Algorithmes et Morphologie Mathématique

Thierry Géraud & Hugues Talbot & Marc Van Droogenbroeck

8 septembre 2010

Chapitre 1

Morphologie et Algorithmes

Chapitre rédigé par Thierry GÉRAUD et Hugues TALBOT et Marc VAN DROOGENBROECK.

Table des matières

1.1. Introduction

Dans ce chapitre nous abordons le problème très important de la mise en œuvre des opérateurs, filtres et méthodologies d'analyse d'images vues dans les chapitre précédents.

D'une manière générale, on présente souvent un nouvel opérateur par sa description mathématique ; toutefois, celle-ci n'est pas toujours simple à traduire sous forme algorithmique. Pourtant, un tel opérateur a normalement un intérêt au delà de sa simple description : on cherche souvent à l'utiliser dans la pratique. Pour cela, il faut réussir à franchir le pas entre les mathématiques pures et la pratique de la programmation. On doit alors exprimer la partie mathématique d'un opérateur sous une forme applicable : un algorithme. Remarquons que si la description mathématique sert à appréhender l'opérateur, cette forme est rarement une aide à l'implantation matériellle ou à l'implémentation logicielle. Ceci explique, sans doute, pourquoi de nombreuses méthodes permettant de réaliser les opérateurs morphologiques ont été proposées. L'évolution des processeurs fait aussi que de nouvelles méthodes sont encore imaginées pour des opérateurs connus depuis plusieurs décennies.

Plutôt que d'inventorier l'ensemble des algorithmes et des structures de données proposés pour les opérations morphologiques, ce qui donnerait un catalogue gigantesque et rébarbatif, nous nous concentrerons dans ce chapitre sur des aspects purement "algorithmiques" de la morphologie mathématique. Nous laisserons donc de côté certains aspects spécifiques à leur réalisation par logiciel et par matériel.

Un algorithme est, depuis les Babyloniens jusqu'à Ada Lovelace [**?**], défini formellement comme une suite d'opérations permettant de résoudre un problème par un calcul. En morphologie mathématique, des filtres (ou opérateurs) s'appliquent généralement à des ensembles ou des fonctions et leur définition est donnée de façon formelle, mathématique.

Un algorithme est donc l'expression d'une solution effective permettant d'obtenir le résultat de l'application de ces opérateurs sur des données en entrée. La traduction en algorithme des mathématiques a pour but de faciliter la mise en œuvre d'un opérateur dans un ordinateur sous forme de programme tout en s'affranchissant de tout langage informatique. Aussi la description algorithmique doit conserver une forme abstraite pour que la mise en oeuvre puisse être réalisée dans n'importe quel environnement (plateforme, langage, boîte à outils, bibliothèque).

Il existe en informatique théorique une modélisation abstraite et générique des ordinateurs réels, appelée la machine de Turing [**?**]. Dans cette modélisation, il est possible, mais peu pratique, de mettre en œuvre tout algorithme correct. Dans l'intérêt de la lisibilité et de la compacité de la description, nous utiliserons une notation relativement plus intuitive qui, en particulier, fait appel à des *structures de données* non triviales.

Dans un premier temps (section 1.2), nous allons discuter de la traduction sous forme informatique des structures et définitions utilisées en morphologie mathématique. Ensuite (section 1.3), nous allons couvrir les différents aspects liés aux algorithmes dans le cadre propre de la morphologie mathématique. Il sera notamment question de taxonomie, de compromis et de classes d'algorithmes. Nous intégrerons ces différents aspects en analysant, en section 1.4, le cas de la reconstruction morphologique. Enfin, la section 1.5 sera consacrée aux perspectives historiques et à un parcours de la bibliographie.

1.2. Traduction définition / algorithme

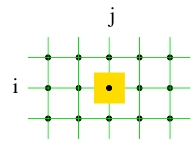
1.2.1. *Structures de données*

Avant de parler d'algorithme, il faut tout d'abord considérer quelles sont les données à traiter et comment elles se matérialisent concrètement lorsqu'elles ne sont plus seulement mathématiques.

Une image f est une fonction d'un espace $\mathcal{E}$ vers un espace $\mathcal{V}$. Comme il est impossible dans la pratique de stocker et manipuler une infinitée de données, ces espaces sont toujours discrétisés en $E \subset \mathcal{E}$ et $V \subset \mathcal{V}$:

$$f : \left\{ \begin{array}{ccc} E & \longrightarrow & V \\ p & \longmapsto & f(p). \end{array} \right.$$

Figure 1.1 – Points définis par une grille carrée et un pixel.



E est souvent doté d'une topologie discrète avec une notion de voisinage. Par convention d'écriture, un élément de E sera noté p (pour "point") et un voisin de p sera noté n. Considérons le maillage inscrit dans $\mathcal{E}$ où p est un nœud et où les arcs traduisent la notion de voisinage. Le cas le plus courant correspond à un échantillonnage régulier d'une partie de $\mathcal{E}$. Le maillage est alors régulier et les points appartiennent à une grille. La figure 1.1 montre une topologie classique d'image bi-dimensionnelle définie sur une grille carré.

Un point p d'une telle image est facilement repérable grâce à deux indices entiers (i, j) et la définition de l'image f, association d'une valeur en chaque point, peut être représentée en mémoire par un tableau à 2 dimensions : $f(p)$ s'apparente alors à $f[i, j]$. Cette représentation, très classique, permet de stocker et de modifier la valeur de f en chaque point de façon indépendante. L'affectation à f d'une valeur $v \in V$ au point $p \in E$ sera notée sous la forme abstraite $f(p) := v$ même si, dans les faits, il s'agira de réaliser $f[i, j] := v$.

Au regard de la réalisation, f n'est plus vraiment "une fonction mathématique" mais une variable (une mémoire) représentant une fonction à un instant t donné de l'exécution d'un algorithme. D'un point de vue formel, un algorithme crée une suite de fonctions f_t et chaque affectation exprime la modification de f_t en f_{t+1}. En langage algorithmique, la variable f permet d'abstraire l'existence de ces multiples fonctions ; c'est une sorte de "fonction" qui évolue au cours de l'algorithme. L'algorithmique commence à prendre ses distances vis-à-vis des mathématiques.

1.2.2. *Forme et étendue du domaine*

En informatique, la représentation d'une image f par un tableau suppose que le domaine de définition de f est fini. Dans la pratique, une image bi-dimensionnelle est généralement rectangulaire et E s'apparente à un sous-ensemble de $\mathbb{Z}^2$ ou de $\mathbb{N}^2$. Itérer sur les points d'une telle image revient à former deux boucles afin de parcourir les cellules du tableau, comme montré ci-dessous à gauche.

for i := 1 **to** n_{lignes}
  for j := 1 **to** $n_{colonnes}$
    ... *// utilisation de f[i,j]*

for_all $p \in E$
  ... *// utilisation de* $f(p)$

Néanmoins, on préférera exprimer la forme abstraite du parcours, ci-dessus à droite, plus proche du mathématicien que de l'informaticien. Cette forme, plus générale que la précédente, ne fait pas apparaître d'hypothèses implicites concernant f : le domaine de définition de f peut ne pas être rectangulaire, ni même bi-dimensionnel. La formulation abstraite d'algorithmes permet de cacher les détails de représentation des données, permettant ainsi de se concentrer sur l'aspect opérationnel des algorithmes. En revanche, cela ne doit pas nous dispenser de préciser les structures de données sous-jacentes. Lorsqu'il s'agit d'évaluer la complexité d'un algorithme, on ne peut pas occulter les opérations élémentaires réalisées sur les données. Si le parcours d'un ensemble de N points, sans contrainte particulière sur l'ordre de parcours, a une complexité en $O(N)$, l'accès aléatoire à une valeur associée à un point p donné a une complexité dépendante de la structure de données utilisée.

Dans le cas d'une image dont les données sont stockées par un tableau en mémoire, on bénéficie d'un accès aléatoire en temps constant : la lecture et l'écriture de la valeur de l'image en un point p sont réalisées en $O(1)$, quelque soit le parcours des points dans lequel s'inscrit cet accès. La représentation des données d'une image sous forme de tableau est généralement passe-partout. Elle est également compacte puisque la mémoire utilisée pour la description de la structure en elle-même, en supplément des données, est très réduite. En revanche, cette représentation n'est pas toujours la meilleure. Considérons une image décrivant le contour d'un objet. L'intérêt d'encoder cette image sous la forme d'un tableau bi-dimensionnel est que l'on peut tester en temps constant si un point du plan discret appartient au contour. D'un autre côté, cet encodage est inefficace sur deux autres aspects. Tout d'abord, ce tableau doit au moins avoir la taille du rectangle englobant le contour, ce qui est beaucoup plus coûteux en mémoire que de stocker la liste des points du contour. Ensuite, le parcours des points du contour nécessite des recherches dans le tableau : pour l'obtention d'un premier point de contour, puis pour tout passage au point suivant, dans le voisinage 2D du point courant. Un algorithme qui fonctionne sur le contour d'un objet, par exemple une dilatation, peut avoir intérêt à utiliser une structure plus adaptée.

Au final, toute structure de données utilisée au sein d'un algorithme, qu'il s'agisse d'une image ou non, est susceptible d'affecter la complexité de cet algorithme.

1.2.3. *Structure d'ensembles de points*

La représentation classique d'une fonction sous la forme d'un tableau permet également de définir sur E des ensembles de points. En effet, lorsque l'espace destination de f est l'ensemble des booléens ($V = \mathbb{B}$), f peut s'interpréter comme la fonction caractéristique d'un ensemble de points, précisemment : $F = \{\, p \in E \,|\, f(p) = \mathit{true} \,\}$. f est alors une image binaire ; l'ensemble $F \subset E$ représente un "objet" et son complémentaire $E \setminus F$ constitue le "fond" de l'image. Le parcours des points de l'ensemble F s'effectue en parcourant le domaine E et en ne retenant que les points p tels que

$f(p) = true$. Dans la suite, nous ne distinguerons pas, dans leur notation, un ensemble ($F \subseteq E$) et sa fonction caractéristique ($F : E \rightarrow \mathbb{B}$).

1.2.4. *Raccourcis d'écriture*

Au final, pour faciliter l'expression des algorithmes nous adopterons les raccourcis d'écriture suivants :

	Forme longue	Raccourcis
Parcours d'un ensemble de points $F \subseteq E$	**for_all** $p \in E$, **if** $F(p) = true$, ...	**for_all** $p \in F$, ...
Copie d'une constante c à toutes les positions d'une image f	**for_all** $p \in E$, $f(p) := c$	$f := c\,\square$
Recopie du contenu de l'image f_1 dans l'image f_2	**for_all** $p \in E$, $f_2(p) := f_1(p)$	$f_2 := f_1$

Avec de telles formes abstraites d'écriture, nous pouvons ignorer certaines difficultés pratiques. Par exemple, rien n'impose qu'une image bi-dimensionnelle soit rectangulaire (son domaine de définition peut-être quelconque). Lorsqu'il s'agit d'examiner les voisins d'un point, il faut pouvoir gérer le cas particulier des points situés sur le bord des images ; néanmoins la forme abstraite d'une itération sur le voisinage $\mathcal{N}(p)$, quelle que soit la structure de données utilisée pour décrire et gérer ce voisinage, s'écrira :

for_all $n \in \mathcal{N}(p)$
  ...

Ainsi le lecteur d'un algorithme pourra se focaliser sur l'aspect algorithmique et ne sera pas pollué par des détails d'implémentation.

Enfin, pour préciser qu'un parcours des points d'une image s'effectue dans l'ordre vidéo classique (pour chaque ligne de haut en bas, pour chaque colonne de gauche à droite), nous utiliserons le symbole $\triangleright$. L'ordre inverse, anti-vidéo (pour chaque ligne de bas en haut, pour chaque colonne de droite à gauche), sera précisé par le symbole $\triangleleft$.

1.2.5. *De la définition à la réalisation*

Comme vu précédemment, si les opérateurs morphologiques sont très souvent exprimés sous la forme d'une expression mathématique, leur traduction en algorithme n'est pas toujours simple et directe. De plus, dans les cas favorables où l'on dispose d'une telle traduction, cette dernière est rarement la traduction la plus appropriée, au sens de l'efficacité du programme résultant. En mathématique, on se préocupe essentiellement du fait que l'algorithme est correct plutôt qu'efficace dans la pratique.

Nous allons prendre l'exemple de la dilatation binaire. La définition d'une dilatation peut s'opérer de plusieurs manières différentes :

$$
\begin{aligned}
X \oplus B &= \bigcup_{b \in B} X_b & (1.1)\\
&= \{\, x + b \in E \mid x \in X,\, b \in B \,\} & (1.2)\\
&= \{\, p \in E \mid \exists b \in B,\, p - b \in X \,\} & (1.3)
\end{aligned}
$$

La définition (1.1) peut se traduire par l'algorithme (1) donné en figure 1.2 (lignes de 1 à 16). En termes informatiques, le point d'entrée est SETDILATION, et utilise la fonction TRANSLATE. En termes algorithmiques, on constate que la méthode reprend exactement la définition, ce qui suffit à justifier que l'algorithme est correct.

Dans une unité de calcul, un facteur de coût essentiel est l'accès à la mémoire, en lecture mais surtout en écriture. On ne va compter ici que les écritures mémoires (affectations) à $true$ pour simplifier. On constate que cette première version nécessite $|B| \times |X| \times 2$ affectations, où la notation $|.|$ désigne le nombre d'éléments d'un ensemble.

La seconde version, qui est issue de la définition (1.2), consomme deux fois moins d'affectations que la précédente. Elle est représentée en figure 1.2 par la routine DILDIRECT (lignes de 33 à 46). Quant à l'algorithme issu de la formule (1.3), il est nettement plus efficace car il ne consomme que $|X \oplus B|$. On a ici remplacé un produit par une somme. Ce dernier algorithme, donné en figure 1.2 par la routine DILREVERSE (lignes de 49 à 65), est l'implémentation “classique” de la dilatation telle qu'on la rencontre dans les bibliothèques logicielles de traitement d'images.

Sur ces exemples simples, on voit bien la distance entre la formulation concise issue des définitions et la mise en œuvre effective en algorithmes, nettement plus verbeuse.

L'exemple de la dilatation est essentiellement utile pour réaliser la distance qui peut séparer définition et algorithme. Nous allons maintenant donner un exemple d'algorithme plus complet pour illustrer diverses stratégies algorithmiques pour un même opérateur.

Figure 1.2 – Dilatations ensemblistes.

```
// algorithme (1)

SETDILATION(X : Image of 𝔹,
            B : Set of Point)
  → Image of 𝔹
begin
  data X_b, U : Image of 𝔹
  // initialisation à l'ensemble vide
  U := false □
  for_all b ∈ B
    // calcul de X_b
    X_b := TRANSLATE(X, b)
    // mise à jour de U
    U := UNION(U, X_b)
  return U
end

TRANSLATE(X : Image of 𝔹,
          b : Point)
  → Image of 𝔹
begin
  data O : Image of 𝔹
  // initialisation à l'ensemble vide
  O := false □
  // calcul de l'ensemble
  for_all p ∈ X
    if p + b ∈ E
      O(p + b) := true
  return O
end
```

```
// algorithme (2)

DILDIRECT(X : Image of 𝔹,
          B : Set of Point)
  → Image of 𝔹
begin
  data O : Image of 𝔹
  O := false □ // initialisation
  for_all p ∈ E
    for_all b ∈ B
      if X(p) = true and p + b ∈ E
        O(p + b) := true
  return O
end


// algorithme (3)

DILREVERSE(X : Image of 𝔹,
           B : Set of Point)
  → Image of 𝔹
begin
  data O : Image of 𝔹
  for_all p ∈ E
    for_all b ∈ B
      if p − b ∈ E and X(p − b) = true
        // existence
        O(p) := true
        goto next
    O(p) := false // pas d'existence
    label next
  return O
end
```

La plupart des algorithmes de morphologie mathématique sont de complexité pseudo-polynomiale. Par exemple, l'algorithme trivial de dilatation (une double boucle) a une complexité de $\mathcal{O}(N \times M)$ avec N le nombre de points de l'image et M le nombre de points de l'élément structurant.

1.3. Taxonomie des algorithmes

Dresser une taxonomie des algorithmes utilisés en morphologie mathématique est une tâche difficile pour plusieurs raisons. D'une part, il est impossible de citer en quelques pages ne serait-ce que parce que la communauté scientifique continue à proposer des nouveaux algorithmes, même pour opérateurs connus depuis les années 70 ! D'autre part, il n'existe pas vraiment un ensemble à la fois réduit et satisfaisant de critères sur lesquels pourraient s'appuyer une telle taxonomie. Enfin, il n'y a pas d'algorithme universel pour l'ensemble des opérateurs morphologiques. Chaque opérateur amène son lot d'algorithmes, de particularités, de compromis, de structures de données, etc.

1.3.1. *Critères de taxonomie*

Les critères de taxonomie qui permettraient de classer les algorithmes sont multiples et variés. Pour illustration, nous en donnons ci-après une première liste, non exhaustive :

– le type utilisé de structures auxiliaires de données (file, arbre ou autre) ;

– la nature du parcours des points dans l'image ;

– la complexité de l'algorithme ;

– l'occupation mémoire requise ;

– les propriétés remarquables de l'algorithme ;

– les conditions (et donc les limites) d'utilisation de l'algorithme ;

– la classe d'opérateurs ou de filtres concernée ;

– la portée de l'algorithme ;

– son intention ;

– son sujet (les données sur lesquelles il porte).

De plus, les contraintes que subit l'utilisateur et qui sont liées à son contexte fournissent également d'autres critères possibles de taxonomie : son domaine d'application, la nature des données, ses objectifs, etc.

À défaut de parcourir cette liste en détail, notons que certains critères permettent d'opposer certains algorithmes tandis que d'autres ne permettent que de les distinguer. Les deux tableaux ci-dessous donnent respectivement quelques exemples de ces deux cas de figure.

critère : portée de l'algorithme	
spécifique	**générale**
décomposition de l'élément structurant (afin d'accélérer le calcul des dilatations et érosions) ;	algorithme de Dijkstra (pour le calcul d'une fonction distance) ; algorithme de Viterbi (pour le filtrage par élagage décrit au chapitre 5.3 du volume 1) ; algorithme de Prim ou de Kruskal (pour la segmentation par calcul de l'arbre de poids minimum décrite au chapitre 7 du volume 1).

critère : propriété	
parallèle	**séquentiel**
algorithme classique de dilatation ou d'érosion par élément structurant (la version (3) décrite plus loin en section 1.2.5) ; détection des points simples	fonction distance de chanfrein (Cf. chapitre 2.4.3 du volume 1) ; filtres alternés séquentiels (Cf. chapitre 1.2.7 du volume 1)

critère : nature des données	
faible quantification[1]	**forte quantification**
tri par distribution ou par base ; utilisation des représentations par arbre (Cf. chapitre 5.2 du volume 1)	tri rapide, par tas

critère : parcours des points	
par front	**par passe**
dilatation par distance ; la majorité des algorithmes de calcul de la ligne de partage des eaux (Cf. chapitres 1.5 et 4 du volume 1)	dilatation triviale ; transformée en tout ou rien (Cf. chapitre 1.1.3 du volume 1)

Le second tableau ci-dessous illustre des critères permettant de discriminer les algorithmes.

critère : structures auxiliaires
tableaux, files, queues de priorité, arbres, graphes, etc.

critère : intention
simplification des données, obtention d'une transformée, calculs / estimations sur les données, partitionnement, etc.

critère : sujet de l'algorithme
pixels, textures, objets, régions, contours, etc.

1.3.2. *Compromis*

La mise en œuvre d'une chaîne de traitement d'images n'est qu'un cas particulier de la réalisation de calculs scientifiques par l'informatique. Lorsqu'un module de cette chaîne est un opérateur morphologique, il ne peut pas échapper aux lois de ce cadre plus général. En particulier, une contrainte que l'on rencontre bien souvent est de trouver un compromis acceptable entre les trois notions antagonistes suivantes.

– La performance attendue en temps d'exécution. Certaines applications nécessitent de tourner en temps-réel ; à l'opposé, d'autres applications peuvent se permettre de prendre tout leur temps. Outre la question du temps nécessaire à l'exécution de l'opération se pose celle de sa variabilité. Une implantation matérielle préfèrera un temps d'exécution constant, quitte à allonger, dans l'absolu, le temps de calcul utilisé.

– Le stockage. Les ressources utilisées en termes d'utilisation mémoire et/ou d'espace disque sont généralement limitées tandis que les données à traiter, pour une même application, sont généralement d'un ordre de grandeur de taille constant. Le choix d'un algorithme et des structures de données qu'il utilise est alors souvent conditionné par les ressources de stockage disponibles.

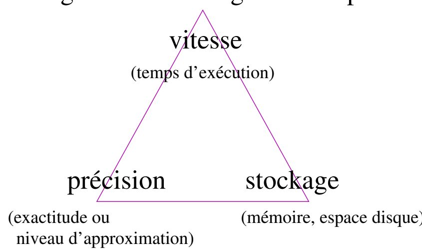
– Le résultat d'un calcul peut être soit exact soit approché et, dans ce dernier cas, un niveau donné de précision est généralement attendu. Dans la pratique, on peut souvent se contenter d'un résultat approché.

Suivant les contraintes imposées par le contexte applicatif, la mise en œuvre d'un opérateur morphologique doit se positionner dans le triangle formé par ces trois notions antagonistes, illustré en figure 1.3. Privilégier la vitesse d'exécution, par exemple, se fait *a priori* en sacrifiant de la précision et/ou des ressources. Notons que les architectures modernes permettent la réalisation en temps réel de nombreuses opérations morphologiques. Dans ce cas, la notion de précision n'est plus pertinente.

1.3.3. *Classes d'algorithmes et canevas*

Un algorithme agissant sur des images doit généralement s'appuyer sur les parties suivantes :

Figure 1.3 – Triangle de compromis.



– un ou plusieurs parcours d'image ;

– des structures de données auxiliaires (d'autres images, ou d'autres structures, classiques ou non) ;

– une logique de traitement, en plusieurs phases, souvent “initialisation, boucles, finalisation” ;

– souvent une notion, importante en morphologie, de voisinage et d'inspection autour d'un point.

Considérons l'ensemble des algorithmes connus qui permettent de mettre en œuvre les opérateurs morphologiques. Nous pouvons alors observer que plusieurs groupes d'algorithmes se forment tels qu'au sein de chaque groupe, les algorithmes se ressemblent. Les algorithmes d'un groupe donné partagent un même schéma algorithmique, c'est-à-dire un même enchaînement d'opérations, une même utilisation de structures de contrôle (comme les boucles), ainsi que les mêmes données auxilaires. Nous sommes en présence d'une classe d'algorithmes.

Nous nommerons *canevas* la représentation d'une classe d'algorithmes sous la forme d'un “algorithme à trous.” Ce canevas, ou motif algorithmique, est exactement comparable à un patron en couture, où le choix de la matière, de la couleur et des ornements reste encore à définir. La figure 1.4 montre deux canevas très classiques de traitement d'images. À gauche, un traitement point-à-point sur les valeurs des pixels d'une image ; la valeur au point p dans l'image de sortie o est obtenue en appliquant une fonction h à la valeur de ce même point p dans l'image d'entrée f : $o(p) := h(\, f(p)\,)$. Avec $h = C$ (fonction de complémentation), ce canevas devient l'opérateur de complémentation ; avec $h = lum$ (fonction de luminance), il s'agit alors d'une conversion d'image couleur à image à niveaux de gris.

Le canevas de droite, figure 1.4, évalue chaque valeur $o(p)$ de l'image de sortie en fonction de l'ensemble des valeurs de l'image d'entrée appartenant à une fenêtre w centrée en p. C'est le canevas des convolutions avec $h(\, \{\, f(q) \,|\, q \in w(p)\, \}\,) = \sum_q g(p-q)\, f(q)$, où g est la fonction noyau de la convolution, et de la dilatation sur les fonctions avec $h = \vee$ (supremum).

Figure 1.4 – Deux canevas : “point-à-point” et “fenêtre glissante.”

```
POINT_WISE(f : Image,
           h : Function)
   → o : Image

begin
   for_all p ∈ E
      o(p) := h(f(p))
end
```

```
SLIDING_WINDOW(f : Image,
               w : Window,
               h : Function)
   → o : Image
begin
   for_all p ∈ E
      o(p) = h( { f(q) | q ∈ w(p) } )
end
```

L’intérêt des canevas algorithmiques est multiple.

– Ils sont généraux. Chaque canevas n’est pas spécifique à un traitement en particulier. Au contraire, chaque canevas est "transposable" afin de réaliser d’autres traitements ; pour cela, il suffit de définir sa partie variable (h dans les exemples ci-avant).

– Ils sont abstraits. Leur expression ne fait aucune hypothèse qui limiterait leur utilisation à certaines natures de données. Par exemple, la présence d’une double boucle sur les coordonnées des points dans les cavenas de la figure 1.4 signifirait implicitement qu’ils sont restreints aux images bi-dimensionnelles, ce qui est clairement faux.

– Ils sont pédagogiques. Ils matérialisent un méta-algorithme dont la compréhension d’un point de vue algorithmique entraîne celle des opérateurs qui vont en être dérivés.

1.4. Exemple de la reconstruction géodésique

Les diverses classes d’algorithmes que nous allons présenter ci-après n’ont pas été proposées simultanément. Elles se situent dans une perpective historique dont nous parlerons dans la section 1.5.

En guise d’illustration de ces classes, nous allons montrer plusieurs façons de mettre en œuvre, en algorithmes, un même opérateur morphologique : la reconstruction géodésique par dilatation d’une fonction (une image en niveaux de gris par exemple) [**?**].

Cet exemple en forme d’exercice de style est significatif ; il permet d’illustrer une grande partie des approches qui sont utilisées par d’autres opérateurs.

1.4.1. *La version mathématique : algorithme parallèle*

Dans un premier temps, nous pouvons mettre en œuvre la reconstruction géodésique par dilatation en partant de sa définition mathématique :

Figure 1.5 – Canevas de reconstruction (partie 1/2). L'algorithme parallèle (à gauche) et l'algorithme séquentiel (à droite) sont décrits respectivement en sections 1.4.1 et 1.4.2.

```
RD_PARALLEL(f : Image,
            g : Image)
   → o : Image
begin

  data
    o′ : Image
    stability : 𝔹

  // initialisation
  o := f

  // itérations
  repeat
    o′ := o // échange

    // opère
    for_all p ∈ E
      o(p) := max{ o′(q) |
                   q ∈ 𝒩(p) ∪ {p} }


    // conditionne
    for_all p ∈ E
      o(p) := min{ o(p), g(p) }

    stability := (o = o′)
  until stability
  return o

end
```

```
RD_SEQUENTIAL(f : Image,
              g : Image)
   → o : Image
begin

  data
    o′ : Image
    stability : 𝔹

  // initialisation
  o := f

  // itérations
  repeat
    o′ := o // mémorisation

    // passe 1
    for_all p ∈ E ▷
      o(p) := min{ max{ o(q) |
                   q ∈ 𝒩⁻(p) ∪ {p} }, g(p) }

    // passe 2
    for_all p ∈ E ◁
      o(p) := min{ max{ o(q) |
                   q ∈ 𝒩⁺(p) ∪ {p} }, g(p) }

    stability := (o = o′)
  until stability
  return o

end
```

$$\mathcal{R}_g^\delta(f) \;=\; \lim_{n\to\infty} \delta_g^n(f) \;=\; \delta_g^\infty(f)$$

où :

$$\begin{array}{lcl} \delta_g^1(f) & = & \delta(f) \bigwedge g, \\ \delta_g^{n+1}(f) & = & \delta(\,\delta_g^n(f)\,) \bigwedge g. \end{array}$$

Cette définition de la reconstruction est à rapprocher de celle donnée au chapitre 1.2.2 du volume 1. Par rapport à cette dernière, ensembliste, il s'agit d'une généralisation aux fonctions. Ici δ est la dilatation géodésique : $\delta(f) = f \oplus B$, où $B = \mathcal{N}(0) \cup \{0\}$, avec $\mathcal{N}$ le voisinage considéré et 0 l'origine de l'espace.

Une hypothèse implicite pour que cet opérateur, ainsi défini, ait un sens est que l'on doit avoir $f \leq g$. Dit autrement, la fonction marqueur f à dilater doit être "sous" la fonction masque g. Comme dans le cas ensembliste, le but de la reconstruction sur les fonctions consiste à retrouver le contenu de g à partir d'une image f.

La définition de cette reconstruction est, en soi, déjà un algorithme : il s'agit d'itérer jusqu'à convergence une dilatation géodésique suivie du conditionnement.

La récursion s'implante naturellement à l'aide d'une boucle et l'algorithme se termine par la stabilité du résultat (aucune modification durant la dernière itération), précisément sur une image d'extension finie. L'algorithme a une convergence garantie, néanmoins très lente en pratique. En effet, au cours des passes successives, on considère de nouveau des zones d'images qui ont déjà atteint une condition locale de convergence. Cet algorithme est illustré par la routine RD_PARALLEL en figure 1.5.

1.4.1.1. *Algorithmes similaires*

Ce type d'algorithme, « modification itérative jusqu'à stabilité, » n'est pas propre au domaine de la morphologie mathématique ; il est utilisé, entre autres, pour réaliser des diffusions.

La complexité d'une passe est ici pseudo-polynomiale avec le nombre de points de l'image et la connexité. La complexité de l'algorithme, dans le pire des cas (une image d'une courbe de Peano), est polynomiale en $\mathcal{O}(M \times N^2)$.

En revanche, cet algorithme est facilement parallélisable puisque les calculs en chaque point à l'intérieur des boucles ne dépendent que de l'image de travail obtenue à l'itération précédente. Contrairement aux autres algorithmes présentés par la suite, il n'existe en effet aucune dépendance entre le calcul courant au point p et les calculs aux points voisins lors d'une même itération.

1.4.2. *Algorithme séquentiel*

Une amélioration de la complexité est possible en remarquant que la reconstruction peut s'exprimer de manière séquentielle. Dans la version précédente parallèle, on calculait chaque dilatation indépendamment (lignes 18 et 20). L'image o se calcule à partir de l'image o' obtenue à l'itération précédente.

Dans cette nouvelle version, il devient inutile de calculer une image o' : la dilatation a lieu en place : elle est calculée dans un voisinage et est inscrite immédiatement dans l'image de travail o. Une modification réalisée au point p peut se propager au cours de la même passe sur l'image.

Dans l'algorithme tel qu'il est donné, pour garder un pendant avec l'algorithme parallèle, on a gardé une copie dans une image o' pour tester la condition de stabilité, mais on peut aussi compter au cours de chaque passe le nombre de pixels modifiés. S'il est nul après une passe (une boucle dans chaque sens), la stabilité est atteinte.

La complexité de cet algorithme n'est théoriquement pas meilleure que dans le cas précédent : on peut en effet exhiber un objet binaire à reconstruire en forme de courbe de PEANO qui exige un nombre de passes sur l'image proportionnel au nombre de points de la courbe. En revanche, dans le cas des objets convexes, on peut obtenir une reconstruction complète en 2 passes sur l'image. Dans la pratique, on constate que les images naturelles comportent de nombreuses parties localement convexes, qui sont reconstruites efficacement.

1.4.2.1. *Algorithmes similaires*

La classe des algorithmes similaires à celui-ci est vaste. Elle comprend en particulier les calculs de carte de distance discrètes, telle les distance de chanfrein, et la distance pseudo-euclidienne de Danielsson [**?**].

1.4.3. *Algorithme à base de file d'attente*

Dans cette version de l'algorithme de reconstruction, on utilise une structure de données supplémentaire : la file d'attente simple (*FIFO* en anglais : *First In, First Out* ; PAPS en français : Premier Arrivé, Premier Servi). L'idée générale est de réaliser la dilatation à l'aide d'un front qui se propage dans l'image, tout en étant conditionnée par l'image g (ligne 90).

L'avantage premier de cette approche est la simplicité de la formulation : la dilatation s'effectue de manière indépendante d'un nombre de passes. On évite ainsi potentiellement de grands parcours sur l'image qui ne sont plus sujets à modification.

La plupart des algorithmes basés sur des files d'attente utilisent le même schéma, donné ci-dessous :

– on initialise la file d'une certaine manière ;

– tant que la file n'est pas vide :
  1) on retire un point de la file,
  2) on effectue le traitement en ce point,
  3) on ajoute conditionnellement les voisins de ce point dans la file.

Figure 1.6 – Canevas de reconstruction (partie 2/2). Les algorithmes à base de file d'attente (à gauche) et hybride (à droite) sont décrits respectivement en sections 1.4.4 et 1.4.3.

```
RD_QUEUE_BASED(f : Image,
               g : Image)
  → o : Image
begin
  data
    q : Queue of Point
    M : Image

  // initialisation
  M := REGIONAL_MAXIMA(f)
  for_all p ∈ M
    for_all n ∈ N(p)
      if n ∉ M
        q.PUSH(p)
  o := f

  // propagation
  while not q.EMPTY()
    p := q.FIRST()
    for_all n ∈ N(p)
      if o(n) < o(p) and o(n) ≠ g(n)
        o(n) := min{o(p), g(n)}
        q.PUSH(n)

  return o
end
```

```
RD_HYBRID(f : Image,
          g : Image)
  → o : Image
begin
  data
    q : Queue of Point

  // initialisation
  o := f

  // une séquence de 2 passes
  for_all p ∈ E ▷
    o(p) := min{ max{ o(q) |
              q ∈ N⁻(p) ∪ {p} }, g(p) }
  for_all p ∈ E ◁
    o(p) := min{ max{ o(q) |
              q ∈ N⁺(p) ∪ {p} }, g(p) }
    // avec mise en queue
    for_all n ∈ N⁺(p)
      if o(n) < o(p) and o(n) < g(n)
        q.PUSH(p)

  // propagation
  while not q.EMPTY()
    p := q.FIRST()
    for_all n ∈ N(p)
      if o(n) < o(p) and o(n) ≠ g(n)
        o(n) := min{o(p), g(n)}
        q.PUSH(n)

  return o
end
```

Ce schéma correspond exactement à un parcours en largeur d'abord dans le graphe de voisinage de l'image. D'autres parcours sont possibles en intervertissant les étapes 1 et 3, ce qui donne localement un parcours en profondeur d'abord.

Ici, l'initialisation part d'une détection des maxima régionaux. Le contour extérieur de ces maxima est mis dans la file. Ces maxima sont propagés par la file d'attente. L'opération réalisée par la file d'attente n'est pas une dilatation complète, au sens où à aucun moment l'opération de propagation ne correspond à une dilatation par élément structurant, même élémentaire. On dilate simplement les maxima. L'opération réalisée est néanmoins une dilatation algébrique (Cf. chapitre 2 du volume 1). De ce fait, l'opération globale incluant le masque g est identique à la reconstruction définie plus haut [**?**].

Le cœur de l'algorithme, présenté en figure 1.6, colonne de gauche, réside dans la propagation : lignes 85–91. On constate facilement qu'on ne traite qu'une unique fois un point p de l'image – la variable du même nom dans les algorithmes décrits. Ce n'est pas le cas des algorithmes précédents. Le cœur de l'algorithme est donc de complexité linéaire.

En revanche, la partie d'initialisation exige de calculer les maxima régionaux (ligne 72) ce qui est relativement coûteux : l'opération est équivalente à un étiquetage, qui est de complexité quasi-linéaire par union-find [**?**].

Plus généralement, l'utilisation d'une structure à accès aléatoire, comme une file d'attente ou une pile, s'avère efficace dans la mesure où une étape préparatoire extrait les bonnes informations. On peut à nouveau faire le lien avec la question de la redondance. Dans un processus de dilatation, morphologique ou géodésique, l'information pertinente est la position et la valeur du maximum local. Les alternatives consistent soit à exploiter les informations pertinentes sans se soucier de leur position et de leur valeur –il s'agit d'un algorithme aveugle au contenu–, soit à trouver préalablement les informations utiles et à les propager dans un second temps. Parmi les approches algorithmiques les plus rapides (mais aussi les plus complexes), on trouve celles qui sélectionnent les informations locales et les propagent directement, en intégrant les phases de détection et de propagation lors d'un unique parcours de l'image. L'algorithme proposé dans [**?**] pour l'ouverture morphologique est de ce type. On analyse les valeurs et, lorsqu'un extremum local est trouvé, la détection laisse place à la propagation jusqu'à trouver un nouvel extremum local.

Nous verrons dans la section suivante une reconstruction plus efficace.

1.4.3.1. *Remarques*

Algorithmes similaires : Parmi les algorithmes similaires en terme d'implémentation on peut citer,

– pour l'exploration en largeur d'abord : fonction distances [**?**], SKIZ, squelettisation [**?**], dilatations ordonnées [**?**],

– et pour l'exploration en profondeur d'abord : le calcul de l'arbre des composantes [**?**].

Utiliser une pile au lieu d'une file : L'usage d'un algorithme à pile (LIFO, PADS), au lieu d'une file, peut s'avérer pertinent lorsque la stucture de pile est utilisée pour stocker des informations relatives à certaines données. Néanmoins, l'utilisation d'une pile est souvent à déconseiller. Le comportement d'un algorithme à pile est récursif puisqu'on empile les traitements en attente avant de les effectuer. Aussi, même si l'algorithme est théoriquement correct, il peut dans certains cas devenir inefficace, car pénalisé par la gestion de la structure de données, dont la taille peut croître jusqu'à la taille de l'image elle-même.

Utilisation d'une file à priorité : Une file d'attente simple ne gère qu'un aspect des objets qui y résident : leur ordre d'entrée dans la pile, qui correspond souvent à un ordonnancement spatial dans les images. Il est utile parfois d'utiliser une priorité plus générale, tenant compte par exemple des niveaux de gris. Une structure spéciale très efficace pour les images à petit nombre de niveaux de gris à été proposée par F. Meyer dans le cadre d'un algorithme de la LPE : la file d'attente hiérarchique [**?**]. Plus généralement, on peut utiliser une *file à priorité*, structure classique en algorithmique, qui peut être implémentée par un *tas* ou un arbre binaire complet par exemple.

Complexité : Suivant les algorithmes et les opérations qu'ils utilisent, certaines structures de données sont plus ou moins appropriées. Une étude des différentes structures classiques utilisables en morphologie a été proposée par E. Breen and D. Monro [], démontrant bien la différence entre la théorie et la pratique dans ce domaine. En particulier les files de Fibonacci [**?**] bien que théoriquement efficaces se révèlent plutôt lentes en pratique. En terme de recommandation, lorsque l'on cherche une file à priorité efficace utilisable avec tout type de donnée, dont le tri est stable (c'est à dire que l'ordre d'entrée dans la pile est préservé), ce qui est assez souvent le cas en imagerie, les *splay-queues* peuvent être un bon choix [**?**].

1.4.4. *Algorithme hybride*

Dans cet algorithme, en figure 1.6, colonne de droite, on retrouve d'abord une partie séquentielle limitée à deux passes. Durant celles-ci, la reconstruction est menée dans les régions convexes de l'image. Dans une seconde partie, on retrouve la propagation de l'algorithme à base de queue qui complète la reconstruction jusqu'à convergence.

L'avantage de cette méthode tient en plusieurs points. La recherche des maxima régionaux est inutile, on placera sur la queue la frontière obtenue à la suite des deux passes séquentielles. D'autre part, une partie majeure de la reconstruction, dans le cas habituel des images naturelles est effectuée au cours des passes séquentielles. La propagation finale, plus coûteuse à cause de la gestion de la structure de donnés de la queue, n'est menée que sur une petite portion de l'image.

Les algorithmes hybrides pour lesquels il existe une réelle synergie entre les différentes approches sont relativement rares dans la littérature.

1.4.4.1. *Remarques*

On retrouve les cas défavorables des algorithmes précédents : courbes fractales par exemple. Dans ce cas, la file reste de taille faible mais on fait beaucoup de boucles. Une façon de rendre cet algorithme plus performant est d'utiliser une file d'attente implémentée par un tableau circulaire (beaucoup plus compacte en mémoire et plus efficace lors des insertions et suppressions d'éléments).

Indépendamment du dernier cas de l'utilisation des files d'attente à priorité, cet exemple met en lumière que l'algorithme effectue une série d'opérations sur l'image et sur la structure de données. Certains auteurs étudient la complexité de leur algorithme en termes de manipulation des données de l'image, oubliant qu'il y a un coût lié à l'utilisation d'une structure de données additionnelle. Sous sa forme simple, cette structure de données est par exemple l'utilisation d'une mémoire-tampon, dont l'utilisation à un coût non négligeable, notamment en écriture. Mais caractériser finement le coût des structures additionnelles est une opération délicate, notamment parce qu'elle est liée au matériel utilisé.

1.4.5. *Algorithme par Union-Find*

L'algorithme d'Union-Find est une recherche de classes d'équivalences sur un graphe. Il est présenté en figure 1.7. L'algorithme est relativement complexe mais on peut sinon l'expliciter, du moins donner sa teneur générale. Il est constitué de trois phases : une initialisation, une étape d'union et une étape d'étiquetage (find).

1.4.5.1. *Idée générale de l'algorithme*

Le cœur de l'algorithme est un changement de représentation de l'image : on passe du domaine des pixels, sans notion de connection autre que locale, à une structure d'arbre, dont la racine est dans les parties basses de l'image et les feuilles dans les parties hautes.

En phase d'initialisation, tous les points de l'image g sont triés par niveau de gris décroissant dans S (ligne 164).

Dans la phase d'union, on parcourt les points de l'image dans cet ordre en parcourant S séquentiellement. Tout point considéré est soit isolé, auquel cas il forme un singleton (un nouveau maximum régional), ou bien connecté à un maximum régional de g, auquel cas il forme la nouvelle racine de l'arbre associé. Un point important est que ce maximum régional de g est lié également à un maximum régional de f. Au fur et à mesure du parcours de S, on crée ainsi une forêt d'arbres, qui correspondent à tous les maxima régionaux de g.

Dans la phase finale (find), on distingue les points qui sont tels que $g > f$, auquels cas ils leur est attribué la valeur maximale de f dans la composante connexe du maximum régional de g, ou bien le cas $g = f$, auquel cas leur valeur est inchangée.

Figure 1.7 – Reconstruction par dilatation avec l'union-find. Cet algorithme est décrit en section 1.4.5.

```
MAKE_SET(p : Point)
begin // crée le singleton { p }
  parent(p) := p
end

IS_ROOT(p : Point) → 𝔹
begin // teste si p est un représentant
  return parent(p) = p
end

FIND_ROOT(p : Point) → Point
begin // trouve le représentant de p
  if IS_ROOT(p)
    return p
  else
    parent(p) := FIND_ROOT(parent(p))
    return parent(p)
end

DO_UNION(n : Point, p : Point)
begin // unit deux arbres
  r := FIND_ROOT(n)
  if r ≠ p
    if g(r) = g(p) or g(p) ≥ o(r)
      parent(r) := p
      o(p) := max(o(r), o(p))
    else
      o(p) := MAX
end

RD_UNION_FIND(f : Image,
              g : Image)
 → o : Image
begin
  data
    parent : Image of Point
    S : Array of Point

  // initialisation
  o := f
  S := SORT(g) // suivant ▷ et g(p) ↓

  // première phase
  for_all p ∈ S
    MAKE_SET(p)
    for_all n ∈ 𝒩(p) if DEJA_VU(n)
      DO_UNION(n, p)

  // seconde phase
  for_all p ∈ S⁻¹
    if is_root(p)
      begin
        if o(p) = MAX, o(p) := g(p)
      end
    else
      o(p) := o(parent(p))

  return o
end
```

Au final, on a dilaté la fonction f sous g, ce qui correspond exactement à l'opérateur de reconstruction.

1.4.5.2. *Détails*

L'image de sortie o est utilisée tout du long. On y stocke l'état de traitement des composantes. L'image o prend ses valeurs finales dans la dernière phase de traitement.

Pour toutes les zones plates où au final on aura $o = g$, on ne gère pas de structure d'arbre (ce sont tous des singletons).

La fonction DEJA_VU s'évalue à la volée sans stockage intermédiaire.

Si les valeurs de g sont faiblement quantifiées (exemple : image sur 8 bits), on peut réaliser un tri linéaire (radix sort ou tri par histogramme). Au final l'algorithme est quasi-linéaire dans le cas où FIND_ROOT opère avec un arbre équilibré et quand on utilise une technique dite de compression de chemin [**?**].

1.4.5.3. *Comparaison avec les méthodes précédentes*

Cet algorithme est quasi-linéaire dans le pire cas, mais n'est pas forcément plus rapide dans la pratique que l'algorithme hybride. Cependant il est emblématique des mises en œuvre modernes des opérateurs connexes : ouverture/fermetures algébriques par attributs, nivellements, ligne de partage des eaux [**?**, **?**, **?**, **?**].

La représentation par forêt d'arbres en particulier permet une description théorique des opérateurs connexes particulièrement riche [**?**, **?**]. Voir également les chapitres 5 et 7 du volume 1.

Dans les méthodes parallèles et séquentielles, le facteur limitant au point de vue complexité était le nombre de passes sur l'image. Pour l'algorithme à base de queue et hybride, le facteur limitant provient de la gestion de la file d'attente, sachant qu'on peut devoir, dans les cas défavorables, parcourir une grande partie de l'image.

Enfin pour le cas de l'union-find, le goulet d'étranglement est la recherche de racine de l'arbre.

1.4.6. *Comparaison de ces algorithmes*

Afin de comparer les cinq algorithmes décrits précédemment, nous pouvons reprendre certains critères de taxonomie donnés en section 1.3.1. Le tableau ci-dessous illustre d'une part la diversité des algorithmes possibles pour traduire un même opérateur et, d'autre part, la difficulté de l'utilisation de ces critères. En effet, certains algorithmes ne sont pas monolithiques ; ils répondent alors à plusieurs critères. L'algorithme hybride, par exemple, n'est qu' "à moitié séquentiel" et utilise à la fois des passes et un front de propagation. Pour l'algorithme par Union-Find, on peut affirmer qu'il est "séquentiel" puisque, avec l'initialisation (le tri) et les deux passes, l'image est parcourue exactement trois fois ; l'ordre de parcours n'est cependant pas celui des séquences "classiques" vidéo et anti-vidéo.

algorithme	propriété	parcours	structures
parallèle	parallèle	passes	aucune
séquentiel	séquentiel	passes	aucune
à base de file	néant	front	file
hybride	1/2 séquentiel	2 passes + front	file
avec Union-Find	"séquentiel"	3 passes	tableau et arbre

Pour comparer les performances de ces cinq algorithmes de reconstruction géodésique par dilatation, nous avons pris pour g l'image `lena` (taille 512×512 pixels) et pour f le conditionnement par g du maximum point à point entre l'image `lena` originale et `lena` tournée de 90° dans le sens horaire. Le voisinage considéré est celui de la 4-connexité. Comme il s'agit d'une comparaison, nous n'avons pas utilisé d'optimisation particulière (l'utilisation d'arithmétique de pointeurs par exemple) ; pour information, ces temps peuvent être facilement réduits d'un même facteur multiplicatif significatif.

algorithme	temps (en sec.)
parallèle	25,28
séquentiel	3,18
à base de file	0,65
hybride	0,34
avec Union-Find	0,34

Ce tableau montre parfaitement à quel point un même opérateur peut se traduire dans la pratique par des performances radicalement différentes suivant l'algorithme qui l'implémente. L'obtention d'un "bon" algorithme, possédant des propriétés en adéquation avec les attentes du praticien, est finalement une science en soi.

1.5. Perspectives historiques et notes bibliographiques

L'historique des divers algorithmes et les liens entre la morphologie mathématique et les autres domaines apparentés à celle-ci prendrait à lui seul un livre entier. Nous devons nous contenter ici de quelques notes.

1.5.1. *Pré- et péri-morphologie*

Comme toutes les disciplines scientifiques, la morphologie mathématique n'est pas née, pas plus qu'elle n'évolue, dans un vide intellectuel. Elle est une étape vers une meilleure connaissance de la représentation spatiale des objets physiques ou virtuels. Avant que Matheron et Serra ne baptisent la morphologie mathématique en 1962 [**?**], l'analyse d'image existait déjà et de nombreux algorithmes avaient été développés dans des disciplines annexes. De même, la morphologie continue à évoluer dans un contexte lui-même mouvant. Il est utile de fixer quelques repères algorithmiques dans ce foisonnement créatif.

1.5.1.1. *Algorithmes à base de graphes*

La représentation des images étant le plus souvent sous forme de graphe régulier, il est peu étonnant que de nombreux algorithmes de morphologie mathématique dérivent d'algorithmes classiques. On peut citer les chemins minimaux Dijkstra [?], le problème de l'arbre de poids minimum [?, ?]. L'algorithme classique d'Union-Find [?] est également un algorithme classique de graphes très utilisé en morphologie mathématique. Nous conseillons l'ouvrage de référence [?] pour tous ces algorithmes.

Il est plus que probable que toute la littérature classique ou récente sur les algorithmes à base de graphes n'ait pas encore été exploitée dans le contexte de la morphologie mathématique. Il s'agit donc d'une mine très riche pour de futurs résultats.

1.5.1.2. *Algorithmes de géométrie et de topologie discrète*

La géométrie discrète est un champ de recherche actif lié à celui de la morphologie mathématique proprement dite. Son but est de redéfinir et d'exploiter, algorithmiquement dans un contexte discret, les objets et opérateurs de la géométrie classique. Par exemple, droites, plans, intersections, vecteurs normaux, etc [?]. Les propriétés de ces objets sont nettement différentes de celles de leurs correspondants dans le continu. Un ouvrage de référence récent en français est [?].

Parmi les algorithmes de la géométrie discrète les plus utiles en morphologie mathématique, on compte en particulier les fonctions distances [?, ?, ?], qui permettent de mettre en œuvre nombre d'opérateurs, tels les érosions et dilatations binaires [?].

La topologie discrète est une discipline cherchant à définir des opérateurs topologiques dans les espaces discrets tels les images, mais également sur des graphes arbitraires, des surfaces ou des complexes [?, ?]. Le lien avec la morphologie mathématique est très fort, en particulier pour les opérateurs d'amincissement [?] et de squelettisation [?]. La ligne de partage des eaux est également un opérateur topologique [?, ?].

1.5.1.3. *Algorithmes inspirés du continu*

Le continu n'est pas représentable exactement dans un ordinateur. Cependant, on peut considérer des objets mathématiques intrinsèquement continus – tels les équations aux dérivées partielles, pour résoudre des problèmes d'analyse d'images. Cette approche permet de produire des algorithmes aux propriétés intéressantes.

En prenant pour point de départ les algorithmes de segmentation, tel celui des contours actifs [?], on trouve des liens avec la squelettisation [?], ainsi que des généralisations de la ligne de partage des eaux incluant des contraintes de courbure [?].

Les algorithmes de marche rapide [?] sont essentiellement équivalents à un algorithme flexible de calcul de la fonction distance géodésique euclidienne [?], en métrique scalaire ou tensorielle [?]. Ils permettent dans certains contextes de proposer une morphologie mathématique du continu [**?**]. Il est a noter que l'algorithme de Sethian n'est précis qu'au premier ordre. Une méthode rapide pour calculer la fonction distance euclidienne géodésique exacte est encore un problème ouvert.

Ces méthodes ont été utilisées en morphologie par exemple en filtrage connectif [**?**], en remplaçant l'opérateur de dilatation par une propagation continue. Des formulations de la LPE dans le continu [**?**, **?**] peuvent être résolues en exploitant les algorithmes de marche rapide.

L'intérêt principal d'une formulation continue est de s'affranchir de la notion de pixel ; dans une certaine mesure, on peut définir une dilatation de rayon arbitraire, et non seulement entier. On peut ainsi également faire de la morphologie sur des variétés arbitraires, par exemple des surfaces triangulées, bien que cette approche ne soit pas la seule.

Nous verrons ci-après d'autres liens avec le continu au travers des algorithmes inspirés par les théories linéaires.

1.5.1.4. *Optimisation discrète et continue pour la segmentation*

Les algorithmes de type « contours actifs » (*snakes*) [**?**], ou « lignes de niveau » (*level-sets*) [**?**, **?**, **?**] sont comparables dans leur approche : il s'agit, dans le domaine de l'analyse d'image, de proposer une formulation d'optimisation continue, basée le plus souvent sur une descente de gradient. Cette formulation peut être exploitée en segmentation, l'objectif étant d'optimiser la position d'un ou de plusieurs contours fermés ou non, en 2D ou de surfaces en 3D, définissant ainsi une région d'intérêt. L'intérêt de ces formulations est la possibilité d'affecter un poids plus ou moins important aux différents aspects d'une segmentation (plus ou moins régulière, intégrant ou non des termes de textures ou de suivi du mouvement, etc), ainsi que la gestion de la topologie. En revanche certaines formulations complexes ne peuvent pas être optimisées de manière globale.

Ces dernières années la communauté de vision et d'analyse d'images ont redécouvert l'intérêt des formulations plus simples mais permettant d'être optimisées globalement, avec les coupures de graphes [**?**], les flots maximaux continus [**?**], ou encore les marches aléatoires [**?**]. En effet, ces formulations sont plus fiables, plus faciles d'emploi et moins sensibles au bruit. Il y a des liens entre ces différentes techniques et la ligne de partage des eaux [**?**, **?**].

1.5.1.5. *Algorithmes d'analyse linéaire*

Par analyse linéaire, on entend tout le domaine lié aux transformées intégrales (Fourier, Radon, ondelettes, etc) qui historiquement descendent du traitement du signal. Dans ce domaine, la structure de base est celle de groupe, avec comme opérateur l'addition. Pour les signaux, et pour certains types d'images ou de problèmes, cette structure est importante. Par exemple, on approxime souvent le bruit d'acquisition par un bruit gaussien additif, qu'une déconvolution permet d'éliminer de manière optimale. En imagerie médicale, comme le cas de la tomographie, la superposition additive des signaux est une hypothèse parfaitement raisonnable.

La morphologie mathématique n'est pas linéaire, la structure de base étant celle de treillis avec le supremum ou l'infimum comme opérateurs. Néanmoins, il existe des liens entre les deux approches, non seulement au niveau des applications, mais aussi de certains outils, comme par exemple l'analyse multi-résolution [**?**], encore appelée *scale-space* [**?**, **?**]. Pour plus d'informations, le lecteur pourra se référer au chapitre 3 du premier volume.

Pour finir, une curiosité : on peut définir la dilatation à partir d'une convolution par

$$\delta_B[I] = (I \star B) > 0, \tag{1.4}$$

où I est une image binaire et B un élément structurant arbitraire. En implémentant la convolution par transformée de FOURIER rapide, cet algorithme permet la dilatation en temps constant par rapport à B. C'est le seul algorithme connu avec cette caractéristique.

1.5.2. *Historique des développements algorithmiques en morphologie mathématique*

Dès le début, le développement de la morphologie mathématique en tant que discipline théorique autant qu'appliquée fut lié à un développement matériel et logiciel : l'analyseur de texture, réalisé à l'école des Mines [**?**]. Par la suite, la plupart des progrès dans cette discipline se sont révélés être le résultat d'une synergie constante entre les applications, la théorie, les algorithmes et les développements matériels.

Serra, dans [**?**], donne un historique des premières années de la morphologie mathématique.

Les développements matériels des premières années, portant sur des architectures dédiées au traitement d'images, étaient une nécessité, du fait des faibles puissances de calcul des ordinateurs d'alors. L'architecture matérielle typique implique un accès séquentiel (lecture vidéo) aux données, et non aléatoire.

1.5.2.1. *Algorithmes parallèles*

Algorithmiquement, cela implique presque nécessairement un traitement *parallèle* des données. Ici le terme désignant des algorithmes qui génèrent un résultat en chaque pixel indépendemment de celui effectué sur les autres pixels, comme l'algorithme de gauche de la figure 1.5 et décrit en section 1.4.1. Ces algorithmes peuvent en effet généralement être distribués facilement entre plusieurs processeurs par exemple, mais ce n'est pas une nécessité. En termes matériels, les algorithmes parallèles s'appliquent bien à une architecture SIMD *Single Instruction Multiple Data* et à la limite au modèle où chaque pixel dispose d'un processeur (rétines artificielles [**?**]). Parmi les développements matériels exploitant des algorithmes parallèles de morphologie mathématique, on compte le Morpho-Pericolor [**?**] et le Cambridge Instrument Quantimet 570 [**?**] ainsi que le processeur dédié (*ASIC*) PIMM1 [**?**]

Les premiers algorithmes de calcul de ligne de partage des eaux, de squelettisation et de filtrage morphologique ont été décrits et mis en œuvre sous forme parallèle, voir par exemple [**?**, **?**].

1.5.2.2. *Algorithmes séquentiels*

Certains algorithmes peuvent être décrits sous forme *séquentielle*, ici désignant un algorithme qui utilise le résultat courant pour dériver le résultat suivant, en adoptant le plus souvent un *ordre de balayage*, comme dans la figure 1.5 de droite, et décrit en section 1.4.2. Les algorithmes séquentiels sont souvent plus rapides que les algorithmes parallèles, du moins sur les ordinateurs standards, car ils exploitent la redondance locale des images. Un algorithme séquentiel typique est celui de la fonction distance [**?**]. Peu d'algorithmes séquentiels de morphologie ont été mis en œuvre au niveau matériel, mais on peut citer l'étude de Lemonier [**?**] qui a, entre autres, proposé un algorithme séquentiel pour la ligne de partage des eaux.

1.5.2.3. *Algorithmes en largeur d'abord*

Plus tard, alors que les ordinateurs montaient en puissance, est apparu comme productif l'idée d'explorer les pixels en partant de la frontière des objets, sans pour autant suivre un ordre imposé par la structure de la mémoire, mais en utilisant une structure de donnée adaptée. Parmi cette famille d'algorithmes, on compte ceux à base de file d'attente [**?**], de lacets [**?**] et de files à priorité [**?**]. Un algorithme typique de ce type d'approches est celui de la ligne de partage des eaux par inondation [**?**, **?**]. Ce type d'algorithme en revanche se prête mal à la mise en œuvre sur matériel dédié de type FPGA. En effet, les structures de données sous-jacentes sont délicates d'utilisation et le goulet d'étranglement n'est pas la vitesse de l'unité centrale, mais celle de l'accès à la mémoire.

1.5.2.4. *Algorithmes inspirés des graphes*

Les algorithmes en largeur d'abord sont classiques dans les problèmes de graphe. L'idée de continuer dans cette direction et d'adapter les algorithmes classiques des graphes est ensuite apparue. Parmi les algorithmes de morphologie mathématique inspirés des graphes, on peut compter l'IFT (*Image Foresting Transform*) [**?**]. Plus fondamentalement, l'idée de considérer une image réellement comme un graphe et, en particulier, celle de valuer également les arêtes, s'est finalement imposée. C'est une idée qui permet par exemple de définir assez naturellement la notion de gradient discret :

la différence numérique entre les deux sommets valués liés par une arête [**?**]. Elle permet également de définir une frontière comme une coupure de graphe et non comme une chaîne de sommets, résolvant ainsi nombre de problèmes topologiques. Cette notion permet de produire de nouveaux algorithmes efficaces et de jeter des ponts avec d'autres disciplines, en particulier au niveau des méthodes de segmentation [**?**].

1.5.2.5. *Algorithmes topologiques*

Outre la notion essentielle de point simple permettant des algorithmes efficaces, préservant l'homotopie des images, de nombreux travaux considèrent de manière essentielle la topologie des images. Parmi les notions essentielles on trouve celle d'arbre des coupes, utilisée en section 1.4.5.1 de ce même chapitre et décrite, de façon détaillée, au chapitre 5 du tome 1. La notion d'arbre des coupes, de par sa représentation efficace des régions et bassins versants d'une image, permet la mise en œuvre de nombreux algorithmes, par exemple la segmentation hiérarchique, les nivellements et encore d'autres filtrages [**?**].

1.5.2.6. *Filtrage*

Les algorithmes de filtrage morphologique forment une classe intéressante en soi. Tout d'abord les références utiles sur le filtrage en morphologie mathématique sont, outre les livres de Serra [**?**, **?**], un article assez complet sur la théorie des filtres morphologiques par Serra et Vincent [**?**] et les articles de Heijmans et Ronse [**?**, **?**]. Un article plus introductif est celui de Heijmans [**?**]. Aucun de ces articles ne traite en détail des aspects algorithmiques, pourtant essentiels. Voici une énumération incomplète mais illustrative de certains problèmes étudiés en morphologie.

1) *Érosions et dilatations rapides*.

De nombreux auteurs ont proposé des méthodes efficaces de calcul des opérateurs de base de la morphologie mathématique. La plupart des bibliothèques logicielles de MM (y compris parmi les plus connues, nous ne citerons pas de noms) calculent un max ou un min sur une fenêtre de manière peu efficace, avec un algorithme avec $O(MN)$ comparaisons, si M est le nombre de pixels dans l'images et N le nombre de pixels dans la fenêtre. Il est souvent possible de décomposer les éléments structurants en parties plus facilement calculables [**?**]. Parmi les éléments structurants les plus courants, on compte les polygones réguliers convexes en 2D. On peut décomposer ceux-ci en opérations séparables par segments de droites. Un travail significatif a consisté à produire des algorithmes de filtrage morphologique par segments en temps constant par rapport à N [**?**, **?**, **?**, **?**]. Le résultat sont des algorithmes capable de calculer les filtres morphologiques avec des ES polygonaux réguliers convexes en temps linéaire (avec $O(N)$ comparaisons), pour tout M. Il est à noter qu'un résultat similaire en 3D ou plus est un problème ouvert à ce jour, sauf pour le cas du parallélépipède dont les faces sont perpendiculaires aux directions de la trame et quelques autres cas particuliers.

Pour les éléments structurants arbitraires en nD, il existe un algorithme en $O(\sqrt[\frac{n}{n-1}]{N}M)$ [**?**]. Un algorithme plus rapide existe en 2-D seulement [**?**]. Plusieurs algorithmes ont été proposés dans le cas binaire [**?**, **?**] de complexité asymptotiquement linéaire.

2) *Ouvertures et amincissements algébriques et arbre des coupes*.

Le filtrage en morphologie mathématique s'appuie plus sur la notion d'ouverture et fermeture que sur celle de dilatation et érosion. Il est courant de définir une notion d'ouverture/fermeture *algébrique* qui n'est pas liée directement à celle d'élément structurant, mais à celle *d'attribut* [**?**] et de connexion [**?**, **?**, **?**]. Cette notion est proche de celle de la reconstruction explicitée dans ce chapitre.

Cette notion permet de définir un grand nombre de filtres très efficaces. Historiquement la première implémentation du filtrage par aire est due à Vincent [**?**, **?**, **?**]. La notion générale a été étendue aux attributs [**?**] non nécessairement croissants, donnant lieu à des opérations appelées *amincissements* algébriques, mais de principe similaire. Une implémentation efficace de ces opérateurs a été proposée dans [**?**], puis une généralisation en partant de l'arbre des coupes par *union-find* dans [**?**]. Récemment, la notion de connectivité a été étendue à la notion d'hyper-connectivité [**?**].

La notion de connectivité par chemins permet également de définir des opérateurs par composition, par exemple en utilisant des familles de droites [**?**] ou de chemins [**?**, **?**] qui permettent le filtrage et la détection d'objets fins, avec de multiples applications comme par exemple en télédétection [**?**].

3) *Filtrage spatialement variable*

Plus récemment, des opérateurs efficaces permettant le filtrage par opérateurs non invariant par translation (ou spatialement variables) ont été proposés[**?**, **?**, **?**]. Ce type de filtrage adapte en chaque pixel la forme de l'élément structurant utilisé en fonction du contenu local de l'image. Cela permet par exemple de tenir compte de la perspective, de la texture [**?**], de l'orientation [**?**], etc. Ces filtres peuvent se révéler efficace dans le cadre du filtrage inverse [**?**, **?**].

4) *Cas nD* Le cadre de la MM s'étend naturellement aux dimensions supérieures [**?**, **?**]. Comme l'écrit Gratin, les difficultés ne sont pas souvent d'ordre théoriques mais pratiques : les images sont plus grandes, la connectivité plus complexe, les problèmes de bord plus nombreux. Un point délicat est l'échantillonnage de directions. En 2-D, de nombreux algorithmes échantillonnent les directions par exemple pour réaliser un filtrage avec des familles de segments avec des orientations isotropes. En 3-D et plus, c'est plus difficile, car un échantillonage isotrope des directions est impossible, résultat connu depuis Platon [**?**]. Du fait de l'amélioration des capteurs, des instruments et des ordinateurs, l'analyse des images en 3D et plus (3D+temps par exemple) est de fait de plus en plus courante dans la pratique. Les grands champs d'applications sont pour le moment l'imagerie médicale, les sciences des matériaux et l'imagerie biologique et bio-moléculaire.

1.6. Conclusions

Dans ce chapitre, nous avons essayé de montrer au lecteur la distance qui existe entre la formulation mathématique d'un opérateur morphologique et sa traduction algorithmique. À partir d'un exemple, nous avons également illustré le fait qu'il n'existe généralement pas un unique algorithme permettant la mise en œuvre d'un opérateur mais plusieurs, dont les caractéristiques peuvent être quelquefois radicalement opposées.

L'algorithmique, qu'elle soit dédiée à la morphologie mathématique ou non, reste un domaine ouvert. Les opérateurs de traitements d'images développés au cours du temps sont de plus en plus exigeants en termes de charge de calculs. À cela s'ajoute la croissance constante du volume des données à traiter. Aussi, on est de plus en plus contraint à attacher une importance particulière à la mise en œuvre algorithmique des opérateurs de traitement d'images, et ce, que cette mise en œuvre soit logicielle ou matérielle.

Les challenges futurs de ce domaine sont de plusieurs natures.

Tout d'abord, les observateurs ont notés la difficulté de la reproductibilité des méthodes et résultats présentés dans la communauté scientifique. Partant d'un article scientifique, le chemin qui mène à l'implémentation est long : à la distance entre les formes mathématique et algorithme s'ajoute également le parfois douloureux passage de l'algorithme à sa mise en œuvre sous forme de programme. En conséquence, la perte de capitalisation d'informations et donc de connaissances est observable. Beaucoup de solutions proposées dans la littérature sont laissées de côté et très peu d'articles présentent des comparaisons entre un nombre significatif de solutions. Un effort devrait être mené afin de doter la communauté du traitement d'images une plate-forme mutualisée d'algorithmes avec leur implémentation.

Un second challenge concerne la mise en œuvre d'algorithmes. Ces derniers sont intrinsèquement abstraits et, dans ce chapitre, nous leur avons conservé cette forme abstraite. En effet, ces algorithmes, tels que décrits, s'appliquent aussi bien sur de classiques images bi-dimensionnelles que sur des signaux (images 1D), volumes (3D) ou des graphes et complexes topologiques. Aucune hypothèse implicite supplémentaire n'est venue réduire le champ d'application de ces algorithmes. Malheureusement, la traduction de la forme algorithmique en programme est souvent accompagnée d'une perte de généralité de l'algorithme. Telle bibliothèque dédiée à la morphologie mathématique ne permettra pas de traiter des images 3D, telle autre de prendre un élément structurant plat de forme quelconque, telle autre d'imaginer doter un espace de couleur d'une relation d'ordre pour accéder à la morphologie, etc. Même une “simple” dilatation, donnée par l'algorithme (3) en figure 1.2, devient, une fois programmée, une dilatation restreinte à un nombre limité de cas. Notons que des solutions existent, permettant d'obtenir la généricité la plus grande possible, sans sacrifier la facilité d'utilisation ni l'efficacité [**?**].

Un troisième challenge est celui de l'effort communautaire. Il est de plus en plus admis que pour être recevable après publication, un algorithme doit être assorti d'une implémentation librement accessible. En effet, tout effort de ré-implémentation est effectivement du temps perdu. L'absence dans la communauté morphologique d'une plate-forme de développement commune est certainement un point qui aura freiné l'adoption de certains algorithmes dans la communauté plus vaste des chercheurs et utilisateurs de l'analyse d'image.

Enfin, un dernier défi réside dans le développement des moyens informatiques. Nous sommes passés à l'heure des processeurs multi-cœurs, des cartes multi-processeurs, et des grappes de calcul. Les outils de traitement d'images, et en particulier les algorithmes, doivent être adaptés à ces nouveaux environnements.