$.push(x); $D^\bullet_{X^\times}(x) := 1$;
  if $D^\bullet_{X^\times}(y) = \infty$ **then** $\mathcal{Q}$.push(y); $D^\bullet_{X^\times}(y) := 1$;
while $\mathcal{Q}$.*isNotEmpty()* **do**
  $x := \mathcal{Q}$.pop() ;
  foreach *vertex* y *in* $G^\bullet$ *adjacent to* x **do** // *i.e.*, when $\{x, y\} \in \mathbb{G}^\times$
    if $D^\bullet_{X^\times}(y) = \infty$ **then** $\mathcal{Q}$.push(y); $D^\bullet_{X^\times}(y) := D^\bullet_{X^\times}(x) + 2$;

processed in parallel, our parallelization strategy on graphs is based on dynamic partitioning. The partition depends on the input set and is iteratively computed during the execution. More precisely, our strategy iteratively considers the successive level-sets of the distance maps, each level set being partitioned and then traversed in parallel. In this section, our parallel algorithm is presented and its complexity is analyzed assuming that partitioning can be done efficiently. Efficient parallel management of partitions is the topic of the next section.

For the sake of simplicity, we only describe the case of vertex-vertex distance maps, but our strategy can also be adapted to edge-edge, vertex-edge and edge-vertex distance maps computations.

Let us first present our algorithm from a high level point of view. To this end, we recall the notion of a level set. Given an integer λ and a (distance) map D from $\mathbb{G}^\bullet$ in the set of integers, the *λ-level set* of D is the set of all elements of value λ for D (*i.e.*, the set $\{x \in \mathbb{G}^\bullet \mid D(x) = \lambda\}$). Given a subset $X^\bullet$ of $\mathbb{G}^\bullet$, after an initialization step where an integer variable λ is set to 0 and where the

elements of $X^\bullet$ are inserted in a variable set E (hence E is the $(\lambda = 0)$-level-set of $D^\bullet_{X^\bullet}$), our algorithm can be sketched as follows:

1. Partition E (*i.e.*, the λ-level set of $D^\bullet_{X^\bullet}$) into p balanced subsets $E_1, \ldots, E_p$.
2. Assign each of the p subsets $E_1, \ldots, E_p$ to one of the p processors
3. Let, in parallel, each processor insert the non already traversed neighbors of the elements in its assigned subset E_i into a private variable set S_i and set the distance map value of the elements in S_i to $\lambda + 2$.
4. Merge the private sets $\{S_i \mid i \in \{1, \ldots, p\}\}$ and store the result in E so that E becomes the $(\lambda + 2)$-level set of $D^\bullet_{X^\bullet}$.
5. Increment λ and repeat steps 1-4 until E becomes empty.

In Step 3, in order to concurrently check if a vertex has been already traversed, we need to equip each vertex with a synchronization Boolean variable that is handled with an atomic *test-and-set* instruction. The test-and-set instruction sets a given variable to true and returns its old value as a single atomic (*i.e.*, non-interruptible) instruction.

Algorithm 4 provides the precise description of our parallel approach. It uses two auxiliary functions called *Partition* and *Union*. In the next section, we provide algorithms for these two functions. The efficiency of Algorithm 4 depends on these functions. As we will see, the function *Partition* runs in linear time with respect to n/p and the function *Union* runs in $O(n/(Kp) + log_2 p)$ amortized time, where n, p and K are the size of the graph, the number of processors, and the number of level-sets of the produced distance map. Furthermore, any class of the produced partition contains either n/p or $n/p + 1$ elements.

Finally, in order to state the time complexity of Algorithm 4, we need to make two assumptions about the graph and the set of vertices under consideration.

The *degree of a vertex x of* $\mathbb{G}$ is the number of edges that contain x (*i.e.*, the cardinality of the set $\{y \in \mathbb{G}^\bullet \mid \{x, y\} \in \mathbb{G}^\times\}$). Let β be any positive integers. We say that $\mathbb{G}$ is *β-balanced* if the degrees of any two vertices of $\mathbb{G}$ differ by at most β. Let $X^\bullet$ be a subset of $\mathbb{G}^\bullet$. We say that $X^\bullet$ is *β-balanced* if every nonempty level-set of $D^\bullet_{X^\bullet}$ contains at least β elements.

Note that when $X^\bullet$ is *p-balanced*, the distance map $D^\bullet_{X^\bullet}$ has at most $|\mathbb{G}^\bullet|/p$ nonempty level-sets, then the while loop at line 7 is executed at most $|\mathbb{G}^\bullet|/p$ times. Furthermore, if a given level set E contains n vertices, any of the $\{E_i \mid i \in \{1, \ldots, p\}\}$ obtained at line 8 contains at most $n/p + 1$ vertices, which allows us to deduce that the loop line 11 runs in $O(|\mathbb{G}^\bullet|/p)$ time since the level-sets of $D^\bullet_{X^\bullet}$ partition $\mathbb{G}^\bullet$. As any of the $\{E_i \mid i \in \{1, \ldots, p\}\}$ contains at most $n/p+1$ vertices, when $\mathbb{G}$ is β-balanced, we can bound the number of edges that contain an element in E_i by $m/p + d_{min} + \beta n/p + \beta$, where m is the total number of edges that contain an element in E_i and where d_{min} is the minimal degree of a vertex of $\mathbb{G}$. Thus, we also have $|S_i| \leq m/p + d_{min} + \beta n/p + \beta$, where S_i is the set obtained after the execution of foreach loop line 11. Hence, since the level sets of $D^\bullet_{X^\bullet}$ partition $\mathbb{G}^\bullet$, it can be shown that the insertion operation on S_i at line 14 is executed at most $(3|\mathbb{G}^\times| + 2\beta|\mathbb{G}^\bullet|)/p$ times by each of the p processors during the overall execution and the continuation condition of the loop at line 12 must be tested less than $3|\mathbb{G}^\times|/p + 2(\beta + 1)|\mathbb{G}^\bullet|/p$ times. Hence, using an array of linked

Algorithm 4: Parallel vertex-vertex distance map.

Data: A connected graph $(\mathbb{G}^\bullet, \mathbb{G}^\times)$, a subset $X^\bullet$ of $\mathbb{G}^\bullet$, the number p of processors.

Result: The distance map $D^\bullet_{X^\bullet}$ to the set $X^\bullet$.

```
1  E := ∅; λ :=0;
2  Set to False all elements of a shared Boolean array Traversed of size |𝔾•|
3  (E_1, ...,E_p):= Partition(X•, p);
4  foreach processor i in {1, ...,p} do in parallel
5      foreach vertex x ∈ E_i do D•_{X•}(x) := λ; Traversed[x] := True; ;
6  E:=Union(p, E_1, ..., E_p);
7  while E ≠ ∅ do
8      (E_1, ...,E_p):= Partition(E, p);
9      foreach processor i in {1, ..., p} do in parallel
10         S_i:=∅;
11         foreach x in E_i do
12             foreach vertex y adjacent to x in 𝔾 do // i.e., when {x, y} ∈ 𝔾×
13                 if test-and-set(Traversed[y]) = False then
14                     S_i:=S_i ∪ {y};
15                     D•_{X•}(y) := λ + 2;
16     E:=Union(p, S_1, ..., S_p);
```

lists to represent the graph $\mathbb{G}$ and using simple arrays for all sets, we deduce that the time complexity of the main part (lines 9 to 15) of Algorithm 4 is linear with respect to $(|\mathbb{G}^\times| + |\mathbb{G}^\bullet|)/p$. Considering also the auxiliary functions *Union* and *Partition*, the overall time complexity of Algorithm 4 can be established.

Theorem 3 *Algorithm 4 outputs a map $D^\bullet_{X^\bullet}$ which is the vertex-vertex distance map to the set $X^\bullet$. Let p be the number of available processors. Let us assume that β is a constant integer such that $\mathbb{G}$ is β-balanced and that $X^\bullet$ is p-balanced. Then, Algorithm 4 runs in $O((|\mathbb{G}^\bullet| + |G^\times|)/p + K \log_2 p)$ time, where K is the number of nonempty level-sets of $D^\bullet_{X^\bullet}$.*

Under the assumption of Theorem 3, the distance map D_X contains at most $|\mathbb{G}^\bullet|/p$ nonempty level-sets. Thus the time complexity of Algorithm 4 is less than $O((|\mathbb{G}^\bullet| + |G^\times|)/p + |\mathbb{G}^\bullet|(\log_2 p)/p)$.

The assumptions in Theorem 3 hold, in general, true when the graph-based morphological operators of [3] are applied to image processing. In particular, when we consider a 2-dimensional image equipped with the 4- or 8- adjacency relation, the degrees of any two vertices are the same (except on the image borders), and the number of distinct level-sets is of the order of $\sqrt{|G^\bullet|}$, meaning that in average, each level set contains $\sqrt{|G^\bullet|}$ vertices. Furthermore, in practice, we generally have $K.\log_2 p \leq (|\mathbb{G}^\bullet| + |\mathbb{G}^\times|)/p$. Thus, roughly speaking, we can say that the time-complexity is, in general, dominated by $(|\mathbb{G}^\bullet| + |G^\times|)/p$.

5 Parallel partition and disjoint union algorithms

In this section, we present efficient parallel algorithms for the partition and union function used in Algorithm 4 and we analyze their time-complexity.

The parallel partition algorithm (see Algorithm 5) consists of computing in parallel, with p processors, a *balanced partition* $\{E_1, \ldots, E_p\}$ of a set E. The partition is balanced in the sense that the k-first sets of the partition contain $|E|/p + 1$ elements whereas the following ones contain $|E|/p$ elements, where k is the remainder in the integer division of $|E|$ by p. The elements of E, stored in an array, are moved to arrays previously allocated for the subsets $E_1, \ldots, E_p$ in the order of their indices: the first set receives the first elements of the array E and so on (see Figure 2). Thus, each processor computes the index of the first and of the last element that must be copied (lines 2 to 7) before actually copying the elements of E located between the computed indices (line 8). The computation of the first and of the last indices can be done in constant time and the copying step is done in linear time with respect to $|E|/p$ (each processor moves at most $|E|/p + 1$ elements).

Algorithm 5: Partition.

Data: An array E of $n = |E|$ elements, the number p of processors.
Result: A balance partition (E_1, ...,E_p) of E.

```
1 foreach processor i in {1, ...,p} do in parallel
2     if i ≤ (n mod p) then
3         start[i] := (i − 1) ∗ (n/p + 1);
4         end[i] := start[i] + n/p;
5     else
6         start[i] := (n mod p) ∗ (n/p + 1) + (i − 1 − (n mod p)) ∗ (n/p);
7         end[i] := start[i] + n/p - 1;
8     foreach j_i in {start[i], ..., end[i]} do E_i[j_i − start[i]] := E[j_i] ;
```

Our parallel *Union* algorithm (see Algorithm 6) computes the union of p disjoint sets $\{S_1, \ldots, S_p\}$ with p processors. The elements of each set are stored in an array and each processor i copies the elements of the array S_i in the array E. The elements of S_i are stored in the resulting array E from the index $start[i]$, where $start[i]$ is the sum of the cardinalities of the sets $S_1, \ldots, S_{i-1}$ (see Figure 3). Thus, our algorithm first computes the values $start[i]$ for any i in $\{1, \ldots, p\}$ (line 1) before actually copying the elements into E (line 3). Given the cardinalities $|S_1|, \ldots, |S_p|$, computing the values $start[i]$ for any i in $\{1, \ldots, p\}$ is known as the prefix-sum problem. It can be solved in parallel with p processors with a $O(\log_2 p)$ running-time algorithm [4]. Then, each processor i copies (line 3) the elements of S_i into E at the correct position. Let us consider the amortized-time complexity of this operation for a sequence of calls to *Union* as used in

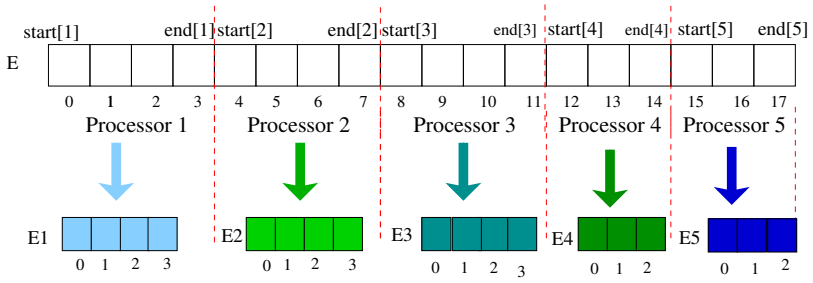

Fig. 2. Illustration of the *Partition* algorithm with $p = 5$ processors.

Algorithm 4, under the assumptions of Theorem 3. Let K be the number of distinct level sets of $D^{\bullet}_{X\bullet}$. There is one call to *Union* for each level-set of the distance map $D^{\bullet}_{X}$. Thus, there are K calls to *Union*. We have seen in Section 4 that there are at most $(3|\mathbb{G}^{\times}| + 2\beta|\mathbb{G}^{\bullet}|)/p$ insertions in S_i. Any element inserted in S_i is considered exactly once at line 3 of Algorithm 6. Thus, the amortized time-complexity of line 3 is $O((|\mathbb{G}^{\bullet}| + |\mathbb{G}^{\times}|)/(Kp))$ and the one of Algorithm 6 is $O((|\mathbb{G}^{\bullet}| + |\mathbb{G}^{\times}|)/(Kp) + \log_2 p)$.

Algorithm 6: Union.

Data: A series $S_1, \ldots, S_p$ of p sets, and the number p of processors.
Result: An array E whose elements constitutes the union of $\{S_1, \ldots, S_p\}$.

1 start = ParallelPrefixSum($|S_1|, \ldots, |S_p|$);
2 **foreach** *processor* i *in* $\{1, \ldots, p\}$ **do in parallel**
3 ⌊ **foreach** j_i *in* $\{0, \ldots, |S_i| - 1\}$ **do** $E[start[i] + j_i] := S_i[j_i]$;

6 Conclusion

In this article efficient sequential and parallel algorithms for the (binary) graph-based mathematical morphology operators defined in [3] have been proposed. These algorithms are based on distance maps computation in unweighted graphs. The sequential algorithms run in linear time with respect to the size of the underlying graph, whereas the parallel algorithms run (under some reasonable assumptions) in $O(n/p + K \log_2 p)$ time, where n, p, and K are the size of the underlying graph, the number of available processors, and the number of distinct level-sets of the distance map, respectively.

From a computational point of view, future work will include experimental studies of the execution times, variations on our parallel algorithms with improved load balancing, as well as algorithms for the so-called "grayscale case" in

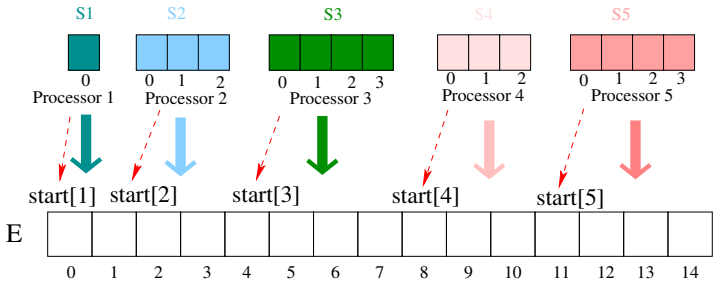

Fig. 3. Illustration of the *Union* algorithm with $p = 5$ processors.

order to filter functions as well as binary sets. On the methodological point of view, the use of distance maps in unweighted graphs opens the door towards the investigation of morphological operators on graphs embedded in metric spaces (or more generally on weighted graphs) where the result of an operator depends on the "length" of the edges according to the metric.

References

1. Chia, T.L., Wang, K.B., Chen, Z., Lou, D.C.: Parallel distance transforms on a linear array architecture. IPL 82(2), 73–81 (2002)
2. Coeurjolly, D.: 2d subquadratic separable distance transformation for path-based norms. In: Discrete Geometry for Computer Imagery. pp. 75–87. Springer (2014)
3. Cousty, J., Najman, L., Dias, F., Serra, J.: Morphological filtering on graphs. CVIU 117(4), 370–385 (2013)
4. Ladner, R.E., Fischer, M.J.: Parallel prefix computation. JACM 27(4), 831–838 (1980)
5. Man, D., Uda, K., Ueyama, H., Ito, Y., Nakano, K.: Implementations of parallel computation of Euclidean distance map in multicore processors and GPUs. In: ICNC. pp. 120–127 (2010)
6. Meyer, F., Angulo, J.: Micro-viscous morphological operators. ISMM pp. 165–176 (2007)
7. Pham, T.Q.: Parallel implementation of geodesic distance transform with application in superpixel segmentation. In: DICTA. pp. 1–8. IEEE (2013)
8. Saito, T., Toriwaki, J.I.: New algorithms for euclidean distance transformation of an n-dimensional digitized picture with applications. Pattern recognition 27(11), 1551–1565 (1994)
9. Shyu, S.J., Chou, T., Chia, T.L.: Distance transformation in parallel. In: Proc. Workshop Combinatorial Math. and Computation Theory. pp. 298–304 (2006)
10. Soille, P., Breen, E.J., Jones, R.: Recursive implementation of erosions and dilations along discrete lines at arbitrary angles. PAMI 18(5), 562–567 (1996)
11. Svolos, A.I., Konstantopoulos, C.G., Kaklamanis, C.: Efficient binary morphological algorithms on a massively parallel processor. In: IPDPS. p. 281 (2000)
12. Vincent, L.: Graphs and mathematical morphology. Sig. Proc. 16(4), 365–388 (1989)