Composants logiciels et algorithmes de minimisation exacte d'énergies dédiés au traitement des images

Jérôme Darbon

Non, dit Mondieur Perle. On dépense plus d'énergie, mais finalement, comme on est parti de plus bas, on arrive au même point ; c'est du gâchis. Évidemment, quand on a vaincu plus d'obstacle on est tenté de croire qu'on a été plus loin. C'est faux. Lutter n'est pas avancer.

passage tiré de L'herbe rouge de Boris Vian.

Table des matières

Préambule

Dans cette thèse nous étudions la minimisation d'énergies markoviennes rencontrées dans les domaines du traitement des images et de la vision par ordinateur. Nous proposons des algorithmes de minimisation exacte pour différents types d'énergies. Ces algorithmes ont l'intérêt de fournir un minimum global quand bien même l'énergie n'est pas convexe. Enfin, nous mettons en évidence quelques liens entre les champs de Markov binaires et la morphologie mathématique. La version finale de ce manuscrit suit les recommandations des rapporteurs.

Partie I : algorithmes de minimisation d'énergie

Dans cette partie nous considérons des énergies souvent utilisées pour modéliser des problèmes de traitement des images. Étant donné une image u à support discret, nous notons u_s la valeur prise par cette image au site s. Nous supposons que la grille discrète est munie d'un système de voisinage et nous considérons les cliques d'ordre deux. Nous étudions les énergies markoviennes qui ont la forme suivante :

$$E(u|v) = \sum_s f_s(u_s - v_s) + \sum_{st} g_{st}(u_s - u_t) \ ,$$

où v est une image observée et u l'image recherchée. La famille de fonctions $\{f_s\}$ est appelée attache aux données et la famille de fonctions $\{g_{st}\}$ est nommée terme de régularisation.

Les objectifs de cette partie sont doubles et dépendent de la forme des attaches aux données et des termes de régularisation. Le premier est de proposer des algorithmes de minimisation exacte des énergies. Le second consiste à tisser quelques liens étroits que les champs de Markov, la minimisation de la variation totale et les filtres issus de la morphologie mathématique entretiennent. Les différents chapitres abordent ces trois objectifs en fonction de la forme des attaches aux données et des termes de régularisation.

Chapitre 1 : Dans le chapitre 1, nous étudions la minimisation de la variation totale (terme de régularisation) avec des attaches aux données modélisées par des fonctions convexes : modèles *Convexe + TV*. Nous proposons une reformulation de ce type d'énergie en utilisant les ensembles de niveau de la variable à optimiser. Nous montrons que cette reformulation correspond à associer un champ de Markov binaire à chaque ensemble de niveau. En utilisant cette formulation, nous proposons un algorithme qui minimise exactement ce type d'énergie. Cet algorithme repose sur l'utilisation de coupures minimales dans un graphe. Les temps de calculs montrent que cet algorithme est rapide. Quelques résultats numériques pour des attaches aux données de type L^1 et L^2 sont également présentés.

Chapitre 2 : Dans le chapitre 2, nous caractérisons la classe d'énergie où l'approche précédente peut être réalisée. Nous appelons cette classe, la classe des *fonctions nivelées*. Cette dernière contient notamment les modèles précédents, à savoir *Convexe + TV*. La classe des fonctions nivelées contient également des énergies qui ne sont*pas convexes*. Nous proposons un algorithme de minimisation exacte pour les fonctions nivelées même si ces dernières ne sont pas convexes. Cet algorithme repose également sur l'utilisation de coupures minimales dans un graphe.

En outre, nous précisons une classe d'énergies nivelées telles que leurs minimisations conduisent à la définition de filtres morphologiques (au sens invariant par changement de contraste). En particulier, nous montrons que le modèle $L^1 + TV$ jouit de cette propriété.

Chapitre 3 : Dans le chapitre 3, nous supposons que les termes de régularisation $\{g_{st}\}$ sont des fonctions convexes. L'étude est séparée en deux parties. Tout d'abord nous traitons le cas où les attaches aux données sont quelconques. Nous proposons un algorithme, à base de coupures minimales, qui minimise de manière exacte ces énergies. Quelques résultats numériques de restauration sont présentés.

Ensuite, nous supposons que les termes d'attaches aux données $\{f_s\}$ sont également des fonctions convexes. Nous proposons un algorithme rapide de minimisation exacte de ces énergies. Cet algorithme repose sur l'utilisation de coupures minimales dans un graphe et sur un procédé de changement d'échelle.

Chapitre 4 : Dans le chapitre 4, deux variations sur le modèle $L^1 + TV$ sont étudiées et quelques liens avec la morphologie mathématique sont exhibés. La première consiste à minimiser l'énergie $L^1 + TV$ en contraignant les formes de l'image à demeurer identiques. Seuls les niveaux de gris des formes peuvent changer. Nous montrons que les résultats peuvent servir de bonne initialisation à des algorithmes de segmentation de plus haut niveau.

La seconde étude décrit une approche pour étendre la morphologie mathématique au cas vectoriel. Cette approche repose sur la minimisation du modèle $L^1 + TV$ sur chaque canal en la contraignant à ne pas créer de nouvelle valeur. Ce problème de minimisation est non convexe. Nous proposons un algorithme qui fournit un minimiseur global de ce problème. Quelques résultats numériques sont présentés pour des images en couleur.

Chapitre 5 : Contrairement au chapitres précédents qui traitent principalement de la minimisation d'énergie, nous proposons dans ce chapitre un algorithme efficace pour mettre en œuvre les filtres connectés. Ces derniers sont des opérateurs morphologiques. Comparé aux algorithmes existants, notre algorithme est la fois le moins gourmand en mémoire vive et le plus rapide pour filtrer des images naturelles.

Partie II : Annexes

Annexe A : Cette annexe présente la liste des publications réalisées durant la thèse.

Annexe B : Dans cette annexe, nous décrivons brièvement les algorithmes de descente de gradient et l'algorithme de projection de Chambolle décrit dans (34), pour minimiser la variation totale.

Annexe C : Cette annexe décrit la construction d'un graphe associé à une énergie Markovienne bianire telle que la coupure minimale de ce graphe donne un étiquetage optimal. Cette approche est originalement proposée par *Kolmogorov et al.* dans (109).

Annexe D : Cette annexe décrit le travail de génie logiciel dédié au traitement des images et partiellement publié dans (49).

Première partie

Algorithmes de minimisation exacte d'une énergie

Motivations

De nombreuses approches en traitement des images et en vision par ordinateur reposent sur la minimisation d'énergies. Il est généralement difficile de trouver un minimum global car ces énergies sont généralement non convexes. La plupart des problèmes en dimension 1 sont optimisables en utilisant la programmation dynamique comme le montrent Amini *et al.* dans (7). Cependant, pour la plupart des problèmes, les énergies associées demeurent difficile à optimiser dans le cas général. Des algorithmes à base de recuit simulé (75) sont alors utilisés malgré la lenteur induite.

A cause de la difficulté à minimiser des énergies non convexes, beaucoup d'auteurs utilisent des modèles convexes. Dans (19), Bouman *et al.* étudient, en particulier, la classe des champs de Markov définis par des gaussiennes généralisées. Rudin *et al.* proposent dans (151) un schéma de restauration d'image reposant sur la minimisation de la variation totale. Dans (16), Blake et Zisserman introduisent l'algorithme *Graduated Non Convexity*. Le principe de cet algorithme est le suivant : on commence tout d'abord à approximer l'énergie par une énergie convexe. Grâce à cette dernière on calcule une bonne solution initiale (qui est indépendante des conditions initiales puisque l'énergie est convexe). On modifie ensuite *graduellement* cette énergie pour qu'elle se rapproche de l'énergie originale à minimiser ; cette modification pouvant faire perdre la propriété de convexité. On minimise cette nouvelle énergie modifiée en utilisant la solution trouvée à l'étape précédente comme initialisation de l'étape courante. Dans certains cas particuliers, on peut prouver la convergence de cette méthode vers un minimum global. Malgré tout il est possible de construire des énergies pour lesquelles le minimum trouvé ne sera qu'un minimum local.

Même quand l'énergie est convexe, des algorithmes rapides et exacts sont recherchés (145).

Dans (81), Greig *et al.* proposent un algorithme qui fournit un minimum exact - au sens de l'estimateur Maximum *a posteriori* (MAP)- pour des champs de Markov dont les variables sont binaires et le modèle est donc celui d'Ising. Cette approche repose sur la construction d'un graphe tel que sa coupure minimale (100) donne l'étiquetage optimal du champ aléatoire. Dans ce travail, Greig *et al.* étudient différentes approches pour calculer le MAP dans le but de restaurer des images binaires. A part la minimisation exacte qui utilise des coupures minimales, les autres méthodes reposent sur l'utilisation du recuit simulé avec des schémas de décroissance de température différents. Les résultats présentés montrent des différences prononcées en fonction des méthodes. L'application d'une méthode *a priori* plutôt "mauvaise" théoriquement (décroissance très rapide de la température) sur le modèle d'Ising suggère que ce modèle est bien adapté à la restauration d'une image binaire. En revanche, la solution exacte montre que ce n'est pas le cas. L'application d'un algorithme "mauvais" sur un modèle "mauvais" peut conduire à un bon résultat. Il faut donc faire la distinction entre le choix du modèle et le choix de l'algorithme pour effectuer la minimisation.

Depuis le travail novateur de Greig *et al.*, l'utilisation des algorithmes de coupures minimales à des énergies plus générales a été abordée par différents auteurs. Boykov *et al.* présentent dans (24) un algorithme efficace, à base de coupures minimales, qui produit un minimum local de bonne qualité, d'une énergie markovienne non convexe. Les termes de régularisation de cette énergie doivent vérifier une condition de semi-métrique pour que l'algorithme soit applicable. Dans (95), Ishikawa propose un algorithme qui minimise de manière exacte une énergie markovienne dont les termes de régularisation sont des fonctions convexes. Dans (109), Kolmogorov *et al.* précisent les champs de Markov binaires qui peuvent

être minimisés en utilisant un algorithme de coupure minimale. D'autres études ont été réalisées pour résoudre de manière exacte des problèmes de vision par ordinateur (21; 96; 150).

Nous proposons dans cette partie des algorithmes de minimisation exacte - à base de coupures minimales - pour différents types d'énergie markovienne. En outre, nous exhibons quelques classes d'énergies qui peuvent être échantillonnées de manière exacte (147).

Chapitre 1

Minimisation exacte de la variation totale

La minimisation de la variation totale (*TV* pour "Total Variation" en anglais) a été initialement introduite par Rudin, Osher et Fatemi en 1992 (151). Par la suite, elle a été très souvent utilisée à des fins de restauration et de déconvolution d'images par (3; 4; 5; 141; 155; 171). La minimisation de TV est actuellement utilisée pour la décomposition d'image en une partie oscillante et une partie géométrique par Aujol *et al.* dans (9) et Vese *et al.* dans (180). Le principal intérêt de la minimisation de la variation totale réside dans le fait que les minimiseurs appartiennent à l'espace des fonctions à variation bornée comme énoncé dans le livre d'Evans et Gariepi (73). Cet espace est d'une grande utilité dans le cadre du traitement des images car il autorise la présence de discontinuités, autrement dit, de contours. Ainsi les images obtenues par cette minimisation présentent des bords francs. Nous décrivons un algorithme de minimisation *exacte* de la variation totale dans le cas où les contraintes sont convexes. Cette classe d'énergies est souvent utilisée pour la restauration d'images. Nous avons également choisi de commencer l'étude par cette classe d'énergies car elle est plus facile à résoudre. Le chapitre suivant lèvera cette hypothèse de convexité de l'attache aux données. La recherche de solution exacte est essentielle afin de pouvoir étudier plus précisément les modèles à base de minimisation de la variation totale. A notre connaissance, ces résultats sont nouveaux. Récemment dans (35), Chambolle a obtenu de manière indépendante dans essentiellement les mêmes résultats (y compris la proposition 1.3) en utilisant des approches reposant sur l'utilisation de fonctions sous-modulaires (132; 156) (alors que nous utilisons des preuves qui appartiennent principalement au domaine du stochastique). Après la réalisation de ces travaux, nous avons eu connaissance de deux travaux précurseurs portant sur la minimisation de la variation totale. Le premier est celui de Hochbaum, présenté dans (90), qui propose une approche de la minimisation de la variation totale par des techniques similaires à celles des flots maximaux *paramétriques*. Cependant aucun résultat numérique n'est présenté. Dans (194), Zalesky propose une minimisation de la variation totale par l'utilisation de coupure minimale. En revanche, il ne propose pas notre approche rapide à base de dichotomie. Dans chacun de ces travaux, une proposition similaire à notre proposition 1.3 est présentée mais les preuves sont différentes.

La partie 1.1 présente le problème de minimisation de la variation totale. L'idée de notre algorithme de minimisation de la variation totale repose sur la représentation d'une image

par ses ensembles de niveaux. Dans la section 1.2, nous reformulons le problème en termes de champs de Markov binaires associé à chacun des ensembles de niveaux d'une image. Cette approche de reformulation d'énergie a initialement été introduite par Strang dans (167; 168). Le problème original se ramène donc à une série d'optimisations (au sens de l'estimateur Maximum *a posteriori*) de champ de Markov binaires. Nous montrons dans la partie 1.3 que ces optimisations peuvent s'effectuer de manière indépendante sur chacun des ensembles de niveaux et que l'on peut reconstruire la solution du problème original à partir de ces solutions binaires. Une fois le cadre théorique posé, nous proposons un algorithme de minimisation exacte dans la section 1.4. Notre algorithme repose sur l'utilisation de coupures minimales dans des graphes. Cette approche a initialement été proposée par Greig *et al* dans (81) dans le but de trouver des solutions globales pour des modèles d'Ising (190). La partie 1.5 est consacrée à l'étude expérimentale de notre algorithme et à la présentation des résultats. Nous comparons les résultats donnés par notre algorithme à ceux obtenus par une approche à base de descente de gradient et ceux obtenus par un algorithme de projection récemment donné par Chambolle (34). Nous exhibons les différences entre les minimiseurs donnés par ces deux algorithmes et le nôtre. Nous étudions la complexité temporelle de notre algorithme. Cette dernière est polynomiale. En pratique elle est linéraire avec le nombre de pixels. Nous mettons en évidence par l'expérience que notre algorithme est rapide tout en fournissant une solution exacte. Nous présentons enfin quelques résultats numériques pour des attaches aux données de type L^2 et L^1.

Une partie de notre travail sur la minimisation variation totale avec des attaches aux données convexes a été publiée dans les actes de la conférence *Combinatorial Image Analysis (IWCIA'2004)* (53) et dans les actes de la conférence *Iberian Conference on Pattern Recognition and Image Analysis (IbPRIA'2005)* (55). Le travail présenté dans ce chapitre est accpeté à la revue *Journal of Mathematical Imaging and Vision* (57).

1.1 Présentation du problème

Nous supposons que l'image u est définie sur un rectangle Ω de $\mathbb{R}^2$. La variation totale $TV(u)$ de u peut s'écrire :

$$TV(u) = \int_\Omega |\nabla u| \ ,$$

où le gradient de u est pris au sens des distributions. Les premières méthodes classiques de minimisation de la variation totale appartiennent au cadre des méthodes variationnelles. Elles reposent sur une descente de gradient. L'équation d'évolution est alors la suivante :

$$\frac{\partial u}{\partial t} = \mathrm{div}\left(\frac{\nabla u}{|\nabla u|}\right) \ .$$

Ce dernier terme correspond à la courbure de u. Afin d'éviter les divisions par zéro, le terme ∇u est généralement remplacé par $\sqrt{|\nabla u|^2 + \epsilon}$, avec $\epsilon > 0$ et petit. Ce procédé a été proposé dans (1; 187). Ce schéma fournit un algorithme qui converge vers la solution du problème quand ϵ tend vers 0, mais la solution exacte demeure inaccessible en pratique.

Une autre formulation de la minimisation de la variation totale utilise les sous-gradients et des techniques de dualités. Ce type d'approche est décrit par Chan *et al.* dans (39), Carter

dans (29) et Chambolle dans (34). L'approche par sous-gradients évite les problèmes numériques de calcul du gradient en considérant l'équation d'Euler-Lagrange. Comme pour les algorithmes à base de descente de gradient, l'algorithme tend vers la solution exacte sans l'atteindre.

Dans (169) les auteurs donnent les solutions exactes et analytiques de la minimisation de la variation totale quand l'attache aux données est L^2 pour le cas des fonctions radiales symétriques.

En dimension 1, un minimiseur peut être calculé en utilisant des algorithmes de programmation dynamique (7), à condition que l'espace des étiquettes (i.e, niveaux de gris) soient discrets. La complexité d'une telle méthode est $\Theta(L^2|\Omega|)$, où L et $|\Omega|$ sont respectivement le cardinal de l'espace des étiquettes et le nombre de points du signal. Dans (145), Pollak *et al.* décrivent un algorithme rapide pour calculer une solution optimale exacte de la minimisation de la variation totale sujette à une contrainte de type L^2 en dimension 1. La complexité de leur algorithme est $\Theta(|\Omega| \ln |\Omega|)$. En revanche, l'algorithme ne fournit qu'une approximation en dimension supérieure ou égale à deux.

Supposons que u et v sont deux images définies sur Ω, et où v correspond à l'image observée. Nous nous intéressons à minimiser la fonctionnelle suivante :

$$E(u|v) = \int_\Omega f\left(u(x),\ v(x)\right) dx + \beta \int_\Omega |\nabla u| \,. \tag{1.1}$$

Dans ce chapitre, nous supposons que le terme d'attache aux données est une fonction convexe de $u(\cdot)$, comme par exemple :

$$f\left(u(x),\ v(x)\right) = |u(x) - v(x)|^p \quad ,$$

pour le cas L^p(avec $p = 1, 2$). Nous supposons également que le terme de pondération β est positif. Dans la suite du manuscrit, nous noterons $L^2 + TV$ le modèle de la variation totale avec une attache aux données de type L^2 ; nous noterons $L^1 + T1$ quand l'attache aux données L^2 est remplacée par la norme L^1. Dans la section suivante, nous reformulons l'équation (1.1) en fonction des ensembles de niveau de u.

1.2 Reformulation en terme d'ensemble de niveaux et de champs de Markov

Pour le reste du chapitre, nous supposons que l'image u est définie sur une grille discrète S et que chaque pixel prend ses valeurs dans l'ensemble discret dans $[\![0, L-1]\!]$. Nous notons u_s la valeur du pixel de l'image u au site $s \in S$. Nous allons décomposer l'image en ses ensembles de niveaux en utilisant le principe de décomposition présenté dans (83), ce qui revient à considérer les versions seuillées u^λ de u :

$$u_s^\lambda = \mathbb{1}_{u_s \leq \lambda} \quad ,$$

avec $\lambda \in [\![0, L-1]\!]$. La reconstruction de u à partir de ses ensembles de niveaux s'obtient alors facilement :

$$u_s = \min\{\lambda, u_s^\lambda = 1\} \quad .$$

Une telle décomposition demeure valide dans le cas où l'espace des labels est continu (82) (il faut alors remplacer les minima dans la formule précédente par des infima). Elle est aussi valide dans le cas où le support Ω de l'image lui aussi est continu (37).

1.2.1 Reformulation en ensembles de niveaux

La formule de la co-aire établit que pour toute fonction u qui appartient à l'espace des fonctions à variation bornée (73), nous avons

$$TV(u) = \int_{\mathbb{R}} P(u^\lambda)\, d\lambda \ , \tag{1.2}$$

presque sûrement pour tout λ et où $P(u^\lambda)$ est le périmètre de u^λ. Le périmètre $P(E)$ d'un sous-ensemble mesurable $E \subset \Omega$ est défini par $P(E) = TV(\mathbb{1}_E)$, où $\mathbb{1}_E$ est la fonction caractéristique de E. Dans le cas discret, nous pouvons écrire

$$TV(u) = \sum_{\lambda=0}^{L-2} P(u^\lambda) \ .$$

Nous remarquons que $u_s^{L-1} = 1$ pour tout $s \in S$ (donc le périmètre $P(u^{L-1}) = 0$), ce qui justifie la sommation précédente de 0 à $L-2$. L'estimation du périmètre peut se faire de manière locale ; cette approche est justifiée dans (22) par Boykov *et al.*. Dans notre cas, nous considérons un voisinage 8-connexes. Les cliques d'ordre deux associées aux voisinages sont notées (s,t). Notre estimation locale du périmètre utilise seulement ces cliques d'ordre deux. Nous obtenons donc

$$TV(u) = \sum_{\lambda=0}^{L-2} \sum_{(s,t)} w_{st}\, |u_s^\lambda - u_t^\lambda| \ . \tag{1.3}$$

Ici, les w_{st} sont fixés à 0.26 pour les quatre voisins de la 4-connexités, et 0.19 pour les voisins diagonaux de la huit connexité. Cette approximation est justifiée dans (136) par des considérations théoriques validées par l'expérience.

Nous introduisons maintenant notre proposition qui reformule l'énergie de (1.1) en terme des ensembles de niveaux de l'image u.

Proposition 1.1 *La reformulation de l'énergie $E(u|v)$ décrite par l'équation (1.1) se réécrit*

$$E(u|v) = \sum_{\lambda=0}^{L-2} E^\lambda(u^\lambda|v) + C \ , \ \textit{avec} \tag{1.4}$$

$$E^\lambda(u^\lambda|v) = \beta \left[\sum_{(s,t)} w_{st}\, ((1-2u_t^\lambda)\, u_s^\lambda + u_t^\lambda) \right] + \sum_{s\in\Omega} (f(\lambda+1, v_s) - f(\lambda, v_s))(1-u_s^\lambda) \ , \tag{1.5}$$

$$\textit{avec} \quad f_s(x) = f(x,\, v_s) \quad \forall s \in S \ \textit{ et } \ C = \sum_{s\in\Omega} f_s(0) \ .$$

Preuve : Remarquons tout d'abord que l'égalité suivante est valide pour des variables binaires a, b :

$$|a-b| = a + b - 2ab \ .$$

Nous repartons de l'égalité (1.3) obtenue par la formule de la co-aire, et en utilisant la formule précédente nous obtenons :

$$TV(u) = \sum_{\lambda=0}^{L-2} \sum_{(s,t)} w_{st} \left((1-2u_t^\lambda)\, u_s^\lambda + u_t^\lambda \right) \ .$$

On a de plus, pour toute fonction g, la décomposition suivante :

$$\forall k \in [0, L-1] \;\; g(k) = \sum_{\lambda=0}^{k-1} (g(\lambda+1) - g(\lambda)) + g(0) = \sum_{\lambda=0}^{L-2} (g(\lambda+1) - g(\lambda)) \mathbb{1}_{\lambda<k} + g(0) \;\; .$$

Remarquons que cette égalité est cohérente pour $k = 0$ et $k = L-1$. Nous définissons alors f_s ainsi

$$f_s(u_s) = f(u_s, v_s) \;\; .$$

Puisque $\mathbb{1}_{\lambda<u_s} = 1 - u_s^\lambda$, nous obtenons donc

$$f(u_s, v_s) = f_s(u_s) = \sum_{\lambda=0}^{L-2} (f_s(\lambda+1) - f_s(\lambda))(1 - u_s^\lambda) + f_s(0) \;\; ,$$

ce qui conclut la preuve. □

Nous avons donc montré que l'énergie (1.1) se décompose sur les ensembles de niveaux de u à une constante près. Nous allons maintenant étudier plus précisément chacun des termes associés à un niveau donné.

1.2.2 Optimisations indépendantes de champs de Markov

Remarquons que chaque terme E^λ est un champ de Markov binaire où l'*a priori* est un modèle d'Ising (190). Seules les interactions binaires sur les composantes de même niveau λ sont considérées. Puisque $\beta \geq 0$, le modèle d'Ising considéré est celui du ferromagnétisme.

Supposons que l'optimisation de chacun des termes $E^\lambda(\cdot|v)$ se fasse de manière indépendante. Nous obtenons donc une famille d'images binaires $\{\hat{u}^\lambda\}_{\lambda=0\ldots L-2}$, où chaque u^λ est un minimiseur de $E^\lambda(\cdot|v)$. Avec cette approche la somme $\sum_{\lambda=0}^{L-2} E^\lambda(\cdot|v)$ sera minimisée et nous obtenons un minimiseur de $E(\cdot|v)$ à condition que la famille $\{\hat{u}^\lambda\}_{\lambda=0\ldots L-2}$ soit monotone, ce qui signifie que nous avons l'inégalité suivante :

$$\hat{u}^\lambda \leq \hat{u}^\mu \;\forall \lambda < \mu \quad . \tag{1.6}$$

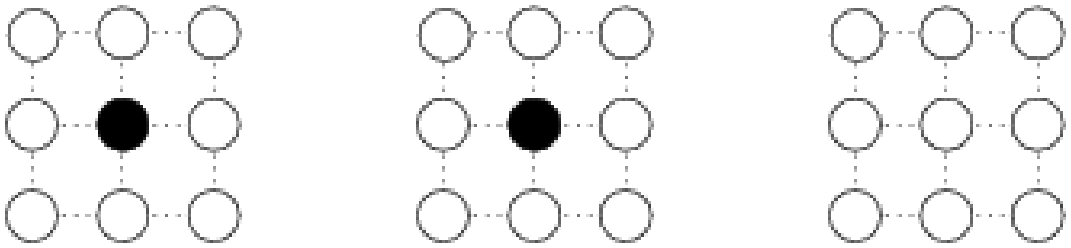
Sous cette condition, la solution optimale est donnée par (83) :

$$\forall s \;\; \hat{u}_s = \min\{\lambda, \hat{u}_s^\lambda = 1\} \;\; .$$

Si la propriété de monotonie précédente (1.6) n'est pas valide, alors la famille $\{u^\lambda\}_{\lambda=0\ldots L-2}$ ne définit pas une fonction et nous ne pouvons donc pas définir une image qui serait une solution du problème (1.1). Le principal intérêt de cette approche est que nous remplaçons un problème de minimisation où la variable est une image à niveau de gris par une série d'optimisations où la variable de chaque problème est binaire. Dans la partie suivante, nous prouvons qu'un tel procédé de minimisation fournit bien un minimiseur global.

1.3 Reconstruction de la solution

Cette partie est consacrée à la preuve de la propriété de monotonie (1.6). Remarquons qu'il est possible que l'énergie (1.1) ne soit pas strictement convexe. C'est par exemple le cas



(a) (b) (c)

FIG. 1.1 – Considérons un cas où l'attache aux données est la norme L^1, ce qui correspond au cas où $E(\cdot)$ n'est pas strictement convexe. Cela signifie que l'unicité des minimiseurs n'est pas garantie. L'image originale est représentée en (a) où la 4-connexité est considérée. Les cercles blancs et noirs représentent les sites dont les valeurs sont respectivement 0 et 1. Si $\beta = 0.25$ alors il existe deux minimiseurs représentés en (b) et (c). L'énergie associée à chacune de ces configurations est 1.

si l'attache aux données est la norme L^1. Dans ce cas, l'unicité des minimiseurs n'est plus garantie dans le cas général. Une telle situation, pour un modèle de type $L^1 + TV$, est décrite par la figure 1.1 où la propriété de monotonie peut être brisée dans certains cas. Cependant, nous montrons qu'il existe au moins un ensemble de solutions binaires $\{\hat{u}^\lambda\}$ qui vérifie la propriété (1.6), et qui est donc une solution du problème.

1.3.1 Un résultat sur les chaînes de Markov couplées

Puisque l'énergie *a posteriori* du champ de Markov est décomposable sur ses ensembles de niveaux, il est pratique de définir les "configurations locales de voisinages" à la fois pour l'image u en niveau de gris et pour les images binaires u^λ :

$$N_s = \{u_t\}_{t\sim s} \ ,$$

et

$$N_s^\lambda = \{u_t^\lambda\}_{t\sim s} \ , \forall \lambda \in [0, L-2] \ .$$

Nous proposons le lemme suivant :

Lemme 1.1 *Si l'énergie conditionnelle* a posteriori *associée à chaque site s s'écrit sous la forme suivante :*

$$E(u_s \mid N_s,\ v_s) = \sum_{\lambda=0}^{L-2} \Big(\Delta\phi_s(\lambda)\, u_s^\lambda \ + \ \chi_s(\lambda) \Big) \tag{1.7}$$

où $\Delta\phi_s(\lambda)$ est une fonction décroissante en λ et où $\chi_s(\lambda)$ ne dépend pas de u_s^λ, alors il existe un algorithme stochastique "couplé" qui minimise chaque énergie a posteriori *$E^\lambda(u^\lambda)$, tout en préservant la condition de monotonie : $\forall s \ \ \forall \lambda \le \mu \ \ u_s^\lambda \le u_s^\mu$.*

En d'autres termes, étant donné $u^\star$, solution binaire du problème E^k, il existe au moins une solution au problème E^l, notée $\hat{u}$, telle que

$$u^\star \le \hat{u} \quad \forall k \le l \ .$$

La preuve de ce lemme repose sur l'utilisation de couplage de chaînes de Markov (63; 147). Cette propriété de couplage a été initialement proposé par Propp et Wilson dans (147), qui ont également exhibé un échantillonneur exact pour le modèle d'Ising (cas du ferromagnétisme).

Preuve : Nous munissons l'espace des images binaires avec la relation d'ordre partielle suivante :

$$u \leq v \text{ si et seulement si } u_s \leq v_s \ \forall s \in S \ .$$

A partir de la décomposition (1.7), l'énergie *a posteriori* conditionnelle locale au niveau λ est

$$E(u_s^\lambda \mid N_s^\lambda, v_s) = \Delta\phi_s(\lambda)\, u_s^\lambda + \chi_s(\lambda) \ .$$

Nous introduisons la probabilité conditionnelle locale *a posteriori* sous forme de distribution de Gibbs, i.e :

$$P_s(\lambda) = P(u_s^\lambda = 1 \mid N_s^\lambda, v_s) = \frac{\exp -\Delta\phi_s(\lambda)}{1 + \exp -\Delta\phi_s(\lambda)} = \frac{1}{1 + \exp \Delta\phi_s(\lambda)} \ . \tag{1.8}$$

En utilisant les conditions du lemme 1.1, cette probabilité est clairement une fonction décroissante en λ.

Nous définissons maintenant $(L-1)$ échantillonneurs de Gibbs couplés pour toutes les $(L-1)$ images binaires de la manière suivante : tout d'abord nous supposons que nous disposons d'un ordre de parcours des sites. Quand un site s est visité, nous tirons uniformément $\rho_s \in [0, 1]$. Puis nous effectuons les affectations suivantes pour chaque valeur de λ :

$$u_s^\lambda = \begin{cases} 1 \text{ si } 0 \leq \rho_s \leq P_s(\lambda) \\ 0 \text{ sinon} \end{cases}$$

Remarquons que cela correspond à tirer une valeur binaire aléatoirement selon sa distribution de probabilité à la différence que le même nombre aléatoire ρ_s est utilisé pour les $(L-1)$ images binaires. Ce procédé permet de maintenir la propriété de monotonie. En effet, d'après la décroissance de (1.8) nous voyons que l'ensemble des valeurs binaires affectées au site s vérifie

$$u_s^\lambda = 1 \Rightarrow u_s^\mu = 1 \ \ \forall \mu > \lambda \ .$$

La propriété de monotonie (équation (1.6))

$$u^\lambda \leq u^\mu \quad \forall\ \lambda < \mu \ ,$$

est donc préservée après chaque échantillonnage.

Cette propriété s'étend de manière immédiate à une série de $(L-1)$ échantillonneurs de Gibbs "couplés" possédant la **même** température T quand ils visitent le site s. Il suffit de remplacer $\Delta\phi_s(\lambda)$ par $\Delta\phi_s(\lambda)$ / T dans (1.8). Ainsi, cette propriété est vérifiée pour une série de $(L-1)$ échantillonneurs de Gibbs couplés qui sont plongés dans un algorithme de recuit simulé (75) où la *même* température T décroît (de manière logarithmique) et tend vers 0. La décroissance s'effectue soit après la visite d'un site s ou au début de chaque parcours de S (190). Ceci conclut la preuve. □

Il est important de remarquer que ce résultat est une condition *suffisante* pour effectuer une minimisation simultanée niveau par niveau des énergie *a posteriori*, tout en préservant la propriété de monotonie (1.6).

Cas L^2 et L^1

Nous montrons maintenant que l'énergie *a posteriori* dans les cas d'une attache aux données L^1 ou L^2 vérifie les conditions du lemme 1.1.

Proposition 1.2 *Le lemme 1.1 s'applique pour les énergies* a posteriori *où les attaches aux données sont L^1 ou L^2.*

Preuve : L'équation (1.5) se réécrit ainsi :

$$E(u_s \mid N_s,\ v_s) \quad = \quad \sum_{\lambda=0}^{L-2} (\ \Delta\phi_s(\lambda)\ u_s^\lambda \ +\ \chi_s(\lambda)\ ) + \sum_{s\in\Omega} f(0, v_s)$$

$$\text{où} \begin{cases} \Delta\phi_s(\lambda) & = & \beta \left[\sum_{t\sim s} w_{st}\ (1 - 2u_t^\lambda) \right] - (f(\lambda+1, v_s) - f(\lambda, v_s)) \\ \chi_s(\lambda) & = & \beta \left[\sum_{t\sim s} w_{st}\ u_t^\lambda \right] + f(\lambda+1, v_s) - f(\lambda, v_s) \end{cases}$$

La contribution du terme issu de la variation totale s'élève à $\Delta\phi_s(\lambda)$, $\sum_{s\sim t} w_{st}\ (1 - 2u_t^\lambda)$, est clairement une fonction décroissante de λ.

• Pour un terme d'attache aux données L^1 nous avons le résultat suivant :

$$f(u_s, v_s) = |u_s - v_s| \quad = \quad \sum_{\lambda=0}^{L} |u_s^\lambda - v_s^\lambda| = \sum_{\lambda=0}^{L} u_s^\lambda (1 - 2v_s^\lambda) + v_s^\lambda \ .$$

La contribution à $\Delta\phi_s(\lambda)$ de $s_s(u_s)$ est donc $(1 - 2v_s^\lambda)$. Ce dernier terme partage les mêmes propriétés de décroissance.

• Considérons maintenant le terme issue d'une attache aux données L^2 :

$$-(f(\lambda+1, v_s) - f(\lambda, v_s)) = -((\lambda + 1 - v_s)^2 - (\lambda - v_s)^2) = -\ (2\ (\lambda - v_s) + 1)$$

qui est également une fonction décroissante de λ. □

En conclusion les énergies *a posteriori* pour une attache aux données L^1 ou L^2 peuvent être minimisée de manière "indépendante" sur les niveaux. Nous donnons maintenant des hypothèses plus simples à vérifier pour le lemme 1.1.

1.3.2 Condition de convexité pour la reconstruction

Nous proposons maintenant des conditions équivalentes à celles requises pour le lemme 1.1. Elles mettent en jeu la convexité des énergies conditionnelles.

Proposition 1.3 *Les conditions requises par le lemme 1.1 sont équivalentes aux conditions suivantes : Pour toutes configurations des voisins u_t et pour toutes données observées v_s , toutes les énergies conditionnelles $E(u_s \mid N_s,\ v_s)$ sont des fonctions* convexes *des niveaux de gris $u_s \in [\![0, L-1]\!]$.*

Preuve : Puisque l'équation (1.4) montre que l'énergie totale se décompose sur ses ensembles de niveaux, les énergies conditionnelles locales se décomposent également ainsi :

$$E(u_s \mid N_s,\ v_s) = \sum_{\lambda=0}^{L-2} E^\lambda(u_s^\lambda \mid N_s^\lambda,\ v_s)\ .$$

En outre, puisque l'énergie conditionnelle locale *a posteriori* au niveau λ est une fonction de la variable binaire u_s^λ, elle satisfait

$$E^\lambda(u_s^\lambda \mid N_s^\lambda,\ v_s) - E^\lambda(u_s^\lambda = 0 \mid N_s^\lambda,\ v_s) = \left(E^\lambda(u_s^\lambda = 1 \mid N_s^\lambda,\ v_s) - E^\lambda(u_s^\lambda = 0 \mid N_s^\lambda,\ v_s)\right) u_s^\lambda$$

ce qui signifie après identification avec (1.7) :

$$\Delta\phi_s(\lambda) = E^\lambda(u_s^\lambda = 1 \mid N_s^\lambda,\ v_s) - E^\lambda(u_s^\lambda = 0 \mid N_s^\lambda,\ v_s)\ . \tag{1.9}$$

Durant la transition $\lambda \rightarrow \lambda + 1$, seule la variable suivante change :

$$u_s^\lambda = 1 \rightarrow u_s^\lambda = 0\ .$$

En considérant la décomposition des énergies conditionnelles sur les niveaux, cela signifie que seul le terme $E^\lambda(u_s^\lambda \mid N_s^\lambda,\ v_s)$ change. Nous avons donc :

$$E(\lambda+1 \mid N_s,\ v_s) - E(\lambda \mid N_s,\ v_s) = E^\lambda(u_s^\lambda = 0 \mid N_s^\lambda,\ v_s) - E^\lambda(u_s^\lambda = 1 \mid N_s^\lambda,\ v_s) = -\Delta\phi_s(\lambda)$$

La condition de décroissance sur $\Delta\phi_s(\cdot)$ est donc équivalente à ce que la fonction suivante (de variable λ)

$$E(\lambda+1 \mid N_s,\ v_s) - E(\lambda \mid N_s,\ v_s)$$

soit une fonction croissante sur $[0, L-2]$. □

Clairement, les énergies conditionnelles locales liées aux modèles $L^1 + TV$ et $L^2 + TV$ jouissent de la propriété de convexité (car c'est une somme de fonctions convexes) ; les conditions d'applications du lemme 1.1 sont donc remplies.

Bien que nous ayons prouvé que la propriété de monotonie (1.6) est valide, cela ne nous fournit pas un algorithme de calcul de la solution optimale. L'utilisation du recuit simulé (10) permet de conclure sur la convergence du processus mais nécessite un nombre infini d'itérations. En pratique, on fixe un critère d'arrêt qui viole la propriété de convergence. La partie suivante présente notre algorithme de minimisation exacte.

1.4 Algorithmes de minimisation

Dans cette partie, nous proposons deux algorithmes de minimisation exacte qui reposent sur les résultats de la partie précédente. Ces deux algorithmes utilisent les méthodes d'optimisations exactes de champs de Markov dont les variables sont binaires. De telles optimisations sont possibles grâce à la transformation du problème initial en un problème de coupure minimale dans un graphe (81). Notre mise en œuvre utilise la construction de graphe décrite par Kolmogorov *et al.* dans (109) et l'algorithme de coupure est celui proposé par Boykov *et al.* dans (23).

1.4.1 Un algorithme séquentiel

D'après la décomposition de l'énergie sur les ensembles de niveaux de u et de la propriété de monotonie énoncée dans le lemme 1.1, un algorithme découlant directement de ces considérations consiste à minimiser l'énergie niveau par niveau. Chacune de ces optimisations correspond à calculer le maximum *a posteriori* d'un champ de Markov binaire. Cette approche conduit donc à effectuer $(L-1)$ optimisations binaires. Si les optimisations sont effectuées de manière indépendante alors la condition de monotonie (équation (1.6)) peut être violée. Afin d'assurer la cohérence des solutions - c'est à dire d'assurer la monotonie des solutions binaires - nous effectuons les optimisations à partir du niveau de gris le plus faible (i.e, 0) au niveau de gris le plus élevé, i.e $(L-1)$. Nous calculons une solution u^λ pour le niveau λ. Soit $P_s = \{s \in S | u_s^\lambda = 1\}$. Pour le niveau $(\lambda+1)$, nous contraignons la solution $u^{\lambda+1}$ afin que la propriété de monotonie (1.6) soit valide. La monotonie est vérifiée si

$$u_s^{\lambda+1} = 1 \; \forall s \in P_s \quad .$$

En effet si $u_s \leq \lambda$ alors $u_s \leq (\lambda+1)$. Du point de vue de la mise en œuvre cela signifie que nous construisons le graphe tel que sa coupure minimale attribue toujours la valeur 1 à $u_s^{\lambda+1}$, ce qui correspond à $u_s^{\lambda+1} = 1$.

1.4.2 Un algorithme de type "diviser pour régner"

Nous présentons maintenant un algorithme qui tire profit des propriétés d'inclusion des solutions binaires.

Diviser pour régner

Supposons que nous avons effectué l'optimisation au niveau λ, et notons $\hat{u}^\lambda$ une solution optimale. Chaque pixel $\hat{u}_s^\lambda$ est affecté à une valeur booléenne qui précise si la valeur optimale au site s est inférieure ou bien strictement supérieure à λ.

Remarquons que les termes présents dans l'équation (1.4)

$$E(u|v) = \sum_{\lambda=0}^{L-2} E^\lambda(u^\lambda|v) + C$$

impliquent seulement les versions seuillées u^λ de u (la constante C est oubliée puisque nous sommes intéressés par la minimisation de cette énergie). La valeur précise de u n'est pas requise. Il est donc inutile de prendre en compte les pixels au dessus de λ pour les optimisations qui ne concernent que les régions de pixels qui sont déjà inférieurs ou égaux à λ. Évidemment, la même considération est valide pour les pixels qui sont en dessous de λ. Nous considérons donc toutes les composantes connexes (qui définissent une partition de l'image) du minimiseur, et nous les optimisons indépendamment les unes des autres.

Ces propriétés nous permettent de proposer un algorithme de type "diviser pour régner" (44). Une telle approche se présente ainsi :

- Tout d'abord, *décomposer* le problème en des sous-problèmes.
- Ensuite, *résoudre* indépendamment chacun de ses *sous-problèmes*.
- Enfin, *recombiner* les solutions des sous-problèmes afin d'obtenir la solution globale.

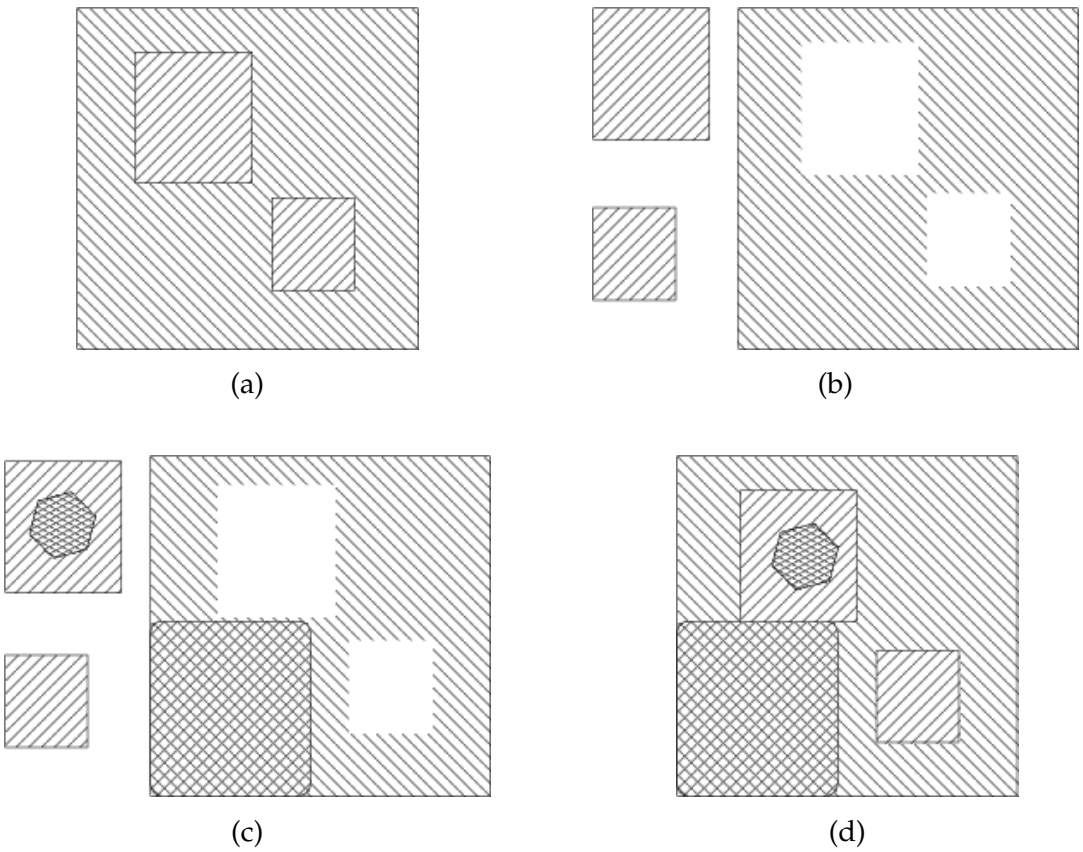


FIG. 1.2 – Illustration de notre algorithme par une approche "diviser pour régner". La partition de l'image après la minimisation au niveau λ est montrée en (a). Les composantes connexes de l'image (a) sont présentées en (b) : cela correspond à la décomposition du problème en sous-problèmes. Chaque sous-problème est résolu de manière indépendante et le résultat est présenté en (c). Enfin, les solutions des sous-problèmes sont recombinées pour donner la solution (d).

D'après ce qui précède, la décomposition en sous-problèmes est effectuée en considérant les composantes connexes du minimiseur du niveau λ. La combinaison des solutions des sous-problèmes est immédiate puisque la décomposition de l'image en composantes connexes est une partition de l'image. Ce processus est décrit sur la figure 1.2.

Lors de la résolution des sous-problèmes, les conditions aux bords doivent faire l'objet d'une attention particulière. Si à une étape les pixels d'une composante sont inférieurs à λ alors les pixels voisins de cette composante connexe sont nécessairement supérieurs à λ. En effet, si ce n'était pas le cas, ces pixels feraient partie de la composante connexe. Un raisonnement identique est mené pour les composantes connexes dont les pixels sont supérieurs à λ.

Choix du seuil

Une bonne stratégie pour le choix du niveau λ auquel l'optimisation doit être effectuée durant le processus "diviser pour régner", est d'utiliser une stratégie de dichotomie. Par

exemple, supposons que le minimiseur soit une image constante, alors il faut effectuer exactement $\log_2(L)$ optimisations binaires pour calculer la solution. C'est un gain considérable comparé aux L optimisations requises pour la méthode séquentielle.

La complexité d'un algorithme de type diviser pour régner est d'autant plus faible que la taille des sous-problèmes est identique. D'autres choix de stratégies pour le choix du seuil telle que la moyenne ou la médiane sur chaque composante connexe reste à tester. Nous laissons ce travail pour le futur.

1.4.3 Complexité en temps

Le choix du coefficient de régularisation β dépend de la quantité de bruit présente dans l'image. D'une manière qualitative plus β est élevé, plus le lissage est fort. Dans un but de clarté nous présentons les résultats en fonction de la taille des images. En effet, la force du filtrage, réglée par le coefficient β, dépend également de la taille de l'image. Ainsi, lorsque l'on divise par 4 la taille de l'image (deux fois moins de lignes et colonnes) le nombre d'interactions binaires est divisées seulement par 2.

Nous présentons les résultats sur des images de taille 512×512 et 256×256. L'image classique *Lena*, l'image de *Barbara* et une image de vue aérienne de Montpellier sont utilisées pour les tests 512×512. Ces images sont présentées sur les figure 1.3-(a), figure 1.3-(b), et figure 1.3-(c). Ces mêmes images réduites par 2 dans chaque dimension sont utilisées pour les tests 256×256 (les valeurs des pixels sont obtenues en faisant par une moyenne sur une fenetre $2x2$ sur l'image originale et en ne considérant qu'un pixel sur deux). En plus, nous testons nos algorithmes sur l'image *girl*, de taille 256×256, et sur une de ses versions bruitées par un bruit gaussien additif de moyenne nulle et d'écart-type 20 (i.e, $\mu = 0$ et $\sigma = 20$). Les tests sont effectués sur un Pentium 4 cadencé à 3Ghz avec 1024 Ko de mémoire cache. Les temps de calculs présentés sont la moyenne des temps d'exécution de 20 lancements du programme.

Les tableaux 1.1 et 1.2 présentent respectivement les temps de calculs avec une attache aux données L^2 pour les images 512×512 et 256×256. Les résultats pour les mêmes images mais avec une attache aux données L^1 sont présentés sur les tableaux 1.3 et 1.4.

TAB. 1.1 – Temps de calcul (en secondes) de nos deux algorithmes avec une attache aux données L^2 pour différentes valeurs de β. Toutes les images ont une taille de 512×512. Les temps pour la méthode séquentielle sont précisés par *(s)* à côté du nom de l'image.

Image	$\beta = 2$	$\beta = 5$	$\beta = 10$	$\beta = 20$	$\beta = 30$	$\beta = 40$
Lena	2,07	2,24	2,53	3,04	3,40	3,75
Lena(s)	52,71	53,98	55,01	56,90	59.25	60,72
Aerien	2.13	2.24	2.45	2.75	3.06	3.28
Aerien(s)	53,79	54,80	55,94	58,53	60,43	62,71
Barbara	2.07	2.26	2.51	2.87	3.22	3.50
Barbara(s)	50,00	51,08	52,27	54,39	56,44	58,05

Les résultats montrent une supériorité très nette pour l'algorithme à base de diviser pour régner. Si toutes les coupures prenaient le même temps, alors le ratio par rapport à la méthode séquentielle devrait être de 32 (si l'image présente 256 niveaux de gris). En pratique le ratio

(a) (b)

(c) (d)

FIG. 1.3 – Images utilisées pour nos tests de performance : (a) image *Lena*, (b) image aérienne de la région de Montpellier, (c) image *girl*, (d) image *Barbara*.

TAB. 1.2 – Temps de calcul (en secondes) de nos deux algorithmes avec une attache aux données L^2 pour différentes valeurs de β. Toutes les images ont une taille de 256×256. Les temps pour la méthode séquentielle sont précisés par *(s)* à côté du nom de l'image.

Image	$\beta = 2$	$\beta = 5$	$\beta = 10$	$\beta = 20$	$\beta = 30$	$\beta = 40$
Lena	0,51	0,54	0,60	0,72	0,8	0,87
Lena(s)	13,41	13,69	14,03	14,51	14,95	15,35
Aerien	0,55	0,57	0,61	0,67	0,74	0,78
Aerien(s)	12,83	13,19	13,53	14,14	14,72	15,18
Girl	0,52	0,55	0,64	0,75	0,85	0,92
Girl(s)	13,43	13,65	13,96	14,44	14,90	15,27
Girl+$\sigma = 20$	0,59	0,60	0,61	0,67	0,71	0,80
Girl+$\sigma = 20$(s)	14,00	14,29	14,66	15,24	15,68	16,06
Barbara	0,53	0,55	0,60	0,69	0,75	0,80
Barbara(s)	13,42	13,76	14,06	14,63	15,10	15,50

TAB. 1.3 – Temps de calcul (en secondes) de nos deux algorithmes avec une attache aux données L^1 pour différentes valeurs de β. Toutes les images ont une taille de 512×512. Les temps pour la méthode séquentielle sont précisés par *(s)* à côté du nom de l'image.

Image	$\beta = 1$	$\beta = 2$	$\beta = 3$	$\beta = 4$	$\beta = 5$	$\beta = 6$
Lena	2,13	2,78	3,47	4,07	4,64	5,24
Lena(s)	49,79	53,87	58,29	62,6	67,40	72,34
Aerien	2,19	2,96	2,72	4,48	5,17	6,01
Aerien(s)	50,53	58,01	66,07	73,63	81,04	88,87
Barbara	2,26	3,25	4,14	4,85	5,56	6,15
Barbara(s)	51,43	57,93	64,22	70,27	77,21	84,38

se situe aux alentours de 20-25. Nous avons vérifié expérimentalement que pour la version séquentielle de notre algorithme, le temps nécessaire pour effectuer une coupure dépend du niveau de gris à laquelle elle est effectuée. Plus le niveau de coupure est extrême, plus le temps nécessaire à la coupure minimale à tendance à être faible. En effet, la construction du graphe que nous utilisons (i.e, celle décrite dans (109)) a tendance à créer des arcs qui relient la source vers un nœud interne ou un nœud interne vers le puits avec de faibles capacités. Ceci justifie partiellement le fait que la coupure minimale soit plus facile à trouver.

Les résultats montrent également une dépendance avec la force de régularisation β, i.e, plus la régularisation est forte plus le temps de calcul est élevé. Cette dépendance avec β est quasi-linéaire.

Dans (23), Boykov *et al.* étudient l'algorithme de coupure minimale que nous utilisons pour différents types de graphe. Ils montrent par l'expérience, que pour le type de graphes que notre algorithme construit, la complexité est quasi-linéaire avec le nombre de pixels de l'image. Nos résultats reflètent également ce comportement.

Tab. 1.4 – Temps de calcul (en secondes) de nos deux algorithmes avec une attache aux données L^1 pour différentes valeurs de β. Toutes les images ont une taille de 256×256. Les temps pour la méthode séquentielle sont précisés par *(s)* à côté du nom de l'image.

Image	$\beta = 1$	$\beta = 1,5$	$\beta = 2$	$\beta = 2,5$	$\beta = 3$	$\beta = 3,5$
Lena	0,55	0,73	0,9	1,06	1,21	1,38
Lena(s)	13,51	15,01	16,59	18,14	19,67	21,30
Aerien	0,57	0,79	1,03	1,25	1,46	1,69
Aerien(s)	13,11	15,96	18,65	20,97	23,17	25,47
Girl	0,55	0,74	0,91	1,06	1,21	1,36
Girl(s)	13,49	14,76	16,19	17,52	19,01	20,63
Girl+$\sigma = 20$	0,67	1,02	1,32	1,57	1,82	2.10
Girl+$\sigma = 20$(s)	14,94	17,09	19,13	21,22	23,46	25,69
Barbara	0,58	0,82	1,01	1,19	1,39	1,58
Barbara(s)	14,08	15,95	17,09	19,81	21,78	23,85

1.5 Résultats

Dans cette partie nous présentons quelques résultats numériques sur différents types d'images. Nous comparons le résultat de notre algorithme avec deux algorithmes de minimisation de la variation totale : le premier s'appuie sur l'équation d'Euler-Lagrange tandis que le second repose sur des techniques de sous-gradients et est dû à A. Chambolle (34). Ces deux algorithmes sont brièvement décrits en annexe B.

1.5.1 Cas d'école

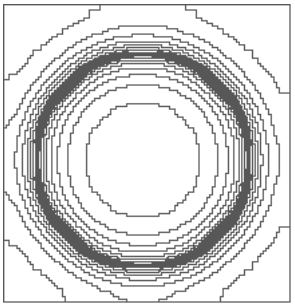
Dans (169), les auteurs donnent les solutions exactes et analytiques des minimiseurs de la variation totale avec une attache aux données quadratiques pour le cas de fonctions radiales. Par exemple, si l'image observée est un disque alors la solution est un disque de même rayon : seul le niveau de gris du cercle est modifié alors que les formes demeurent identiques. Ce résultat théorique est initialement énoncé par Meyer dans (128). Nous vérifions que notre algorithme fournit bien la solution attendue, et à titre de comparaison nous présentons également les solutions obtenues par l'algorithme de descente de gradient et celle obtenue par l'algorithme de projection de Chambolle. La figure 1.4 présente les solutions fournies par ces différents algorithmes ainsi que leurs lignes de niveaux. Remarquons que *visuellement* les résultats sont quasiment identiques. En revanche, à notre connaissance, seul notre algorithme fournit un cercle comme solution. Les autres méthodes produisent des minimiseurs qui présentent beaucoup plus de lignes de niveaux que ne l'énonce la théorie.

1.5.2 Cas L^2

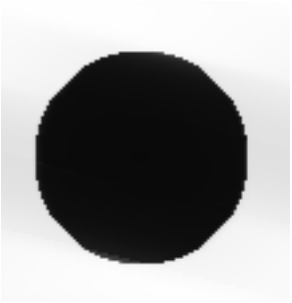
Nous appliquons la minimisation de la variation totale dans le but de restaurer les images corrompues par un bruit additif gaussien. La figure 1.5 présente une image de texte manuscrit et sa version dégradée par un bruit additif gaussien de moyenne nulle et de variance $\sigma = 15$ identiquement distribué. La figure 1.6 présente les résultats pour les différentes méthodes.

(a) Original image

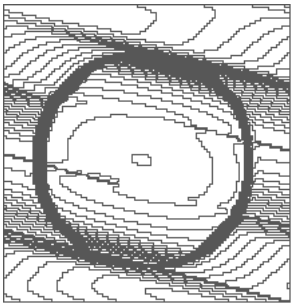
(b)



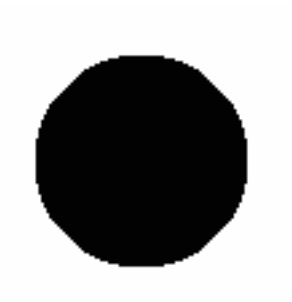
(c)



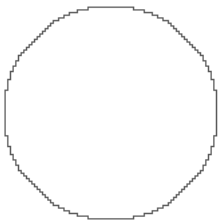
(d)



(e)

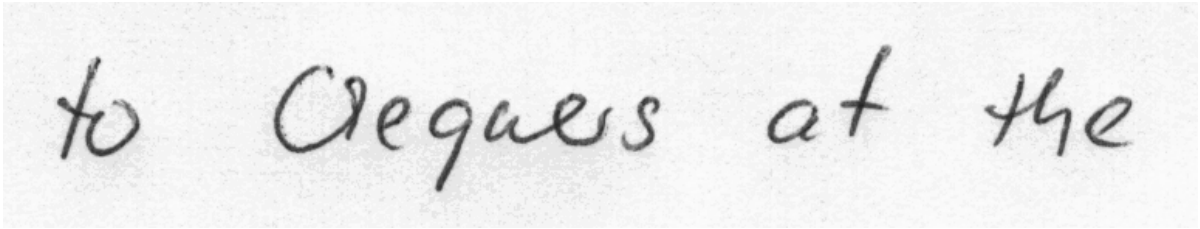


(f)

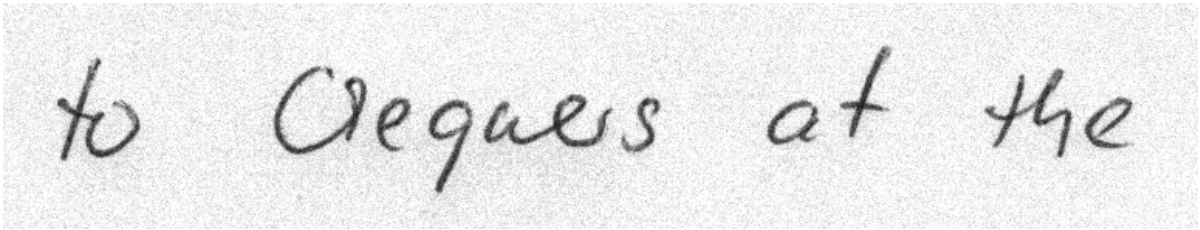


(g)

FIG. 1.4 – Minimiseurs de la variation totale avec une attache aux données L^2 ($\beta = 100$). L'image originale est présentée en (a). Les solutions produites par la descente de gradient est présentée en (b), celle par la méthode de Chambolle est présenté en (d) et le résultat pour notre algorithme en (f). Les lignes de niveaux associées à chacun de ces minimiseurs sont respectivement présentées en (c), (e) et (f).



(a)



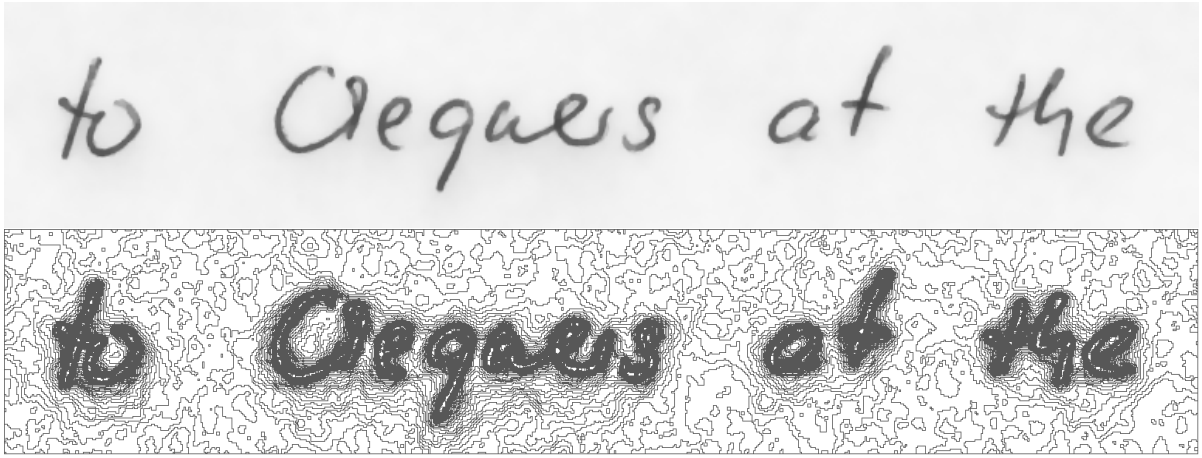
(b)

FIG. 1.5 – Une image manuscrite originale en (a) et sa version dégradée par un bruit gaussien additif ($\mu = 0$, $\sigma = 15$) en (b).

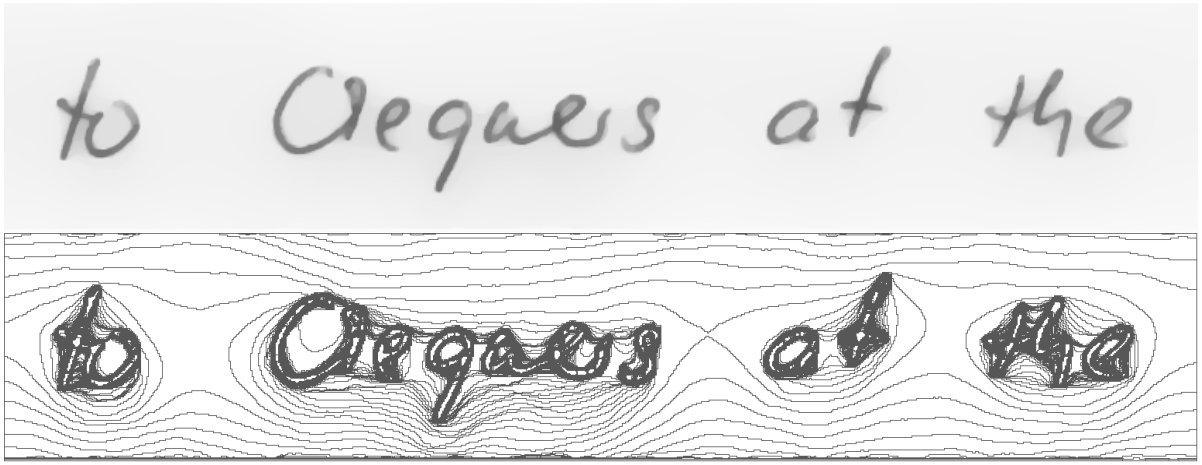
Nous observons tout d'abord que le fond du minimiseur exact est quasiment uniforme : i.e, avec très peu de lignes de niveau. Les autres méthodes en produisent beaucoup plus. Ce comportement est similaire au cas du cercle. En revanche, l'approche de Chambolle donne un résultat où ces lignes de niveaux sont plus régulières que celles données par la descente de gradient. Sur l'image du minimiseur exact, nous observons une concentration de lignes de niveaux près des discontinuités.

L'image *Girl* présentée sur la figure 1.3-(c) est corrompue par un bruit additif gaussien de moyenne nulle et avec une variance (i.e, $\sigma = 12$ et $\sigma = 20$). Ces deux images bruitées sont présentées respectivement sur les figures 1.7-(a) et 1.7-(c) tandis que les résultats de la restauration sont présentés sur les figures 1.7-(b) et 1.7-(d). Le coefficient de régularisation β est choisi pour donner le minimum de l'erreur quadratique moyenne. Nous voyons le phénomène de marches d'escalier apparaître dans les images résultats. Ce comportement a été décelé et décrit par Chan *et al.* dans (36), Dobson *et al.* dans (65) et Durand *et al.* (70). Nous observons également une légère perte de contraste dans les images résultats comparées à l'image originale. Ce phénomène est énoncé par Chan *et al.* dans (37) et il est principalement dû à l'utilisation de la norme L^2.

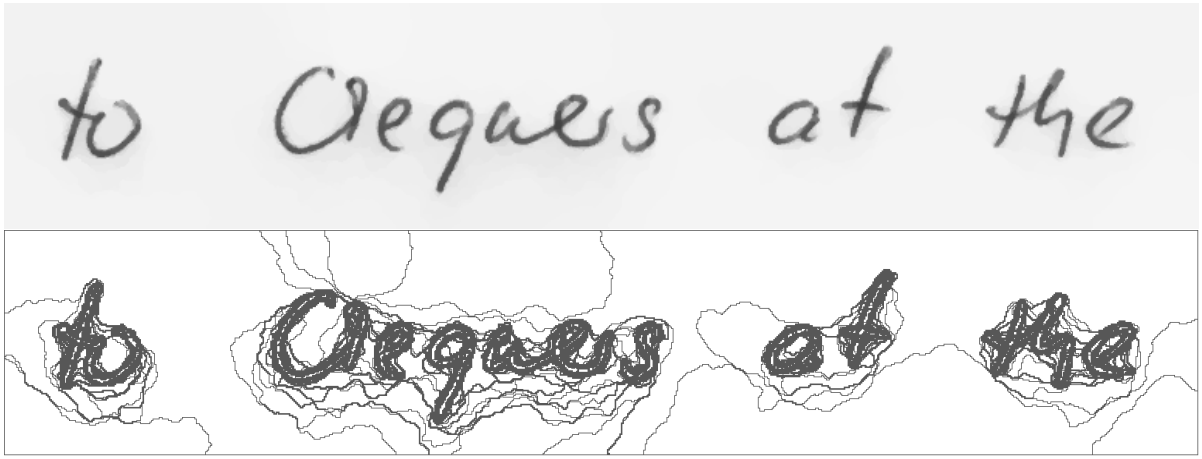
La figure 1.8-a) présente le minimiseur pour le coefficient de régularisation $\beta = 40$. La différence entre l'image originale et la version filtrée est présentée sur la figure 1.8-b). Ce type d'approche (filtrage et différence d'image) est utilisé pour effectuer de la décomposition d'images (179). L'objectif est d'obtenir deux images où la première contient l'information géométrique tandis que l'autre contient les textures. Notons que d'autres approches, comme celles décrites par Aujol *et al.* dans (9), Vese *et al* dans (179; 180), qui utilisent des modèles plus élaborés que $L^2 + TV$, donnent de bien meilleurs résultats.



(a)



(b)



(c)

FIG. 1.6 – Minimiseurs de la variation totale et ses lignes de niveaux avec une attache aux données L^2 ($\beta = 100$). Le minimiseur obtenu par descente de gradient est présenté en (a), par l'algorithme de Chambolle en (b), et notre résultat en (c).

(a) (b)

(c) (d)

FIG. 1.7 – Les images de *Girl* corrompues par un bruit gaussien additif de moyenne nulle $\mu = 0$, d'écart-type $\sigma = 12$ et $\sigma = 20$ sont respectivement présentées en (a) et (c). Les minimiseurs obtenus par le modèle $L^2 + TV$ sont montrés en (b) et (d) avec $\beta = 23.5$ et $\beta = 44.5$ respectivement.

(a)

(b)

FIG. 1.8 – Le minimiseur du modèle $L^2 + TV$ pour l'image *Barbara* avec $\beta = 2$ est présenté en (a). La différence entre l'image originale et l'image filtrée (a) est montrée en (b) (le zéro est fixé au niveau de gris 128).

1.5.3 Cas L^1

Comme le suggère Chan *et al.* dans (37), nous remplaçons la norme L^2 par L^1, et ce, afin de préserver le contraste dans les images solutions. Nous illustrons ici les bonnes propriétés de simplification d'images d'une telle minimisation.

L'image *Main* qui représente une main sur un fond texture est présentée sur la figure 1.9. Nous observons que plus le coefficient de régularisation β est fort, plus l'image est simplifiée. Autrement dit, la texture disparaît alors que le contraste demeure préservé. La figure 1.10 présente l'image *woman*. Cette figure montre également les minimiseurs et leurs lignes de niveaux pour différents coefficients de régularisation. Nous formulons les mêmes observations que pour l'image *Main*. Les détails du visage et les micro textures disparaissent et le fond tend à devenir homogène.

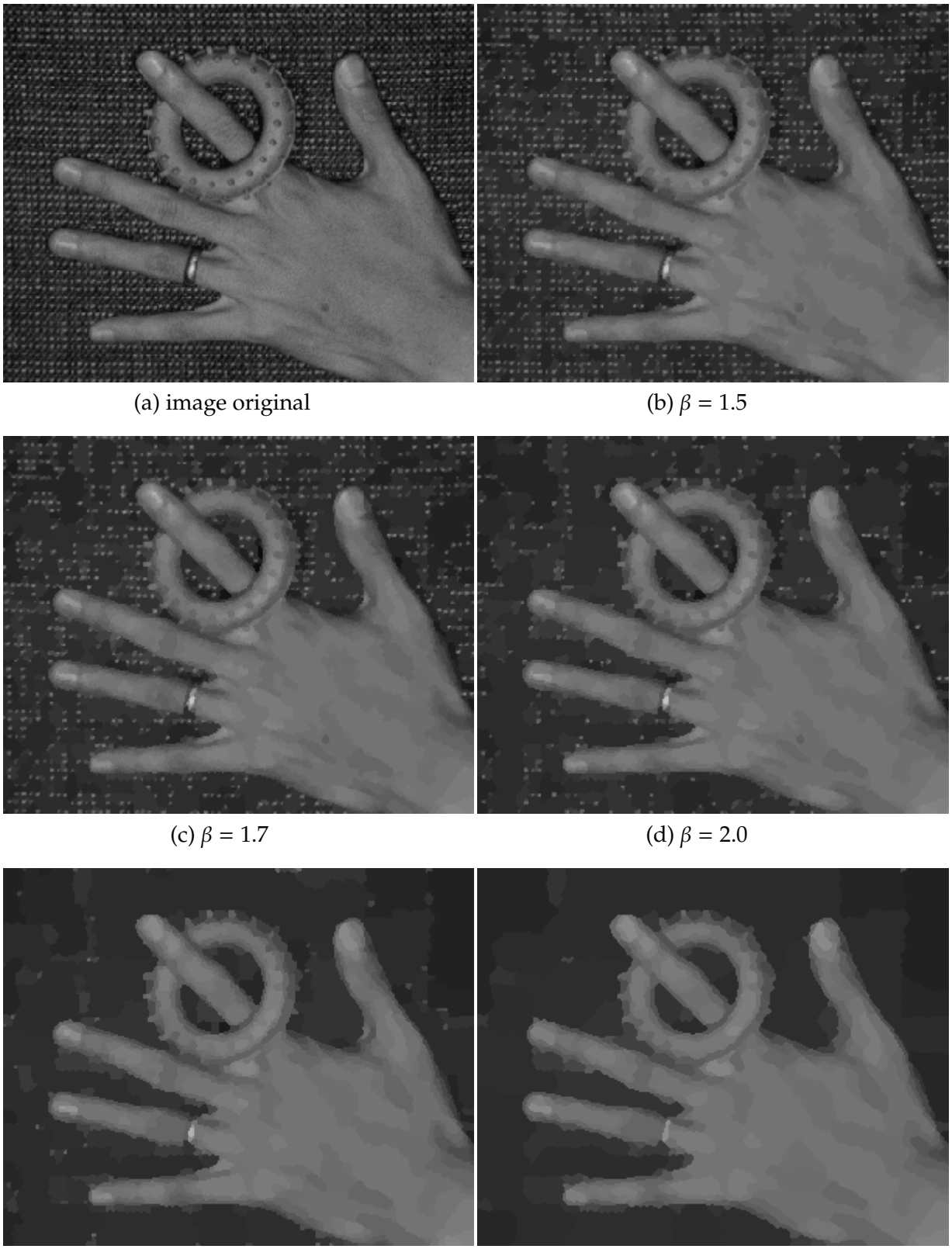
La figure 1.11-a) présente le résultat de la minimisation sur l'image *Barbara* avec un coefficient de régularisation $\beta = 2$. Comme pour l'image de la main, nous observons que la texture disparaît tout en préservant le contraste. L'image présentée sur la figure 1.11-b) correspond à la différence entre l'image observée et l'image filtrée. Dans (191), Yin *et al.* étudient ce modèle dans le but de faire de la décomposition d'images. Nous observons que le résultat de la décomposition est de bien meilleure qualité qu'avec le modèle $L^2 + TV$ (figure 1.8). En effet, le visage de *Barbara* apparaît beaucoup moins dans la composante de texture. De plus la texture sur la chaise est mieux captée par le modèle $L^1 + TV$.

Nous avons vu qu'un filtrage de type $L^1 + TV$ tend à faire disparaître les textures alors que le contraste est préservé. Nous montrons dans le chapitre 2 que la minimisation de la variation totale avec une attache aux données modélisée par la norme L^1 conduit à un filtre morphologique. Ce filtre est morphologique au sens où il est invariant par changement de contraste.

1.6 Conclusion

Dans ce chapitre, nous avons proposé un algorithme original de minimisation de la variation totale quand l'attache aux données est convexe. Cette dernière condition est restrictive puisque de nombreux modèles de bruits se modélisent par des attaches aux données non convexes. Notre algorithme repose sur la reformulation de l'énergie en fonction des lignes de niveaux d'une image et de champs de Markov binaires. La minimisation exacte de ces énergies est obtenue par l'utilisation de coupures minimales dans des graphes. Nous avons montré par l'expérience que les temps de calcul de notre algorithme à base de diviser pour régner était suffisamment faibles pour l'utiliser en pratique.

Dans le chapitre suivant, nous généralisons notre approche de la décomposition de l'énergie sur les ensembles de niveaux de l'image. Nous montrons que le cas de la variation totale comme terme de régularisation est un cas particulier d'une approche plus générale. En outre la condition de convexité de l'attache aux données est levée.



(a) image original (b) $\beta = 1.5$

(c) $\beta = 1.7$ (d) $\beta = 2.0$

(d) $\beta = 2.5$ (e) $\beta = 3.0$

FIG. 1.9 – Minimiseurs de la variation totale avec une attache aux données de type L^1.

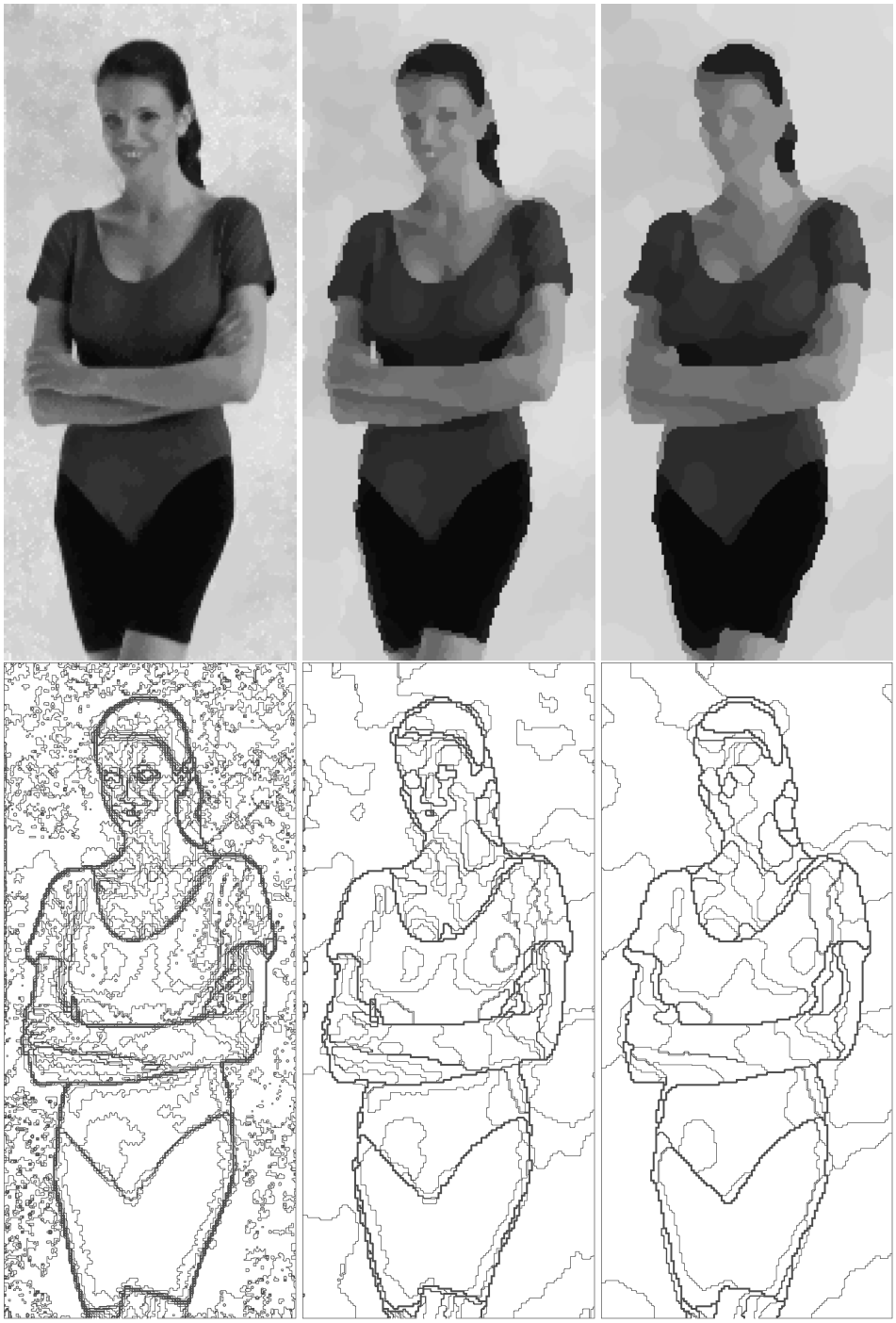


FIG. 1.10 – Minimiseurs de la variation totale avec une attache aux données L^1. De la gauche vers la droite : les minimiseurs pour $\beta = 1$, $\beta = 2.1$. $\beta = 3$. Les images des lignes de niveau des minimiseurs sont également présentées (seules les lignes de niveau multiples de 10 sont affichées).

(a)

(b)

FIG. 1.11 – Un minimiseur du modèle $L^1 + TV$ pour l'image *Barbara* avec $\beta = 2$ est présenté en (a). La différence entre l'image originale et l'image filtrée (a) est montrée en (b) (le zéro est fixé au niveau de gris 128).

Chapitre 2

Les fonctions nivelées : propriétés et minimisation exacte

Dans le chapitre précédent, nous avons étudié le cas où régularisation était la variation totale de l'image. Nous désirons généraliser cette approche à une classe plus grande de régularisation. C'est l'objet de ce chapitre. Dans la partie 2.1, nous étendons les résultats de décomposition d'énergies *a posteriori* sur les ensembles de niveaux à des termes de régularisation plus généraux que la variation totale. Nous verrons que cette classe contient des énergies non-convexes. Dans la partie 2.2 nous proposons un algorithme de minimisation *exacte* pour cette classe d'énergies, et ce, malgré le manque de convexité. Quelques exemples de restauration d'image radar, et de restauration d'image corrompue par du bruit impulsif sont présentés dans la partie 2.3. Nous nous concentrons sur ces types de bruits car un bruit additif gaussien de moyenne nulle correspond à une attache aux données L^2 et un bruit de Laplace (de moyenne nulle) à une attache aux données L^1. Les algorithmes de restaurations d'images corrompues par ces bruits ont été présentés dans le chapitre précédent. Enfin dans la partie 2.4, nous étudions théoriquement le cas de la minimisation de la variation totale avec une attache aux données L^1. Nous exhibons également une sous-classe d'énergies telles que leur minimisation définisse un filtre morphologique. Le caractère morphologique de ces filtrages est à prendre au sens où ils sont invariants par changement de contraste. En particulier le modèle $L^1 + TV$ jouit de cette propriété. Après la rédaction de ces travaux, nous avons pris connaissance des travaux de Zalesky présenté dans (195). Zalesky étudie, sous un autre nom, les propriétés des énergies que nous présentons dans ce chapitre. Il propose également un algorithme pour minimiser ces fonctions mais sa mise en œuvre demeure délicate et aucun résultat numérique n'est présenté. Contrairement à Zalesky, nous étudions plus particulièrement une sous classe de ces énergies dédiées au traitement des images et nous proposons un algorithme de minimisation efficace.

L'invariance par changement de contraste est publiée pour le cas $L^1 + TV$ en continu dans les actes de *4th IEEE International Symposium on Image and Signal Processing and Analysis (ISPA 2005)* (47), tandis que le reste est décrit dans l'article en deux parties accpeté à la revue *Journal of Mathematical Imaging and Vision* (57; 58).

2.1 Énergies nivelées

Afin de généraliser les résultats du chapitre précédent, nous présentons une classe de fonctions que nous nommons fonctions *nivelées*. Notre approche repose sur les idées d'intégration de fonctions au sens de Lebesgue. Nous exhibons également les liens étroits qu'elles entretiennent avec les champs de Markov binaires.

2.1.1 Décomposition sur les ensembles de niveaux

Nous définissons tout d'abord notre concept de fonctions *nivelées*.

Définition 2.1 *Une fonction d'une ou plusieurs variables* $x,\ y\ldots \in [0, L-1]$ *est* nivelée *si et seulement si elle peut s'écrire comme une somme, sur tous ses ensembles de niveaux, de combinaisons de fonctions caractéristiques du niveau courant, i.e, :*

$$f(x,\ y\ldots) = \sum_{\lambda=0}^{L-1} \psi(\lambda,\ \mathbb{1}_{\lambda<x},\ \mathbb{1}_{\lambda<y}\ldots)$$

Comme il a déjà été remarqué dans le chapitre précédent, une fonction d'une variable est toujours nivelée.

Proposition 2.1 *Une fonction d'une variable est nivelée.*

Preuve : La preuve repose sur des idées classiques d'intégration :

$$\begin{aligned} \forall k \in [0, L-1]\ \ f(k) &= \sum_{\lambda=0}^{k-1} (f(\lambda+1) - f(\lambda)) \ +\ f(0) \\ &= \sum_{\lambda=0}^{L-2} (f(\lambda+1) - f(\lambda))\ \mathbb{1}_{\lambda<k} \ +\ f(0) \end{aligned} \tag{2.1}$$

Remarquons que la somme sur λ ne se poursuit que jusqu'à $\lambda = L-2$. Ceci produit bien un résultat cohérent pour $k = 0$ et $k = L-1$, puisque

$$\forall k \in [0, L-1]\ ,\ \mathbb{1}_{\lambda<k} = 0 \text{ quand } \lambda = L-1.$$

□

Les résultats et théorèmes suivants précisent les connexions entre une énergie nivelée et sa version sous forme de Gibbs.

Proposition 2.2 *Les deux propositions suivantes sont équivalentes :*

a) L'énergie totale sous forme de Gibbs est nivelée.

b) Chaque énergie associée à une clique est une fonction nivelée.

Preuve : Cas b) $\Rightarrow$ a). L'ensemble des fonctions nivelées jouit d'une structure d'espace vectoriel.

Cas a) $\Rightarrow$ b). Nous nous limitons aux énergies associées à des cliques d'ordre inférieur ou égal à deux. Cependant, la preuve s'étend facilement par induction aux énergies impliquant un ordre quelconque fini de cliques. Dans ce but, nous écrivons l'énergie totale ainsi :

$$\begin{aligned} E(u_s,\ u_t,\ u_r \ldots) &= \sum_{(s,t)} U(u_s,\ u_t) + \sum_s T(u_s) \\ &= \sum_{\lambda=0}^{L-2} \psi(\lambda,\ \mathbb{1}_{\lambda<u_s},\ \mathbb{1}_{\lambda<u_t},\ \mathbb{1}_{\lambda<u_t}, \ldots)\ , \end{aligned}$$

Puisque nous nous restreignons aux cliques d'ordre deux, nous affectons $u_r = 0\ \forall r \notin \{s,\ t\}$ dans la formule précédente. Nous rappelons que $\{s, t\}$ signifie que les sites s et t sont voisins. Nous obtenons alors

$$\begin{aligned} U(u_s,\ u_t) + T(u_s) + T(u_t) + \mathrm{C} &= \sum_{\lambda=0}^{L-2} \underbrace{\psi(\lambda,\ \mathbb{1}_{\lambda<u_s},\ \mathbb{1}_{\lambda<u_t},\ 0,\ 0 \ldots)}_{\downarrow} \\ &= \sum_{\lambda=0}^{L-2} \psi_{st}(\lambda,\ \mathbb{1}_{\lambda<u_s},\ \mathbb{1}_{\lambda<u_t})\ , \end{aligned}$$

Comme chaque fonction d'une seule variable est nivelée (Cf. proposition 2.1), le résultat est acquis pour les termes impliquant des cliques d'ordre deux : $U(u_s,\ u_t)$. □

Le résultat suivant établi la forme que doivent prendre les énergies associées aux cliques afin qu'elles jouissent de la propriété de nivellement. Nous donnons ici la forme que prennent les énergies nivelées associées aux cliques d'ordre deux.

Proposition 2.3 *Une fonction* symétrique *de deux variables, $U(x, y)$, est nivelée* si et seulement si *elle s'écrit sous les deux formes équivalentes suivantes :*

$$U(x, y) = S(\max(x,\ y)) - S(\min(x,\ y)) + D(x) + D(y) \tag{2.2}$$

$$= f(\max(x,\ y)) - g(\min(x,\ y))\ , \tag{2.3}$$

où $S,\ D,\ f = S + D$ et $g = S - D$ sont des fonctions $[0,\ L-1] \mapsto \mathbb{R}$.

Preuve :
• Condition suffisante - Remarquons tout d'abord que

$$\mathbb{1}_{\lambda<\min(x,\ y)} = \mathbb{1}_{\lambda<x} \times \mathbb{1}_{\lambda<y}\ .$$

Nous avons donc

$$\begin{aligned} S(\max(x,\ y)) - S(\min(x,\ y)) &= S(x) + S(y) - 2\ S(\min(x,\ y)) \\ &= \sum_{\lambda=0}^{L-2} (S(\lambda+1) - S(\lambda))\ \left(\mathbb{1}_{\lambda<x} + \mathbb{1}_{\lambda<y} - 2\ \mathbb{1}_{\lambda<x} \times \mathbb{1}_{\lambda<y} \right) \\ &= \sum_{\lambda=0}^{L-2} (S(\lambda+1) - S(\lambda))\ |\ \mathbb{1}_{\lambda<x} - \mathbb{1}_{\lambda<y}\ |\ . \end{aligned}$$

• Condition nécessaire - L'argument précédent nous incite à développer le terme $U(x,\ y)$ sur la base

$$\{\ \mathbb{1}_{\lambda<x},\ \mathbb{1}_{\lambda<y}\ \}_\lambda \text{ et } \{\ |\ \mathbb{1}_{\lambda<x} - \mathbb{1}_{\lambda<y}\ |\ \}_\lambda\ ,$$

plutôt que sur la base

$$\{\ \mathbb{1}_{\lambda<x},\ \mathbb{1}_{\lambda<y}\ \}_\lambda \text{ et } \{\ |\mathbb{1}_{\lambda<x} \times \mathbb{1}_{\lambda<y}\ |\ \}_\lambda\ .$$

Comme U est nivelée, en la développant sur la base précédente, il existe R et θ tels que

$$U(x,\ y) = \sum_{\lambda=0}^{L-2} R(\lambda)\ |\ \mathbb{1}_{\lambda<x} - \mathbb{1}_{\lambda<y}\ | + \sum_{\lambda=0}^{L-2} \theta(\lambda)\ [\ \mathbb{1}_{\lambda<x} + \mathbb{1}_{\lambda<y}\ ] \tag{2.4}$$

$$= \sum_{\lambda=0}^{L-2} (S(\lambda+1) - S(\lambda))\ |\ \mathbb{1}_{\lambda<x} - \mathbb{1}_{\lambda<y}\ | + \sum_{\lambda=0}^{L-2} (D(\lambda+1) - D(\lambda))\ [\ \mathbb{1}_{\lambda<x} + \mathbb{1}_{\lambda<y}\ ] \tag{2.5}$$

Par identification entre les équations (2.4) et (2.5), nous avons $S(\lambda+1) - S(\lambda) = R(\lambda)$ et $D(\lambda+1) - D(\lambda) = \theta(\lambda)$. Nous pouvons respectivement voir $S(\cdot)$ et $D(\cdot)$ comme des "primitives" de $R(\cdot)$ et $\theta(\cdot)$, i.e,

$$S(\lambda) = \sum_{\mu<\lambda} R(\mu)\ \ \forall \lambda \geq 1\ ,$$

et

$$D(\lambda) = \sum_{\mu<\lambda} \theta(\mu)\ \ \forall \lambda \geq 1\ ,$$

avec

$$D(0) = S(0) = 0\ .$$

La valeur de $S(0)$ et $D(0)$ n'importent pas (comme pour une primitive qui est définie à une constante près) puisque que ces quantités disparaissent lors des différences successives dans l'équation (2.5). En utilisant la forme introduite dans (2.1) pour une fonction nivelée d'une variable, ceci implique, à une constante près, que :

$$\begin{aligned} U(x,\ y) &= S(\max(x,\ y)) - S(\min(x,\ y)) + D(x) + D(y) \\ &= S(\max(x,\ y)) - S(\min(x,\ y)) + D(\max(x,\ y)) + D(\min(x,\ y))\ . \end{aligned}$$

Ceci conclut la preuve. □

Nous donnons maintenant le comportement des potentiels de cliques d'ordre deux sous de faibles conditions (dans le contexte du traitement des images).

Proposition 2.4 *Supposons que les prérequis suivants pour un potentiel de clique d'ordre deux soient satisfaits :*

1. $U(x,\ y)$ *est une fonction* nivelée symétrique *en* x, y.
2. $\forall y \in [0, L-1]$, $U(x, y)$ *atteint son minimum en* $x = y$.

Alors, dans les fonctions f, g *et* $S = (f + g)/2$ *définies dans la proposition 2.3, sont des fonctions* croissantes *sur* $[0, L-1]$. *Dans ce cas, l'énergie associée s'écrit :*

$$\begin{aligned} U(x,\ y) &= |S(x) - S(y)| + D(x) + D(y) \\ &= \sum_{\lambda=0}^{L-2} R(\lambda) |\mathbb{1}_{\lambda<x} - \mathbb{1}_{\lambda<y}| + D(x) + D(y)\ , \end{aligned}$$

où $R(\lambda) = S(\lambda + 1) - S(\lambda)$ *est une fonction positive sur* $[0, L-1]$.

Preuve : La condition de minimum d'énergie pour $x \geq y$ est $U(x, y) \geq U(y, y)$. D'après les conditions de la proposition 2.3 on a :

$$U(x,\ y) = f(x) - g(y) \geq U(y,\ y) = f(y) - g(y) \Rightarrow f(x) \geq f(y)\ \ \forall x \geq y\ . \tag{2.6}$$

Réciproquement, pour $x \leq y$:

$$U(x,\ y) = f(y) - g(x) \geq U(y,\ y) = f(y) - g(y) \Rightarrow g(x) \leq g(y)\ \ \forall x \leq y\ . \tag{2.7}$$

Ces inégalités doivent être satisfaites $\forall y \in [0, L-1]$. Par conséquent f, g et $S = \dfrac{f+g}{2}$ doivent être des fonctions croissantes sur $[0, L-1]$.

□

En d'autres termes, $\forall \lambda \in [0, L-1]$, *nousavons* :

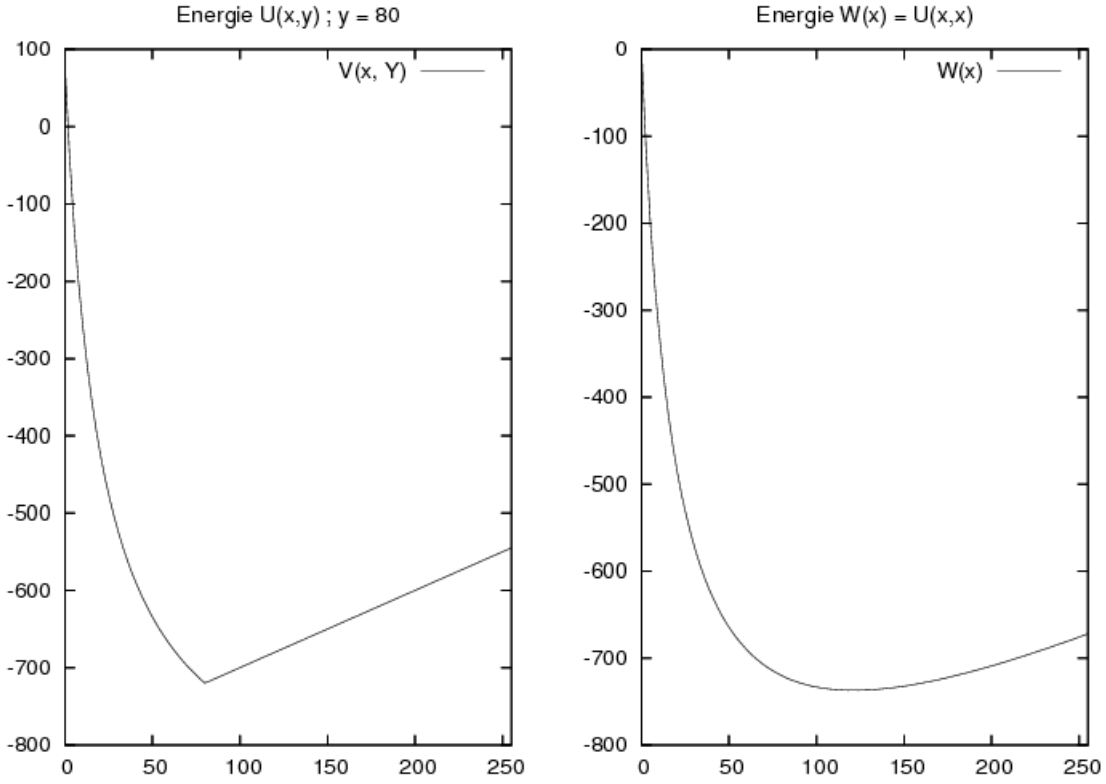
$$U(x, y) = \begin{cases} f(y) - g(x) & \text{est une fonction décroissante en } x \quad \text{pour } x \leq y \\ f(x) - g(y) & \text{est une fonction croissante en } x \quad \text{pour } x \geq y\ . \end{cases} \tag{2.8}$$

Ces deux dernières inégalités signifient que $U(\cdot, y)$ doit être une fonction *quasi-convexe* (20; 78) qui atteint son minimum en y, $\forall y \in [0,\ L-1]$. Une fonction est quasi-convexe si et seulement ses ensemble de niveaux sont des ensembles convexes. La figure 2.1 présente quelques exemples d'une telle énergie. De plus l'ensemble des énergies nivelées, convexes et vérifiant $D(x) = 0$ (i.e, aucun biais sur les niveaux de gris n'est introduit) se réduit à aux fonctions linéaires

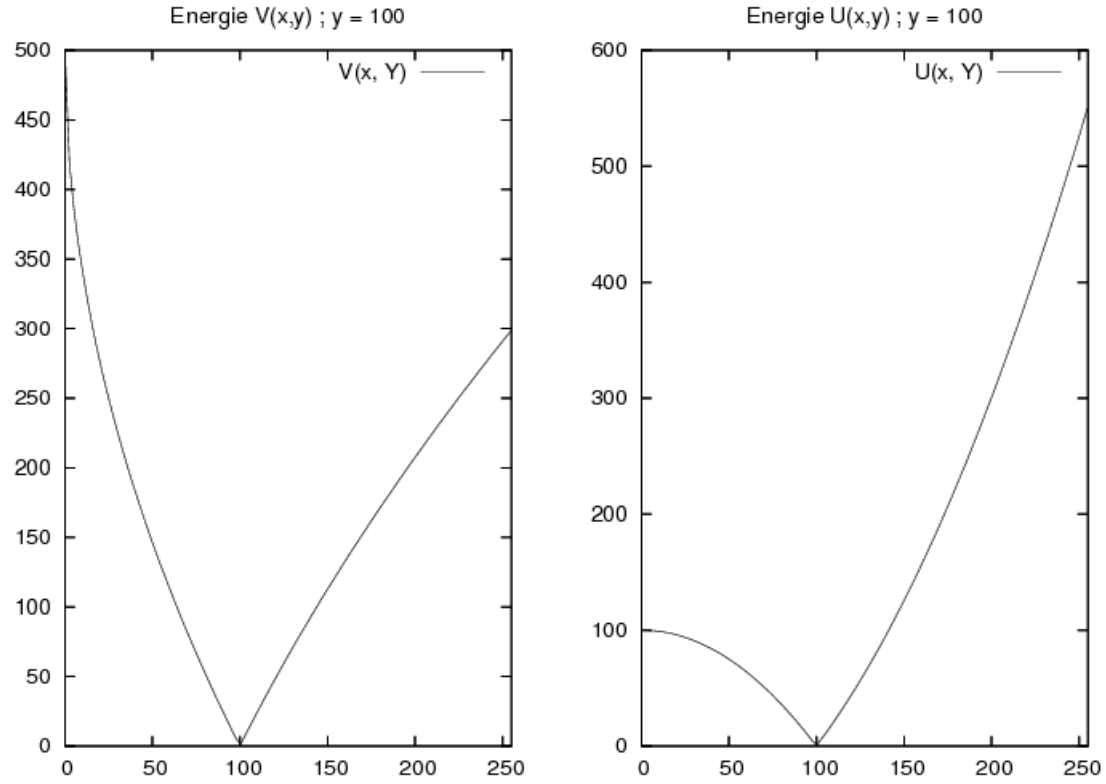
$$S(x) = \beta\, x\ .$$

En effet, d'après l'équation (2.6) $S(x) = f(x)$ doit être convexe et d'après l'équation (2.7) $S(x) = g(x)$ doit être concave. Nous obtenons donc la variation totale. Nous retrouvons ici l'importance de la variation totale.

Remarquons qu'en général, aucune condition n'est imposée sur D. Seule la somme $S \pm D$ doit être une fonction croissante sur $[0, L-1]$.



a) gauche : $f(x) = x^2/100 \quad g(x) = 50\ \sqrt{x}$
droite : $f(x) = 50\ \sqrt{x} \quad g(x) = x^2/100$



b) gauche : $f(x) = g(x) = 50\ \sqrt{x}$ droite : $f(x) = g(x) = x^2/100$

FIG. 2.1 – $U(x, y)$ en tant que fonction de x ($y = 100$) .

2.1.2 Structure des énergies nivelées markoviennes

Pour la clarté de l'exposé, nous supposons que les potentiels de cliques d'ordre deux satisfont les conditions des propositions 2.3 et 2.4 selon une connexité donnée. Les potentiels de cliques d'ordre deux seront pondérés différemment selon l'orientation de la clique considérée. Pour cela nous introduisons quelques notations. Nous notons γ_i le nombre de voisins de type $i \in [1, I]$, $i(s,t) = i$ la i^{e} clique d'ordre deux, et enfin $U_i(u_s, u_t)$ les énergies de régularisation associées. Nous supposons que les termes U_i satisfont les hypothèses de la propriété 2.3. Nous nous intéressons donc à minimiser l'énergie suivante :

$$E(u|v) = \sum_{i=1}^{I} \sum_{\substack{(s,t) \\ i(s,t)=i}} U_i(u_s, u_t) + \sum_s f(u_s, v_s) \ , \tag{2.9}$$

Remarquons que chaque terme $D_i(u_s)$ apparaît dans toutes les énergies $U_i(u_s, u_t) \mid t \sim s$ avec $i(s,t) = i$. Sa contribution dans l'énergie totale de régularisation s'élève donc à $\gamma_i D_i$. Nous noterons donc

$$D = \sum_{i=1}^{I} \gamma_i \, D_i \ .$$

montre que l'énergie totale *a posteriori* prend la forme nivelée suivante :

$$E(u \mid v) = \sum_{i=1}^{I} \sum_{\substack{(s,t) \\ i(s,t)=i}} |S_i(u_s) - S_i(u_t)| + D_i(u_s) + D_i(u_t) + \sum_s f_s(u_s) \tag{2.10}$$

$$= \sum_{\lambda=0}^{L-2} \left\{ \sum_{i=1}^{I} \sum_{\substack{(s,t) \\ i(s,t)=i}} R_i(\lambda) \mid u_s^\lambda - u_t^\lambda \mid + \sum_s \delta(\lambda, v_s) \, (1 - u_s^\lambda) \right\} + \tilde{C} \tag{2.11}$$

où

- la constante $\tilde{C}$ vaut $\tilde{C} = \sum_s (f_s(0) + D(0))$,
- le terme $R_i(\lambda)$ qui joue le rôle du coefficient de régularisation au niveau λ vaut :

  $$R_i(\lambda) = S_i(\lambda + 1) - S_i(\lambda)$$

  et est une fonction *positive* sur $[0, L-1]$ $\forall i \in [1, I]$,
- et enfin $\delta(\lambda, v_s)$, qui joue le rôle du coefficient *apparent* d'attache aux données, vaut :

  $$\delta(\lambda, v_s) = D(\lambda + 1) - D(\lambda) \ + \ f_s(\lambda + 1) - f_s(\lambda) \ .$$

Encore une fois, insistons sur le fait qu'aucune condition n'est imposée sur les deux fonctions $\lambda \mapsto D(\lambda)$ et $\lambda \mapsto f_s(\lambda)$.
D'après l'équation (2.11), nous voyons que toutes les énergies locales *a posteriori* sont elles-

mêmes nivelées, i.e,

$$E(u_s \mid N_s,\ v_s) \quad = \quad \sum_{\lambda=0}^{L-2} E^{\lambda}(u_s^{\lambda} \mid N_s^{\lambda},\ v_s) \quad \text{avec} \tag{2.12}$$

$$E^{\lambda}(u_s^{\lambda} \mid N_s^{\lambda},\ v_s) \quad = \quad \sum_{i=1}^{I} \sum_{\substack{t \sim s \\ i(s,t)=i}} R_i(\lambda) \mid u_s^{\lambda} - u_t^{\lambda} \mid + \sum_{s} \delta(\lambda, v_s)\ (1 - u_s^{\lambda})\ . \tag{2.13}$$

Cette formulation est la représentation générale d'une énergie *a posteriori* markovienne nivelée à interactions binaires.

2.1.3 Quelques propriétés

La proposition suivante montre qu'étant donné deux configurations binaires ordonnées, nous pouvons effectuer un échantillonnage qui préserve cet ordre pour les énergies nivelées.

Proposition 2.5 *Supposons que les énergies de régularisation satisfont les conditions de la proprosition 2.4. Considérons deux configurations binaires, u et ũ, satisfaisant*

$$\tilde{u}^{\lambda} \preceq u^{\lambda} \Leftrightarrow \tilde{u}_s^{\lambda} \leq u_s^{\lambda} \ \ \forall s \in S\ ,$$

à un niveau λ quelconque. Alors, quelque soit le terme d'attache aux données, nous avons l'inégalité suivante :

$$P(\tilde{u}_s^{\lambda} = 1 \mid \tilde{N}_s^{\lambda},\ v_s) \leq P(u_s^{\lambda} = 1 \mid N_s^{\lambda},\ v_s)\ .$$

Preuve : Par restriction, nous avons

$$\tilde{u}^{\lambda} \preceq u^{\lambda} \Rightarrow \forall t \in \tilde{N}_s^{\lambda}\ \tilde{u}_t \leq u_t\ .$$

Maintenant, d'après les équations (1.9) et (2.13) la quantité $\Delta\phi_s(\lambda)$ se calcule facilement ainsi :

$$\begin{aligned} \Delta\phi_s(\lambda) \quad &= \quad E^{\lambda}(u_s^{\lambda} = 1 \mid N_s^{\lambda},\ v_s) - E^{\lambda}(u_s^{\lambda} = 0 \mid N_s^{\lambda},\ v_s) \\ &= \quad -\delta(\lambda,\ v_s) + \left(\sum_{i=1}^{I} \sum_{\substack{t \sim s \\ i(s,t)=i}} R_i(\lambda)\ (1 - 2\ u_t^{\lambda}) \right) \end{aligned}$$

Puisque par hypothèse nous avons $R_i(\lambda) \geq 0\, \forall i \in [1, I]$, $\Delta\phi_s(\lambda)$ est donc une fonction décroissante de la configuration locale du voisinage $N_s^{\lambda} = \{u_t^{\lambda}\}_{t \sim s}$ au niveau λ. Par conséquent, nous avons

$$P(u_s^{\lambda} = 1 \mid N_s^{\lambda},\ v_s) = \frac{1}{1 + \exp \Delta\phi_s(\lambda)} \tag{2.14}$$

qui est une fonction croissante en N_s^{λ}, et ce, quelque soit le terme d'attache aux données $\delta(\lambda,\ v_s)$. Ceci conclut la preuve. □

Nous présentons maintenant une propriété de monotonie pour un sous-ensemble des énergies nivelées. Celle propriété est l'analogue de la propriété de monotonie pour le cas de la Variation Totale avec une attache aux données convexe.

Proposition 2.6 *Si pour touts configurations des voisins u_t et pour toutes données observées v_s, toutes les énergies conditionnelles $E(u_s|N_s, v_s)$ sont des fonctions* convexes *des niveaux de gris $u_s \in [\![0, L-1]\!]$, alors il existe un algorithme stochastique "couplé" de minimisation qui préserve la propriété de monotonie.*

Preuve : Nous reprenons le schéma de la preuve de la proposition 2.6. Rappelons que nous avons :

$$\begin{aligned} \Delta\phi_s(\lambda) &= E^\lambda(u_s^\lambda = 1 \mid N_s^\lambda,\, v_s) - E^\lambda(u_s^\lambda = 0 \mid N_s^\lambda,\, v_s) \\ &= -\delta(\lambda,\, v_s) + \left(\sum_{i=1}^{I} \sum_{\substack{t \sim s \\ i(s,t)=i}} R_i(\lambda)\,(1 - 2\,u_t^\lambda) \right) . \end{aligned}$$

D'après la convexité des énergies conditionnelles, $\Delta\phi_s(\cdot)$ est une fonction décroissante. Par conséquent la probabilité suivante

$$P(u_s^\lambda = 1 \mid N_s^\lambda,\, v_s) = \frac{1}{1 + \exp \Delta\phi_s(\lambda)} \ , \tag{2.15}$$

est décroissante en fonction de λ. Le reste de la preuve est identique à celle du lemme 1.1. C'est-à-dire, $(L-1)$ échantillonneurs Gibbs couplés sont définis pour les $(L-1)$ images binaires de la manière suivante : tout d'abord nous supposons que nous disposons d'un ordre de parcours des sites. Quand un site s est visité, nous tirons uniformément $\rho_s \in [0, 1]$. Puis nous effectuons les affectations suivantes pour chaque valeur de λ :

$$u_s^\lambda = \begin{cases} 1 \text{ si } 0 \leq \rho_s \leq P_s(\lambda) \\ 0 \text{ sinon} \end{cases}$$

Cet échantillonnage maintient la propriété de monotonie. L'extension à $(L-1)$ échantillonneurs de Gibbs couplés possédant la même température T quand ils visitent le site s est immédiate. Le plongement de ces échantillonneurs dans un recuit simulé où la décroissance de la température T s'effectue de manière logarithmique et où le température tends vers 0 permet de conclure.

□

Nous présentons maintenant nos algorithmes de minimisation exacte pour des énergies nivelées.

2.2 Algorithmes de minimisation et expériences

Nous présentons deux algorithmes de minimisation exacte d'une énergie nivelée. Le premier correspond au cas où l'énergie totale est convexe, tandis que le second s'applique dans où les termes de régularisation sont des fonctions nivelées.

2.2.1 Cas convexe

Remarquons que la proposition 2.6 généralise le lemme 1.1 (qui traite le cas spécifique de la variation totale avec des attaches aux données convexes) au cas des fonctions nivelées dont les énergies locales sont convexes. L'algorithme de minimisation est donc essentiellement celui présenté dans la partie 1.4. Seuls les arcs correspondant aux termes de régularisation changent. Leur valeur dépend du niveau considéré λ alors qu'ils prenaient une valeur indépendante du niveau λ dans le cas de variation totale.

2.2.2 Reformulation énergétique et représentation par graphe

Nous repartons de l'équation (2.11), à savoir :

$$E(u \mid v) = \sum_{\lambda=0}^{L-2} \left\{ \sum_{i=1}^{I} \sum_{\substack{(s,t) \\ i(s,t)=i}} R_i(\lambda) \mid u_s^\lambda - u_t^\lambda \mid + \sum_s \delta(\lambda, v_s)\,(1 - u_s^\lambda) \right\} + \tilde{C} \ . \tag{2.16}$$

Puisque nous sommes intéressés par la minimisation de $E(u \mid v)$, nous pouvons retirer la constante $\tilde{C}$. Nous rappelons que nous avons supposé que R_i est une fonction *positive*. Puisque l'énergie n'est pas convexe en général (non seulement à cause des termes non convexes d'attaches aux données mais aussi de la régularisation), la monotonie des $\{u_s^\lambda\}$ n'est pas assurée en général ; autrement dit, la propriété de monotonie (équation (1.6)) n'est pas toujours vraie :

$$\forall s\, \forall \lambda \leq \mu \ \ u_s^\lambda \leq u_s^\mu \ .$$

Remarquons que cette dernière condition peut s'exprimer de manière plus locale (i.e, sans utiliser la variable supplémentaire μ) :

$$\forall s\, \forall \lambda \ \ u_s^\lambda \leq u_s^{\lambda+1} \ . \tag{2.17}$$

Il est aisé de voir que la minimisation de (2.16) est équivalente à l'optimisation du problème suivant :

$$\begin{cases} \displaystyle \operatorname*{argmin}_{\{u^\lambda\}} \sum_{\lambda=0}^{L-2} \left\{ \sum_{i=1}^{I} \sum_{\substack{(s,t) \\ i(s,t)=i}} R_i(\lambda) \mid u_s^\lambda - u_t^\lambda \mid + \sum_s \delta(\lambda, v_s)\,(1 - u_s^\lambda) \right\} \\ \text{sous la contrainte } \forall s\, \forall \lambda \ \ u_s^\lambda \leq u_s^{\lambda+1} \end{cases}$$

Maintenant, nous transformons ce problème d'optimisation sous contrainte en un problème d'optimisation sans contrainte. Pour cela nous ajoutons un terme H à la fonction objectif qui forcera la validité des contraintes (équation (2.17)) :

$$H(u_s^\lambda - u_s^{\lambda+1}) \leq 0 \ ,$$

où $H : \mathbb{R} \mapsto \mathbb{R}$ est la fonction d'Heaviside définie ainsi :

$$H(x) = \begin{cases} 0 \text{ si } x \leq 0 \ , \\ 1 \text{ sinon.} \end{cases} \tag{2.18}$$

Définissons alors l'énergie $E_\alpha(\{\cdot^\lambda\}|v)$ avec $\alpha > 0$ de la manière suivante :

$$E_\alpha(\{u^\lambda\}|v) = \sum_{\lambda=0}^{L-2} \left\{ \sum_{i=1}^{I} \sum_{\substack{(s,t) \\ i(s,t)=i}} R_i(\lambda) \mid u_s^\lambda - u_t^\lambda \mid + \sum_s \delta(\lambda, v_s)\,(1 - u_s^\lambda) + \sum_s \alpha H(u_s^\lambda - u_s^{\lambda+1}) \right\} . \tag{2.19}$$

Remarquons que $E_\alpha(\{\cdot^\lambda\}, v)$ est une énergie dont les variables sont $L-1$ images binaires tandis que la variable de l'énergie $E(\cdot, v)$ est une image à niveaux de gris, i.e, $\Omega \mapsto [L]^\Omega$. Nous montrons maintenant que cette énergie est représentable par graphe au sens de Kolmogorov *et al.* (109). Pour cela, il suffit que chaque terme

$$R_{st}^{i,\lambda}(u_s^\lambda, u_t^\lambda) = R_i(\lambda) \mid u_s^\lambda - u_t^\lambda \mid$$

et

$$H_{st}^{\lambda}(u_s^\lambda, u_t^\lambda) = \alpha H(u_s^\lambda - u_s^{\lambda+1})$$

vérifient la propriété de régularité, i.e,

$$R_{st}^{i,\lambda}(0,0) + R_{st}^{i,\lambda}(1,1) \leq R_{st}^{i,\lambda}(0,1) + R_{st}^{i,\lambda}(1,0) \ , \tag{2.20}$$

et

$$H_{st}^{\lambda}(0,0) + H_{st}^{\lambda}(1,1) \leq H_{st}^{\lambda}(0,1) + H_{st}^{\lambda}(1,0) \ . \tag{2.21}$$

C'est effectivement le cas car nous avons

$R_{st}^{i,\lambda}(0,0)$	$R_{st}^{i,\lambda}(0,1)$
$R_{st}^{i,\lambda}(1,0)$	$R_{st}^{i,\lambda}(1,1)$

$=$

0	$R_i(\lambda)$
$R_i(\lambda)$	0

et

$H_{st}^{\lambda}(0,0)$	$H_{st}^{\lambda}(0,1)$
$H_{st}^{\lambda}(1,0)$	$H_{st}^{\lambda}(1,1)$

$=$

0	0
1	0

L'inégalité (2.20) est vérifiée car R_i est une fonction positive. D'après la définition de H et puisque $\alpha > 0$, l'inégalité (2.21) est également vérifiée. L'énergie totale $E_\alpha(\{u^\lambda\}|v)$ définie par (2.19) est représentable sous forme de graphe. En effet, la somme des termes est représentable sous forme de graphe (109).

Nous montrons dans la sous-section suivante comment la minimisation de l'énergie totale (2.19) peut conduire à un minimiseur de l'équation (2.16).

2.2.3 Satisfaction de la propriété de monotonie

Nous procédons en deux étapes. D'abord nous supposons que les contraintes (2.17) sont satisfaites pour un minimiseur de $E_\alpha(\{\cdot^\lambda\}|v)$. Puis nous exhibons une condition qui implique nécessairement la satisfaction de ces contraintes.

Cas où la contrainte est satisfaite

La proposition suivante montre comment construire un minimiseur de $E(\cdot|v)$ à partir d'un minimiseur de $E_\alpha(\{\cdot^\lambda\}|v)$ qui satisfait la contrainte de monotonie.

Proposition 2.7 *Supposons que $\{\hat{u}^\lambda\}$ soit un minimiseur de $E_\alpha(\{\cdot^\lambda\}|v)$ tel que*

$$\forall s \, \forall \lambda \;\; \hat{u}_s^\lambda \leq \hat{u}_s^{\lambda+1} \;\; .$$

Alors l'image $\hat{u}$ définie par

$$\forall s \;\; \hat{u}_s = \min\{\lambda, \hat{u}_s^\lambda = 1\} \;\; ,$$

est un minimiseur de $E(\cdot|v)$.

Preuve : Remarquons que le minimiseur $\{\hat{u}^\lambda\}$ vérifie la propriété de monotonie. Nous pouvons donc définir l'image $\hat{u}_s$ par la formule de reconstruction (83) :

$$\forall s \;\; \hat{u}_s = \min\{\lambda, \hat{u}_s^\lambda = 1\} \;\; .$$

Il est aisé de voir qu'avec ces conditions nous avons :

$$E(\hat{u}|v) = E_\alpha(\{\hat{u}^\lambda\}|v) \;\; ,$$

puisque $\forall s \;\; \alpha H(\hat{u}_s^\lambda - \hat{u}_s^{\lambda+1}) = 0$, et que les autres termes des énergies (2.16) et (2.19) sont identiques. Ceci conclut la preuve. □

Il nous reste maintenant à nous placer dans le cas où les hypothèses de la proposition 2.7 sont toujours vérifiées.

Choix de α pour satisfaire la contrainte

Nous exhibons maintenant une valeur finie pour $\alpha > 0$ telle que la contrainte de monotonie (2.17) soit vérifiée. Nous montrons tout d'abord que nous pouvons borner l'énergie d'un minimiseur de $E(\cdot|v)$. En effet, soit $\overline{0}$ l'image nulle, alors tous les termes de régularisation donnent une valeur et il reste seulement les termes d'attache aux données. Nous avons donc pour tout minimiseur $\hat{u}$ de $E(\cdot, v)$ l'inégalité suivante :

$$E(\hat{u}|v) \leq E(\overline{0}|v) \;\; .$$

Maintenant, nous fixons $\alpha > E(\overline{0}|v)$. Nous montrons qu'avec un tel α, un minimiseur de $E_\alpha(\{\cdot^\lambda\}, v)$ vérifie toujours les contraintes (2.17).

Proposition 2.8 *Soit $\alpha > E(\overline{0}|v)$. Un minimiseur $\{\hat{u}^\lambda\}$ de $E_\alpha(\{\cdot^\lambda\}, v)$ vérifie l'inégalité suivante :*

$$\forall s \, \forall \lambda \;\; u_s^\lambda \leq u_s^{\lambda+1} \;\; .$$

Preuve : En effet, supposons qu'un minimiseur $\{\hat{u}^\lambda\}$ de $E_\alpha(\{\cdot^\lambda\}|v)$ vérifie :

$$\exists s \, \exists \lambda \;\; H_\alpha(\hat{u}_s^\lambda - \hat{u}_s^{\lambda+1}) > 0 \;\; .$$

Puisque tous les termes de l'énergie (2.19) sont positifs, nous avons nécessairement :

$$E_\alpha(\{\hat{u}^\lambda\}|v) > E(\overline{0}|v) \ .$$

Or $\{\hat{u}^\lambda\}$ n'est pas un minimiseur de $E_\alpha(\{\cdot^\lambda\}|v)$ puisque :

$$E_\alpha(\{\overline{0}^\lambda\}|v) = E(\overline{0}|v) < E_\alpha(\{\hat{u}^\lambda\}|v) \ .$$

Ceci conclut la preuve. □

Nous remarquons que nous pourrions ajuster localement la valeur de α pour chaque site s. Nous laissons ce travail pour le futur. En pratique, nous avons observé que α n'a nul besoin d'être à une valeur aussi élevée que celle imposée théoriquement.

2.2.4 Remarques sur le graphe construit

Nous comparons brièvement le graphe construit par la méthode d'Ishikawa (95) et la nôtre. Pour la clarté de la comparaison, nous nous limitons au cas d'un signal en une dimension. La figure 2.2 présente le graphe construit par Ishikawa quand le terme de régularisation est la variation totale. Sa méthode repose sur la représentation unaire des étiquettes. Nous voyons que le graphe construit présente une structure en couches. La relation de voisinage qui permet de décrire les arcs entre les nœuds met en jeu à la fois les voisinages spatiaux et les voisinages liés aux étiquettes. En effet, les voisins d'un nœud n_p^λ (i.e, ce nœud représente le pixel au site p qui prend la valeur λ) sont :

- les nœuds dont le label est identique au sien et qui lui sont voisins spatialement, i.e, $\{n_q^\lambda | p \sim q\}$;
- et ceux qui ont le même site et un label qui vaut $\lambda + 1$ ou $\lambda - 1$, i.e, $\{n_p^\mu | |\mu - \lambda| = 1\}$.

Un voisin tire un arc vers chacun de ses voisins au sens défini ci-dessus. Le nombre de noeuds nécessaires pour la méthode d'Ishikawa s'élève à $|\Omega| \times L$ tandis que le nombre d'arcs nécessaires est de l'ordre de $\theta(|\Omega| \times \Gamma)$, où nous rappelons que Γ est la connectivité utilisée.

Contrairement au graphe d'Ishikawa, le graphe que nous construisons n'est pas du tout en couche, bien au contraire. En revanche il présente tout de même des analogies en termes d'arcs qui connectent les nœuds. Le graphe construit est présenté sur la figure 2.3. Comme pour le graphe d'Ishikawa, un nœud est créé pour chaque valeur u_s^λ que peut prendre un pixel au site s. Chacun de ces nœuds est relié aux nœuds qui lui sont voisins spatialement et qui sont au même niveau λ que lui. Chacun des nœuds au niveau λ est également relié aux nœuds qui ont le même site et qui sont au niveau $\lambda + 1$ ou $\lambda - 1$. En revanche, *tous* les nœuds sont connectés au puits, et la source les relie. Le nombre de nœuds qui doit être créé est le même que pour la méthode d'Ishikawa. L'ordre de grandeur du nombre d'arcs est également le même. En revanche, la longueur du chemin minimal qui relie la source au puit vaut 2. Pour nos mises en œuvre, nous utilisons l'algorithme de coupure minimale de Boykov *et al.* (23). Ce dernier repose sur l'itération de la recherche d'un chemin reliant la source au puits. Comme la longueur de ces chemins vaut 2, il est envisageable que la découverte d'une coupure minimale dans notre graphe soit plus facile, et donc plus rapide, que par la méthode d'Ishikawa. La partie suivante confirme cette conjecture sur quelques exemples.

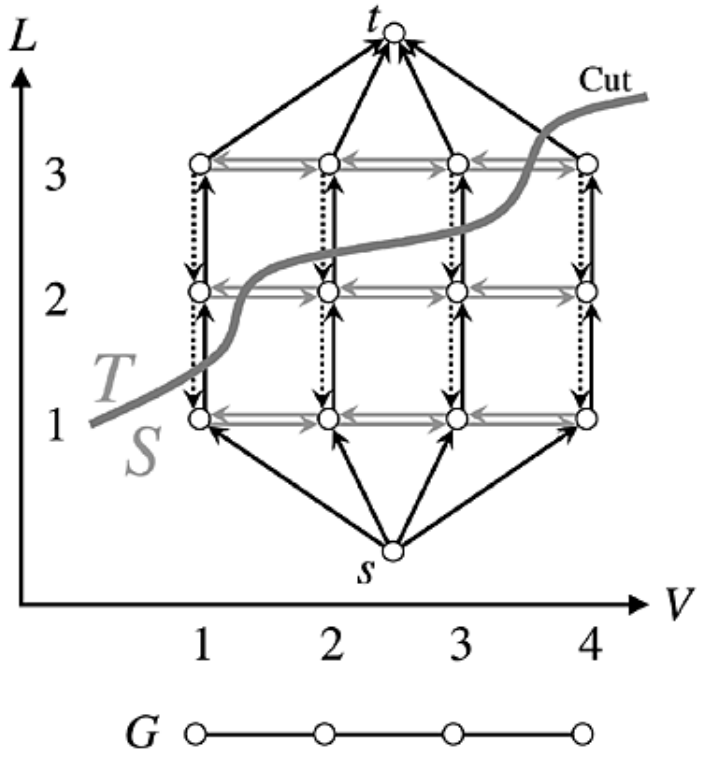


FIG. 2.2 – Graphe construit par la méthode d'Ishikawa pour un signal en une dimension avec un terme de régularisation TV. L'axe V représente les sites des pixels tandis que l'axe L représente les étiquettes. Cette image est tirée de (95) *Exact Optimization for Markov Random Fields with Convex Priors*, H. Ishikawa, IEEE Transaction on Pattern Analysis and Machine Intelligence, 25(10), 2003.

2.3 Application au débruitage

Nous présentons quelques résultats numériques sur des images corrompues par un bruit impulsif. Puis nous restaurons des images radar. Pour ce dernier, nous modélisons l'attache aux données par une loi de Nakagami (91). A chaque fois nous considérons la variation totale comme terme de régularisation.

2.3.1 Le cas du bruit impulsionnel

Nous présentons quelques résultats de débruitage quand l'image est corrompue par un bruit impulsif appliqué à p pour cent des pixels. L'attache aux données f_s aux sites s est donc définie ainsi :

$$f_s(u_s) = f(u_s, v_s) = \begin{cases} -\ln\left((1-p) + \frac{p}{L}\right) & \text{si } u_s = v_s \ , \\ -\ln \frac{p}{L} & \text{sinon} \ . \end{cases}$$

L'utilisation de ce type d'attache aux données implique que l'énergie totale $E(\cdot|v)$ est hautement non convexe. Chaque test a été effectué 30 fois. L'ordinateur utilisé pour effectuer ces tests est un Pentium 4 cadencé à 3Ghz avec 1024 Ko de mémoire cache. La 4-connexité est utilisée pour estimer la variation totale. Le temps de calcul, pour un jeu de paramètres

Image	p impulsif	Nivelé	Ishikawa	Gain
Lena	0,20	114,78	425,14	3,70
Lena	0,40	159,14	633,09	3.98
Lena	0,70	252,67	1203.22	4.76
Girl	0,20	114,09	469,44	4.11
Girl	0,40	171,68	648,72	3.78
Girl	0,70	272,72	1553,66	5.70

TAB. 2.1 – Temps de calcul (en secondes) de la minimisation de la Variation Totale avec une attache aux données correspondant à un bruit impulsif et une régularisation de type TV. Les temps pour notre algorithme et celui d'Ishikawa sont présentés pour des images de taille 256×256. Le gain apporté par notre algorithme (temps nivelé / temps Ishikawa) est également précisé.

Image (sous-quantifiée par 2)	p impulsif	Nivelé	Ishikawa	Gain
Lena	0,20	40,95	84,21	2,06
Lena	0,40	60,14	123,39	2,05
Lena	0,70	86,77	163,61	1,89
Girl	0,20	44,48	89,63	2,01
Girl	0,40	65,42	132,23	2,02
Girl	0,70	95,44	172,23	1,80

TAB. 2.2 – Les images *Lena* et *Girl* sont quantifiées par 2 : autrement dit elles présentent 128 étiquettes au lieu de 256. Temps de calcul (en secondes) de la minimisation de la variation totale avec une attache aux données correspondant à un bruit impulsif et une régularisation de type TV. Les temps pour notre algorithme et celui d'Ishikawa sont présentés pour des images de taille 256×256 . Le gain apporté par notre algorithme (temps nivelé / temps Ishikawa) est également précisé.

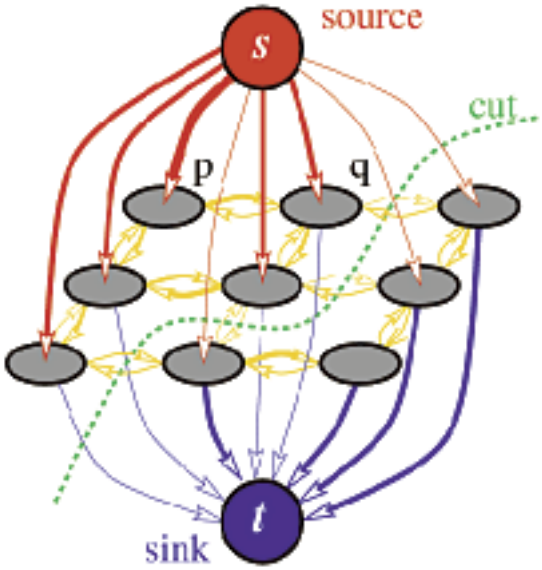


FIG. 2.3 – Graphe construit par la méthode de Kolmogorov *et al.* pour un signal en une dimension avec un terme de régularisation TV et une coupure possible. Cette image est tirée de *Fast Approximate Energy Minimization via Graph Cuts*, Y. Boykov ,O. Veksler and R. Zabih, IEEE Transaction on Pattern Analysis and Machine Intelligence, 23(11), 2001.

donné, est la moyenne de ces 30 temps d'exécutions. Les temps de calculs pour l'algorithme d'Ishikawa et le nôtre sont présentés sur le tableau 2.3.1 pour les images *lena* et *girl*, chacune de taille 256×256. Nous observons que notre graphe procure un gain d'environ 4 par rapport à la méthode d'Ishikawa.

Les images restaurées sont présentées sur les figures 2.4, 2.5 et 2.6 pour des bruits impulsifs de pourcentages respectifs 20%, 40% et 70%. A chaque fois coefficient de régularisation β a été choisi pour donner le meilleur résultat visuel. Les minimiseurs sont présentés sur les figures 2.4, 2.5 et 2.6. Nous observons que certains pixels qui n'ont pas été suffisamment régularisés. Nous vérifions expérimentalement en fait que ces pixels ne changent pas de valeurs par rapport aux images originales. Ces résultats sont à mettre en correspondance avec les observations de Nikolova dans ses études (137; 139) sur la détection des "outliers". A titre de comparaison, nous présentons, sur la figure 2.7, les résultats obtenus par un filtrage médian, dans le cas où 70% des pixels sont touchés par le bruit.

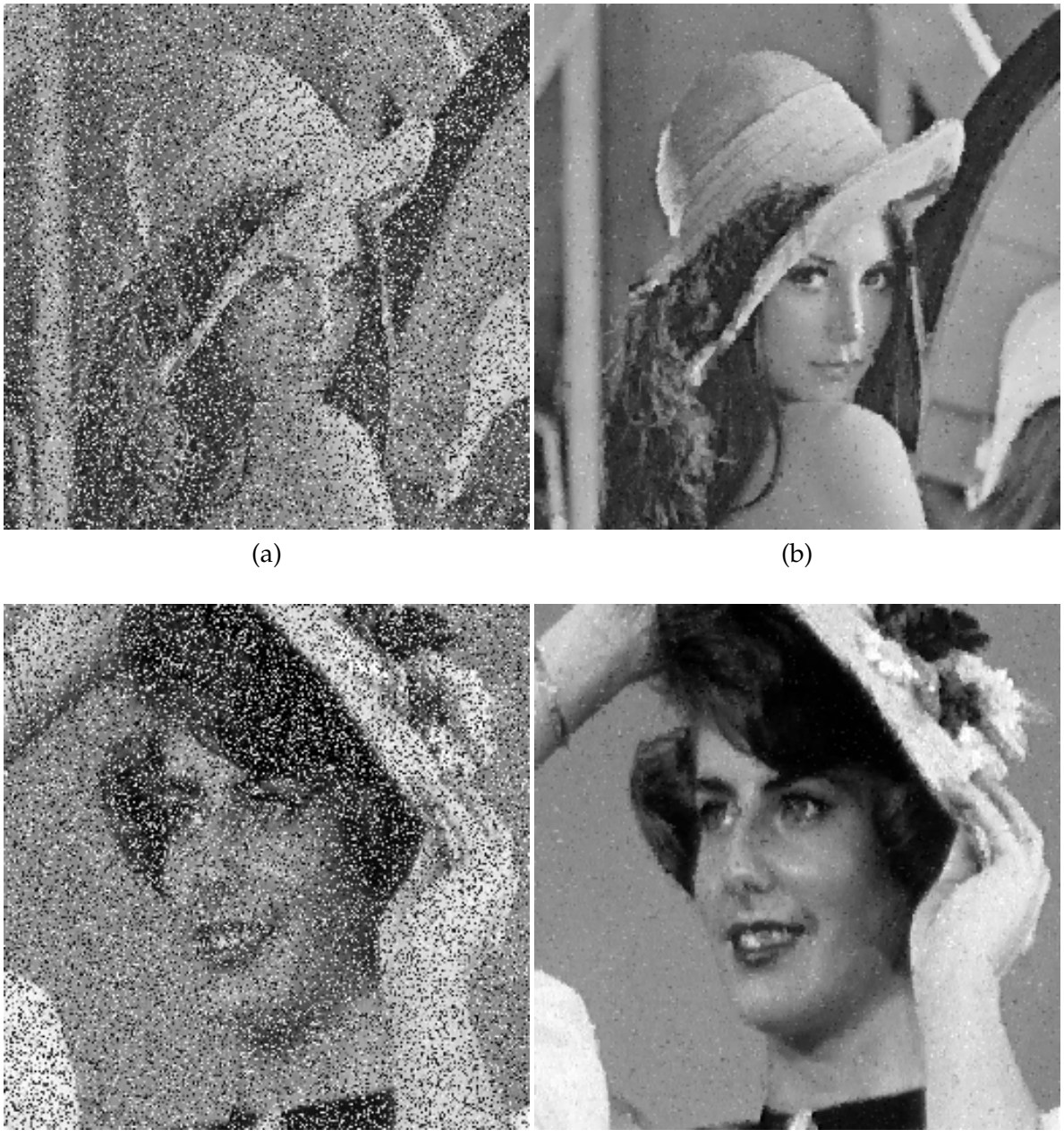
2.3.2 Le cas d'une loi de Nakagami

Nous présentons maintenant quelques résultats sur images corrompues par du bruit de chatoiement ("speckle" en anglais). Nous modélisons le bruit observé sur l'image d'intensité par une loi de Nakagami. Les images observées sont le fruit de M vues avec ici $M = 3$. Le

(a) (b)

(c) (d)

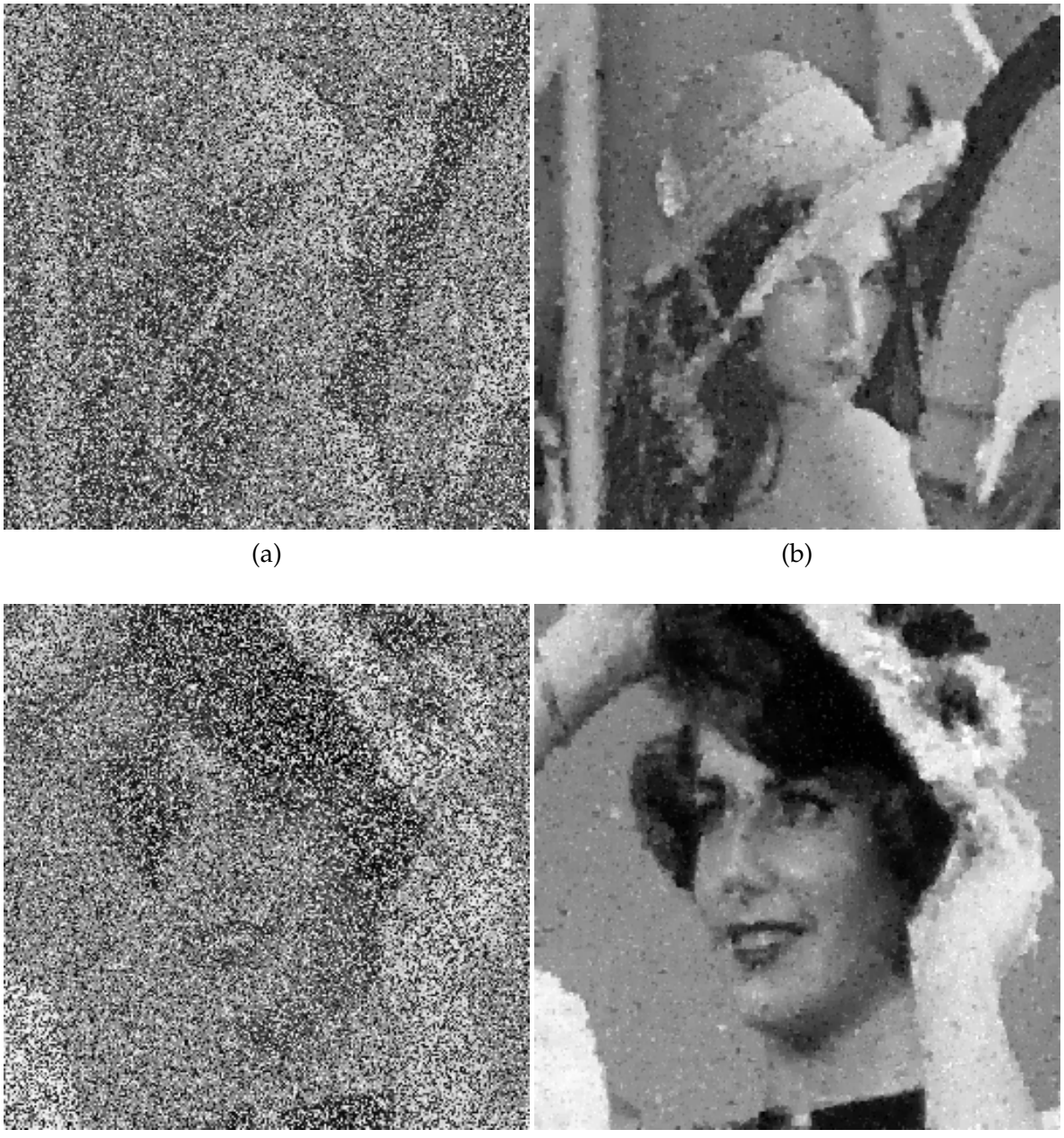
FIG. 2.4 – Restauration de *lena* et *girl* corrompues par un bruit impulsif ($p = 0.2$). Les images corrompues sont présentées en (a) et (c) et images restaurées en (b) et (d). Le coefficient de régularisation est fixé à $\beta = 0,20$.



(a) (b)

(c) (d)

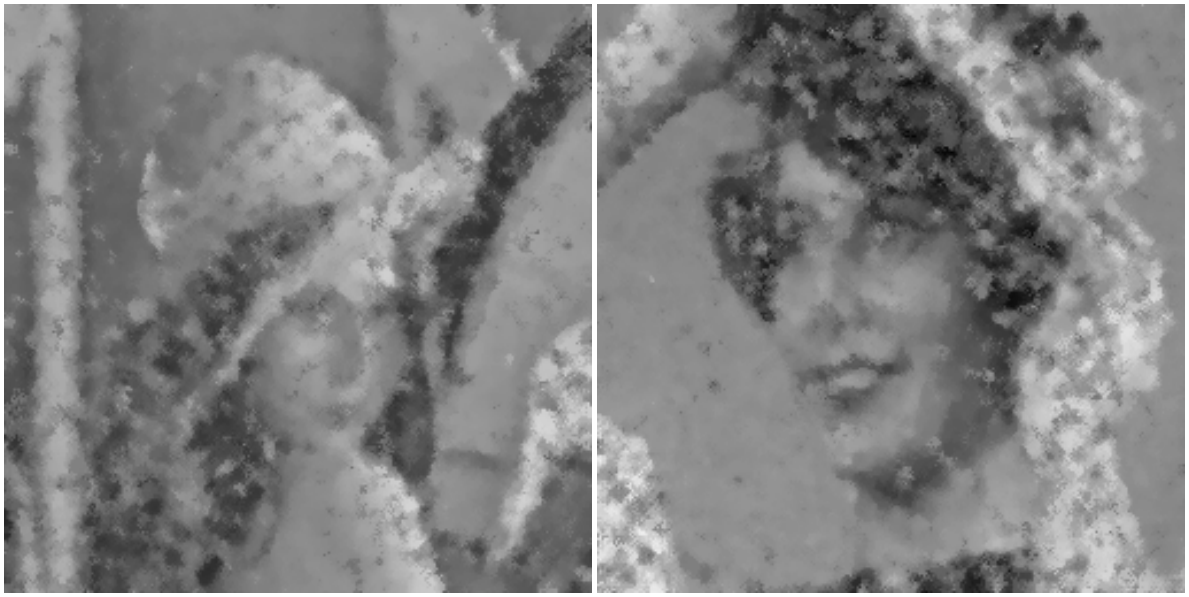
FIG. 2.5 – Restauration de *lena* et *girl* corrompues par un bruit impulsif ($p = 0.4$). Les images corrompues sont présentées en (a) et (c) et les images restaurées en (b) et (d). Le coefficient de régularisation est fixé à $\beta = 0,20$.



(a) (b)

(c) (d)

FIG. 2.6 – Restauration de *lena* et *girl* corrompue par un bruit impulsif ($p = 0.7$). Les images corrompues sont présentées en (a) et (c) et les images restaurées en (b) et (d). Le coefficient de régularisation est fixé à $\beta = 0,40$.



(a) (b)

FIG. 2.7 – Résultat de la restauration de *lena* et de *girl* corrompue par un bruit impulsif ($p = 0.7$) par un filtrage médian. La fenêtre utilisée est un disque de rayon 4. Le résultat pour *lena* est présenté en (a) et en (b) pour *girl*.

termes d'attache aux données prend donc la forme suivante :

$$f(u_s, v_s) = (\frac{{v_s}^2}{u_s} + 2\log u_s) \times M \ .$$

Nous effectuons les premiers tests sur une image synthétique. Cette dernière est une mire qui se compose de quatre carrés de taille différentes, et superposés les uns sur les autres. Elle est présentée en figure 2.8 ainsi que sa version bruitée selon un modèle de loi de Nakagami. Les résultats pour différentes valeurs du coefficient de régularisation β sont également présentés. Afin d'observer la qualité des minimiseurs nous présentons les niveaux de gris sur une ligne. A titre de comparaison la figure 2.9 présente le résultat obtenu par le filtre de Lee (121). Nous observons que pour ce cas d'école, la géométrie de l'image est assez bien reconstruite. La restauration est moins bonne pour les sites de l'image dont les niveaux de gris sont élevés. Ce comportement est attendu puisque le bruit de chatoiement bruite d'autant plus fort les sites dont les valeurs sont élevées. Nous remarquons qu'une perte de contraste est obtenue pour ces niveaux de gris élevés. Ceci est principalement dû au fait que nous utilisons des images codées sur 8 bits : les valeurs bruités qui dépassent 255 sont ramenées à 255. Par conséquent, notre filtre à tendance a sous-estimer les valeurs du signal pour ces grandes valeurs bruitées.

La figure 2.10 présente une image radar réelle. Les résultats 2.11 de la restauration pour des coefficients de régularisation $\beta = 0,02$ et $\beta = 0,05$ sont présentés respectivement sur les figures 2.11 et 2.12. A titre de comparaison, le resulat de la restauration par le filtre de Lee (121) est présenté sur la figure 2.13.

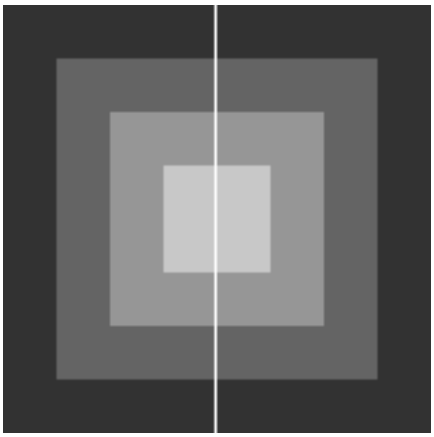
2.4 Étude théorique du modèle $L^1 + TV$

L'utilisation de la variation totale avec une attache aux données L^1 a déjà été étudiée par de nombreux auteurs. Dans (3; 4; 5), Alliney restreint son étude au cas unidimensionnel et discret. Il fournit un algorithme de minimisation à base de filtres médians récursifs. Dans (137; 138; 139), Nikolova étudie des fonctionnelles avec des *a priori* et des attaches aux données non différentiables dans le but de détecter les "outliers". Elle présente de très bons résultats pour le débruitage d'images corrompues par du bruit impulsif ou de type "poivre et sel". Le cas $L^1 + TV$ est un cas particulier des fonctionnelles de son étude. Dans (138), elle remarque que la minimisation de TV avec une attache aux données L^1 produit une image où certains pixels gardent leur niveau de gris original. Dans (37), Chan *et al.* étudient directement le problème en continu. Ils montrent que l'énergie de l'attache aux données est discontinue en fonction des minimiseurs. Dans (38), les auteurs utilisent $L^1 + TV$ afin de minimiser des énergies non convexes. Dans (19), Bouman *et al.* étudient la classe d'énergie définie par les gaussiennes généralisées : le cas $L^1 + TV$ est un cas particulier de leur étude.

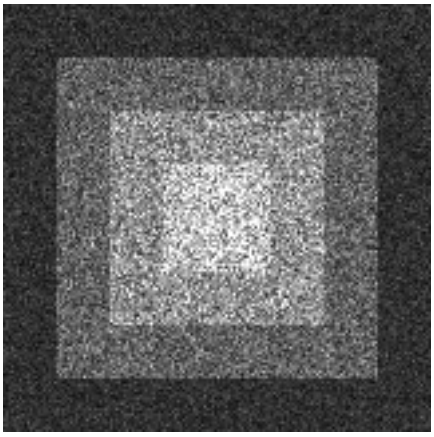
Les résultats théoriques suivants sont originaux à notre connaissance. Tout d'abord, nous exhibons une condition suffisante sur les énergies nivelées telles que leurs minimisations conduisent à un filtre invariant par changement de contraste. Puis nous étudions la non unicité de ces minimiseurs pour le modèle $L^1 + TV$.

2.4.1 Filtres invariants par changement de contraste

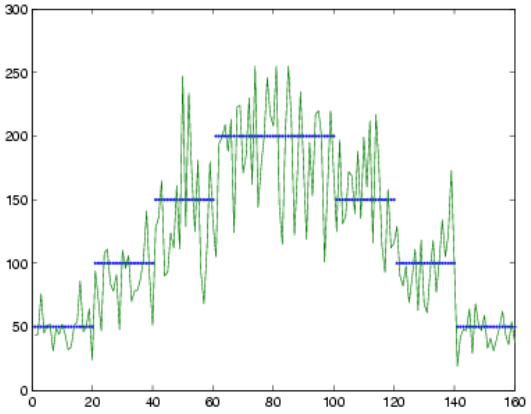
Nous exhibons une classe de champs de Markov dont l'optimisation conduit à des filtres invariants par changement de contraste. Nous définissons tout d'abord la notion de change-



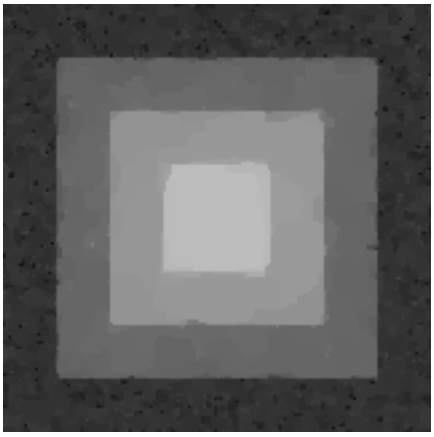
(a)



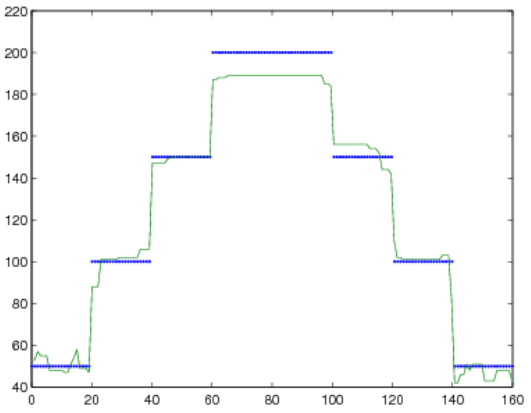
(b)



(c)



(d)



(e)

FIG. 2.8 – L'image de la mire originale (160×160) est présentée en (a) et une version bruité en (b). Le résultat avec une régularisation TV et une attache aux données de type Nakagami est présenté en (d) pour un coefficient de régularisation $\beta = 0.1$. Les images (c) et (e) représentent respectivement les niveaux de gris des images (b) et (d) le long de la ligne verticale indiquée en blanc sur (a). Les niveaux de l'image originale sont également représentés sur ces deux dernières images.

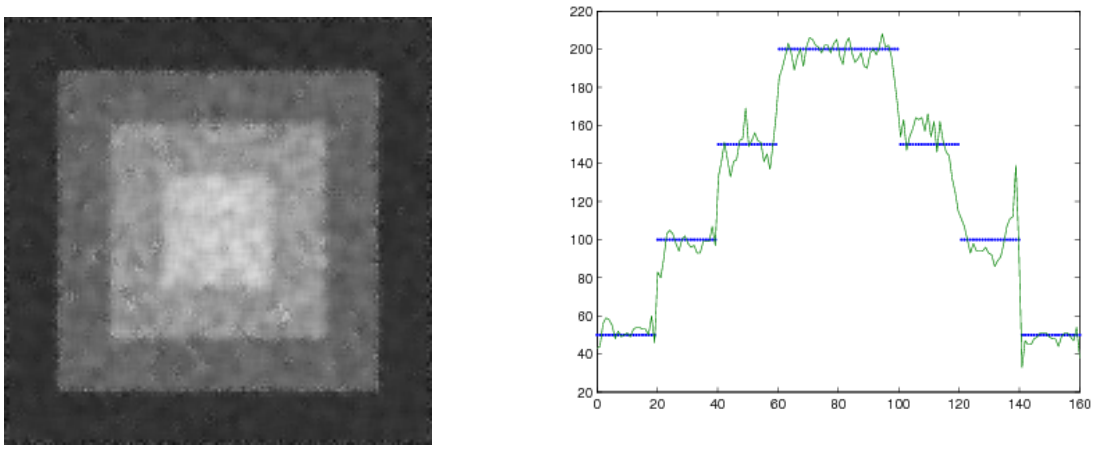


FIG. 2.9 – Débruitage de la mire bruitée présentée sur la figure 2.8-(b) en utilisant le filtre de Lee. L'image restaurée est présentée à gauche tandis que les niveaux de gris du signal le long la ligne verticale sont présentés sur la figure 2.8-(a).

ment de contraste continu (83) :

Définition 2.2 *Une fonction continue et croissante* $h : \mathbb{R} \mapsto \mathbb{R}$ *, est appelée un changement de contraste continu.*

Nous pouvons maintenant introduire la notion de filtre invariant par changement de contraste.

Définition 2.3 *Un filtre* $\mathcal{T}$ *est dit invariant par changement de contraste s'il vérifie la relation suivante :*

$$h(\mathcal{T}(u)) = \mathcal{T}(h(u)) \ ,$$

où u *est une image et* h *un changement de contraste.*

Nous relions maintenant les lignes de niveau de deux images qui diffèrent par un changement de contraste. Pour cela, nous présentons un lemme prouvé dans (82).

Lemme 2.1 *Supposons que* h *soit un changement de contraste continu et que* u *soit une fonction réelle définie sur* Ω*. La proposition suivante est valide :*

$$\forall \lambda \ \exists \mu \ (h(u))^{\lambda} = u^{\mu} \ .$$

En d'autres termes, après avoir appliqué un changement de contraste h sur une image u, les ensembles de niveau de $h(u)$ correspondent à certains ensembles de niveau de l'image u.

Nous rappelons la formulation de l'énergie totale $E(\cdot|v)$ sous forme nivelée donnée par l'équation (2.11) :

$$E(u|v) = \sum_{\lambda=0}^{L-2} \left\{ \sum_{i=1}^{I} \sum_{\substack{(s,t) \\ i(s,t)=i}} R_i(\lambda) \mid u_s^{\lambda} - u_t^{\lambda} \mid + \sum_{s} \delta(\lambda, v_s) \, (1 - u_s^{\lambda}) \right\} + \tilde{C} \ . \tag{2.22}$$

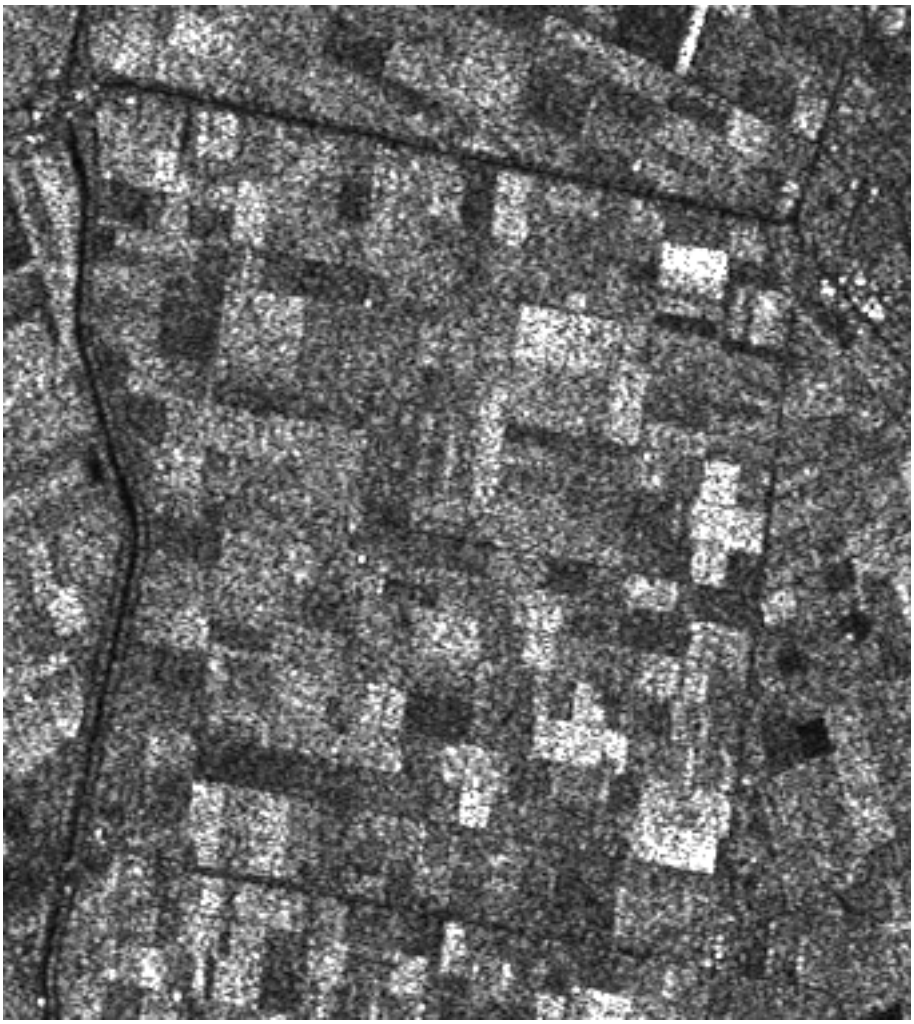


FIG. 2.10 – Une image radar (340 × 480). Région de Wieringermeer. (Flevoland) Copyright : ESA image ERS-1.

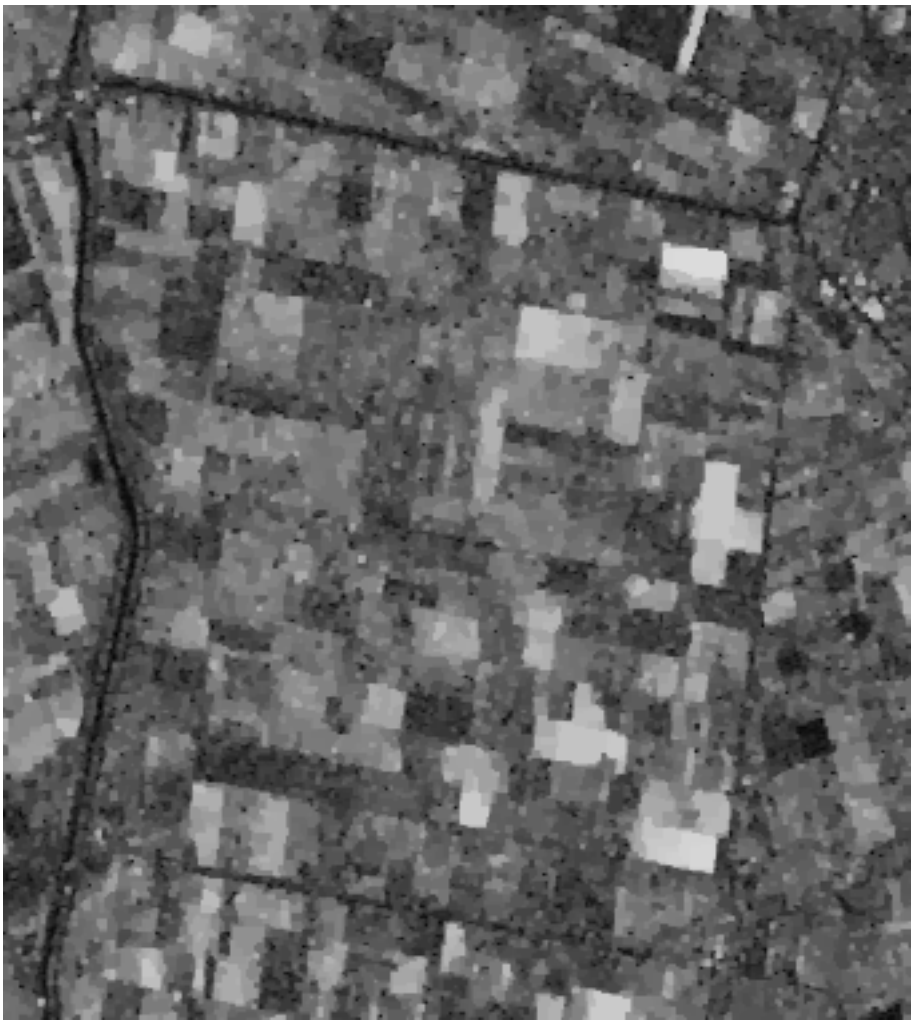


FIG. 2.11 – Résultat de la restauration de l'image radar présenté en figure 2.10. Le modèle utilisé est TV pour la régularisation et une loi de Nakagami pour l'attache aux données. Le coefficient de régularisation est fixé à $\beta = 0,02$.

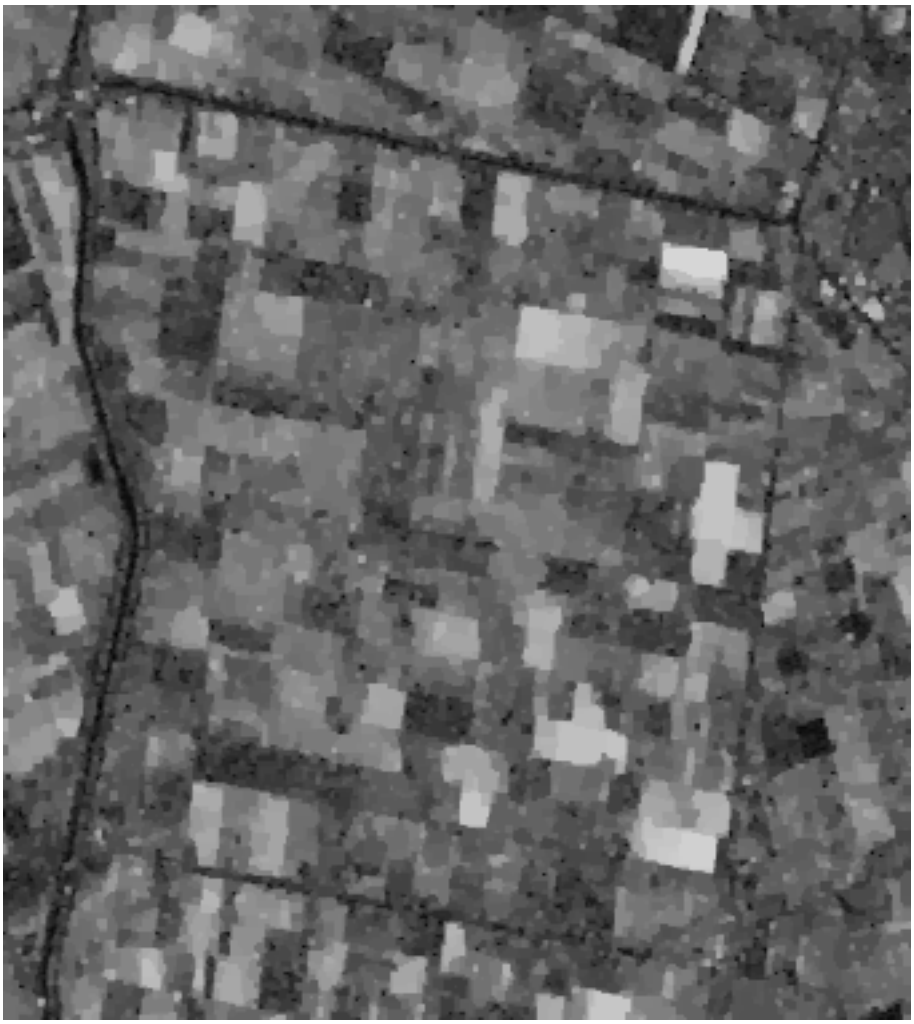


FIG. 2.12 – Résultat de la restauration de l'image radar présenté en figure 2.10. Le modèle utilisé est TV pour la régularisation et une loi de Nakagami pour l'attache aux données. Le coefficient de régularisation est fixé à $\beta = 0,025$.

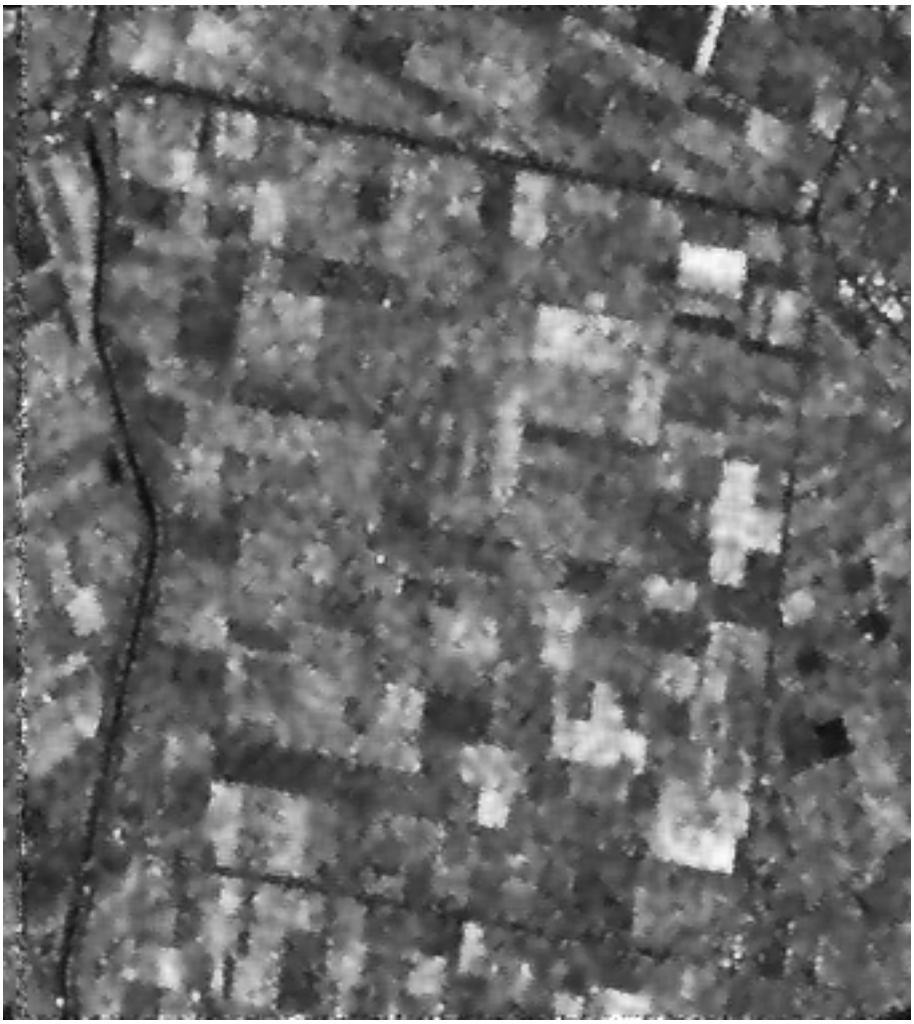


FIG. 2.13 – Résultat de la restauration par le filtrage de Lee de l'image radar présenté en figure 2.10.

Remarquons que la constante $\tilde{C}$ s'écrit $\tilde{C} = \sum_{\lambda=0}^{L-2} \frac{\tilde{C}}{L-1}$. Nous supposons maintenant que

$$\exists C' \ \forall \lambda \ \ R(\lambda) = C' \ , \tag{2.23}$$

et qu'il existe des fonctions k_s associées à chaque site s telles que

$$\forall \lambda \ \ \delta(\lambda, v_s)(1 - u_s^\lambda) = k_s(u_s^\lambda, v_s^\lambda) \ . \tag{2.24}$$

Nous pouvons donc réécrire l'équation (2.22) sous la forme suivante :

$$E(u|v) = \sum_{\lambda=0}^{L-2} E^\lambda(u^\lambda|v^\lambda) \ , \tag{2.25}$$

où

$$E^\lambda(u^\lambda|v^\lambda) = \sum_s \left\{ \sum_{i=1}^{I} \sum_{\substack{(s,t) \\ i(s,t)=i}} C' |\, u_s^\lambda - u_t^\lambda \,| + k_s(u_s^\lambda|v_s^\lambda) \right\} \ .$$

Il est important de remarquer que chaque terme $E_v^\lambda(u^\lambda|v^\lambda)$ implique seulement les images seuillées u^λ et v^λ. En d'autres termes, cette décomposition réécrit l'énergie comme une somme de quantité sur les niveaux de gris.

Nous sommes maintenant prêts à montrer que les filtres définis comme minimiseur d'une énergie qui s'écrit sous la forme (2.25) sont invariants par changement de contraste.

Proposition 2.9 *Supposons que l'énergie $E(\cdot|v)$ s'écrive sous la forme (2.25). Soit v une image observée et h un changement de contraste continu. Supposons que u soit un minimiseur de $E(\cdot|v)$. Alors $h(u)$ est un minimiseur de $E(\cdot|h(v))$.*

Preuve : Il suffit de prouver que pour tout niveau λ, un minimiseur pour l'énergie $E(\cdot|h(v)^\lambda)$ est $h(u)^\lambda$. D'après le lemme 2.1, il existe μ tel que $v^\mu = h(v)^\lambda$. Un minimiseur de $E_v^\mu(\cdot|v^\mu)$ est u^μ. Par conséquent, u^μ est un minimiseur de $E_v^\mu(\cdot|g(v)^\lambda)$. Et nous avons $u^\mu = g(u)^\lambda$. Ceci conclut la preuve. □

Le modèle $L^1 + TV$ est un cas particulier remarquable de modèles qui vérifient les hypothèses de la propriété 2.9. En effet, dans ce cas il est facile de montrer que l'énergie s'écrit ainsi

$$E_{L^1+TV}(u|v) = \sum_{\lambda=0}^{L-2} \left\{ \beta \sum_{i=1}^{I} \sum_{\substack{(s,t) \\ i(s,t)=i}} |\, u_s^\lambda - u_t^\lambda \,| + \sum_s |u_s^\lambda - v_s^\lambda| \right\} \ .$$

Dans (47) la preuve d'invariance par changement de contraste pour le modèle $L^1 + TV$ en continu est établie. L'approche est absolument identique à la preuve de la proposition 2.9 ; Il suffit d'exprimer la variation totale en utilisant la formule de la co-aire, le reste de la preuve étant direct.

Définition 2.4 *Un filtre $\mathcal{T}$ est dit auto-dual s'il vérifie la relation suivante :*

$$\mathcal{T}(u) = -\mathcal{T}(-u) \ ,$$

où u est une image.

Il est trivial de montrer la propriété d'auto-dualité pour ce modèle.

Proposition 2.10 *Soit v une image observée et u un minimiseur de l'énergie $E_{L^1+TV}(\cdot|v)$. Alors $-u$ est un minimiseur de l'énergie $E_{L^1+TV}(\cdot|-v)$.*

Nous étudions maintenant plus en détail le modèle $L^1 + TV$.

2.4.2 Unicité *versus* non unicité des solutions

Nous avons vu que le cas $L^1 + TV$ appartient à la classe des énergies nivelées puisque le coefficient de régularisation β est positif. D'après la formule suivante

$$|u_s - v_s| = \sum_{\lambda=0}^{N-2} |u_s^\lambda - v_s^\lambda| \ ,$$

le terme de l'énergie à un niveau $\lambda \in [0, L-2]$ s'écrit de la manière suivante :

$$E^\lambda(u^\lambda) = \sum_s |u_s^\lambda - v_s^\lambda| + \beta \sum_{(s,t)} w_{st} \, |u_s^\lambda - u_t^\lambda|.$$

Dans le cas isotrope (i.e, $w_{st} = 1$) cela correspond à un modèle *ferromagnétique* d'Ising avec la *même* constante de couplage $J = \beta/2 > 0$ et un champ magnétique d'amplitude $B = 1/2$ (dont le signe local au site s dépend de v_s^λ) sur tous les niveaux.

Le cas spécifique du modèle binaire de l'*échiquier*, i.e, un modèle d'Ising ferromagnétique en 4-connexité où la donnée observée v^λ est un échiquier binaire, présente une propriété de transition de phase (142; 162). Quand la longueur A du coté d'une case carrée de l'échiquier satisfait :

$$A > 4J/B \ \ (= 4\beta) \ ,$$

alors la configuration unique de l'énergie minimale ("ground state" en anglais) est l'échiquier binaire original observé, et ce, peu importe les conditions aux bords. Dans l'autre cas, les deux états périodiques de plus basses énergies correspondent aux cas binaires uniformément blancs et uniformément noirs. Physiquement, cela signifie que tout objet dont le périmètre est plus grand que 4β est parfaitement préservé tandis que les objets qui ne présentent pas cette caractéristique sont perdus dans le "fond".

Pour le cas général des images à niveaux de gris, nous considérons une image d'échiquier avec comme valeur minimale m et valeur maximale M. Les images binaires v^λ qui correspondent aux seuillages de u par λ peuvent prendre trois valeurs :

- $v^\lambda = \overline{0}$ si $0 \leq \lambda < m$,
- $v^\lambda = c$ si $m \leq \lambda < M$,
- $v^\lambda = \overline{1}$ si $M \leq \lambda \leq L-2$,

où c est l'image d'un échiquier binaire définit par $c = \{\mathbb{1}_{v_s=M}\}_{s\in S}$. La minimisation des énergies E^λ pour des niveaux λ en dehors de l'intervalle $[m, M[$ conduit aux solutions suivantes :

$$u^\lambda = \overline{0} \text{ pour } 0 \leq \lambda < m$$

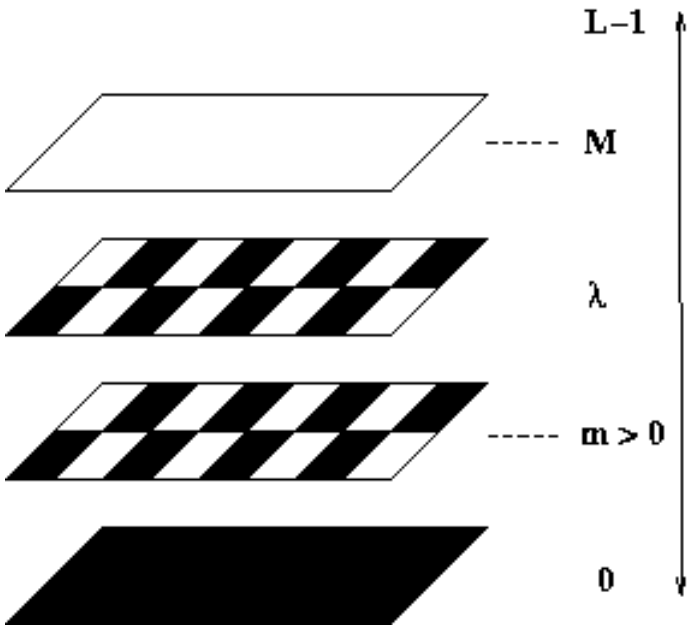


FIG. 2.14 – Configurations niveau par niveau d'énergie minimale pour le modèle de l'échiquier à niveaux de gris.

et

$$u^\lambda = \overline{1} \text{ pour } M \leq \lambda \leq L-2 \ ,$$

puisque la minimisation consiste à restaurer les images binaires observées uniformément noires ou blanches. Autrement dit, aucune *valeur à niveau de gris* en dehors de l'intervalle $[m, M[$ n'est créée.

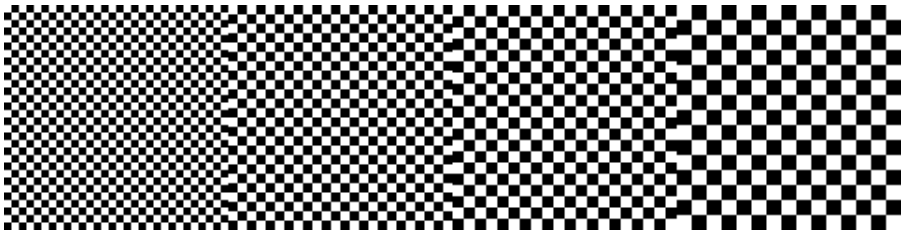
Pour les niveaux intermédiaires ($m \leq \lambda < M$) la même image de l'échiquier binaire $v^\lambda = c$ doit être restaurée en utilisant la même énergie sur les niveaux et en assurant la condition de monotonie sur les u^λ. Pour les niveaux où $A > 4\beta$, alors la solution est l'image binaire originale $u^\lambda = c$ (Fig. 2.14 sous l'hypothèse que $m > 0$). Ainsi, la restauration produit l'image originale.

Quand la taille des cellules n'est pas stationnaire, alors les conditions précédentes ne sont pas valides en général. L'analogie avec une "transition de phase" pour une image à niveau de gris peut être observée en fonction de β et de la taille des cellules ; ce phénomène est présenté sur la figure 2.15. En revanche nous n'avons pas prouvé ce résultat de manière théorique. Nous laissons ce travail pour le futur.

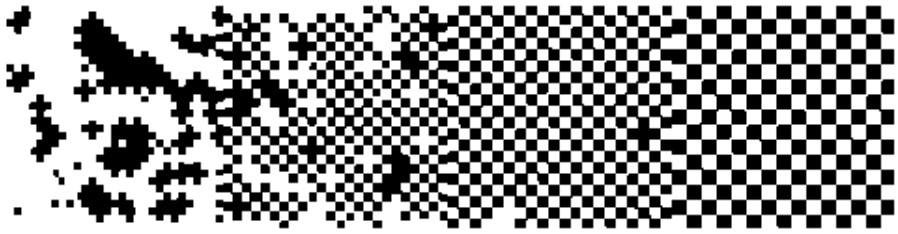
2.5 Conclusion

Dans ce chapitre, nous avons introduit le concept de fonctions nivelées. Il s'agit de l'ensemble des fonctions qui peuvent se décomposer à l'aide d'une somme sur des ensembles de niveaux. Nous avons proposé un algorithme de minimisation exacte pour ces fonctions.

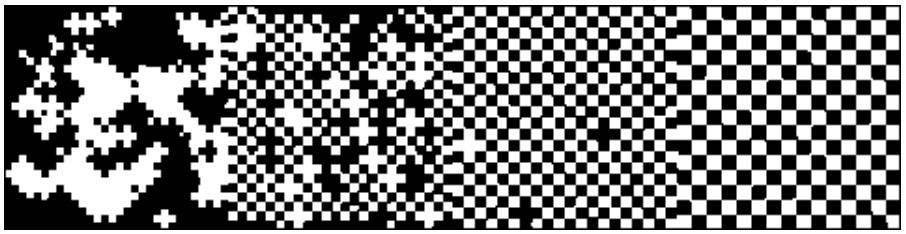
Nous rappelons que le graphe que nous construisons (sous-section 2.2.4) pour effectuer la minimisation, présente des spécificités intéressantes : le chemin de la source à la racine vaut 2. De plus, puisque la propriété de monotonie doit être vérifiée, cela signifie que des contraintes



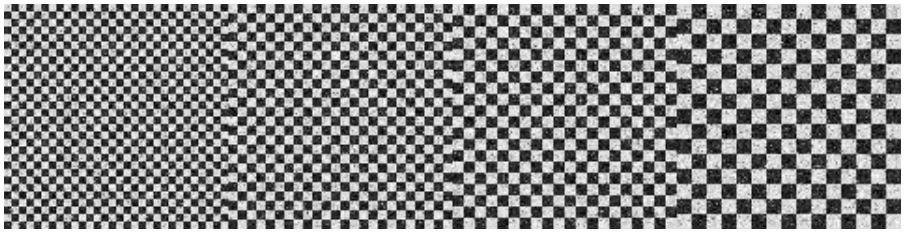
(a) Image binaire originale avec différentes tailles de carré : 4,5, 6 et 8 (de gauche à droite).



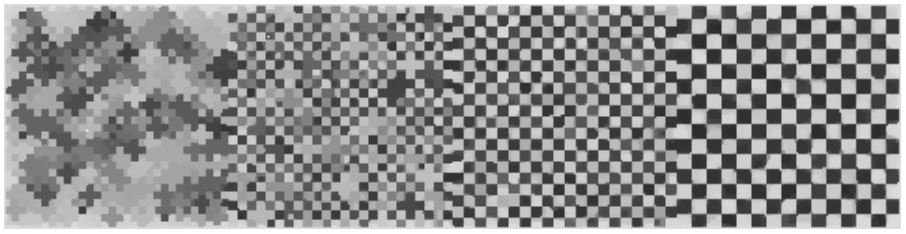
(b) Image binaire restaurée avec des conditions aux bords positives.



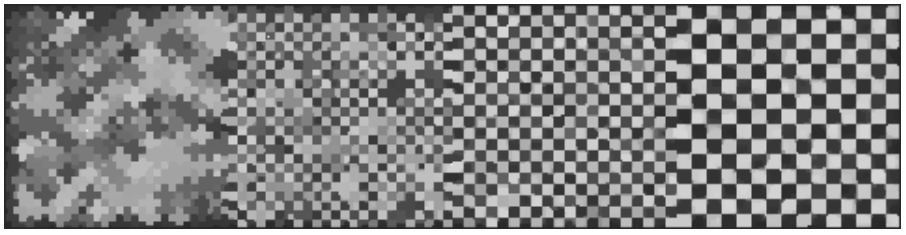
(c) Image binaire restaurée avec des conditions aux bords négatives.



(a′) Image originale avec les mêmes tailles de carré que (a) et les niveaux gris 40 et 220.



(b′) Image restaurée avec des conditions aux bords fixées à 220.



(c′) Image restaurée avec des conditions aux bords fixées à 40.

FIG. 2.15 – Configurations d'énergie minimale obtenue par recuit simulé. Température initiale $T_0 = 16$ avec une décroissance géométrique de 0.98, $\beta = 1.5$ (4-connexité).

existent sur les coupures minimales possibles ; autrement dit, nous pouvons réduire *a priori* l'ensemble des coupures minimales possibles. Nous avons utilisé un algorithme de coupure minimale qui fonctionne dans le cas d'un graphe quelconque. Il serait intéressant de proposer un algorithme de coupure minimale spécifique au graphe que nous avons construit. Nous conjecturons qu'un tel algorithme serait plus rapide que ceux disponibles et qui fonctionnent dans le cas général. Remarquons que des algorithmes de coupures minimales de faible complexité on été présentés dans le cas où le graphe est planaire ou si les arcs ont tous une capacité unitaire (13; 112; 114; 175). Nous avons laissé ce travail pour le futur.

Il est facile de voir que les fonctions nivelées ne contiennent pas les énergies dont le terme de régularisation est quadratique. Dans le chapitre suivant, nous introduisons une nouvelle classe d'énergie, différente des énergies nivelées, qui contient toutes les énergies dont les termes de régularisation sont convexes.

Chapitre 3

Champs de Markov avec des régularisations convexes : propriétés et minimisation exacte

Nous poursuivons l'approche menée dans le chapitre précédent. Cette fois-ci, au lieu de représenter le terme de régularisation avec une seule somme sur les niveaux, nous en autorisons deux, ce qui permet de représenter toutes les fonctions de deux variables à l'aide de variables binaires. Avant de présenter cette approche, nous introduisons quelques définitions et propriétés de fonctions convexes discrètes (132) dans la parite 3.1. Dans la partie 3.2 nous effectuons la décomposition des termes de régularisation avec une double somme. Nous proposons également dans cette partie un algorithme de minimisation exacte quand ces termes de régularisation sont des fonctions convexes. Dans (95), Ishikawa présente un algorithme qui résout le même problème. Comme pour les fonctions nivelées, aucune hypothèse n'est nécessaire sur les termes d'attache aux données. Nous utiliserons un algorithme sensiblement similaire à celui pour les fonctions nivelées (exposé dans le chapitre précédent), et qui repose donc sur l'utilisation de coupures minimales. La construction de notre graphe est différente de celle d'Ishikawa (95). Ensuite, nous supposons dans la partie 3.3 que les termes d'attaches aux données et ceux de régularisation sont également des fonctions convexes. Nous exhibons un algorithme qui fournit un minimiseur global. Cet algorithme est itératif et chaque étape repose encore une fois sur l'utilisation de coupures minimales. Nous proposons également une version rapide de cet algorithme en utilisant des procédés de "scaling". Quelques résultats numériques pour la restauration d'image sont présentés pour les champs de Markov convexes. A notre connaissance ces résultats et algorithmes sont nouveaux.

Nos travaux sur les champs de Markov avec des régularisations convexes et des termes d'attache aux données quelconques sont acceptés à la revue *Journal of Mathematical Imaging and Vision* (58).

3.1 Quelques propriétés des fonctions convexes discrètes

Dans ce chapitre nous présentons quelques résultats simples de convexité discrète. Une fonction discrète unidimensionnelle f est une fonction de support $A \subset \mathbb{Z}$ et qui prend ses

valeurs dans $\mathbb{R}$, i.e, $f : A \subset \mathbb{Z} \to \mathbb{R}$. Nous nous restreignons au cas des fonctions unidimensionnelles discrètes. Elles suffisent pour notre étude car les fonctions discrètes de plusieurs variables que nous étudierons dans la suite de ce chapitre, sont des sommes de fonctions unidimensionnelles. Nous renvoyons le lecteur au livre de Murota (132) pour une étude très complète et poussée des différents types de convexité discrète pour des fonctions à plusieurs variables.

Nous donnons maintenant la définition de convexité pour une fonction discrète en considérant la positivité de la dérivée seconde.

Définition 3.1 *Soit une fonction discrète unidimensionnelle* $f : \mathbb{Z} \mapsto \mathbb{R}$. *Cette fonction est dite* convexe *si et seulement si l'inégalité suivante est vérifiée*

$$\forall x \quad 2f(x) \leq f(x-1) + f(x+1) \ . \tag{3.1}$$

Nous proposons d'autres caractérisations de la convexité d'une fonction discrète unidimensionnelle. Celles-ci seront très utiles par la suite.

Proposition 3.1 *Soit une fonction discrète unidimensionnelle* $f : \mathbb{Z} \mapsto \mathbb{R}$. *Les trois propositions suivantes sont équivalentes :*

a) $\forall x \quad 2f(x) \leq f(x-1) + f(x+1)$.

b) $\forall x \ \forall y > x \quad f(x+1) - f(x) \leq f(y) - f(y-1)$.

c) $\forall x \ \forall y \geq x \ \forall d \geq 0 \ \ d \leq (y-x) \quad f(x) + f(y) \geq f(x+d) + f(y-d)$.

Preuve : Nous traitons les différents cas possibles.

• *Cas a)* $\Rightarrow$ *b)* . Puisque $y > x$ il existe $K > 0$ tel que $y = x + k$. Nous avons donc :

$$\sum_{l=1}^{k-1} 2f(x+l) \leq \sum_{l=1}^{k-1} \left(f(x+l+1) + f(x+l-1)\right) \ . \tag{3.2}$$

Nous séparons en deux la somme de la partie droite, cela donne :

$$\sum_{l=1}^{k-1} 2f(x+l) = \sum_{l=1}^{k-1} f(x+l+1) + \sum_{l=1}^{k-1} f(x+l-1) \ .$$

Nous effectuons le changement de variable $l \leftarrow l+1$ pour la somme de gauche et $l \leftarrow l-1$ sur la somme de droite, on obtient :

$$\sum_{l=1}^{k-1} 2f(x+l) = \sum_{l=2}^{k} f(x+l) + \sum_{l=0}^{k-2} f(x+l) \ .$$

Nous développons les termes précédents, ce qui donne :

$$\sum_{l=2}^{k} f(x+l) + \sum_{l=0}^{k-2} f(x+l) = f(x) + f(x+1) + f(x+k-1) + f(x+k) + \sum_{l=2}^{k-2} 2f(x+l) \ . \tag{3.3}$$

En limitant le parcours des sommes de 2 à $k-2$, on obtient pour la somme de gauche de l'équation (3.2) :

$$\sum_{l=1}^{k-1} 2f(x+l) = 2f(x+1) + 2f(x+k-1) + \sum_{l=2}^{k-2} 2f(x+l) \ ,$$

En injectant cette égalité et celle donnée par (3.3) dans l'inégalité (3.2) on obtient :

$$2f(x+1) + 2f(x+k-1) \leq f(x+k) + f(x+k-1) + f(x) + f(x+1) \ ,$$

ce qui termine la preuve du premier cas.

• *Cas b)* $\Rightarrow$ *c)* . Nous séparons la preuve en deux cas. Le premier correspond au cas où nous avons $x \leq x+d \leq y-d \leq y$, et le second au cas $x \leq y-d \leq x+d \leq y$.

Cas 1 : Supposons que $x+d-1 \leq y-d+1$. Nous avons donc en appliquant *b)*

$$\forall k \in [\![0, d-1]\!] \quad f(x+k+1) - f(x+k) \leq f(y-k) - f(y-k-1) \ .$$

En sommant sur $[\![0, d-1]\!]$, nous obtenons :

$$\sum_{k=0}^{d-1} f(x+k+1) - f(x+k) \leq \sum_{k=0}^{d-1} f(y-k) - f(y-k-1) \ ,$$

ce qui est équivalent à :

$$f(x+d) - f(x) \leq f(y) + f(y-d) \ .$$

Ceci sert de conclusion au premier cas.

Cas 2 : Nous supposons que nous avons $x+d-1 \geq y-d+1$. Nous avons donc $2d \geq y-x-2$, c'est-à-dire $d \geq \frac{y-x}{2} - 1$. Remarquons qu'il existe D tel que $x+D = y-d$. De plus nous avons $D \leq \frac{y-x}{2}$. Nous pouvons donc appliquer *c)* avec $k \in [\![0, D-1]\!]$ puisque dans ce cas $y-k > x+k$:

$$\forall k \in [\![0, D-1]\!] \quad f(x+k+1) - f(x+k) \leq f(y-k) - f(y-k-1) \ .$$

En sommant sur $[\![0, D-1]\!]$ et en simplifiant on obtient :

$$f(x+D) - f(y-D) \leq f(x) + f(y) \ .$$

Puisque $x+D = y-d$ on obtient :

$$f(y-d) - f(x+d) \leq f(y) + f(y-d) \ .$$

Ceci conclut le cas *b)* $\Rightarrow$ *c)* .

• *Cas c)* $\Rightarrow$ *a)* .
Soit $x \in \mathbb{Z}$. D'après *c)*, on a pour $x = y$ et $d = 1$

$$f(x-1) + f(x+1) \geq f(x) + f(x) \ .$$

Ceci conclut la preuve. □

Ces différentes formulations de la convexité d'une fonction sont utilisées dans les deux parties suivantes.

3.2 Champs de Markov avec des *a priori* convexes

Dans ce chapitre nous nous intéressons à minimiser l'énergie suivante :

$$E(u|v) = \sum_s f_s(u_s, v_s) + \sum_{(s,t)} g_{st}(u_s - u_t) \ , \tag{3.4}$$

où v est l'image observée, les termes $\{f_s\}$ sont les attaches aux données, et les termes $\{g_{st}\}$ sont les termes de régularisations. Nous supposons que ces derniers sont des fonctions *convexes* et que $g(0) = 0$.

L'approche que nous suivons est similaire à celles du cas des fonctions nivelées. Notre but est de réécrire l'énergie $E(\cdot|v)$ uniquement en fonction de variables binaires. Nous montrons ensuite que nous pouvons minimiser $E(\cdot|v)$ par une coupure minimale à partir de la nouvelle énergie qui ne fait intervenir que des variables binaires.

3.2.1 Reformulation énergétique

Avant de reformuler l'énergie $E(\cdot|v)$, nous considérons d'abord les termes de régularisation. Nous décomposons une fonction de deux variables à l'aide de deux sommes sur les versions seuillées de ces deux variables. L'approche est maintenant classique. Nous effectuons d'abord la décomposition sur la première variable :

$$\forall k, l \in [\![0, L-1]\!]^2 \quad g(k,l) \ = \ \sum_{\lambda=0}^{L-2} \left(g(\lambda+1, l) - g(\lambda, l)\right) \mathbb{1}_{\lambda<k} \ + \ g(0,l) \ . \tag{3.5}$$

Puis nous effectuons la décomposition sur la seconde variable, nous obtenons :

$$\forall k, l \in [\![0, L-1]\!]^2 \quad g(k,l) \ = \ \sum_{\mu=0}^{L-2}\sum_{\lambda=0}^{L-2} G(\lambda,\mu)\, \mathbb{1}_{\lambda<k}\mathbb{1}_{\mu<l} + \sum_{\lambda=0}^{L-2} \left(g(\lambda+1) - g(\lambda)\right) \mathbb{1}_{\lambda<k} \ + \ g(0,l) \ ,$$

avec

$$G(\lambda,\mu) = g(\lambda+1, \mu+1) - g(\lambda, \mu+1) - g(\lambda+1, \mu) + g(\lambda, \mu) \ . \tag{3.6}$$

En développant encore les termes nous obtenons finalement :

$$\begin{aligned} g(k,l) \ = \ & \sum_{\mu=0}^{L-2}\sum_{\lambda=0}^{L-2} \left\{g(\lambda+1, \mu+1) - g(\lambda, \mu+1) - g(\lambda+1, \mu) + g(\lambda, \mu)\right\} \mathbb{1}_{\lambda<k}\mathbb{1}_{\mu<l} \\ + \ & \sum_{\lambda=0}^{L-2} \left(g(\lambda+1, 0) - g(\lambda, 0)\right) \mathbb{1}_{\lambda<k} \\ + \ & \sum_{\mu=0}^{L-2} \left(g(0, \mu+1) - g(0, \mu)\right) \mathbb{1}_{\mu<l} \\ + \ & g(0,0) \ . \end{aligned} \tag{3.7}$$

Cette décomposition est valable pour toute forme de g. Nous supposons maintenant que la

fonction g s'exprime comme fonction de la différence de ses deux variables (comme supposé dans la forme de l'équation (3.4)), i.e,

$$g : (x, y) \mapsto g(x - y) \ .$$

En injectant cette hypothèse supplémentaire dans la définition de $G(\cdot, \cdot)$ donnée par l'équation 3.6, nous obtenons alors :

$$G(\lambda, \mu) = 2g(\lambda - \mu) - g(\lambda, \mu + 1) - g(\lambda + 1, \mu) \ .$$

En injectant cette dernière équation dans l'équation (3.7), nous obtenons la forme suivante :

$$\begin{aligned} g(k,l) &= \sum_{\mu=0}^{L-2} \sum_{\lambda=0}^{L-2} \{2g(\lambda - \mu) - g(\lambda, \mu + 1) - g(\lambda + 1, \mu)\} \ \mathbb{1}_{\lambda<k} \mathbb{1}_{\mu<l} \\ &+ \sum_{\lambda=0}^{L-2} (g(\lambda + 1) - g(\lambda)) \, \mathbb{1}_{\lambda<k} \\ &+ \sum_{\mu=0}^{L-2} (g(-\mu - 1) - g(-\mu)) \, \mathbb{1}_{\mu<l} \\ &+ g(0) \ . \end{aligned} \tag{3.8}$$

Si de plus nous supposons que g est une fonction convexe, comme énoncé sous l'équation (3.4), alors nous avons :

$$2g(\lambda - \mu) - g(\lambda - \mu - 1) - g(\lambda - \mu + 1) \leq 0 \ . \tag{3.9}$$

Nous réécrivons alors l'énergie définie par l'équation (3.4) en utilisant des variables binaires. Tout d'abord nous rappelons que d'après la proposition 2.1, toute fonction d'une variable est nivelée. Nous obtenons donc la décomposition suivante pour tous les termes d'attache aux données f_s :

$$f_s(u_s, v_s) = \sum_{\lambda=0}^{L-2} \{f_s(\lambda + 1, v_s) - f_s(\lambda, v_s)\} \ \mathbb{1}_{\lambda<k} + f_s(0, v_s) \ . \tag{3.10}$$

Nous pouvons maintenant réécrire l'énergie $E(\cdot|v)$ sur les ensembles de niveaux.

Proposition 3.2 *L'énergie (3.4) s'écrit de la manière suivante :*

$$E(u|v) = \sum_{(s,t)} \left\{ \sum_{\mu=0}^{L-2} \sum_{\lambda=0}^{L-2} E_{st}^{\lambda,\mu}(u_s^\lambda, u_t^\mu) + \sum_{\lambda=0}^{L-2} D_{st}^{+}(u_s^\lambda) + \sum_{\mu=0}^{L-2} D_{st}^{-}(u_t^\mu) \right\} + \sum_s \sum_{\lambda=0}^{L-2} D_s^\lambda(u_s^\lambda|v) + C \ , \tag{3.11}$$

avec :

$$E_{st}^{\lambda,\mu}(u_s^\lambda, u_t^\mu) = R_{st}(\lambda, \mu) \, (1 - u_s^\lambda)(1 - u_t^\mu) \ ,$$

où

$$R_{st}(\lambda, \mu) = 2g_{st}(\lambda - \mu) - \{g_{st}(\lambda - \mu + 1) - g_{st}(\lambda - \mu - 1)\} \ \leq 0 \ ,$$

et

$$D_{st}^{+}(u_s^\lambda) = (g_{st}(\lambda + 1) - g_{st}(\lambda)) \, (1 - u_s^\lambda) \ ,$$

$$D^-_{st}(u^\mu_t) = (g_{st}(-\mu - 1) - g_{st}(-\mu))\,(1 - u^\mu_t) \ ,$$

$$D^\lambda_s(u^\lambda_s|v) = (f_s(\lambda + 1, v_s) - f_s(\lambda, v_s))\,(1 - u^\lambda_s) \ ,$$

et enfin

$$C = \sum_s f_s(0, v_s) \ .$$

Preuve : Le travail est presque abouti en écrivant l'équation (3.8) pour les termes de régularisation. Nous injectons alors cette égalité ainsi que celle donnée pour les termes d'attache aux données (équation (3.10)) dans l'équation (3.4). En outre, en remarquant que $\mathbb{1}_{\lambda<u_s} = (1 - u^\lambda_s)$, nous obtenons le résultat voulu. □

Le fait que les termes de régularisation $\{g_{st}\}$ soient des fonctions convexes est d'une importance capitale pour l'algorithme de minimisation que nous proposons maintenant. En revanche la forme de l'attache aux données n'est pas une contrainte.

3.2.2 Représentation par graphe

L'approche considérée ici est absolument identique à celle proposée pour les énergies nivelées. Nous définissons tout d'abord l'énergie E_α, où $\alpha > 0$ et dont les variables sont binaires et notées $\{u_\lambda\}$:

$$E_\alpha(\{u_\lambda\}|v) = \sum_{(s,t)} \left\{ \sum_{\mu=0}^{L-2} \sum_{\lambda=0}^{L-2} E^{\lambda,\mu}_{st}(u^\lambda_s, u^\mu_t) + \sum_{\lambda=0}^{L-2} D^+_{st}(u^\lambda_s) + \sum_{\mu=0}^{L-2} D^-_{st}(u^\mu_t) \right\} \tag{3.12}$$

$$+ \sum_s \sum_{\lambda=0}^{L-2} D^\lambda_s(u^\lambda_s|v) \tag{3.13}$$

$$+ \sum_s \sum_{\lambda=0}^{L-2} \alpha H(u^\lambda_s - u^{\lambda+1}_s) \ , \tag{3.14}$$

où H est la fonction d'Heaviside (équation (2.18)). Nous voyons que la somme des termes des équations (3.12) et (3.13) sont les termes de l'énergie $E(\cdot|v)$ donnés dans la proposition 3.2 (équation 3.11). Le dernier terme présenté dans l'équation (3.14) sert à satisfaire la propriété de monotonie (équation (1.6)), i.e,

$$\forall \lambda \ u^\lambda_s \leq u^{\lambda+1}_s \ .$$

Proposition 3.3 *Soit $\alpha > 0$. L'énergie $E_\alpha(\cdot|v)$ est représentable par graphe au sens de Kolmogorov et al. (109).*

Preuve : Chacun des termes suivants est représentable par graphe au sens de Kolmogorov *et al.* (109) :

- H : la justification est donnée dans la partie 2.2.
- D^+_{st}, D^-_{st} et D^λ_s sont des fonctions d'une variable binaire et sont donc représentables par graphe.

- $E_{st}^{\lambda,\mu}(u_s^\lambda, u_t^\mu)$ est représentable par graphe car

$$\begin{array}{|c|c|}\hline E_{st}^{\lambda,\mu}(0,0) & E_{st}^{\lambda,\mu}(0,1) \\ \hline E_{st}^{\lambda,\mu}(1,0) & E_{st}^{\lambda,\mu}(1,1) \\ \hline \end{array} = \begin{array}{|c|c|}\hline 0 & 0 \\ \hline 0 & R_{st}(\lambda,\mu) \\ \hline \end{array} .$$

Puisque $\forall\lambda, \forall\mu\ R_{st}(\lambda,\mu) \leq 0$, nous avons bien la condition de régularité de vérifiée, i.e,

$$E_{st}^{\lambda,\mu}(0,0) + E_{st}^{\lambda,\mu}(1,1) \leq E_{st}^{\lambda,\mu}(1,0) + E_{st}^{\lambda,\mu}(0,1) \ .$$

Dans (109), Kolmogorov *et al.* montrent que la somme de fonctions représentables par graphe est également représentable par graphe. C'est le cas de l'énergie $E_\alpha(\{\cdot^\lambda\}|v)$. Ceci conclut la preuve. □

Nous pouvons donc construire un graphe associé à $E_\alpha(\{\cdot^\lambda\}|v)$ tel que sa coupure minimale donne un minimiseur global. Comme pour les énergies nivelées, nous donnons maintenant une valeur finie pour α telle que nous pouvons construire un minimiseur pour $E(\cdot|v)$ à partir d'un minimiseur de $E_\alpha(\{\cdot^\lambda\}|v)$.

Proposition 3.4 *Soit $\overline{0}$ l'image constante où tous ses pixels valent 0. Soit $\alpha > E(\overline{0}|v)$ et $\{\hat{u}^\lambda\}$ un minimiseur global de $E_\alpha(\{\hat{u}^\lambda\}|v)$. Alors $\hat{u}$ défini par :*

$$\forall s\ \hat{u_s} = \min\{\lambda, u_s^\lambda = 1\}$$

est un minimiseur global de $E(u|v)$.

Preuve : La preuve est similaire à celle des propositions 2.7 et 2.8. □

Cette dernière proposition montre donc qu'un champ de Markov avec des termes de régularisations convexes est optimisable exactement par coupure minimale.

3.2.3 Optimisation stochastique

Nous présentons maintenant une propriété de monotonie pour les champs de Markov avec des termes de régularisation convexe. Cette proriété est similaire a celle donnée par la proposition 2.6 pour les énergies nivelées. Contrairement au cas des énergies nivelées, il faut désormais prendre en compte la dépendance avec le niveau de gris des voisins $\{u_t\}_{t\sim s}$, où nous rappelons que $s \sim t$ signifie que les sites s et t sont voisins.

Proposition 3.5 *Une condition nécessaire et suffisante pour que la propriété de monotonie soit respectée est que les énergies conditionnelles $E(u_s|\{u_t\}_{t\sim s}, v_s)$ sont des fonctions convexes.*

Preuve : Pour chaque énergie conditionnelle locale l'énergie peut être décomposée de la manière suivante :

$$\begin{aligned} E(u_s|\{u_t\}_{t\sim s}, \{u_t\}_{t\sim s}, v) &= \sum_{\lambda=0}^{L-2} \left(E(\lambda+1\,|\{u_t\}_{t\sim s}, v_s) - E(\lambda\,|\{u_t\}_{t\sim s}, v_s)\right)(1-u_s^\lambda) + E(0\,|\{u_t\}_{t\sim s}, v_s) \\ &= \sum_{\lambda=0}^{L-2} (\ \Delta\phi_s(\lambda)\ u_s^\lambda\ +\ \chi_s(\lambda)\ ) \\ \text{où } \Delta\phi_s(\lambda) &= -(E(\lambda+1\,|\{u_t\}_{t\sim s}, v_s) - E(\lambda\,|\{u_t\}_{t\sim s}, v_s))\ . \end{aligned} \tag{3.15}$$

La distribution de Gibbs locale à un niveau donné s'écrit de la manière suivante :

$$P(u_s^\lambda = 1 \,|\{u_t\}_{t\sim s}, v_s) = \frac{1}{1+\exp\{\Delta\phi_s(\lambda)\}} \ .$$

Pour utiliser les échantillonneurs de Gibbs "couplés" le lemme 1.1 il faut que

$$\forall \{u_t\}_{t\sim s},\ v_s,\ P(u_s^\lambda = 1 \mid \{u_t\}_{t\sim s},\ v_s) \text{ soit une fonction croissante en } \lambda \ ,$$

et d'après l'équation (3.15), nous avons :

$$\forall \{u_t\}_{t\sim s},\ v_s,\ -\Delta\phi_s(\lambda) = E(\lambda+1\,|\{u_t\}_{t\sim s}, v_s) - E(\lambda\,|\{u_t\}_{t\sim s}, v_s) \text{ qui est croissante en } \lambda \ .$$

Par conséquent, $\forall \{u_t\}_{t\sim s}\ v_s, E(\cdot|\ N_s, v_s)$ doit être une fonction convexe sur $]0, L-1[$. □

Nous présentons dans la partie suivante un algorithme rapide dans le cas où les termes d'attache aux données sont également convexes.

3.3 Champs de Markov convexes

Nous supposons maintenant dans cette partie que les termes d'attache aux données $\{f_s\}$ et les termes de régularisation $\{g_{st}\}$ sont des fonctions convexes. Nous montrons tout d'abord une proposition qui signifie qu'il existe toujours une descente locale qui fait décroître l'énergie tant que nous ne sommes pas sur un minimiseur global. D'autre part, nous montrons que cette descente locale nous rapproche toujours (au sens d'une norme définie dans cette partie) d'un minimiseur global. Nous proposons un algorithme de minimisation exacte et efficace à partir de ces considérations.

3.3.1 Théorème de proximité du minimiseur

Nous montrons tout d'abord un lemme qui nous sera utile par la suite. Il généralise la notion de convexité discrète unidimensionnelle aux champs de Markov convexes. Auparavant nous définissons la notation suivante : $\mathcal{D} = \{-1, 0, 1\}^{|S|}$.

Lemme 3.1 *Soient u et $\hat{u}$ deux images. Soit une image $\delta \in \mathcal{D}$ définie de la manière suivante :*

$$\forall s \in S\ \delta_s = sign(\hat{u}_s - u_s) \ .$$

L'inégalité suivante est vraie :

$$E(u|v) + E(\hat{u}|v) \geq E(u+\delta|v) + E(\hat{u}-\delta|v) \ . \tag{3.16}$$

Preuve : Nous montrons tout d'abord que l'inégalité est vraie pour les termes d'attache aux données f_s puis pour les termes de régularisation.

• *termes d'attache au données.* Si $\delta_s \geq 0$ alors nous avons l'inégalité suivante en appliquant le résultat de la proposition 3.1-(c) à chaque site s :

$$f_s(u_s, v_s) + f_s(\hat{u}_s, v_s) \geq f(u_s + \delta_s, v_s) + f(\hat{u}_s - \delta_s, v_s) \ ,$$

d'où le résultat. Le cas où δ_s est négatif se traite de manière similaire.

• *termes de régularisation.* Soient $X_{st} = u_s - u_t$, $Y_{st} = \hat{u}_s - \hat{u}_t$ et $D_{st} = \delta_s - \delta_t$. Supposons que $X_{st} \leq Y_{st}$. Nous avons alors $X_{st} \leq X_{st} + D_{st} \leq Y_{st}$. En appliquant la proposition 3.1-(c), nous avons alors pour chaque terme de régularisation g_{st} :

$$g_{st}(X_{st}) + g_{st}(Y_{st}) \geq g_{st}(X_{st} + D_{st}) + g_{st}(Y_{st} - D_{st}) \ .$$

Le cas $X_{st} > Y_{st}$ se traite de manière identique. Ceci conclut donc la preuve. □

Nous introduisons maintenant une norme $\|u\|_\infty$ pour une image u de la manière suivante :

$$\|u\|_\infty = \max_{s \in S} |u_s| \ . \tag{3.17}$$

Autrement dit, la norme $\|.\|_\infty$ correspond au maximum des valeurs absolues sur les pixels. Cette norme donne un sens à la notion de voisinage entre deux images. Deux images u et v seront dites voisines l'une de l'autre si et seulement si $\|u - v\|_\infty = 1$.

Nous présentons maintenant un théorème qui précise le lien entre un minimum local (la localité étant définie par le voisinage entre deux images au sens ci-dessus) et un minimum global. Comme pour les fonctions convexes continues (20), cette proposition énonce que si un minimum est local alors c'est un minimum global.

Proposition 3.6 *Soit u une image telle que*

$$E(u|v) > \min_u E(u|v) \ .$$

Il existe $\hat{u}$ un minimiseur global de $E(\cdot|v)$ et $\delta \in \mathcal{D}$ tels que

$$E(u|v) > E(u + \delta|v) \ ,$$

et de plus nous avons

$$\|(u + \delta) - \hat{u}\|_\infty = \|u - \hat{u}\|_\infty - 1 \ .$$

Preuve : Soit $\mathcal{M}$ l'ensemble des minimiseurs de $E(\cdot|v)$. Parmi tous les minimiseurs qui sont les plus proches de u au sens de $\|\cdot\|_\infty$, nous en prenons un, noté $\hat{u}$:

$$\hat{u} \in \operatorname*{argmin}_{\tilde{u} \in \mathcal{M}} \|u - \tilde{u}\|_\infty \ . \tag{3.18}$$

Définissons δ de la manière suivante :

$$\forall s \ \delta_s = sign(\hat{u}_s - u_s) \ .$$

Nous avons bien $\|\delta\|_\infty = 1$ et $\|(u + \delta) - \hat{u}\|_\infty = \|u - \hat{u}\|_\infty - 1$. Nous pouvons donc appliquer le résultat du lemme 3.1 (équation 3.16) avec $u, \hat{u}$ et δ. Nous obtenons

$$E(u|v) - E(u + \delta|v) \geq E(\hat{u} - \delta|v) - E(\hat{u}|v) \ . \tag{3.19}$$

Or $E(\hat{u} - \delta|v) - E(\hat{u}|v) \geq 0$ puisque $\hat{u}$ est un minimiseur global de $E(\cdot|v)$. Nous injectons cette inégalité dans l'inégalité (3.19) et on obtient donc :

$$E(u|v) - E(u + \delta|v) \geq 0 \ .$$

Il nous faut maintenant prouver que cette dernière inégalité est stricte. Supposons nous ayons

$$E(\hat{u} - \delta|v) = E(\hat{u}|v) \ ,$$

alors nous avons une contradiction avec le choix de $\hat{u}$ donné par l'équation (3.18). En effet, nous aurions $(\hat{u} - \delta) \in \mathcal{M}$ et $\|(\hat{u} - \delta) - u\|_\infty = \|\hat{u} - u - \delta\| < \|\hat{u} - u\|_\infty$. Ceci conclut la preuve. □

Cette proposition énonce donc que si une image u n'est pas un minimiseur global, alors il existe une descente locale qui permet de faire décroître l'énergie. D'autre part, il précise également que cette descente conduit à une image dont la distance à un minimiseur global est toujours réduite de 1 (au sens de $\|.\|_\infty$). Remarquons que Murota (132) donne un théorème équivalent au précédent pour des fonctions discrètes bien plus générales que le cas que nous étudions. Dans (14) Bioucas *et al.* montre également qu'un minimiseur local est global, mais ils ne présentent pas de propriété qui montre qu'on se rapproche toujours d'un minimiseur. Dans le contexte des images radars, ces auteurs proposent une méthode pour effectuer le déroulement de phase. Cette méthode repose sur la minimisation d'une énergie convexe.

Cependant, cette dernière proposition ne précise pas le procédé de calcul pour calculer la descente (car la définition de la descente nécessite la connaissance d'un minimiseur). Nous présentons maintenant une proposition qui permettra de trouver la bonne direction de descente. Avant de donner cette proposition nous introduisons l'ensemble des images à une distance au plus n d'une image u noté $I_n(u)$:

$$I_n(u) = \{u' | u_s \leq u'_s\} \ .$$

Nous rappelons également la notation $\mathcal{D} = \{-1, 0, 1\}^{|S|}$ qui correspond à l'espace des descentes locales.

Proposition 3.7 *Soit u une image. Soit $\hat{u}$ un minimiseur global de $E(\cdot|v)$ sur $I_n(u)$. Alors il existe une image $u^\star$ minimiseur global de $E(\cdot|v)$ sur $I_{n+1}(u)$ telle que*

$$\|\hat{u} - u^\star\|_\infty \leq 1 \ .$$

En d'autre termes, cela signifie que $u^\star$ est atteignable à partir de $\hat{u}$ par descente locale, où l'espace des descentes locales est donnée par $\mathcal{D}$.

Preuve : Soit $\mathcal{M}_{n+1}$ l'ensemble des minimiseurs de $E(\cdot|v)$ sur I_{n+1}. Parmi tous ces minimiseurs, nous en prenons un qui soit le plus proche de $\hat{u}$ au sens de $\|\cdot\|_\infty$, et que nous notons $u^\star$:

$$u^\star \in \underset{\tilde{u} \in \mathcal{M}_{n+1}}{\operatorname{argmin}} \|u - \hat{u}\|_\infty \ . \tag{3.20}$$

Pour prouver cette proposition nous raisonnons par l'absurde. Nous supposons donc l'inégalité suivante :

$$\|\hat{u} - u^\star - \| > 1 \ . \tag{3.21}$$

D'après le lemme 3.1 (équation 3.16) appliqué avec $\hat{u}$ et $u^\star$ et δ définit ainsi

$$\forall s \ \ \delta_s = sign(u^\star - \hat{u}) \ ,$$

nous avons :

$$E(\hat{u}) - E(u^\star - \delta) \geq E(\hat{u} + \delta) - E(u^\star) \ .$$

Or $E(\hat{u}+\delta)-E(u^\star) > 0$. En effet, nous avons $E(\hat{u}+\delta)-E(u^\star) \geq 0$ car $u^\star$ est minimiseur global. Si $E(\hat{u}+\delta) = E(u^\star)$ alors cela contredirait l'hypothèse (3.21) puisqu'on aurait $\|u^\star - \hat{u}\|_\infty = \|\hat{u}+\delta-\hat{u}\|_\infty = 1$. Nous avons donc

$$E(\hat{u}) > E(u^\star - \delta) \ .$$

Or $(u^\star - \delta) \in I_n$. En effet,

- Si $u_s^\star = (n+1)$ alors $\delta_s = sg(n+1-u_s) = 1$ puisque $|u_s| \geq n$, et donc $|u_s^\star - delta| \geq n$. Le cas $u_s^\star = -(n+1)$ se traite de manière similaire.
- Si $u_s^\star = n$ alors $\delta_s = sg(n-u_s) \geq 0$ puisque $|u_s| \geq n$, et donc $|u_s^\star - \delta| \geq n$. Le cas $u_s^\star = -n$ se traite de manière similaire.
- Si $|u_s^\star| < n$ alors $|u_s^\star - delta| \geq n$ puisque $|\delta_s| \leq 1$.

Ceci contredit donc le fait que $\hat{u}$ soit un minimiseur global de $E(\cdot|v)$. Par conséquent, l'inégalité donnée par l'équation (3.21) est fausse. Ceci conclut la preuve. □

Récemment, Kolmogorov a prouvé de manière indépendante ce théorème dans (108). En revanche, sa preuve repose sur la sous-modularité du champs de Markov convexe. Notre preuve ne requiert pas cette hypothèse. Cette dernière proposition est à la base de notre algorithme de minimisation. En effet, étant donné une image initiale u, on trouve par descente de plus grande pente un minimiseur global de $E(\cdot|v)$ restreint à $I_1(u)$. En recalculant une descente de plus grande pente à partir de ce minimiseur on obtient un nouveau minimiseur global pour $E(\cdot|v)$ restreint à $I_2(u)$. En itérant ce processus on obtient un minimiseur global de $E(\cdot|v)$. Il nous reste maintenant à savoir calculer une descente de plus grande pente locale. C'est l'objet de la sous-section suivante.

3.3.2 Algorithme de calcul de la plus grande pente locale

Nous montrons maintenant comment calculer un déplacement local qui fait décroître strictement l'énergie. L'algorithme que nous présentons retourne non seulement un déplacement qui fait décroître l'énergie, mais c'est aussi un déplacement qui fait décroître le *"plus possible"* l'énergie. Notre algorithme repose sur des coupures minimales. Récemment une approche similaire a aussi été utilisée, par Bioucas *et al* dans (15), pour effectuer ce calcul. Ici, nous tirons parti de la forme de l'énergie du champ de Markov (équation 3.4). En effet, bien que K. Murota énonce la propriété 3.6 pour des fonctions plus générales que les nôtres, il ne propose pas de calcul efficace pour calculer cette descente.

Nous montrons tout d'abord que le champ de Markov, défini par l'équation (3.4), possède une propriété de représentation par graphe au sens de Kolmogorov *et al.*. Nous présentons ensuite un algorithme original pour calculer la descente locale de plus grande pente. Enfin nous présentons un nouvel algorithme pour minimiser $E(\cdot|v)$.

Représentation par graphe

Étant donnée une valeur d, la propriété suivante énonce que si chaque pixel a le choix entre se déplacer de la quantité d ou rester à sa valeur actuelle, alors nous pouvons calculer un déplacement qui conduit à un état d'énergie minimale. Nous montrons que ce calcul peut s'effectuer par coupure minimale.

Proposition 3.8 *Considérons l'énergie du champ de Markov convexe définie par l'équation (3.4) :*

$$E(u|v) = \sum_s f_s(u_s, v_s) + \sum_{t\sim s} g_{st}(u_s - u_t) \ .$$

Supposons que chaque pixel u_s puisse soit garder sa valeur u_s, soit prendre une nouvelle valeur $u_s + d$. Soit la famille $\{b_s\}$ qui indique pour chaque site si la nouvelle valeur a été prise ou non ($b_s = 1$ si la nouvelle est choisie au site s, et $b_s = 0$ sinon). L'énergie suivante, qui définit un champ de Markov binaire, est représentable par graphe au sens de Kolmogorov et al..

$$E(\{b_s\}|v) = \sum_s f_s(u_s + b_s d, v_s) + \sum_{t\sim s} g_{st}(u_s - u_t + b_s d - b_t d) \ . \tag{3.22}$$

Preuve : L'équation (3.22) définit un modèle d'Ising ferromanétique. Cette propriété est vraie si tous les termes de régularisation sont représentables par graphe au sens de Kolmogorov *et al.*. Nous définissons les termes énergétiques binaires suivants :

$$E^{\delta}_{st}(b_s, b_t) = g_{st}(u_s - u_t + b_s d - b_t d) \ , \tag{3.23}$$

où a et b sont des variables binaires. Chacun de ces termes est représentable par graphe au sens de Kolmogorov *et al.*, i.e,

$$E^{\delta}_{st}(0,0) + E^{\delta}_{st}(1,1) \leq E^{\delta}_{st}(1,0) + E^{\delta}_{st}(0,1) \ .$$

Ceci correspond à montrer que :

$$2g_{st}(u_s - u_t) \leq g_{st}(u_s - u_t - d) + g_{st}(u_s - u_t + d) \ .$$

Cette dernière inégalité est vraie par convexité des $\{g_{st}\}$, et conclut la preuve. □

Autrement dit, étant donné un champ de Markov convexe et un déplacement, nous pouvons calculer l'état d'énergie minimale. Nous montrons ci-dessous comment appliquer ce résultat au calcul de la descente locale de plus grande pente.

Algorithme de calcul de la plus grande pente locale

Avant de présenter notre algorithme de calcul de la plus grande pente nous présentons un lemme qui servira à le prouver. Ce lemme énonce qu'un champs de Markov convexe est une fonction sous-modulaire. Une fonction $f : \mathbb{Z}^n \to \mathbb{R}$ est dite sous-modulaire si et seulement si

$$\forall x, y \in \mathbb{Z}^n \ f(x) + f(y) \geq f(x \wedge y) + f(x \vee y) \ , \tag{3.24}$$

où $(x \wedge y)_s = max(x_s, y_s)$ et $(x \vee y)_s = min(x_s, y_s)$.

Lemme 3.1 *Un champs de Markov convexe est une fonction sous-modulaire.*

Preuve : • Sous-modularité d'un champs de Markov convexe. Rappelons la définition d'un champs de Markov convexe donné par l'équation 3.4 :

$$E(u|v) = \sum_s f_s(u_s, v_s) + \sum_{(s,t)} g_{st}(u_s - u_t) \ .$$

Pour prouver la sous-modularité de E, il suffit de prouver la sous-modularité pour chaque terme d'attache aux données et pour chaque terme de régularisation.
• *termes d'attache aux données.* Immédiat.
• *termes de régularisation.* Considérons deux image u et $\hat{u}$. Il nous faut montrer que :

$$g_{st}(u_s - u_t) + g_{st}(\hat{u}_s - \hat{u}_t) \geq g_{st}((u \wedge \hat{u})_s - (u \wedge \hat{u})_t) + g_{st}((u \vee \hat{u})_s - (u \vee \hat{u})_t) \tag{3.25}$$

Nous montrons que cette dernière égalité est équivalente à la convexité de g_{st}. Nous considérons les différents cas possibles

- Si $(u \vee \hat{u})_s = u_s$ et $(u \vee \hat{u})_t = u_t$, alors $(u \wedge \hat{u})_s = \hat{u}_s$ et $(u \wedge \hat{u})_t = \hat{u}_t$ et l'inégalité 3.25 est trivialement vérifiée.
- Le cas où $(u \vee \hat{u})_s = \hat{u}_s$ et $(u \vee \hat{u})_t = \hat{u}_t$ se traite de la même manière que le précédent.
- Supposons $(u \vee \hat{u})_s = u_s$ et $(u \vee \hat{u})_t = \hat{u}_t$. Il faut donc montrer que :

  $$g_{st}(u_s - u_t) + g_{st}(\hat{u}_s - \hat{u}_t) \geq g_{st}(\hat{u}_s - u_t) + g_{st}(u_s - \hat{u}_t) \ .$$

  En notant $X_{st} = u_s - u_t$, $Y_{st} = \hat{u}_s - \hat{u}_t$ et $\delta = \hat{u}_s - u_s$ l'inégalité précédente est équivalent à :

  $$g_{st}(X_{st}) + g_{st}(Y_{st}) \geq g_{st}(X_{st} + \delta) + g_{st}(Y_{st} - \delta) \ .$$

  En remarquant que nous avons $X_{st} \geq Y_{st}$, et $\delta \leq 0$, nous voyons que cette dernière inégalité correspond à la convexité de g_{st} d'après la proposition 3.1-c).
- Enfin, le cas où $(u \vee \hat{u})_s = \hat{u}_s$ et $(u \vee \hat{u})_t = u_t$ se traite de la même manière que le cas précédent.

Ceci conclut la preuve. □

Notre algorithme de calcul de la descente de plus grande pente repose sur l'application de deux coupures minimales. Nous notons $\overline{1}$ l'image constante qui vaut 1 pour tous les pixels. La proposition suivante présente l'algorithme de descente. Avant de donner la proposition nous définissons $\mathcal{D}^+ = \{0,1\}^{|S|}$ et nous rappelons que $\mathcal{D} = \{-1,0,1\}^{|S|}$.

Proposition 3.9 *Soit u une image. L'algorithme suivant retourne $\delta \in D$ tel que*

$$E(u + \delta|v) = \min_{\delta' \in \mathcal{D}} E(u + \delta') \ .$$

Algorithme :

1. *Soit* $u^- = u - \overline{1}$.
2. *Calculer* $\hat{u}^1$ *tel que*
$$\hat{u}^1 = \min_{\delta \in \mathcal{D}^+} E(u^- + \delta|v) \ .$$
3. *Calculer* $\hat{u}$ *tel que*
$$\hat{u} = \min_{\delta \in \mathcal{D}^+} E(\hat{u}^1 + \delta|v) \ .$$
4. *Retourner* $\hat{u} - u$.

Preuve : Les étapes 2 et 3 sont effectuées par coupure minimale en utilisant le résultat de la proposition 3.8.

Considérons l'énergie $\tilde{E}(\cdot|v)$ qui est la restriction de l'énergie $E(\cdot|v)$ aux images u' telles que $\|u' - u\|_\infty \leq 1$. Nous avons $\|\hat{u} - u\|_\infty \leq 1$ par définition des étapes 1,2 et 3. Pour prouver

l'algorithme, il suffit de prouver que $\hat{u}$ est un minimiseur global de $\tilde{E}(\cdot|v)$. Nous notons $\tilde{\mathcal{M}}$ l'ensemble des minimiseurs de $\tilde{E}(\cdot|v)$. Par définition de u^-, nous avons

$$\min_{\tilde{u}\in\tilde{\mathcal{M}}} \|u^- - \tilde{u}\|_\infty \leq 2 \ .$$

Si $\exists u' \in \mathcal{M}$, $|u^- - u'|_\infty \leq 1$ alors, ce minimiseur est trouvé par l'étape 2 (car c'est une image qui produit une plus grande pente), et $\hat{u}$ est donc bien un minimiseur global de $\tilde{E}(\cdot|v)$.

Il nous reste à traiter le cas où $\forall u' \in \mathcal{M}$, $\|u^- - u'\|_\infty = 2$. D'après le lemme 3.1, $\tilde{E}$ est une fonction sous-modulaire, nous avons donc :

$$\forall\ u' \in \mathcal{M},\ \ \tilde{E}(u') + \tilde{E}(\hat{u}^1) \geq \tilde{E}(u' \vee \hat{u}^1) + \tilde{E}(u' \wedge \hat{u}^1) \ .$$

Nous avons donc :

$$\forall\ u' \in \mathcal{M},\ \ \tilde{E}(\hat{u}^1) - \tilde{E}(u' \wedge \hat{u}^1) \geq \tilde{E}(u' \vee \hat{u}^1) - \tilde{E}(u') \ .$$

Remarquons que nous avons $\|(u' \vee \hat{u}^1) - u^-\|_\infty \leq 1$, et $\tilde{E}(u' \vee \hat{u}^1) - \tilde{E}(u') \leq 0$. Si $\tilde{E}(u' \vee \hat{u}^1) - \tilde{E}(u') > 0$, alors $\tilde{E}(\hat{u}^1) - \tilde{E}(u' \wedge \hat{u}^1) > 0$. Cette dernière inégalité contredit le fait que $\hat{u}^1$ soit une descente de plus grande pente depuis u. Nous avons donc, $\tilde{E}(u' \vee \hat{u}^1) = \tilde{E}(u')$. Autrement dit $(u' \vee \hat{u}^1)$ est un minimiseur global. Et $(u' \vee \hat{u}^1)$ est bien atteignable depuis $\hat{u}^1$ en appliquant l'étape 2. Donc $\hat{u}$ est bien un minimiseur global. Ceci conclut la preuve. □

L'algorithme ci-dessus requiert deux coupures minimales sur un champ de Markov binaire pour calculer une descente locale de plus grande pente pour l'énergie $E(\cdot|v)$. Il nous faut maintenant montrer que cette descente locale de plus grande pente se rapproche bien d'un minimiseur global.

Nous pouvons maintenant présenter notre premier algorithme de minimisation qui utilise ces derniers résultats.

3.3.3 Premier algorithme de minimisation

Notre premier algorithme est une application directe des résultats théoriques précédents. Il consiste à effectuer localement des descentes locales de plus grande pente jusqu'à convergence.

Proposition 3.10 *Soit $u^{(0)}$ une image. L'algorithme suivant fournit un minimiseur global de l'énergie $E(\cdot|v)$:*

1. $n = 0$.
2. *Calculer*
$$u^{(n+1)} = \underset{\delta\in\mathcal{D}}{argmin}\ E(u^{(n)} + \delta|v) \ .$$
3. *Si $u^{(n+1)} = u^{(n)}$, alors retourner $u^{(n+1)}$, sinon $n \leftarrow n+1$ et retourner à l'étape 2 .*

De plus, le nombre d'étapes à effectuer est $\|\hat{u} - u^{(0)}\|_\infty$.

Rappelons que nous considérons que les images sont à valeurs dans l'ensemble discret $[\![0, L-1]\!]$. Nous initialisons l'image de départ $u^{(0)}$ à $\overline{\lceil \frac{L}{2} \rceil}$, i.e, tous les pixels valent $\lceil \frac{L}{2} \rceil$, où $\lceil \cdot \rceil$ est le plus proche entier supérieur. Avec une telle initialisation, il est aisé de voir que cet algorithme retourne le résultat en $\lceil \frac{L}{2} \rceil$ itérations dans le pire cas. Nous proposons maintenant une amélioration de cet algorithme qui réduit la complexité dans le pire cas initialement linéaire en complexité logarithmique dans tous les cas.

3.3.4 Deuxième algorithme : amélioration des performances par un changement d'échelle

Nous montrons comment nous pouvons réduire drastiquement le nombre de coupures minimales à effectuer en utilisant un procédé de changement d'échelle. Tout d'abord nous introduisons la notion de changement d'échelle d'une fonction. Ensuite nous montrons comment minimiser l'énergie $E(\cdot|v)$ à partir de ces versions dilatées. Pour cela nous démontrons une proposition analogue à la proposition 3.6 (proximité du minimiseur) pour les énergies soumises à un changement d'échelle.

Nous introduisons la notion de changement d'échelle pour une fonction. C'est un concept utilisé notamment par la théorie des ondelettes (118).

Definition 3.2 *Soient f une fonction $f : \mathbb{R} \mapsto \mathbb{R}$ et n un entier positif. La version de la fonction f soumise à un changement d'échelle par n, notée f^n est définie ainsi :*

$$f^n(x) = f(nx) \ .$$

Nous montrons que l'ensemble des fonctions unidimensionnelles et convexes sont fermées par l'opération de changement d'échelle.

Proposition 3.11 *Soit f une fonction unidimensionnelle discrète et convexe $f : \mathbb{Z} \mapsto \mathbb{R}$. Le changement d'échelle de f par un entier positif n est également convexe.*

Preuve : Il suffit de considérer l'inégalité de la propriété 3.1-c) pour $x - n$, $x + n$ et $d = n$. Ceci conclut la preuve. □

Nous introduisons maintenant l'énergie soumise à un changement d'échelle. Soit un entier n positif et l'énergie $E^n(\cdot|v)$ définit de la manière suivante :

$$E^n(u^n|v) = \sum_s f_s^n(u_s^n, v_s) + g_{st}^n(u_s^n - u_t^n) \ , \tag{3.26}$$

où $f_s^n(u_s^n, v_s) = f_s(nu_s, v_s)$ et $g_{st}^n(u_s^n - u_t^n) = g_{st}(nu_s - nu_t)$. Comme les termes d'attache aux données $f_s(\cdot, v_s)$ et les termes de régularisation g_{st} sont des fonctions unidimensionnelles discrètes et convexes, leurs versions *dilatées* le sont également. L'équation (3.26) est donc aussi un champ de Markov *convexe*. Par extension de la notation définie dans la définition 3.2, nous avons l'égalité suivante $E^n(u|v) = E(nu|v)$. Nous pouvons donc appliquer les résultats des propositions 3.9 et 3.10 pour l'énergie $E^n(\cdot|v)$.

Pour cela nous proposons un lemme similaire à celui du lemme 3.1.

Lemme 3.2 *Soit d un entier positif. Soient u et $\hat{u}$ deux images. Soit une image $\delta \in \{-d, 0, d\}^{|S|}$ définie de la manière suivante :*

$$\forall s \in S \ \delta_s = d * sign([\frac{\hat{u}_s - u_s}{d}]) \ ,$$

où $[\cdot]$ désigne de l'entier le plus proche. L'inégalité suivante est vraie :

$$E(u|v) + E(\hat{u}|v) \geq E(u + \delta|v) + E(\hat{u} - \delta|v) \ . \tag{3.27}$$

Preuve : La preuve est une simple adaptation de celle du lemme 3.1. □

Nous montrons maintenant la propriété fondamentale de proximité pour les énergies soumises à un changement d'échelle qui est à la base de notre algorithme rapide.

Proposition 3.12 *Soit $\hat{u}^n$ un minimiseur global de $E^n(\cdot|v)$. Alors il existe un minimiseur global $\hat{u}$ de $E(\cdot|v)$ tel que :*

$$\|n\hat{u}^n - \hat{u}\|_\infty < n \ .$$

Preuve : Soit $\mathcal{M}$ l'ensemble des minimiseurs de $E(\cdot|v)$. Parmi tous les minimiseurs qui sont les plus prôches de nu^n au sens de $\|\cdot\|_\infty$, nous en prenons un, noté $\hat{u}$:

$$\hat{u} = \underset{\tilde{u} \in \mathcal{M}}{\operatorname{argmin}} \|nu^n - \tilde{u}\|_\infty \ . \tag{3.28}$$

Il existe $D \in \{-1, 0, 1\}^{|S|}$ et δ tels que

$$\hat{u} = \hat{u}^n + nD + \delta \ ,$$

avec $\forall s \ |\delta_s| < n$. On applique le lemme 3.2 avec $n\hat{u}^n$, $\hat{u}$ et nD, on obtient alors (équation (3.27)) :

$$E(nu^n|v) + E(\hat{u}|v) \geq E(nu^n + nD|v) + E(\hat{u} - nD|v) \ .$$

Ce qui est équivalent à :

$$E^n(u^n|v) + E(\hat{u}|v) \geq E^n(u^n + D|v) + E(\hat{u} - nD|v) \ .$$

On obtient donc :

$$E^n(u^n|v) - E^n(u^n + D|v) \geq E(\hat{u} - nD|v) - E(\hat{u}|v) \ .$$

Le terme de droite est positif car $\hat{u}$ est un minimiseur global. Il est même strictement positif car s'il valait 0, alors nous aurions une contradiction avec le choix de $\hat{u}$: l'égalité (3.28) serait violée. Ceci conclut la preuve. □

Malheureusement, cette dernière proposition ne donne aucune information sur la proximité de deux minimiseurs à des échelles différentes. De telles bornes sont par exemple données par *Hochbaum et al.* dans (89), mais elles sont inutilisables en pratique pour le traitement des images car elles font intervenir le nombre de pixels et ce, de manière linéaire.

Malgrès cette dernière considération, nous pouvons émettre l'heuristique suivante : la distance entre deux minimiseur à deux échelles proches n'est pas trop éloigné. L'idée de notre deuxième algorithme consiste à minimiser localement une version dilatée de l'énergie $E(\cdot|v)$ par un entier positif n. Nous supposons que n est de la forme 2^k pour simplifier la suite de l'exposé. L'adaptation à un entier quelconque ne pose aucune difficulté. Une fois que nous

avons trouvé un tel minimiseur, nous minimisons la nouvelle énergie $E^{\frac{n}{2}}(\cdot|v)$ en partant de ce minimiseur.Nous itérons ce processus jusqu'à ce que n atteigne 1. Nous obtenons alors un minimiseur global.

Proposition 3.13 *Soient $u^{(0)}$ une image qui prend ses valeurs dans l'ensemble discret $[\![0, L-1]\!]$ et n un entier positif de la forme $n = 2^k$. L'algorithme suivant fournit un minimiseur global de l'énergie $E(\cdot|v)$:*

1. *Soit $u^{(k-1)} = \overline{\frac{L}{2}}$, i.e, $\forall s \;\; u_s = \frac{L}{2}$*
2. *Pour $l \leftarrow k-1$ à 1*
   Calculer
   $$u^{(l-1)} = \underset{\|\delta\|_\infty \leq 1}{argmin}\; E(u^{(l)} + l \times \delta|v) \;\; .$$
3. *Retourner $u^{(0)}$.*

Si l'heuristique émise se révèle vraie, nous pouvons espérer que cet algorithme requiert $\log_2 L$ itérations contrairement au premier qui en réclame $\frac{L}{2}$. Nous le montrons par l'expérience, dans la sous-section suivante, que cette heuristique se vérifie en pratique pour les modèles utilisé en traitement des images.

3.3.5 Expériences

Nous présentons brièvement quelques résultats numériques pour la restauration d'images avec des modèles à régularisation convexes et champs de Markov convexes. Nous avons choisi un modèle de régularisation proposé par Nikolova dans (139). Il s'agit de remplacer la variation totale par la régularisation suivante $|\nabla \cdot|^{1+\epsilon}$, où ϵ est une valeur proche de zéro. Pour nos expériences nous fixons $\epsilon = 0,2$. A la différence de la variation totale, le terme de régularisation est désormais différentiable en zéro, et implique que l'effet d'escalier est moins visible. Ce phénomène est observé en pratique.

Nous présentons des meilleurs résultats visuels de restaurations de l'image *girl* corrompue par un bruit gaussien additif sur la figure 3.1. Le coefficient de régularisation β a été choisi pour donner visuellement le meilleur résultat. Nous observons que l'effet escalier est bien *moins* présent que pour les minimiseurs du modèle de la variation totale (voir figure 1.7), et que le résultat est de meilleure qualité visuelle.

Les temps de calculs pour différentes valeurs du coefficient de régularisation sont présentés sur le tableau 3.1 pour des images de taille 256×256 et sur le tableau 3.2 pour des images de taille 512×512. Les temps correspondent à la moyenne de 20 lancements du programme. Les tests sont réalisés sur un Pentium 4 cadencé à 3GHz avec 1Mo de mémoire cache. L'algorithme présente le même comportement que celui de la minimisation de la variation totale. Nous observons une dépendance avec la force de régularisation. Plus le coefficient est élevé, plus les temps de calculs sont grands. Nous voyons que le contenu des images influence très peu les temps de calculs. Il faut noter que notre mise en œuvre précalcule les termes de régularisations pour les différentes entrées possibles, et ce, afin d'éviter à chaque fois les appels de fonctions pour calculer une exponentielle. Les résultats montrent que la dépendance des algorithmes en fonction du nombre de pixels est quasiment linéaire. Le ratio théorique que nous devrions obtenir pour l'algorithme à base de changement d'échelle, *avec* la condition que l'heuristique de proximité des minimiseurs à des échelles proches est vraie, par rapport

(a) (b) $\beta = 20$

(c) (d) $\beta = 30$

FIG. 3.1 – Les images de *girl* corrompues par un bruit gaussien additif d'écart-type $\sigma = 12$ et $\sigma = 20$ sont présentées sur (a) et (d). Les minimiseurs obtenus pour le modèle de type $L^2 + |\nabla \cdot|^{1,2}$ sont respectivement montrés en (b) et (b) avec $\beta = 15$ et $\beta = 30$.

à la version à un pas, s'élève à $\frac{L}{log_2 L}$. Pour nos expériences, $L = 256$ et le ratio vaut donc 32. En pratique, il s'élève aux alentours de 15 ce qui représente déjà une amélioration considérable.

Tab. 3.1 – Temps de calcul (en secondes) de nos deux algorithmes pour champ de Markov convexe avec différentes valeurs du coefficient de régularisation β. Le modèle utilisé est $L^2 + |\nabla .|^{1,2}$. Toutes les images ont une taille de 256×256. La présence de (1) à côté du nom de l'image précise que le premier algorithme est utilisé, tandis que son absence signifie que la version par *dilatation* est utilisée.

Image	$\beta = 5$	$\beta = 10$	$\beta = 20$	$\beta = 30$	$\beta = 40$	$\beta = 50$
girl	1,61	1,86	2,26	2,57	2,82	2,98
girl(1)	24,86	29,32	35,12	39,76	42,62	45,34
lena	1,62	1,88	2,24	2,49	2,71	2,90
lena(1)	23,36	30,77	36,34	40,36	43,29	45,74
barbara	1,58	1,80	2,12	2,38	2,58	2,75
barbara(1)	24,66	28,20	33,49	37,60	40,48	42,78

Tab. 3.2 – Temps de calcul (en secondes) de nos deux algorithmes pour champ de Markov convexe avec différentes valeurs du coefficient de régularisation β. Le modèle utilisé est $L^2 + |\nabla .|^{1,2}$. Toutes les images ont une taille de 512×512. La présence de (1) à côté du nom de l'image précise que le premier algorithme est utilisé, tandis que son absence signifie que la version par *dilatation* est utilisée.

Image	$\beta = 5$	$\beta = 10$	$\beta = 20$	$\beta = 30$	$\beta = 40$	$\beta = 50$
lena	6,22	7,34	8,94	10,14	11,21	12,19
lena(1)	101,59	121,00	145,21	163,01	177,0	189,67
aérien	5,93	6,84	8,10	9,02	9,77	10,46
aérien(1)	92,12	107,03	126,51	140,11	150,81	159,67
barbara	6,05	7,01	8,54	9,62	10,63	11,43
barbara(1)	95,06	111,26	132,98	150,55	163,43	174,65

3.4 Conclusion

Dans ce chapitre nous avons étudié le modèle général suivant :

$$E(u|v) = \sum_s f(u_s, v_s) + \sum_{t \sim s} g_{st}(u_s - u_t) \ ,$$

où les termes g_{st} sont des fonctions convexes (le modèle TV est donc inclus en tant que cas particulier). Deux cas ont été traités :

– *Attache aux données quelconque*. Nous avons proposé un algorithme de minimisation exacte en ajoutant des termes de contraintes à l'énergie pour assurer les propriétés de monotonie.

– *Attache aux données convexe.* Nous avons proposé un algorithme rapide d'optimisation à base de changement d'échelle.

Différentes extensions sont envisagées pour l'avenir. Tout d'abord, l'utilisation de termes de régularisation non stationnaire (i.e, les termes $\{g_{st}\}$ dépendent de leur position dans l'image) est à considérer pour résoudre des problèmes de segmentation. Cette approche est considérée par Bresson *et al.* dans (26) pour trouver des minimiseurs de modèles de contours actifs (40) et des solutions pour certains modèles de type à base de la fonctionnelle de Mumford et Shah (59). L'extension de la décomposition de l'énergie à l'aide de somme sur les niveaux de gris à des énergies de régularisation non convexe, est un travail que nous réservons pour le futur.

Dans le chapitre suivant, nous proposons des algorithmes de minimisation exacte pour des énergies à base de modèle $L^1 + TV$. Ces travaux utilisent les résultats que nous avons prouvés pour les champs de Markov convexes. Contrairement aux cas étudiés dans les chapitres 1 et 2, nous ajoutons des contraintes à la minimisation pour obtenir des propriétés supplémentaires.

Chapitre 4

Retours sur le modèle $L^1 + TV$ et la Morphologie Mathématique

Dans ce chapitre nous utilisons le modèle $L^1 + TV$

$$E(u|v) = \int_\Omega |u(x) - v(x)|\, dx + \beta \int_\Omega |\nabla u| \quad , \tag{4.1}$$

déjà étudié dans le chapitre 1 et 2 en ajoutant des contraintes supplémentaires dans le processus de minimisation. Nous montrons quelques liens étroits que ces minimisations entretiennent avec la morphologie mathématique. Ces nouvelles contraintes dans le processus d'optimisation permettent d'obtenir de nouvelles propriétés sur les filtres induits par ces minimisations.

Dans la partie 4.1, nous proposons de minimiser l'énergie définie par le $L^1 + TV$ sous la contrainte supplémentaire que la forme des ensembles de niveaux demeure identique. Autrement dit, seuls les niveaux de gris associés aux ensembles de niveaux peuvent changer tandis que leurs formes demeurent identiques. La structure de données qui permet d'extraire les formes contenues dans une image est la transformée en ensembles de niveaux (30) (*Fast Level Sets Transform* en anglais (FLST)). Cette transformée conduit à un arbre qui représente l'image et correspond à une fusion du *Min-Tree* et du *Max-Tree* (c.f. chapitre 5) proposé par Salembier *et al.* dans (152) en un seul arbre, et ce, afin de ne pas privilégier les niveaux clairs ou foncés. Nous montrons sur quelques exemples, que ce filtrage est une très bonne initialisation pour effectuer ultérieurement une segmentation. Dans la partie 4.2, nous proposons une extension de la morphologie mathématique au cas des images contenant des vecteurs. La principale caractéristique de cette extension est qu'elle ne réclame pas l'usage d'un ordre sur les éléments vectoriels.

Ces résultats sont, à notre connaissance, nouveaux. La partie qui traite de la minimisation de $L^1 + TV$ sur l'arbre issu de la *Fast Level Sets Transform* est publiée dans les actes de la conférence *Image and Signal Processing and Analysis (ISPA'2005)* (47). Notre extension de la morphologie mathématique au cas vectoriel est publié dans les actes de la conférence *International Symposium on Visual Computing (ISVC'2005)* (51).

4.1 Champs de Markov sur les ensembles de niveaux

Nous rappelons que la minimisation de la variation totale avec une attache aux données de type L^2 est un modèle classique dans le cadre de la restauration d'image (151). Cependant l'utilisation de la norme L^2 conduit à une perte de contraste dans l'image résultat comme énoncé par Chan *et al.* dans (37). Ces auteurs suggèrent donc de remplacer la norme L^2 par la norme L^1. Ce changement de norme définit un filtre qui est invariant par changement de contraste et qui donc préserve le contraste. Ce fait a été énoncé de manière empirique par Chan *et al* dans (37) et nous en avons montré la justification théorique dans le chapitre 2 et dans (47).

Nous rappelons que Meyer prouve dans (128) qu'un carré ne peut être solution de la minimisation de $L^2 + TV$. Les coins du carré sont arrondis. La même remarque est de rigueur pour le modèle $L^1 + TV$ (37). Dans (61; 62; 107), les auteurs proposent un algorithme itératif pour minimiser la variation totale où seuls les niveaux de gris des ensembles de niveaux peuvent changer tandis que leurs formes restent identiques. Ainsi, durant la minimisation, un carré demeure un carré. L'algorithme repose sur la carte topographique (*topographic map* en anglais) introduite par Caselles *et al.* dans (30). C'est une représentation de l'image sous la forme d'un arbre. Cette représentation est également invariante par changement de contraste. Cet arbre est calculé efficacement par l'algorithme de la *Fast Level Sets Transform* décrit par Monasse *et al.* dans (131). Cette carte est également connue sous le nom d'arbre d'inclusion de formes.

Nous détaillons brièvement la structure de cet arbre. Puis nous reformulons le modèle $L^1 + TV$ sur cet arbre. Nous proposons ensuite un algorithme de minimisation et nous présentons quelques résultats.

4.1.1 L'arbre de la "Fast Level Set Transform"

Une présentation détaillée d'un algorithme rapide de construction de l'arbre issue de la *Fast Level Sets Transform* est faite par Monasse dans (130; 131). L'idée repose sur l'inclusion des ensembles de niveau. En effet, nous avons vu que les ensembles de niveau inférieur (respectivement supérieur) sont croissants (respectivement décroissants) au sens de l'inclusion. Rappelons que nous notons $L_\lambda(u)$ les ensembles de niveau inférieur d'une image u, et $U^\lambda(u)$ ses ensembles de niveau supérieur. Nous avons donc

$$L_\lambda(u) \subset L_\mu(u) \ \forall \lambda \leq \mu \, ,$$

et

$$U^\lambda(u) \subset U^\mu(u) \ \forall \lambda \geq \mu \ \ .$$

Ces propriétés d'inclusion permettent de représenter l'image d'une manière complète et non redondante sous la forme d'un arbre. Nous considérons les composantes connexes des ensembles de niveau dont les trous ont été bouchés : chacun de ces éléments est appelé *forme*. Un nœud de l'arbre issu de la *Fast Level Sets Transform* correspond à une forme. Le parent d'un nœud est la forme la plus petite qui contient la forme correspondant à ce nœud : les descendants d'un nœud sont toutes les formes qui sont incluses dans la forme correspondant à ce nœud. Puisque l'arbre issu de la *Fast Level Sets Transform* considère à la fois les ensembles de niveau inférieur et supérieur, chaque forme porte un drapeau qui précise si elle provient

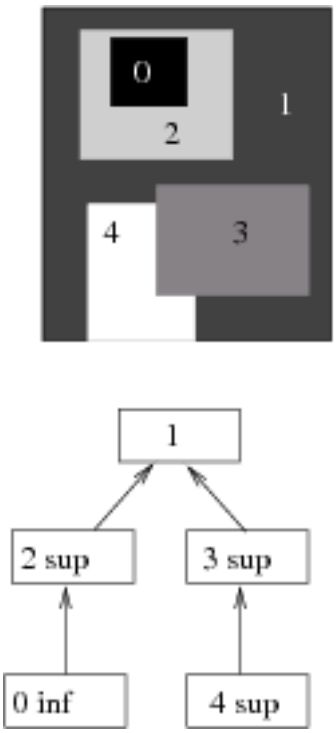


FIG. 4.1 – Une image simple et son arbre issu de la *Fast Level Sets Transform*. Les formes issues des ensembles de niveau inférieur et supérieur sont respectivement représentés par *inf* et *sup*.

d'un ensemble de niveau inférieur ou supérieur. La figure 4.1 montre l'arbre issu de la *Fast Level Sets Transform* sur une image simple. Par conséquent, cette représentation décompose une image u en une série de N formes $S_1, ..., S_N$, où S_N est la racine de l'arbre.

De nombreux attributs peuvent être calculés et associés à chacune des formes. Nous nous intéressons seulement aux attributs qui permettent de reformuler TV en utilisant la formule de la co-aire donnée par l'équation (1.2). Nous nous limitons dans cette partie aux attributs suivants pour chaque forme S_i :

- son niveau de gris,
- son périmètre P_i,
- l'ensemble des pixels qui appartiennent à S_i,
- l'ensemble des pixels qui appartiennent à S_i mais qui n'appartiennent *pas* à ses descendants. On note cet ensemble D_i et son cardinal $|D_i|$.

Enfin, remarquons que par sa définition, l'arbre issu de la *Fast Level Sets Transform* n'implique que les ensembles de niveau. Par conséquent, si deux images ne diffèrent que par un changement de contraste, alors la *Fast Level Sets Transform* génère le même arbre (seuls les niveaux de gris des formes changent). La propriété d'auto-dualité est également valide.

4.1.2 Reformulation markovienne de $L^1 + TV$ sur l'arbre de la FLST

Nous considérons l'arbre issu de la *Fast Level Sets Transform* à partir de l'image observée v. Puisque cet arbre est une représentation complète de l'image, nous l'utilisons pour reformuler le modèle $L^1 + TV$ (équation (4.1)). Nous minimisons donc $L^1 + TV$ sous la contrainte supplémentaire suivante : seuls les niveaux de gris des formes peuvent changer. Les frontières des formes ne bougent pas. Dans ce qui suit, le niveau de gris d'une forme S_i est noté $\bar{v}_i$ pour l'image originale et v_i pour le minimiseur. Le niveau de gris du parent de la forme S_i est noté v_i^p. Bien entendu, le parent de la racine n'est pas défini. Nous reformulons tout d'abord le terme de fidélité L^1 puis la variation totale. Nous montrons ensuite que c'est un champ de Markov.

Reformulation sur l'arbre de la FLST

L'idée repose sur la définition d'une partition de Ω en utilisant l'arbre issu de la *Fast Level Sets Transform*. Une telle partition est facilement obtenue de la manière suivante : pour chaque forme S_i nous avons gardé les pixels qui appartiennent à la forme S_i mais qui n'appartiennent pas à ses descendants. Nous rappelons qu'un tel ensemble est noté D_i. Par conséquent la famille $\{D_i\}$ est une partition de Ω. Les termes d'attache aux données L^1 s'écrivent donc ainsi :

$$\sum_{i=1}^{N} |D_i|\,|v_i - \bar{v}_i|.$$

En utilisant les formes fournies par l'arbre de la *Fast Level Sets Transform* et en utilisant la formule de la coaire (73), nous réécrivons la variation totale comme ceci :

$$\sum_{i=1}^{N-1} P_i\,|v_i - v_i^p| \ .$$

Cette formule ne dépend que de la différence entre les niveaux de gris d'une forme et de celui de son parent, le tout étant pondéré par le périmètre de la forme considérée.

Par conséquent, nous sommes intéressés par trouver une famille $v_0 ... v_N$ qui minimise l'énergie suivante :

$$\sum_{i=1}^{N} |D_i|\,|v_i - \bar{v}_i| + \beta \sum_{i=1}^{N-1} P_i\,|v_i - v_i^p|. \tag{4.2}$$

Comme nous pouvons le voir, cette énergie correspond à un champ de Markov où les interactions sont binaires. Plus précisément, les sites du champ de Markov sont les nœuds de l'arbre issu de la *Fast Level Sets Transform* et les étiquettes sont les niveaux de gris des formes. Les voisins sont définis par les parents et les descendants. Les cliques d'ordre deux sont considérées. On adapte très facilement la proposition 2.9 qui montre l'invariance par changement de contraste pour le modèle $L^1 + TV$ à notre cas. La seule différence est que nous remplaçons Ω par les nœuds de l'arbre issu de *Fast Level Sets Transform*. Par conséquent, l'intégrale sur Ω est remplacée par une somme sur les nœuds de l'arbre issu de la *Fast Level Sets Transform*.

TAB. 4.1 – Temps de calcul (en secondes) pour le filtrage. Le temps nécessaire pour construire l'arbre issu de la *Fast Level Sets Transform* et celui requis pour effectuer la minimisation sont respectivement présentés dans la deuxième et troisième colonne.

Image	FLST	Minimization
Lena (256x256)	0.18	0.11
Lena (512x512)	1.09	1.04
Woman (522x232)	0.39	0.06
Squirrel (209x288)	0.24	0.19

4.1.3 Algorithme de minimisation et résultats

Nous présentons brièvement notre algorithme de minimisation pour l'énergie (4.2) et puis nous présentons quelques résultats.

Algorithmes de minimisation

Nous sommes intéressés par minimiser *exactement* l'énergie (4.2), afin de garantir les propriétés morphologique du filtrage. De nombreux algorithmes sont disponibles puisque cette énergie est convexe (c.f. chapitre 1). Remarquons que nous pourrions tirer parti de la structure d'arbre pour la minimisation, et utiliser par exemple un algorithme de type Viterbi comme décrit par Salembier *et al.* dans (154). Cependant, ce calcul est insurmontable parce que de nombreux nœuds de l'arbre issu de la *Fast Level Sets Transform* ont beaucoup de descendants (typiquement plus de 200 dans des images naturelles).

Nous avons adapté notre algorithme décrit dans la section 1.4 du chapitre 1, à base de "diviser pour régner". Nous avons utilisé la mise en œuvre de la *Fast Level Sets Transform* disponible dans la bibliothèque de traitement d'images *Megawave* (124). Les temps de calcul (en secondes sur un Pentium 4 cadencé à 3GHz) pour l'image *lena*, l'image *woman* (figure 4.2) et l'image *écureuil* (figure 4.5) sont présentés sur le tableau 4.1. Comme nous pouvons le voir, cet algorithme est très rapide.

Expériences

Nous montrons, sur quelques exemples, que la minimisation de la variation totale avec une attache aux données de type L^1 et également contrainte par l'arbre de la *Fast Level Sets Transform* conduit à une bonne simplification d'image.

Les figures 4.2 et 4.3 montrent les minimiseurs pour l'image *woman* avec différents coefficients de régularisation β. Comme nous pouvons le voir, plus le coefficient β est élevé, plus l'image est simplifiée. Le fond de l'image tend à devenir homogène et les détails du visage sont effacés. Remarquons que le contraste est préservé. Ceci est principalement dû au caractère morphologique du filtrage. Les résultats suggèrent que ces images peuvent être utilisées comme initialisation pour des algorithmes itératifs de segmentation de plus haut niveau (fusion de régions par exemple) (165). Enfin, peu de régions subsistent quand un très fort coefficient β est utilisé. Les lignes de niveau d'un tel minimiseur superposée à l'image originale sont présentées sur la figure 4.3. Remarquons que les frontières ne bougent pas et que les contours les plus pertinents préservés.

Les figures 4.5 et 4.6 présentent les minimiseurs pour une image texturée qui représente un écureuil. Sur cette image, plus le coefficient β est élevé, plus les textures, à la fois du fond et de l'écureuil disparaissent. Cependant, le fond et l'écureuil ne fusionnent presque pas. La fusion a lieu là où la queue forme un trou et là où elle est sombre à cause de l'ombre. A part ces deux écueils, les résultats sont assez bons. Remarquons que la séparation entre les deux textures (fond/écureuil) en utilisant un fort coefficient de régularisation β est extrêmement bonne. Ce dernier résultat pourrait être un résultat de segmentation en lui-même à part les deux problèmes énoncés ci-dessus ainsi la mauvaise segmentation de l'œil qui est trop petit.

A titre de comparaison, les résultats de la méthode de segmentation proposée par Rousson *et al.* dans (149) pour les mêmes images sont présentés sur la figure 4.4 pour l'image *woman* et sur la figure 4.7 pour l'image de l'écureuil. Cette méthode fait évoluer des contours actifs et estime de manière adaptative les informations régions.

Nous laissons pour une étude future l'utilisation des *Min/Max-Trees* introduit par Salembier *et al* dans (152) à la place de l'arbre issu de la *Fast Level Sets Transform*. Nous proposons dans la partie suivante une extension de la morphologie mathématique au cas vectoriel.

4.2 Un filtre morphologique vectoriel

Dans cette partie, nous proposons une extension de la morphologie mathématique au cas vectoriel. Les difficultés de cette extension surviennent parce qu'un ordre total entre les éléments est requis en utilisant l'approche classique par les treillis (*lattice* en anglais) (159). Des extensions au cas vectoriel en suivant cette approche ont été proposées en utilisant des concepts de classement de vecteurs (80; 160). Nous proposons un filtre auto-dual et morphologique pour des images vectorielles. Notre approche repose sur la minimisation de la variation totale. Elle ne réclame pas de relation d'ordre entre les éléments mais seulement une norme.

Beaucoup de travaux ont été présentés dans le but de créer des opérateurs morphologiques pour les images en *couleurs* en tant que cas particulier d'images vectorielles. La plupart des approches consiste à choisir un espace approprié pour la représentation des couleurs, et à définir une relation d'ordre entre les éléments de cet espace (85; 68). Dans (33), Chambolle propose une définition pour l'invariance par changement de contraste pour les opérateurs agissant sur les images en couleurs. Cette définition est utilisée par Caselles *et al.* dans (31) et les auteurs présentent un filtre morphologique pour les images couleur.

Les filtres médians vectoriels sont une autre approche pour définir des filtrages vectoriels. Cette méthode a été originalement présentée par Astola *et al.* dans (8). Le procédé consiste à remplacer la valeur d'un pixel par le médian des valeurs des pixels voisins contenus dans une fenêtre. La valeur médiane est définie comme la valeur qui minimise la norme L^1 avec les autres valeurs. Ce type d'approche a été très souvent utilisé dans des buts de restauration d'images (115; 116; 117). Dans (31), Caselles *et al.* tissent des liens théoriques entre les filtres médians vectoriels, les opérateurs morphologiques et les équations aux dérivées partielles. Ils considèrent un ordre lexicographique sur les éléments pour obtenir ces connexions - cette considération technique est nécessaire pour leur approche. On trouvera des résultats complémentaires dans (82) pour le cas scalaire.

Dans (17), Blomgren *et al.* proposent des extensions de la variation totale au cas vectoriel. Ils étudient ces extensions dans le but de faire de la restauration d'images couleur. En

(a) (b)

FIG. 4.2 – L'image originale est montrée en (a). Un minimiseur de $L^1 + TV$ sur l'arbre de la *Fast Level Sets Transform* est présenté en (b) ($\beta = 3$).

(a) (b)

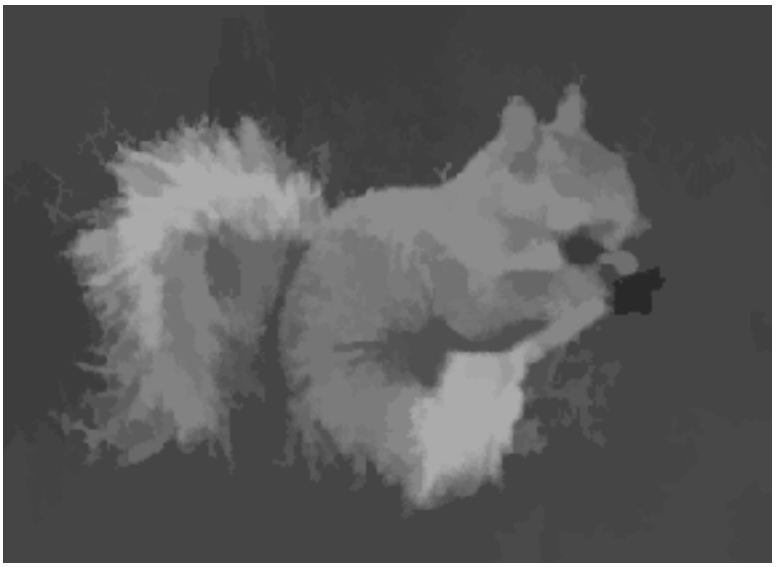
FIG. 4.3 – Minimiseurs de $L^1 + TV$ sur l'arbre de la *Fast Level Sets Transform* pour l'image *woman* ($\beta = 15$) en (a). L'image (b) présente les lignes de niveau d'un minimiseur ($\beta = 30$) superposé sur version atténuée de l'image originale.

FIG. 4.4 – Résultat de la méthode proposée par Rousson et Deriche pour l'image *woman*.

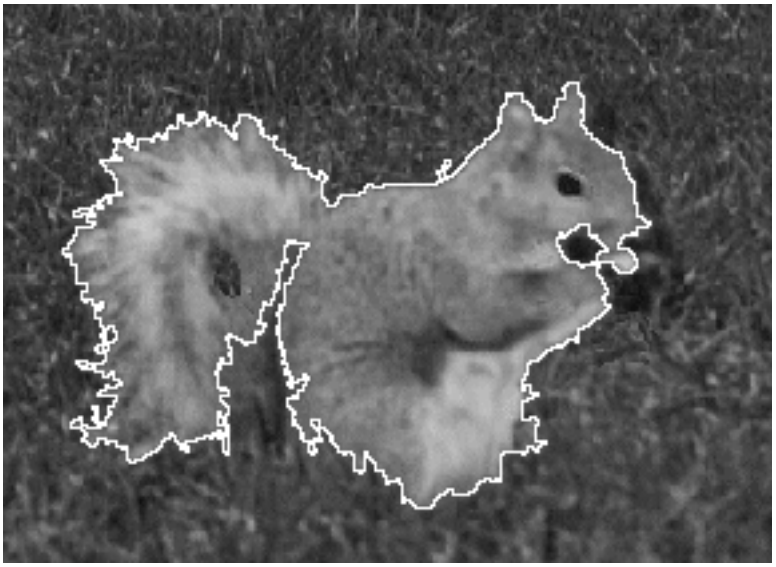
(a)

(b)

FIG. 4.5 – L'image originale est montrée en (a). Un minimiseur de $L^1 + TV$ sur l'arbre de la *Fast Level Sets Transform* est présenté en (b) ($\beta = 1$).



(c)



(d)

FIG. 4.6 – Minimiseurs de $L^1 + TV$ sur l'arbre de la *Fast Level Sets Transform* pour l'image *woman* ($\beta = 2$) en (a). L'image (b) présente les lignes de niveau d'un minimiseur ($\beta = 10$) superposé sur l'image originale.

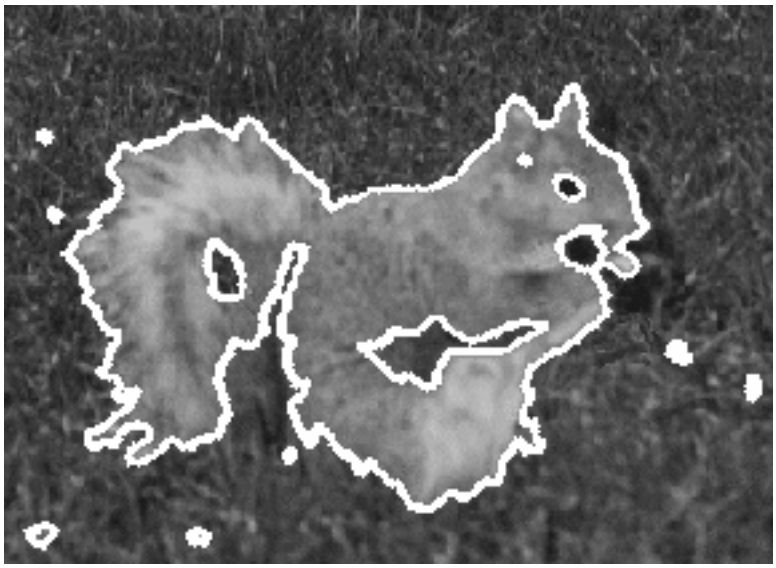


FIG. 4.7 – Résultat de la méthode proposée par Rousson et Deriche pour l'image *écureuil*.

revanche, aucune relation avec la morphologie mathématique n'est proposée.

Nous présentons maintenant notre approche.

4.2.1 Notre approche à base du modèle $L^1 + TV$

Nous considérons une image vectorielle, $u = (u^1, ...u^N)$, définie sur Ω qui prend des valeurs dans $\mathbb{R}^N$. Nous définissons la norme L^1 $\|u\|_{L^1}$ pour une image vectorielle u comme la somme des normes L^1 sur chaque canal, i.e :

$$\|u\|_{L^1} = \sum_{i=1}^{N} \int_{\Omega} |u^i(x)| dx.$$

Nous étendons la variation totale $\overrightarrow{tv}(u)$ d'une image vectorielle u de la même manière que pour l'attache aux données, i.e :

$$\overrightarrow{tv}(u) = \sum_{i=1}^{N} \int_{\Omega} |\nabla u^i|.$$

Notre généralisation du modèle $L^1 + TV$ scalaire (équation (4.1)) au cas vectoriel consiste à appliquer *indépendamment* le modèle $L^1 + TV$ sur chaque canal, i.e :

$$E(u|v) = \|u - v\|_1 + \beta \overrightarrow{tv}(u) \quad . \tag{4.3}$$

Cependant, il est aisé de voir que si aucune contrainte n'est ajoutée, alors la minimisation de l'énergie (4.3) conduit à un minimiseur qui de nouvelles valeurs vectorielles (nouvelles, par rapport à celles observées dans l'image originale). Par conséquent, cela brise la propriété

morphologique qui énonce qu'un filtre morphologique ne doit pas créer de nouvelle valeur. Nous ajoutons donc une contrainte qui assure que cette propriété est préservée. Nous notons C l'ensemble des valeurs observées dans l'image originale v. Notre but est donc de trouver une solution au problème suivant :

$$(\mathrm{P}) \begin{cases} \underset{u}{\operatorname{arginf}} \|u - v\|_1 + \beta \overrightarrow{tv}(u) \\ \text{s. c. } \forall x \in \Omega \;\; u(x) \in C \end{cases}$$

Nous voyons que notre extension au cas vectoriel se réduit au cas du modèle $L^1 + TV$ scalaire quand les images observées sont scalaires. Nous donnons maintenant notre définition d'invariance par changement de contraste dans le cas vectoriel.

Définition 4.1 *Toute fonction continue $g : \mathbb{R}^N \mapsto \mathbb{R}^N$ est appelée un changement de contraste continu vectoriel si et seulement si sa restriction à un axe canonique est un changement de contraste continu (Cf. définition 2.2).*

Nous voyons aisément que le problème (P) définit un filtre morphologique vectoriel d'après notre définition de changement de contraste continu vectoriel (4.1).

Bien que la minimisation du modèle scalaire $L^1 + TV$ définit par (4.1) soit un problème convexe, *ce n'est plus le cas pour le problème* (P). En effet, la fonction objectif est toujours convexe, mais les contraintes ne le sont pas. Nous présentons maintenant un algorithme de minimisation exacte pour le problème (P) non convexe.

4.2.2 Algorithme de minimisation et résultats

Nous présentons tout d'abord un algorithme qui fournit un minimiseur global au problème (P), puis nous présentons quelques résultats. Nous exprimons l'énergie (4.3) de manière discrète de la manière suivante :

$$E(u|v) = \sum_{i=1}^{N} \left(\sum_{s} |u_s^i - v_s^i| + \sum_{(s \sim t)} |u_s^i - u_t^i| \right) .$$

Nous présentons maintenant notre algorithme de minimisation.

Minimisation par coupures minimales

Notre algorithme repose sur l'utilisation d'un algorithme à base d'α-expansions. Cet algorithme a été proposé par Boykov *et al.* dans (24). Étant donné un étiquetage courant, une α-expansion est définie comme ceci : étant donné une valeur α, chaque pixel garde soit sa valeur courante, soit prend la valeur α. Nous sommes intéressés par trouver l'α-expansion optimale à partir de l'étiquetage courant qui minimise l'énergie. Originalement, cette méthode est proposée pour minimiser de manière approximative une énergie markovienne non convexe (24).

Pour résoudre le problème (P), nous itérons la recherche des α-expansions optimales. A chaque itération nous appliquons une α-expansion optimale où α appartient à l'ensemble des valeurs observées C de l'image v. Le parcours des valeurs observées C s'arrête quand aucune α-expansion ne peut faire décroître l'énergie. Notre algorithme est présenté sur la

```
Commencer avec un étiquetage de u tel que ∀s  u_s ∈ C
do
   succes ← false
   forall α ∈ C
        u' = argmin_û E(û)  où  û est une α−expansion de u
        if E(u'|v) < E(u|v)
             u ← u'
             success ← true
while success ≠ false
```

FIG. 4.8 – Pseudo-code de notre algorithme de minimisation.

figure 4.8. Nous prouvons maintenant que notre algorithme fournit un minimiseur global du problème non convexe (P). L'approche est similaire à celle présentée dans la partie 3.3 (chapitre 3) pour les champs de Markov convexes.

Proposition 4.1 *Soit u une image telle que*

$$E(u|v) > \inf_{u'} E(u'|v) \ .$$

Il existe u^α qui est à une $\alpha-$expansion de u, tel que

$$E(u) > E(u^\alpha) \ .$$

Preuve : Avant de donner la preuve, nous rappelons que pour une fonction unidimensionnelle discrète $f : \mathbb{Z} \mapsto \mathbb{R}$, nous avons l'inégalité suivante (c.f. proposition 3.1) :

$$\forall x \, \forall y \forall d \mid (y \geq x) \wedge (0 \leq d \leq (y - x)), \quad f(x) + f(y) \geq f(x + d) + f(y - d) \ . \tag{4.4}$$

Étant donné $\alpha \in C$, nous définissons l'image δ ainsi :

$$\forall s \ \ \delta_s = \begin{cases} \alpha - u_s & \text{si } \forall i \ \ \alpha^i \in [\![u^i_s, \hat{u}^i{}_s]\!] \text{ ou } \alpha^i \in [\![\hat{u}^i{}_s, u^i_s]\!] \ , \\ 0 & \text{sinon.} \end{cases} \tag{4.5}$$

• Nous montrons tout d'abord que l'inégalité suivante est valide :

$$E(u) + E(\hat{u}) \geq E(u + \delta) + E(\hat{u} - \delta) \ . \tag{4.6}$$

Nous montrons cette inégalité pour l'attache aux données et pour les termes de régularisation de manière indépendante et pour chaque dimension. Pour faciliter la lecture de la preuve, nous commettons l'abus de notation suivant : $\delta^i = \delta$. Autrement dit, nous montrons l'inégalité pour chaque canal.
- Attache aux données : Puisque la valeur absolue est une fonction convexe, nous avons l'inégalité suivante (obtenue en utilisant l'équation (4.4)) pour tous les termes de fidélité :

$$|u_s - v_s| + |\hat{u}_s - v_s| \geq |u_s + \delta_s - v_s| + |\hat{u}_s - \delta_s - v_s| \ .$$

Ceci conclut la preuve pour le premier cas.
- Termes *a priori* : Soient $X_{st} = u_s - u_t$, $Y_{xt} = \hat{u}_s - \hat{u}_t$ et $D_{st} = \delta_s - \delta_t$. Supposons que $X_{st} \leq Y_{st}$.

Nous avons donc par définition de δ (équation (4.5)), $X_{st} \leq X_{st} + D_{st} \leq Y_{st}$. En appliquant l'inégalité (4.4) ,nous obtenons le résultat désiré. Le cas $X_{st} > Y_{st}$ se traite de manière similaire.
• Soit $\mathcal{M}$ l'ensemble des minimiseurs de $E(\cdot|v)$. Nous introduisons la norme $\|u\|_1$ sur une image u de la manière suivante : $\|u\|_1 = \sum_s |u_s|$. Soit $u^\star$ définit ainsi :

$$u^\star = \underset{u' \in \mathcal{M}}{\operatorname{argmin}} \|u' - u\|_1 \ . \tag{4.7}$$

D'après l'équation (4.6), nous avons

$$E(u) - E(u + \delta) \geq E(\hat{u} - \delta) - E(\hat{u}) \geq 0 \ ,$$

puisque $\hat{u}$ est un minimiseur global. Si l'inégalité est stricte alors la preuve est terminée. Si ce n'est pas le cas, alors nous avons $E(\hat{u} - \delta) = E(\hat{u})$. Cependant, il est aisé de montrer que $\|u + \delta - \hat{u}\|_1 < \|u - \hat{u}\|_1$. Ceci est en contradiction avec la définition de $\hat{u}$ (equation (4.5)). Ceci conclut la preuve. □

Nous utilisons un algorithme de coupure minimale pour calculer l'$\alpha-$expansion optimale (24). La coupure minimale est calculée sur un graphe attribué qui correspond à l'énergie associé à l'$\alpha-$expansion.

4.2.3 Résultats

Nous présentons quelques résultats sur des images couleurs. Nous but est seulement d'illustrer l'efficacité de notre filtre sur des images vectorielles. Nous appliquons notre filtre en utilisant l'espace RGB pour la représentation des couleurs, bien que de nombreux espace de représentation de couleurs existent tels que Lab, HSI. . . (144). Le choix de meilleurs espaces de représentation des couleurs est laissé pour une étude future. La figure 4.9 présente les résultats de la minimisation pour l'image *Main*. Nous remarquons que plus le coefficient β est élevé, plus l'image est simplifiée ; les détails de la texture disparaissent tandis que la géométrie de l'image a tendance à être préservée. En outre, les couleurs du fond et de la main ne fusionnent pas. Enfin, la couleur de la bague est assez bien préservée. Ce résultat suggère également que l'utilisation de ce filtre pour une segmentation est appropriée. Ces résultats suggèrent également l'utilisation de ce filtre pour faire de la décomposition d'images. Dans (191), Yin *et al.* utilisent le modèle scalaire $L1 + TV$ pour faire cette décomposition. Notre extension de ce modèle au cas vectoriel suggère une décomposition au cas des images vectorielles. La figure 4.9 montre le résultat de cette décomposition. Nous avons appliqué un changement de contraste sur les niveaux de gris pour mettre en évidence le contenu de l'information de l'image de texture. Remarquons que cette décomposition est plutôt assez bonne. Une étude plus fournie devrait être réalisée pour justifier cette décomposition. La figure 4.10 présente une image corrompue par une compression JPEG (101). Le résultat après filtrage montre que les artéfacts dûs à la compression JPEG ont été grandement réduits. L'utilisation de la minimisation de la variation totale pour enlever les artéfacts dûs à JPEG est proposée dans par Alter *et al.* dans (6). La figure 4.10 présente le résultat d'un minimiseur sur une image de texte ancien corrompue par une compression JPEG.

4.3 Conclusion

Dans ce chapitre nous avons proposé quelques filtres morphologiques à base du modèle $L^1 + TV$. Nous avons montré comment utiliser l'arbre issu de la *Fast Level Sets Transform* pour contraindre les formes à rester identiques durant la minimisation. Ce filtre présente l'intérêt de bien simplifier l'image. L'utilisation des *Min/Max-Trees* à la place de cet arbre est une possibilité que nous n'avons pas investiguée. Il est également envisageable de reformuler les filtres connectés attribués (c.f. chapitre 5) par un modèle markovien sur ces arbres. Nous laissons ce travail pour le futur.

Nous avons proposé une extension de la morphologie mathématique pour les images vectorielles. Le principal intérêt de notre approche est qu'elle ne nécessite pas une relation d'ordre entre les éléments vectoriels. Nous avons également proposé un algorithme de minimisation pour le modèle $L^1 + TV$ vectoriel et morphologique. En revanche, un algorithme plus rapide tirant partie de la convexité de la fonction objective ($L^1 + TV$) est envisageable. Ceci permettrait d'enlever l'itération des α-expansions sur les couleurs par une descente locale. Nous n'avons pas étudié cette possibilité en détail. Le cas particulier des images couleur, et spécifiquement des espaces de représentation de la couleur, est également un sujet que nous n'avons pas examiné. Nous laissons ce travail pour le futur.

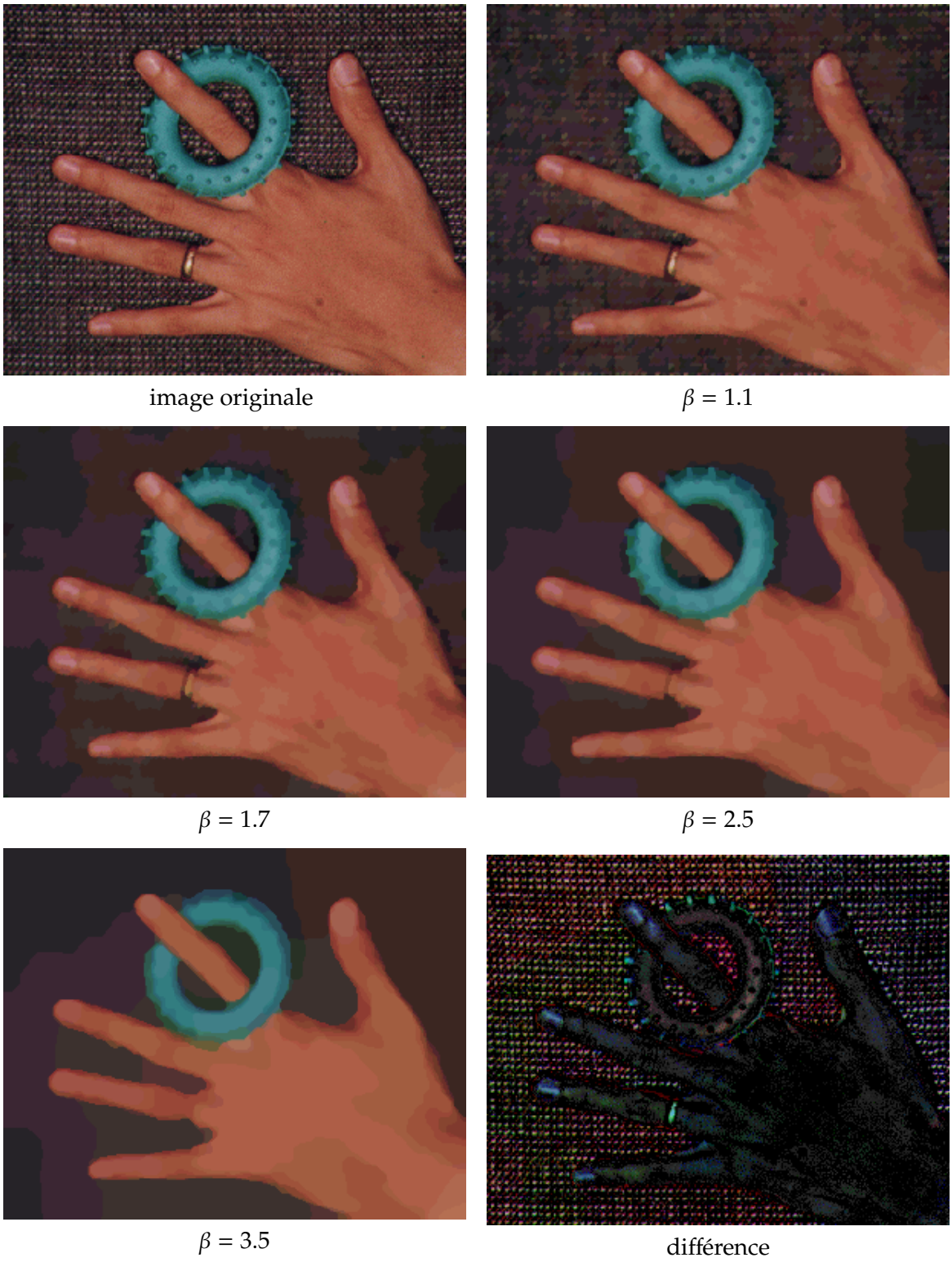


FIG. 4.9 – Minimiseur du problème P avec différents coefficients de régularisation pour l'image *Main*. La dernière image *différence* est la différence entre l'image originale et celle régularisée avec un coefficient $\beta = 3,5$. Nous avons appliqué un changement de contraste sur les niveaux de gris pour mettre en évidence le contenu.

Image originale

$\beta = 2.0$

(c) zoom sur une partie de (a)

(d) zoom sur une partie de (b)

FIG. 4.10 – Restauration d'une image de texte ancien corrompue par une faible compression JPEG.

Chapitre 5

Algorithmes rapides pour les filtres morphologiques connectés et attribués

Dans le chapitre précédent nous avons défini un filtre morphologique définit par l'utilisation du modèle $L^1 + TV$ sur l'arbre de la *Fast Level Sets Transform*. Ce dernier présente l'avantage de ne pas lisser les contours de l'image. De tels filtres sont dénommés connectés. Dans ce chapitre nous présentons des algorithmes rapides pour les filtres morphologiques connectés et attribués. A la différence du chapitre précédant les considérés dans ce chapitre ne sont pas construit en utilisant la *Fast Level Sets Transform* mais le *Max-Tree* ou le *Min-Tree* décrit par Salembier *et al.* dans (152).

Rappelons que la principale caractéristique des filtres morphologiques (159) est leur invariance par changement de contraste. Parmi ceux-ci, les filtres connectés (119; 87; 153) ont reçu un intérêt considérable ces dernières années après que leurs assises théoriques sont présentées dans (161). Le principal intérêt de ces filtres est qu'ils ne déplacent pas les contours (par définition de ces filtres), et ils n'en créent pas de nouveaux. La principale caractéristique des filtres morphologiques connectés est qu'ils agissent sur les composantes connexes plutôt que sur des pixels. Les ouvertures et fermetures par reconstruction furent les premiers filtres à appartenir à cette classe (184). Dans ce chapitre, nous présentons deux algorithmes rapides pour effectuer des filtrages morphologiques connectés.

La partie 5.1 présente les différentes approches pour mettre en œuvre les filtres connectés morphologiques. Notre algorithme est détaillé en partie 5.2. En fait, il existe deux versions de notre algorithme. Le premier procède directement au filtrage, tandis que le second repose sur la construction d'un arbre qui représente exactement l'image à filtrer. Cette représentation par arbre repose sur l'inclusion des ensembles de niveau. L'élagage de cet arbre conduit à un filtre morphologique connecté. L'avantage de ce dernier algorithme est de pouvoir filtrer plusieurs fois une image avec différents critères sans refaire tout le travail à chaque fois. Les complexités en temps et mémoire à la fois théoriques et par l'expérience sont présentées dans la partie 5.3.

Ce travail est en grande partie publié dans les actes de la conférence *European Signal Processing Conference (EUSIPCO 2005)* (48).

5.1 Introduction

Les filtres d'attributs connectés issus du domaine de la morphologie mathématique sont connus par leur pouvoir de simplifier l'image sans déplacer les contours (127; 154). De fait, ce sont des filtres flexibles dans le sens où ils peuvent être utilisées dans diverses applications : restauration et filtrage (41; 182; 185), segmentation (45; 164). Dans (2), les auteurs ont montré comment les utiliser dans un but de classification. Le filtrage de séquence d'images constitue également une niche d'applications pour ces filtres (152). De plus, les filtres morphologiques connectés permettent de construire des espaces multi-échelles non-linéaires (2). L'approche classique du filtrage par éléments structurant viole le principe de causalité dans des espaces de dimension strictement supérieure à un (97). Une approche par des filtres connectés permet de récupérer la causalité (11; 12) - il n'y a pas de nouvelles caractéristiques créés.

Nous définissons tout d'abord une ouverture d'attribut pour une image binaire définie sur Ω. Nous appelons fonction attribut toute fonction A qui travaille sur des ensembles et qui retourne un réel. Nous supposons de plus que cette fonction est croissante, i.e,

$$\forall E \forall F,\ E \subset F \Rightarrow A(E) \leq A(F)\ .$$

Étant donné un ensemble E, nous notons $CC(p, E)$ la composante connexe incluse dans E et qui contient le point p.

Définition 5.1 *Soient E un ensemble binaire, A une fonction d'attribut croissante et $\lambda > 0$. Une ouverture binaire ϕ_λ^A de E avec le paramètre λ est défini ainsi*

$$\phi_\lambda^A(E) = \{p \in \Omega \mid A(CC(p, E)) \geq \lambda\}$$

La fermeture est définie par dualité (164). L'extension aux images par niveau de gris se fait par le principe de décomposition (82; 83). Autrement dit, étant donné une image à niveaux de gris u, nous considérons ses versions seuillées supérieurement $u^\mu = \{x \in \Omega | u(x) \geq \mu\}$ pour toutes les valeurs de $\mu \in \mathbb{R}$. Nous effectuons l'ouverture d'aire pour chacune des images u^μ. Nous remarquons que nous avons $\forall \mu \leq \mu'\ u^{\mu'} \subset u^\mu$. Quand nous avons ces inclusions, nous pouvons reconstruire l'image par la formule suivante :

$$u(x) = \sup\{\mu \mid u^\mu(x) = 1\}\ .$$

Puisque la fonction attribut est croissante, les images binaires filtrées présentent les mêmes inclusions et le filtre est algébrique (164). Et nous pouvons donc reconstruire l'image résultat en niveau de gris.

L'ouverture (respectivement fermeture) d'aire est un filtrage de référence pour les opérateurs connectés d'attributs (182) car c'est le premier exemple de la littérature. Dans ce cas, la fonction attribut renvoie simplement l'aire de la composante connexe. L'intérêt de l'utilisation d'une ouverture d'aire est motivée sur un exemple simple. La figure 5.1 illustre une telle utilisation dans le but d'effectuer une segmentation. Une approche classique, en utilisant les outils de la morphologie mathématique, consiste à appliquer la ligne de partage des eaux (164) sur l'image de la norme du gradient. Cependant, cela conduit très souvent à une sur-segmentation à cause des nombreux minima présents dus au bruit. Nous filtrons donc

l'image de la norme du gradient par une fermeture d'aire. Nous observons sur la figure 5.1, que le nombre de régions est considérablement réduit au fur et à mesure que la valeur de l'attribut croît. On remarque également que les contours ne sont pas déplacés.

Nous présentons différentes approches pour mettre en œuvre une ouverture d'aire. La première approche pour calculer une ouverture d'aire repose sur l'utilisation d'une queue pour effectuer une propagation (182; 183). Une autre approche repose sur l'utilisation d'un arbre qui représente l'image. Chaque nœud de l'arbre correspond à une composante connexe de niveau supérieur. L'arbre lui-même est orienté vers les maxima régionaux de l'image (ces maxima régionaux sont les feuilles de l'arbre tandis que la racine représente tous les points de l'image) à niveaux de gris et il est donc nommé arbre de maxima (152) ("max-tree" en anglais). Enfin, la dernière approche utilise les techniques de type *union-find* (172) qui permettent de maintenir des ensembles disjoints.

Dans ce qui suit nous notons $\mathcal{N}_p$ le voisinage d'un point p. Nous notons N le nombre de pixels de l'image et nous supposons que les pixels des images prennent des valeurs dans l'ensemble discret $[\![0, L-1]\!]$. L'image originale est notée I. Nous présentons brièvement les trois approches, à savoir : celle à base de queue, celle reposant sur les algorithmes de type *union-find* et enfin celle construisant le *Max-Tree*. L'algorithme à base de queue est historiquement le premier.

5.1.1 Algorithme à base de queue

L'approche initiale du calcul d'une ouverture d'aire est due à Vincent (182; 183) et repose sur l'utilisation d'une queue de priorité.

Nous décrivons le fonctionnement de cet algorithme. Tout d'abord, un parcours de l'image est effectué afin de remplir une liste qui contient un point par chaque maximum régional. Une fois que cette liste est créée, chacun de ses points est traité de manière séquentielle. On retire le premier point de la liste (notons le p) qui a le niveau μ. On effectue une croissance de région à partir de ce point. Cette croissance de région correspond à extraire la composante connexe de l'ensemble de niveau supérieur h qui contient le point p : autrement dit il s'agit d'extraire $CC(p, u^\mu)$. Cette croissance s'effectue en ajoutant les voisins d'un point considéré dans une queue de priorité. Seuls les voisins qui ne sont pas encore dans cette queue sont ajoutés. La priorité (i.e, la relation ordre) est donnée par le niveau de gris du pixel : dans le cas d'une ouverture, plus un pixel a un niveau de gris élevé plus sa priorité est grande. Quand on retire un point de la queue, on retire celui qui possède la plus haute priorité. La propagation s'effectue tant que les deux critères suivants sont vérifiés :

1. on ne rencontre pas de voisins dont le niveau de gris est strictement supérieur,
2. on n'a pas encore atteint le critère d'aire suffisant.

Si jamais ces conditions sont vérifiées et que l'on a extrait $CC(p, u^\mu)$ alors on continue la propagation pour extraire $CC(p, u^{\mu-1})$. Si jamais une de ces conditions n'est pas vérifiée alors on arrête la propagation et on passe au maximum régional suivant. Dans ce dernier cas, si l'aire de composante extraite est suffisamment grande alors le critère est vérifié et son niveau est donc connu. Si jamais un pixel voisin a un niveau de gris strictement supérieur, cela signifie que nous ne sommes plus en train d'extraire un pic. Il faut donc s'arrêter et aller extraire au préalable ce nouveau pic.

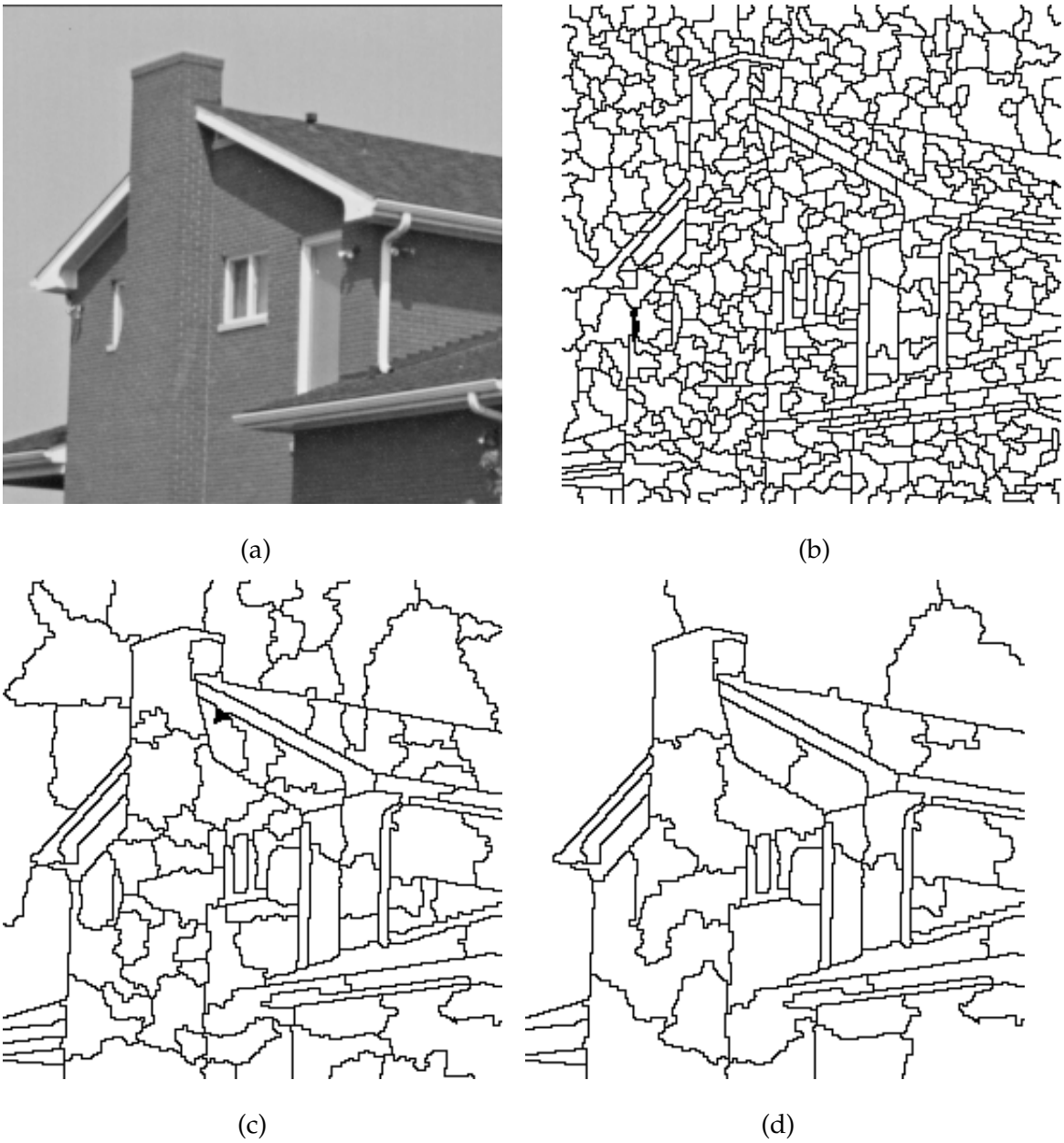


FIG. 5.1 – Illustration de l'effet d'une ouverture d'aire. L'image originale est montrée en (a). Une ouverture d'aire est appliquée sur l'image de la norme de gradient de l'image (a) et puis la ligne de partage des eaux est calculée. Les résultats pour une d'aire de 10, 50 et 100 sont respectivement montrés en (b), (c) et (d).

```
1 // Liste L contenant un point dans chaque maximum regional
2   forall p ∈ L
3    {
4       ajouter p a Q
5       ajouter p a Component
6       λ = u[p]
7       area = 1
8       // extraction de U^λ(u,p)
9       while (area < λ)
10        {
11         p = recuperer le point de Q
12         λ = u[p]
13         forall q ∈ N_p tel que q ∉ Q
14            if  u[q] > λ
15               break
16            add q to Q
17        }
18      forall p ∈ Component
19        u[p] = λ
20    }
```

FIG. 5.2 – Algorithme d'ouverture d'aire par queue de priorité. L'image originale est notée u et le critère d'aire est noté λ. La queue de priorité est notée Q. Un conteneur *Component* contient tous les pixels extraits.

La figure 5.2 présente l'algorithme d'ouverture d'aire par queue de priorité. Dans (189), Wilkinson *et al.* montre que la complexité dans le pire cas de cette approche est $\theta(N^2 \log N)$. L'extension de cet algorithme à d'autres types d'attribut, est présenté dans (25) par Breen *et al.*. Dans ce travail le cas des amincissement, *thinning* en anglais, et épaississement, *thickening* en anglais, est également traité. Ces dernières classes de filtres correspondent au cas où l'attribut n'est plus une fonction croissante de la composante connexe. Du fait du mauvais comportement de cette approche dans le pire cas d'une part, et d'autre part, du fait du bon comportement des autres algorithmes que nous présentons ci-dessous, nous ne présenterons pas les résultats pour l'approche à base de queue. Il a été montré dans (126), que les résultats sont toujours bien moins bons que pour les autres approches.

Nous présentons maintenant l'approche par *union-find*.

5.1.2 L'approche par *union-find*

L'approche par *union-find* pour effectuer une ouverture d'attribut est originalement présentée par Meijster et Wilkinson dans (125; 126; 189). Leur méthode traite les extrema régionaux simultanément, contrairement à l'approche à base de queue de priorité qui les traite séquentiellement. Les composantes sont créées et fusionnées à la volée et lorsque c'est justifié. Quand une composante satisfait le critère elle cesse de grandir.

Les algorithmes de type *union-find* décrits par Tarjan dans (172) sont très efficaces pour maintenir des ensembles disjoints et les fusionner. L'idée est la suivante : pour chaque ensemble un représentant est sélectionné arbitrairement. Chacun de ces éléments est la racine

d'un arbre. Chaque élément qui appartient à un ensemble et qui n'est pas le représentant pointe vers un parent. La racine pointe vers une valeur spéciale. Deux éléments p et q appartiennent au même ensemble si et seulement si ils ont le même représentant - c'est à dire que la racine de l'arbre auxquels ils appartiennent est la même. Les opérations élémentaires pour maintenir ces ensembles sont les suivantes :

- *create*(p) : crée un nouvel ensemble qui ne contient qu'un seul élément p
- $find\_root(p)$: retourne le représentant de l'ensemble auquel p appartient.
- *union*(p, q) : effectue la fusion des deux ensembles qui contiennent p et q.

Nous utilisons un tableau `parent` de taille N dont chaque case contient un pointeur vers son parent. Nous retenons l'encodage de la relation de parent par la méthode donnée par Meijster *et al.* dans (126). Ce pointeur peut être encodé par un entier qui correspond à la position de la case de son parent dans le tableau `parent` : c'est donc un entier positif. Nous présentons la mise en oeuvre d'une ouverture/fermeture *d'aire*. C'est la cas le plus avantageux pour la méthode à base de *union-find*. En effet, quand l'attribut est l'aire scalaire, alors le tableau parent peut être également utilisé pour stocker ce scalaire en cours d'extraction. Au final, soit la racine pointe vers une valeur spéciale, soit elle peut contenir l'aire (c'est possible car l'aire est un entier et le pointeur vers le parent est également un entier). Si cet entier est positif alors c'est un pointeur vers le parent, s'il est négatif alors il représente le scalaire. Si jamais l'attribut n'est pas un scalaire (autrement dit, nous avons besoin de plusieurs entiers pour représenter l'attribut), alors nous avons besoin d'un tableau supplémentaire `attribut` pour stocker les attributs. Afin de savoir si deux éléments appartiennent à la composante, nous avons besoin d'une opération *satisfy*(p, q) qui précise si les points p et q appartiennent ou non à la même composante. La figure 5.3 présente les opérations élémentaires de l'approche par *union-find* dans le cas d'une ouverture ou fermeture d'aire.

Nous voyons que l'opération *find_root* effectue une compression de chemin à chaque fois qu'elle est appelée. Autrement dit, quand on cherche à savoir quel est le représentant d'un élément, on met à jour tous les nœuds intermédiaires pour qu'ils pointent vers le représentant. Ainsi, si on refait une demande *find_root* sur le même élément ou sur un élément appartenant à un chemin déjà examiné alors l'accès au représentant (racine) sera plus court. Ce type de compression est le point crucial pour les algorithmes de type *union-find*. La complexité d'une telle approche est décrite par Seidel *et al.* dans (158) : elle est quasi-linéaire avec le nombre d'éléments (plus précisément elle fait intervenir la réciproque de la fonction d'Ackerman qui est une fonction qui croît extrêmement rapidement). Notons que d'autres heuristiques pour cette compression peuvent être utilisées.

Nous pouvons maintenant décrire l'algorithme qui effectue une ouverture d'attribut. Tous d'abord les pixels sont triés en fonction de leur niveau de gris et par ordre décroissant. Les pixels qui ont le même niveau de gris sont triés par l'ordre classique de parcours des points de l'image (*raster* en anglais).. On itère sur les points dans l'ordre ci-dessus. Quand un pixel est traité, alors on crée un nouvel ensemble. Ensuite, on regarde si un de ces voisins a déjà été traité : un tel voisin est soit de niveau de gris inférieur, soit de même niveau que lui mais déjà traité. Dans ce cas, on lance la procédure de fusion des deux composantes. Une fois que la phase de fusion des composantes est effectuée, il nous faut procéder à une phase de résolution du filtrage. Nous parcourons l'image dans l'ordre inverse de la phase de fusion des composantes. Si un pixel est une racine, alors le niveau de gris de l'image de sortie à ce point est celui de l'image originale, au même point. Si le point n'est pas une racine, il suffit de propager le niveau de gris du parent. Le niveau de gris est forcément connu car nous effectuons le parcours en sens inverse. Le code est présenté sur la figure 5.4.

```
1  create(p)
2  {
3    parent[p] = −1;
4  }
5
6  find_root(p)
7  {
8    if (parent[p] ≥ 0)
9      return parent[p] = find_root(parent[p])
10   return parent[p]
11 }
12
13 statisfy(p,q)
14 {
15   // p est supposé être une racine
16   return (I[p] == I[q]) || (−parent[p] < λ)
17 }
18
19 union(p,q)
20 {
21   // q est supposé être une racine
22   rp = find_root(p)
23   if (rp ≠ q)
24     if satisfy(rp,q)
25       {
26         parent[q] += parent[rp]
27         parent[rp] = q
28       }
29     else
30       parent[rp] = −λ
31 }
```

FIG. 5.3 – Opérations élémentaires pour l'approche par *union-find* dans le cas d'une ouverture d'aire.

```
1  // I est l'image originale
2  // S est un tableau qui contient les pixels triés
3  for (i = 0; i < N; ++i)
4    p = S[i]
5    create(p)
6    forall q in N_p
7      if (I[p] < I[q]) || ((I[p] == I[q]) && (q < p))
8        union(q,p)
9
10 // phase de résolution
11 for (i = N − 1; i ≥ 0; −−i)
12   p = S[i]
13   if (parent[p] \geq 0)
14     parent[p] = parent[parent[p]]
15   else
16     parent[p] = I[p]
```

FIG. 5.4 – Code pour effectuer une ouverture d'aire avec l'approche *union-find*.

Nous présentons maintenant l'approche par *Max-Tree*.

5.1.3 L'approche par *Max-Tree*

Dans (152), Salembier *et al.* présentent un algorithme en trois étapes qui permet d'effectuer une ouverture d'aire : création d'un arbre, filtrage (élagage de cet arbre) et calcul de l'image de sortie. Le point capital de cette approche repose sur la création de l'arbre. Cette étape repose sur une procédure de "flooding" récursive. Nous décrivons brièvement cette approche. L'approche que nous proposons ci-dessous est largement inspirée de cette méthode. Dans l'algorithme de Salembier *et al.* une queue (FIFO pour first-in-first-out en anglais) de priorité hiérarchique (186) est utilisée. Nous modélisons une queue de priorité hiérarchique par un ensemble de L queues (une queue par niveau de gris), comme proposé par Vincent dans (186). Une telle queue au niveau h est utilisée pour effectuer la propagation au niveau h.

Un tableau, appelé `status`, de la même taille que l'image à traiter contient les informations relatives aux pixels. C'est-à-dire, un pixel au niveau h peut être dans l'un de ces trois états : `NOT_ANALYZED` qui signifie que le pixel n'a pas encore été traité, `IN_THE_QUEUE` qui signifie que le pixel est dans la queue de priorité hiérarchique, ou affecté à la valeur k qui détermine à quel k^e nœud ce pixel appartient, i.e, à quelle k^e composante au niveau k il appartient. Un autre tableau auxiliaire de taille L, nommé `NUMBER_NODES` est utilisé. La valeur `NUMBER_NODES`$[h]$ stocke l'indice de la composante connexe, de niveau h, qui est en train d'être traitée.

Selon le choix d'effectuer une ouverture ou une fermeture, la procédure d'inondation commence au niveau de gris le plus faible ou le plus élevé et se décompose en deux étapes. La première correspond, à un niveau courant donné, à l'exploration de tous les pixels de la composante connexe. Une fois cette phase d'innondation terminée, la recherche du parent est effectuée et c'est à ce moment que la construction de l'arbre proprement dit est effectuée.

Une fois que la phase d'innondation terminée et la relation de parent mise à jour, nous devons évaluer l'attribut (dans notre cas, il s'agit de l'aire) associé à chaque composante de niveau inférieur - c'est-à-dire pour chaque noeud de l'arbre, afin de filtrer l'image à niveau

de gris. Nous décidons ensuite si une composante conserve ou non son niveau de gris. Si elle ne le conserve pas, elle doit être fusionnée avec son parent. La décision s'effectue en testant l'attribut contre un seuil défini au préalable. Enfin une passe sur tous les pixels de l'image est réalisée afin de savoir à quel niveau de gris sont affectés les pixels.

Par dualité (164), une fermeture d'aire est calculée de manière similaire mais avec une légère modification : au lieu que l'arbre soit orienté vers le maxima, il est orienté vers le minima de l'image. Il est donc appelé *Min-Tree*. Le filtrage pour une fermeture est exactement le même ; c'est-à-dire que le calcul des attributs et la prise de décision sont les mêmes que pour une ouverture.

Nous avons présenté la version originale de la construction du *Max-Tree* à base d'une queue de priorité hiérarchique. Dans (88), Hesselink montre que cet arbre peut être calculé par un parcours en largeur sur un graphe construit à partir de l'image à traiter. Dans (135), Najman *et al.* montrent que l'on peut remplacer la queue de priorité hiérarchique par une approche de type *union-find*. Aucun temps de calcul n'est présenté dans ces deux papiers.

Contrairement aux approches qui construisent *explicitement* l'arbre pour effectuer le filtrage, comme c'est le cas dans (92; 99; 135; 152), nous proposons une méthode qui s'appuie largement sur la méthode de Salembier *et al.* mais qui ne construit *pas* l'arbre. Ce travail peut donc être considéré comme une approche directe de filtrage. Nous montrons que cette approche conduit à de bonnes performances algorithmiques.

5.2 Notre algorithme à base de propagation

Notre algorithme repose sur la phase d'innondation du *Max-Tree* proposé par Salembier *et al.* dans (152). Cependant rappelons que nous ne sommes pas intéressés par la construction de l'arbre mais seulement par le filtrage. Notre but est donc de combiner la phase de construction de l'arbre avec celle de son filtrage.

Avant de décrire complètement notre algorithme, nous présentons comment nous mettons en œuvre la phase de propagation. Notre approche s'appuie sur une méthode proposée par Knuth dans (103) pour mettre en oeuvre des queues sur des ordinateurs qui ne possèdent pas de pointeurs dynamiques - ce qui était quasiment toujours le cas pour les premiers ordinateurs. Autrement-dit, nous montrons comment, avec un tableau, nous pouvons simuler une queue. Ce procédé nous permet notamment d'économiser énormément de mémoire comparé aux autres méthodes. Nous présentons ensuite notre algorithme.

5.2.1 La phase de propagation

Notre idée repose sur l'utilisation du tableau `status` pour stocker différents types d'informations. Ce type d'approche est utilisé par la méthode de type *union-find* de Meijster *et al.* (126) pour stocker soit un pointeur vers le parent soit une aire. Dans notre cas, il peut stocker un point qui correspond à un pointeur vers un point et ce, dans le but de simuler une queue ; ou il peut stocker un niveau de gris. Nous stockons un point sous la forme d'un entier défini par $y * width + x$ où x et y sont les coordonnées du pixel (x, y) et où $width$ est la largeur (en nombre de pixels) de l'image originale. Remarquons que c'est un entier *positif*.

Nous utilisons les entiers négatifs pour stocker les niveaux de gris. Dans notre algorithme un niveau de gris h est encodé par l'entier négatif $-h - 1$. Cet entier est bien négatif car nous

avons supposé que chaque pixel de l'image originale prenait ses valeurs dans l'ensemble discret $[\![0, L-1]\!]$.

Durant la phase de propagation à un niveau donné, dès qu'un point est retiré de la queue, nous le faisons pointer vers un point qui est le représentant de la composante actuellement en cours d'extraction. Les composantes sont fusionnées tant que leurs aires n'ont pas atteint la valeur du seuil choisi au préalable. L'aire de la composante actuellement en cours d'extraction est stockée grâce à un tableau supplémentaire, de taille L, que nous appelons *attribut*.

Pour mettre en œuvre une queue en utilisant le tableau `status`, nous avons besoin d'un tableau supplémentaire, nommé `last`, de taille L. Il contient le dernier élément qui a été ajouté dans la queue. Chaque case de ce tableau est initialisée à une valeur spéciale `NONE`. Cette valeur signifie qu'aucun élément n'est actuellement dans la queue. Nous illustrons sur la figure 5.5, la dynamique des tableaux `status` et `last` pour différentes opérations sur une queue.

Afin de maintenir un représentant pour chaque composante extraite à un niveau donné, nous avons encore besoin d'un autre tableau de taille L que nous nommons `representative`. A chaque fois qu'une nouvelle composante à extraire au niveau h est découverte, le premier point rencontré qui appartient à cette composante est stocké dans ce tableau à la cellule `representative[`h`]`. Ce tableau est initialisé à la valeur spéciale `NONE` qui signifie qu'aucune composante à ce niveau n'est actuellement en cours d'extraction, et qu'il n'y a donc aucun représentant.

5.2.2 Le canevas pour un filtrage direct

Nous pouvons maintenant décrire le cœur de notre algorithme. Comme pour la méthode de Salembier *et al.*, nous lançons la phase d'innondation avec un pixel dont le niveau de gris est le plus faible (cas d'une ouverture). Chaque case du tableau `status` est initialisée à `NOT_ANALYZED`. Nous supposons que chaque case du tableau `attribut` contient un objet qui dispose des méthodes suivantes : `update(p)` signifie que le point p appartient à la composante que l'on est en train d'extraire, `merge(att)` qui met à jour l'attribut quand deux composantes sont fusionnées, `satisfies(`λ`)` retourne une valeur binaire qui précise si la composante satisfait le critère avec le seuil λ et enfin `reset()` qui réinitialise l'attribut. L'algorithme en entier est montré sur la figure 5.6.

Une fois qu'un élément est retiré de la queue (lignes 4-5), il ne peut plus être ajouté de nouveau dans la queue. Par conséquent, `status[`p`]` est affecté à l'élément qui représente la composante de niveau h actuellement extraite (ligne 7). Si un pixel q voisin de p, i.e, $q \in \mathcal{N}_p$, qui n'a pas encore été traité est rencontré alors il est ajouté dans la queue de niveau $I[q]$ (lignes 14-15) ; et si c'est une nouvelle composante qui est découverte alors nous mettons à jour le tableau `representative` (lignes 11-12). Enfin dans tous les cas, si le niveau de gris du pixel q est plus élevé que le niveau de gris courant h auquel l'extraction de la composante est en train d'être effectuée, alors nous relançons récursivement l'extraction de la composante de que nous venons de découvrir (lignes 16-18).

Une fois que la propagation au niveau h est terminée, nous devons savoir si elle garde son niveau de gris h ou non. Tout d'abord, nous cherchons, si elle existe, la composante de niveau m qui a le plus petit niveau de gris et qui est également strictement supérieur à h (lines 23-24). Si une telle composante n'existe pas, alors elle est racine de l'arbre et elle garde son niveau

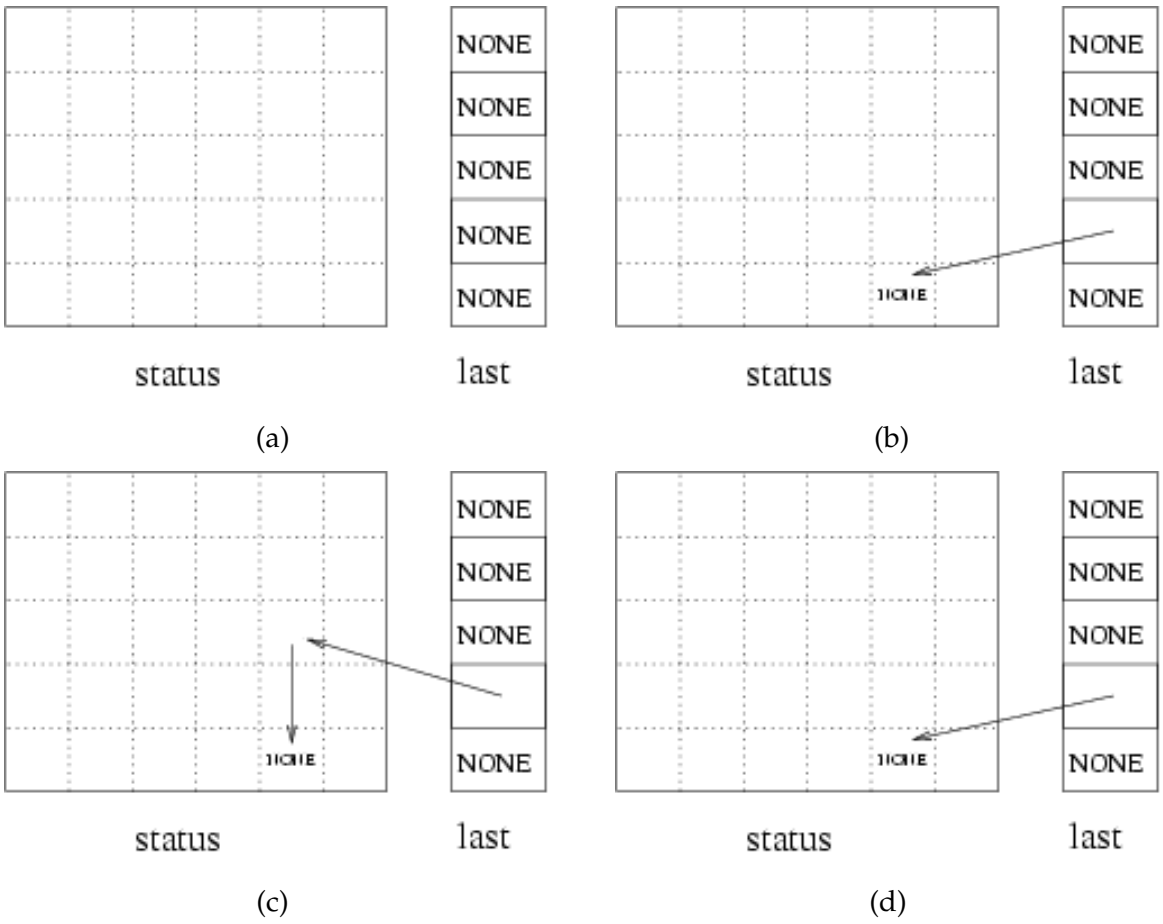


FIG. 5.5 – Illustration de la mise en œuvre de la queue dans l'image. Les états des tableaux `status` et `last` sont précisés après différentes opérations sur des points qui sont au même niveau de gris : l'état original est présenté en (a), l'état après un "push" du point (5,5) est montré en (b), l'état après un "push" du point (3,5) est représenté en (c), enfin (d) montre l'état après un "pop".

de gris h - ce niveau est le niveau le plus faible présent dans l'image (line 35). Si une telle composante existe, alors cette composante est le parent de composante que nous venons d'extraire. Nous regardons si l'attribut associé à cette composante de niveau h satisfait ou non le critère. Si elle ne le satisfait pas, alors nous la fusionnons avec son parent, i.e, avec la composante de niveau m (lignes 27-28). Si elle satisfait le critère alors elle garde son niveau de gris qui sera son niveau après filtrage (ligne 33). Dans tous les cas, l'attribut du parent est mis à jour (ligne 28). Enfin, puisque l'extraction de la composante au niveau h est terminée, nous réinitialisons les informations associées au niveau de gris h (lignes 25-27).

A la fin de ce processus, chaque pixel dans le tableau `status` contient soit un pointeur (représenté par un entier positif) vers un nœud ou un niveau de gris (représenté par un entier strictement négatif). Par conséquent, nous devons effectuer un phase de résolution pour que chaque pixel récupère son niveau de gris (toujours stocké par un entier strictement négatif). Ce processus est décrit dans la figure 5.7. Une passe finale sur l'image convertit les niveaux de gris stocké sous forme négative en niveau de gris réel.

5.3 Complexités théoriques et expériences

Nous rappelons que nous notons N le nombre de pixels de l'image et que les pixels de l'image prennent leur valeur dans l'ensemble discret $[\![0, L-1]\!]$. Pour les images à niveau de gris que nous traitons, L vaut 256. Nous considérons tout d'abord la complexité en mémoire avant de présenter les résultats de nos expériences.

5.3.1 Complexité en mémoire

Notre algorithme est celui qui se comporte le mieux, parmi les trois décrits précédemment, du point de vue de la consommation de la mémoire. En effet, à part l'image originale, l'approche par *union-find* requiert deux tableaux de taille N. Le premier est utilisé pour trier les pixels tandis que le second maintient la structure d'ensembles disjoints et stocke l'aire. Il faut donc $2N$ entiers. L'approche de Salembier *et al.* est la plus gourmande en mémoire. Elle requiert une queue hiérarchique (N eniters), un tableau de N entiers pour le tableau `status`, et enfin N nœuds pour l'arbre. Chaque nœud contient un pointeur vers son parent, l'aire de la composante connexe et enfin son niveau de gris final après filtrage (soit 3 entiers pour chaque nœud). En tout, cette méthode nécessite $5N$. Notre algorithme n'a besoin que d'un tableau de taille N. Remarquons que puisque $L \ll N$, nous négligeons la place utilisée par les tableaux de tailles L à la fois pour notre méthode et celle de Salembier *et al.*.

Plus de mémoire est nécessaire si l'attribut n'est plus l'aire. Supposons que l'attribut nécessite K entiers pour être calculé. Comparé à l'ouverture d'aire, $K-1$ tableaux supplémentaires de taille N sont requis pour l'approche *union-find*. Chaque nœud du l'arbre "max" se voit rajouter $K-1$ entiers. Notre algorithme ne requiert que $K-1$ tableaux supplémentaires de taille L, ce qui est négligeable.

5.3.2 Complexité en temps

D'un point de vue théorique, la complexité de l'approche "union/find" est $\Theta(N \log N)$ dans le pire cas et quasi-linéaire dans le cas moyen, comme Meijster *et al.* le montre dans (126). La complexité de l'approche de Salembier et de notre algorithme est largement dominée par la

```
1  flood(h)
2     while (last[h] == NONE)
3        // propagation au niveau h
4        p = last[h]
5        last[h] = status[last[h]]
6        // affectation à son représentant
7        status[p] = representative[h]
8        for all q ∈ N_p such that q ∈ Ω
9           if (status[q] == NOT_ANALYSED)
10          // affectation à son représentant s'il n'existe pas
11             if (representative[h] == NONE)
12                representative[h] = p
13             val_q = I[q]
14             status[q] = last[q]
15             last[val_q] = q
16             if (val_q > h)
17                m = val_q
18                do {m = flood(m)} while (m > h)
19       attribut[h].update(p)
20
21 // affectation vers les parents
22 m = h−1
23 while ((m >= 0) and (representative[m] == NONE))
24       −−m
25 if (m>=0)
26   if (attribut[h].satisfies(λ))
27      status[representative[h]]= representative[m]
28      attribut[m].merge(attribut[h])
29   else
30      attribut[m] = λ
31      status[representant[h]] = −h − 1
32 else
33    status[representant[h]] = −h − 1
34 // réinitialison des attributs au niveau h
35 attribut[h].reset()
36 last[h] = NONE
37 representative[h] = NONE
38 return m
```

FIG. 5.6 – Notre algorithme pour la phase d'innondation.

```
1 for all p ∈ Ω
2    root = status[p];
3    while (status[root] >= 0)
4       root = status[root];
5    val = status[root];
6    while (p != root)
7       tmp = status[p]; status[p] = val; p = tmp;
```

FIG. 5.7 – Notre algorithme pour la phase de résolution.

FIG. 5.8 – L'image *lena* de taille 512x512.

phase d'innondation qui est linéaire, i.e, $\Theta(N)$. Du point de vue théorique, notre algorithme et celui de Salembier présente la meilleur complexité. Bien que notre algorithme soit en théorie plus économe (en temps de calcul et en mémoire), il faut veiller à ce que la constante qui est cachée dans ces complexités ne soit pas trop grande. En considérant les algorithmes étudiés, cela dépend principalement de la mise en œuvre. Tous les algorithmes ont été mis en œuvre en C ou C++. Nous avons donc attaché une attention particulière à l'optimisation de chacun de ces algorithmes afin de ne pas introduire de biais.

Nous mesurons les temps de calculs des différents algorithmes sur quelques images de test. La première image est une image synthétique d'échiquier de taille ($512x512$), et dont le côté des cases est de 2 pixels. Les autres images sont des images naturelles. La première est l'image classique de *lena* de taille (512x512), présentée sur la figure 5.8. La seconde est une image de tapis texturée, de taille (715x1024), et montrée sur la figure 5.9. Enfin nous utilisons une image satellitaire, de taille (1780x1380), présentée sur la figure 5.10.

Pour chaque image, les temps de calculs (en secondes) pour une ouverture d'aire avec différentes valeurs de seuil sont présentés : la figure 5.11 pour l'image *échiquier*, la figure 5.12 pour l'image *lena*, la figure 5.13 pour l'image *tapis* et la figure 5.14 pour l'image satellitaire. Les temps de calculs sont obtenus en prenant la moyenne des temps de calculs sur 20 lancements des programmes. L'ordinateur utilisé est un Pentium 4 cadencé à 3.0GHz avec 1024 Kb de mémoire cache. Nous avons vérifié que le comportement des algorithmes ne changent pas en utilisant un ordinateur de capacité beaucoup plus faible, à savoir un Celeron 2.6GHz avec 256 Kb de mémoire cache.

Nous observons que notre algorithme est le plus rapide pour traiter les images naturelles. Le seul cas où l'approche par *union-find* est supérieure à la nôtre survient pour le filtrage de l'image synthétique *échiquier* qui ne contient que deux niveaux de gris. La supériorité

FIG. 5.9 – L'image *tapis* de taille 715x1024.

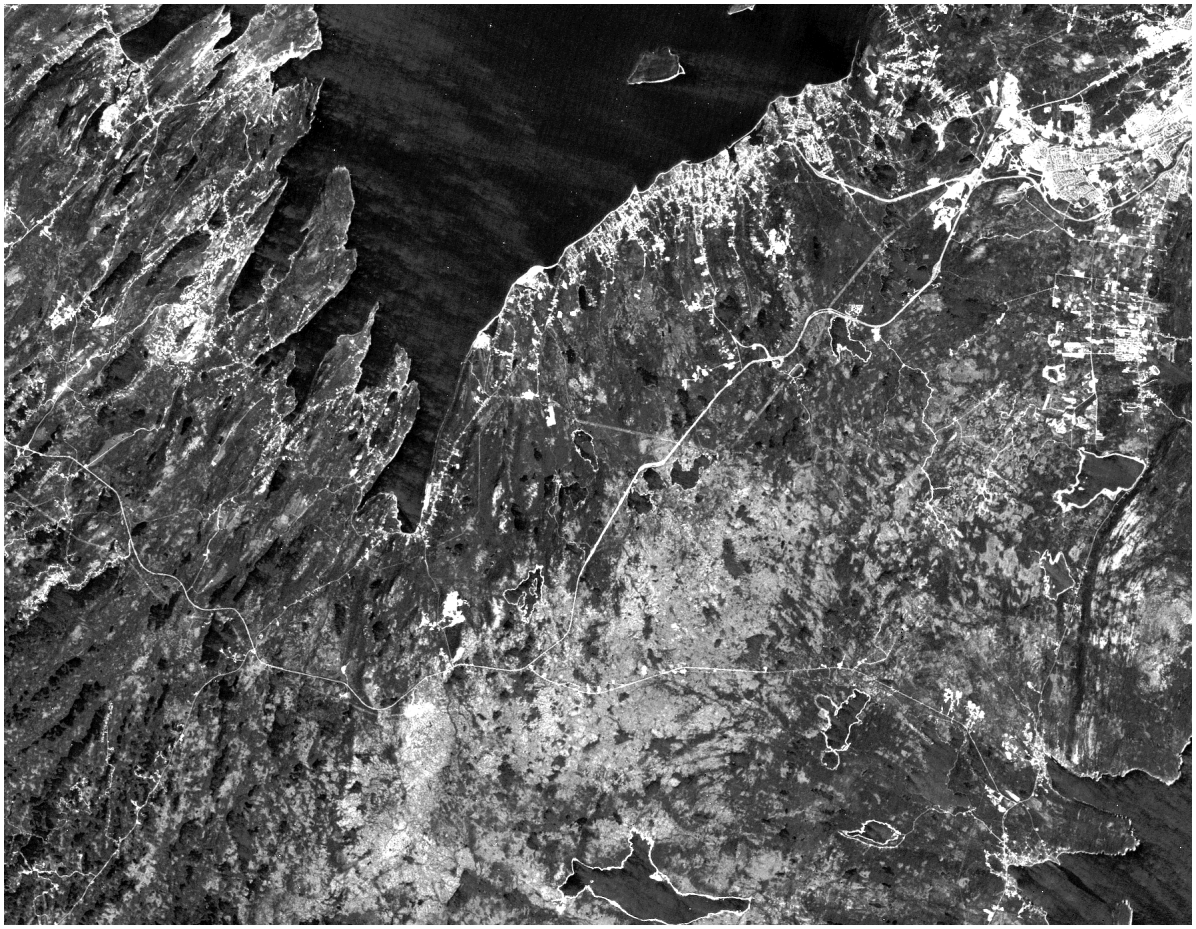


FIG. 5.10 – l'image satellitaire de taille 1780x1380. Cette image originale est sous le droit suivant "*Copyright © 2000. Government of Canada with permission from Natural Resources Canada"* (`http ://geogratis.cgdi.gc.ca/`).

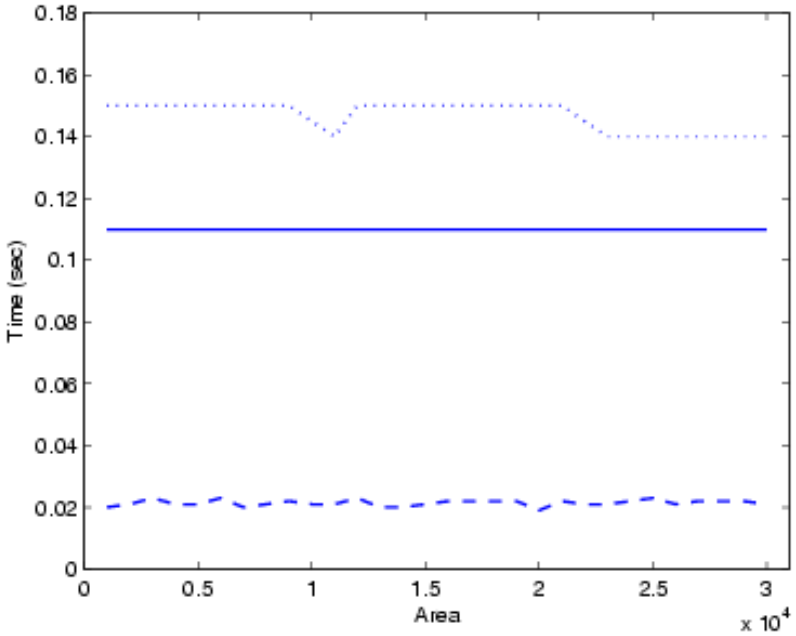


FIG. 5.11 – Temps de calcul pour différentes ouvertures d'aire pour l'image *échiquier*. Les résultats sont présentés en pointillé pour la méthode à base de *Max-Tree* , en tiret pour l'approche par *union-find* et en trait plein pour notre algorithme.

de notre algorithme dépend donc du contenu de l'image traitée (principalement données synthétiques par rapport à des données naturelles), mais pas de la taille des images. Il est intéressant de remarquer que pour des images naturelles, l'approche par *union-find* ne surpasse l'algorithme à base de *Max-Tree* que pour l'image satellitaire qui ne possède que 30 niveaux de gris. Cela suggère que l'algorithme à base de *union-find* est d'autant plus rapide que l'image à traiter présente peu de niveaux de gris.

Les temps d'exécution de notre algorithme est indépendant du choix de seuil de l'aire. L'algorithme à base de *Max-Tree* présente le même comportement. Une très faible dépendance vis-à-vis du seuil est observé pour l'approche par *union-find*.

Nous avons également effectué les tests avec le moment d'inertie comme attribut à la place de l'aire. Les résultats et les comportements des différents algorithmes sont les mêmes que pour le cas de l'aire.

5.4 Conclusion

Nous avons présenté un algorithme efficace pour effectuer des ouvertures et fermetures connectées d'attribut. Cet algorithme se comporte bien, du point de vue de sa complexité temporelle, pour traiter des images naturelles. En outre, il réclame beaucoup moins de mémoire que les autre algorithmes disponibles.

Nous avons montré que le comportement de l'algorithme à base de *union-find* dépend plus (par rapport aux autres) du contenu de l'image (et particulièrement du nombre de niveau de gris présents). Nous émettons maintenant une conjecture. Nous pensons que les

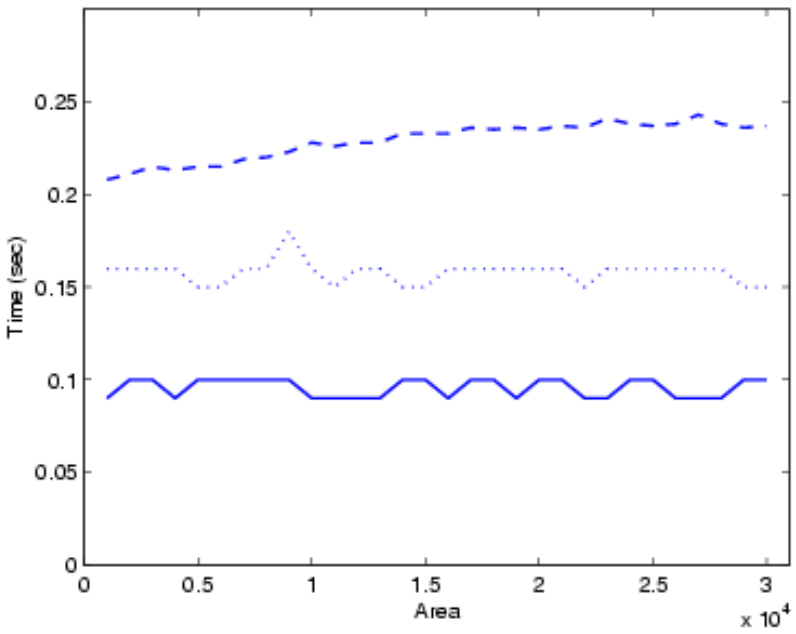


FIG. 5.12 – Temps de calcul pour différentes ouvertures d'aire pour l'image *lena*. Les résultats sont présentés en pointillé pour la méthode à base de *Max-Tree* , en tiret pour l'approche par *union-find* et en trait plein pour notre algorithme.

différences de performances proviennent principalement de la gestion de la mémoire cache. Nous rappelons que la mémoire cache assure des accès très rapides à des cases mémoires proches les unes des autres. En effet, plus le nombre de niveau de gris est faible, plus de grandes zones constantes (zones plates) ont des chances d'être présentes dans les images. Puisque l'approche par *union-find* parcourt ces zones par niveau de gris, la mémoire cache est très efficace dans de tels cas. En revanche, quand de nombreux niveaux de gris sont présents (comme c'est le cas dans une image naturelle brute, non filtrée), le parcours des pixels par niveau de gris tend à devenir un parcours aléatoire. Dans ce cas, la mémoire cache ne peut remplir son rôle. Une telle conjecture doit être vérifiée par l'expérience. Une étude plus poussée doit être menée pour comprendre ce phénomène.

L'extension de notre approche à des critères non nécessairement croissants est à mettre en oeuvre. Ces cas correspondent à des filtrages connus sous le nom d'amincissement (*thinning* en anglais) et épaississement (*thickening* en anglais) (25). Une extension de plus grande envergure consiste à utiliser notre approche pour mettre en œuvre le filtre de grain, décrit par Caselles *et al.* dans (31). Ce dernier repose non plus le *Min/Max-Tree* mais sur la fusion du *Min-Tree* et du *Max-Tree* en un seul arbre. La description de cet arbre est décrite par Monasse dans (131) et dans le chapitre 7.

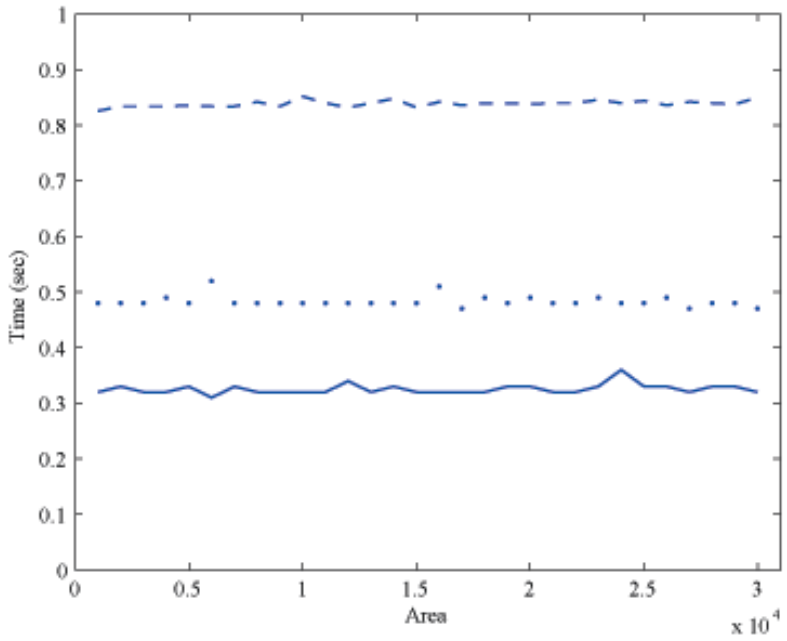


FIG. 5.13 – Temps de calcul pour différentes ouvertures d'aire pour l'image *tapis*. Les résultats sont présentés en pointillé pour la méthode à base de *Max-Tree* , en tiret pour l'approche par *union-find* et en trait plein pour notre algorithme.

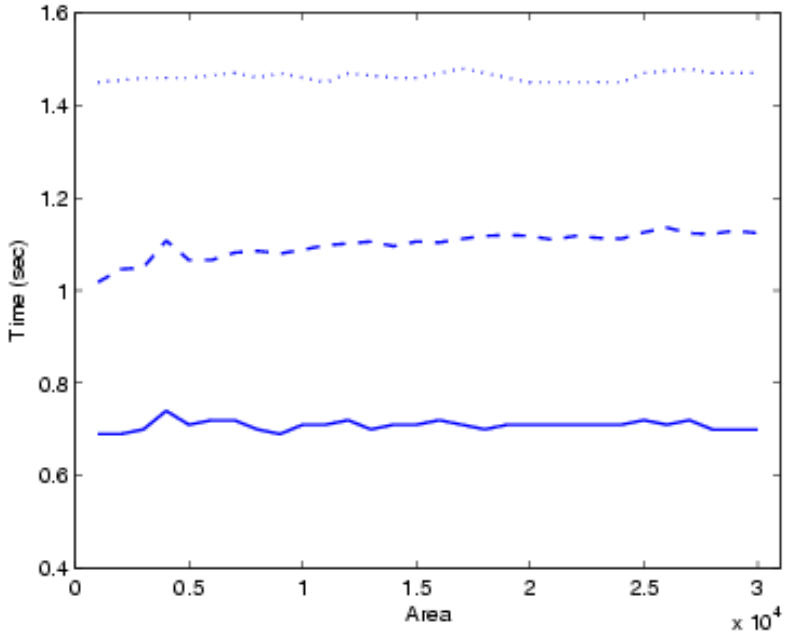


FIG. 5.14 – Temps de calcul pour différentes ouvertures d'aire pour l'image satellitaire. Les résultats sont présentés en pointillé pour la méthode à base de *Max-Tree* , en tiret pour l'approche par *union-find* et en trait plein pour notre algorithme.

Conclusion

Bilan

Ce mémoire a traité de la minimisation exacte d'énergies utilisées dans le domaine du traitement des images. Nous avons supposé que les termes de régularisations sont des fonctions convexes de la différence des étiquettes. Notre approche repose sur la représentation d'une fonction à l'aide de ses ensembles de niveaux. Cette approche permet de reformuler les énergies à l'aide de variables binaires. En revanche, toute configuration de ces variables binaires ne définit pas nécessairement une fonction : nous travaillons donc dans un espace plus grand que celui original. Une fonction peut être définie à partir de ces variables binaires à condition qu'une relation de monotonie soit vérifiée. Nous avons présenté des algorithmes de minimisation pour différentes énergies où cette relation d'inclusion pouvait être préservée. Cette approche nous a également permis d'exhiber des échantillonneurs exacts pour certaines énergies.

Nous avons tout d'abord étudié la minimisation de la variation totale avec des attaches aux données convexes. Nous avons montré que sous ces hypothèses, il était possible de ramener ce problème de minimisation à plusieurs minimisations indépendantes où chacune d'elles ne fait intervenir que des variables binaires. Un algorithme efficace, reposant sur l'utilisation de coupures minimales dans un graphe, a été présenté.

Nous avons ensuite étendu l'approche menée pour la minimisation de la variation à une classe plus grande d'énergies. Pour cela, nous avons introduit la notion de fonctions nivelées : c'est l'ensemble des fonctions qui peuvent s'écrire comme la somme sur les niveaux des fonctions caractéristiques des ensembles de niveaux. Nous avons proposé un algorithme de minimisation exacte à base de coupure minimale quand bien même l'énergie considérée n'est pas convexe. Nous avons également exhibé une condition sur ces énergies telles que leurs distributions de Gibbs associées soient échantillonnables exactement. Enfin, nous avons montré que le modèle $L^1 + TV$ présentait la propriété d'invariance par changement de contraste - propriété que possède un filtre morphologique.

Le cas des énergies dont les termes de régularisation sont des fonctions convexes a ensuite été étudié. Nous avons suivi la voie ouverte par l'approche considérée pour les fonctions nivelées. Nous avons utilisé une double somme sur les niveaux pour représenter une fonction de deux variables. La construction d'un graphe a été décrite telle que sa coupure minimale fournisse un minimiseur exact de l'énergie à minimiser. L'hypothèse de convexité des termes de régularisation permet de considérer la minimisation comme un problème de coupure minimale dans un graphe. Nous avons proposé un algorithme itératif qui calcule un minimiseur exact, sous l'hypothèse supplémentaire de convexité des termes d'attaches aux données. Une condition sur les énergies a également été exhibée afin de pouvoir échantillonner exactement

ces énergies.

Enfin nous avons étudié quelques modélisations à base du modèle $L^1 + TV$. La première modélisation consiste à remplacer le support de l'image par les nœuds de l'arbre issus de la *Fast Level Sets Trasform*. L'intérêt d'une telle minimisation réside dans le fait que les formes des ensembles de niveaux sont préservées durant la minimisation. Nous avons montré que ce filtrage produit une bonne initialisation pour des algorithmes de segmentation de plus haut niveau. Un filtrage vectoriel et morphologique a ensuite été proposé. Ce dernier est défini par la minimisation du modèle $L^1 + TV$ sur chaque canal sous la contrainte qu'aucune nouvelle valeur n'est créée. Bien que ce problème soit non convexe, nous avons proposé un algorithme qui fournit un minimiseur exact. Contrairement à la plupart des autres approches, la nôtre ne requiert pas un ordre sur les éléments.

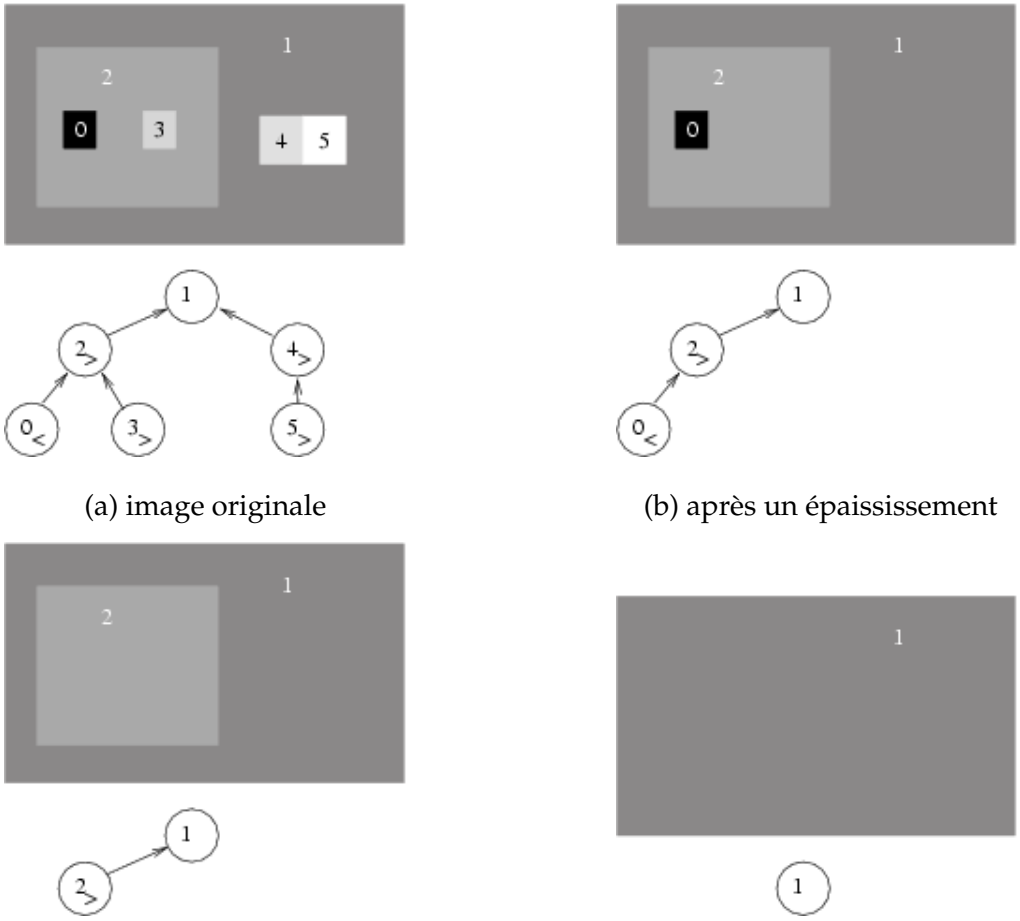
Enfin nous avons ensuite présenté un algorithme pour effectuer un filtrage connecté morphologique. Notre algorithme présente l'intérêt de consommer moins de mémoire que les autres actuellement disponibles. Les expériences suggèrent qu'il est également plus rapide quand il s'agit de traiter des images naturelles.

Perspectives

Les perspectives liés à ce travail sont multiples. La première consiste à appliquer notre approche pour effectuer le filtrage connecté morphologique pour calculer l'arbre de la *Fast Level Sets Transform*. Les autres enjeux concernent la minimisation et l'échantillonnage exacts d'énergie.

Une approche pour calculer l'arbre de la *Fast Level Sets Transform*. Dans ce mémoire, nous avons étudié et présenté notre algorithme pour effectuer une ouverture/fermeture connectée. Nous pourrions l'adapter pour calculer à la place le *Min-Tree* ou la *Max-Tree*. L'extension à des critères non croissants doit encore être considérée ; cela correspond à des amincissement ou à des épaississements. En outre, cette approche peut être utilisée pour calculer l'arbre de la *Fast Level Sets Transform*. En effet, cet arbre peut être calculé en alternant des amincissements et des épaississements. En effet, les feuilles de cet arbre correspondent aux plus petits ensembles de niveaux (inférieurs ou supérieurs) qui n'ont pas de trous. Leurs parents sont les formes qui les contiennent et qui sont d'aires minimales. Considérons le critère suivant : *ne pas avoir de trou*. En itérant les épaississements/amincissements avec ce critère on élague l'arbre de la *Fast Level Sets Transform*. Il suffit de détecter la disparation d'une forme pour être capable de construire l'arbre de la *Fast Level Sets Transform*. Le nombre d'itérations à effectuer correspond au maximum du nombre de transitions inférieur/supérieur et supérieur/inférieur quand on part d'une feuille et que l'on remonte jusqu'à la racine. Nous avons vérifié que pour des images naturelles, ce nombre est de l'ordre de 5. Une telle approche est décrite sur la figure 5.15 pour une image simple.

Algorithmes pseudo-polynomiaux pour résoudre des problèmes NP-complets de minimisation. Rappelons tout d'abord qu'un algorithme est de complexité polynomiale quand le nombre d'opérations effectuées est polynomial avec la taille de l'entrée. Il est important de noter que la taille de l'entrée est exprimée en nombre de bits d'information pour décrire l'entrée.



(a) image originale (b) après un épaississement

(c) après un amincissement (d) après un épaississement

FIG. 5.15 – Illustration des alternances des épaississements/amincissements sur une image simple. A chaque étape, l'image considérée et son arbre associé sont montrés.

Les algorithmes que nous avons proposés à base de coupures minimales pour résoudre les problèmes de minimisations non-convexes n'ont donc pas des complexités polynomiales. En effet, il existe des algorithmes de coupure minimale qui ont des complexités polynomiales mais nous les avons utilisés sur des graphes où leur taille est proportionnelle avec le nombre de labels L. Or, le nombre de bits nécessaires pour décrire L est $\log_2 L$. De tels algorithmes sont appelés pseudo-polynomiaux. Un exemple d'algorithme pseudo-polynomial est celui du test de primalité d'un nombre n. Cet algorithme consiste à déterminer si un des nombres compris entre 1 et $\sqrt{n}$ est un diviseur de n. De tels algorithmes présentent peu d'intérêt théorique puisque l'enjeu est de déterminer des algorithmes polynomiaux.

En revanche, ces algorithmes peuvent se révéler intéressants en pratique à condition que le nombre d'étiquettes demeure suffisamment faible. En outre, ce type d'algorithme laisse la voie ouverte à la résolution de problèmes NP-complets sans pour autant signifier que $P = NP$. En particulier, l'enjeu actuel est de proposer un algorithme de minimisation exact pour des modèles où les termes *a priori* sont des fonctions non convexes. C'est le cas par exemple des modèles de Potts ferromagnétiques et anti-ferromagnétiques.

Un autre enjeu consiste à minimiser exactement des énergies dont les valeurs associées aux sites sont des vecteurs. La difficulté principale réside dans le fait que les valeurs vectorielles ne présentent pas de relation d'ordre naturel.

Echantillonnage exact. Une autre perspective concerne l'échantillonnage des distributions de Gibbs associées aux champs de Markov. Deux approches peuvent être considérées. La première consiste à créer un schéma de *coupling from the past* en suivant les idées de Propp et Wilson (147). Ces derniers ont montré que le modèle d'Ising était échantillonnage exactement par un procédé de *coupling from the past*. Les décompositions des énergies sur les ensembles de niveaux que nous avons proposées dans ce manuscrit conduisent à des champs de Markov binaires associés à chaque ensemble de niveaux et chacun d'eux peut être échantillonné de manière exacte en suivant l'approche de Propp et Wilson. Cette approche suppose l'indépendance de deux champs aléatoire associé à deux niveaux de gris différents et notre décomposition viole cette hypothèse. Il serait intéressant de pouvoir construire un échantillonneur exact en utilisant cette approche. Une autre approche de l'échantillonnage exact, due à M. Huber (93), permet de s'affranchir de l'ordre partiel sur l'espace des configurations. L'idée repose sur l'utilisation d'une étiquette dit "encore inconnu" pour chaque site. Un échantillonneur est alors créé avec cet état supplémentaire. En pratique avec cette approche, on ne suit qu'une seule configuration qui résume l'état entier du système. Quand tous les sites sont affectés à une étiquette réelle, i.e, aucun site n'est assigné à l'étiquette "encore inconnu", alors un c'est un échantillon exact. Ce procédé a été utilisé avec succès pour créer des échantillons exacts pour des modèles de Potts anti-ferromagnétiques.

Avec de tels échantillonneurs, il est envisageable de pouvoir estimer automatiquement les hyper-paramètres des modèles de restauration et de segmentation d'images.

Deuxième partie

Annexes

Annexe A

Publications

Les publications sont données par ordre anti-chronologique et par catégorie.

Article de revue

1. *Image Restoration with Discrete Constrained Total Variation Part I : Fast and Exact Optimization*, **J. Darbon** and M. Sigelle. Journal of Mathematical Imaging and Vision, 2005. (57)

2. *Image Restoration with Discrete Constrained Total Variation Part II : Levelable Functions, Convex and Non-Convex cases*, **J. Darbon** and M. Sigelle. Journal of Mathematical Imaging and Vision, 2005. (58)

3. *Shape-Based Hand Recognition*, E. Konukoglu, E. Yorük, B. Sankur and **J. Darbon**, IEEE Transactions on Image Processing. (193)

Conférences internationales avec comité de relecture

1. *A Vectorial Self-Dual Morphological Filter based on Total Variation Minimization*, **J. Darbon** and S. Peyronnet, Accepté à International Symposium on Visual Computing (ISVC 2005). (51)

2. *Total Variation Minimization with L^1 Data Fidelity as a Contrast Invariant Filter*, **J. Darbon**, Accepted to the 4th IEEE International Symposium on Image and Signal Processing and Analysis (ISPA 2005), September 15-17, 2005, Zagreb, Croatia. (47)

3. *An efficient Algorithm for Attribute Openings and Closings*, **J. Darbon** and C.B. Akgul, Accepted to the 13th European Signal Processing Conference (EUSIPCO 2005),Antalya, Turkey, September 4-8, 2005. (48)

4. *A Fast and Exact Algorithm for Total Variation Minimization*, **J. Darbon** and M. Sigelle, In the proceedings of the 2nd Iberian Conference on Pattern Recognition and Image Analysis (IbPRIA 2005), LNCS series vol. 3522, pp 351–359, Estoril, Portugal, June 7-9, 2005. (55)

5. *Exact Optimization of Discrete Constrained Total Variation Minimization Problems*, **J. Darbon** and M. Sigelle, In the proceedings of the Tenth International Workshop on Combinatorial Image Analysis (IWCIA 2004), R. Klette and J. Zunic (Eds.) : LNCS series vol. 3322, pp. 548-557. Auckland, New Zealand 1-3 december 2004. (53)

6. *Person Authentication Based in Hand Shape*, E. Yörük, E. Konukoglu, B. Sankur, and **J. Darbon**, In the proceedings of the 12th European Signal Processing Conference (EUSIPCO 2004), Vienna, Austria, September 6-10, 2004. (192)

7. *Generic Algorithmic Blocks dedicated to Image Processing*, **J. Darbon**, T. Géraud and P. Bellot, ECOOP-PhD Workshop. Oslo, Novège, Juin 2004. (49)

8. *Generic Implementation of Morphological Image Operators*, **J. Darbon**, T. Géraud and A. Duret-Lutz, In the Proceedings of the International Symposium on Mathematical Morphology VI (ISMM'2002), pages 175–184, Sydney, New South Wales, Australia, April 2002. (50)

9. *Color Image Segmentation Based on Automatic Morphological Clustering*, T. Géraud, P.Y. Strub, et **J. Darbon**, IEEE International Conference on Image Processing (ICIP' 2001), vol. 3, page 70-73, Thessaloniki, Greece, October 2001. (76)

10. *Error Correcting Code Performance for Watermark Protection*, **J. Darbon**, B. Sankur, and H. Maître, In the Proceedings of the 13th Symposium SPIE on Electronic Imaging 2001 (EI'2001) – Security and Watermarking of Multimedia Contents III (EI27), vol. 4314, pages 663-672, San Jose, CA, USA, January 2001. (52)

Articles de revue en préparation

1. *Fast and Exact Optimization of Convex Markov Random Fields*, **J. Darbon**, International Journal of Computer Vision.

2. *Efficient Algorithms for Min/Max Trees and the Fast Level Sets Transform* **J. Darbon** and C.B. Akgul. IEEE Transactions on Pattern Analysis and Machine Intelligence.

Présentations orales

1. Présentation orale du papier *Total Variation Minimization with L^1 Data Fidelity as a Contrast Invariant Filter*, Zagreb, Croatie, September 2005.

2. Présentation orale du papier *An efficient Algorithm for Attribute Openings and Closings*, Antalya, Turquie, Septembre 2005.

3. Présentation orale avec M. Sigelle au séminaire Méthodes Mathématiques du Traitement d'Images. *Restauration d'image avec régularisation par variation totale : échantillonnage et optimisation exacts*, Janvier 2005.

4. Présentation du poster *Exact Optimization of Discrete Constrained Total Variation Minimization Problems*, Auckland, Nouvelle-Zélande, Décembre 2004.

5. Présentation orale du papier *Generic Algorithmic Blocks dedicated to Image Processing*, Oslo, Novège, Juin 2004.

6. Présentation du poster *Generic Implementation of Morphological Image Operators*, Sydney, Australie, Avril 2002.

7. Présentation orale du papier *Error Correcting Code Performance for Watermark Protection*, San Jose, USA, Janvier 2001.

Rapports de recherche et autres papiers

1. *A Fast and Exact Algorithm for Total Variation Minimization*, **J. Darbon** and M. Sigelle, ENST Research Report, Report 2005D002, january 2005. (56)

2. *Exact Optimization of Discrete Constrained Total Variation Minimization*, **J. Darbon** and M. Sigelle, ENST Research Report, Report 2004C004, october 2004. (54)

3. *Unified Texture Management for Arbitrary Meshes* S. Lefebvre, **J. Darbon** and F. Neyret, INRIA research report RR-5210, mai 2004. (54)

4. *Segmentation d'Images en Couleur par Classification Morphologique Non Supervisée*, T. Géraud, P.Y. Strub et **J. Darbon**, International Conference on Image and Signal Processing (ICISP'2001),pages 387-394, edited by the Faculty of Sciences at Ibn Zohr University, Agadir, Moroco, May 2001. (77)

Relectures

1. Relecteur pour la revue *Annales des Télécommunications.*.
2. Relecteur pour la revue *Signal Processing*.
3. Relecteur pour la conférence *European Conference on Siganl Processing (EUCIPCO 2005)*.
4. Relecteur pour *International Workshop on Biometric Recognition Systems (IWBRS 2005)*.
5. Relecteur pour *International Symposium on Visual Computing (ISVC 2005)*.

Annexe B

Algorithmes de minimisation de la variation totale

Nous présentons tout d'abord un algorithme de minimisation de la variation totale par une descente de gradient, puis nous décrivons brièvement l'algorithme de Chambolle.

B.1 Algorithme par descente de gradient

La minimisation de la variation totale par descente de gradient nécessite d'approximer localement la norme du gradient. La variation totale de u, $tv(u)$, est remplacée par

$$tv_\epsilon(u) = \int_\Omega |\nabla_\epsilon u| \ ,$$

où

$$|\nabla_\epsilon u| = \frac{\left(\epsilon(a-b)^2 + (c-d)^2 + (a-c)^2 + (b-d)^2 + \frac{1}{2}(a-d+b-c)^2 + \frac{1}{2}(a-d-b+c)^2\right)^{\frac{1}{2}}}{2+\sqrt{2}} \ ,$$

et où les valeurs des pixels sont définies ainsi

a	b
c	d

.

Ce schéma d'approximation est celui utilisé dans la mise en œuvre fournie dans la bibliothèque Megawave (124). Le pas du gradient est initialisé à 1. Si l'énergie ne décroît pas, alors le pas du gradient est multiplié par 0.8. Le critère d'arrêt de l'algorithme paramétré par e est l'arrêt est provoqué dès que $\|u(n) - u(n-1)\|_2 < e$ ($u(n)$ fait référence à l'image à la n^{e} itération). Pour nos expériences nous avons fixé $\epsilon = 1$ et $e = 0.1$.

B.2 Algorithme de projection de Chambolle

Nous présentons brièvement la méthode de Chambolle, décrite dans (34), pour minimiser la variation totale avec une attache aux données quadratique, i.e,

$$\inf_u \int_\Omega (u(x) - v(x))^2 + \frac{1}{2\lambda} TV(u) \ . \tag{B.1}$$

La transformée de Legendre-Fenchel est définie ainsi

$$f^*(x) = \sup_y \left(< x, y > -f(x) \right) \ ,$$

où $< x, y >$ représente le produit scalaire euclidien associé à l'espace où vivent x et y. Remarquons que la variation totale est homogène de degré un, i.e,

$$TV(\lambda u) = \lambda TV(u) \ \forall u \text{ et } \lambda > 0 \ .$$

La transformée de Legendre-Fenchel, f^*, d'une fonction homogène f est la fonction indicatrice d'un ensemble convexe fermé K (72). Chambolle montre que la solution de (B.1) est donné par

$$v - \frac{\lambda}{2} P_{\lambda K}(v) \ ,$$

où P est le projecteur orthogonal sur λK. Dans (34) Chambolle un algorithme itératif pour calculer $\frac{\beta}{2} P_{\lambda K}(v)$.

On discrétise l'image et les 4 voisins sont considérés. On note s un site de l'image et Ω le support discret. Le projeté peut s'écrire ainsi

$$\min\{\lambda \text{div}(u) - v\|^2 \mid |u_s| \leq 1, \forall s \in \Omega\} \ .$$

Cette minimisation est effectuée par une méthode itérative de recherche de point fixe. On part de $u^{(0)} = 0$ et on applique la règle suivante

$$u_s^{(n+1)} = \frac{u_s^{(n)} + \eta \left(\nabla(\text{div}(u^{(n)} - \frac{v}{\lambda}) \right)_s}{1 + \eta \left| \left(\nabla(\text{div}(u^{(n)} - \frac{v}{\lambda}) \right)_s \right|} \ .$$

Chambolle montre dans (34) que cet algorithme converge vers $P_{\lambda K}(v)$.

Annexe C

Minimisation de Champs de Markov binaires par coupure minimale

Nous présentons brièvement la méthode de Kolmogorov *et al.* (109) pour construire un graphe associée à une énergie markovienne dont les variables sont binaires. La coupure minimale de ce graphe fournit un minimiseur global.

Une énergie E de n variables binaires est dite représentable par graphe au sens de Kolmogorov *et al.* si il existe un graphe $G = (N, A)$ (i.e, composé de N nœuds et de A arcs) tel que pour toute configuration des variables binaires , $u_1 \dots u_n$, la valeur de l'énergie $E(u_1 \dots u_n)$ est égale à la coupure minimale entre S et T, où les ensembles S et T sont définis ainsi : $n_i \in S$ si $u_i = 0$ et $n_i \in T$ si $u_i = 1$.

Toute fonction d'une variable binaire est représentable par graphe. En effet, considérons un graphe qui contient 3 nœuds (s, t, n_1) avec les arcs suivants :

- un arc de s vers n_1 avec la capacité $E(1) - E(0)$ si $E(1) - E(0)$,
- un arc den_1 vers t avec la capacité $E(0) - E(1)$ si $E(1) - E(0) < 0$.

Kolmogorov *et al.* montrent qu'une fonction de deux variables binaires est représentable par graphe si et seulement l'inégalité suivante est vérifiée

$$E(0,0) + E(1,1) \leq E(1,0) + E(0,1) \ .$$

Le graphe associée à cette énergie est composé de 4 nœuds $(s, t, n_1, n2)$ et des arcs suivants :

- un arc de s vers n_1 avec la capacité $E(1,0) - E(0,0)$ si $E(1,0) - E(0,0) \geq 0$, ou un arc de n_1 vers t avec la capacité $E(0,0) - E(1,0)$ si $E(1,0) - E(0,0) < 0$.
- un arc de s vers n_2 avec la capacité $E(1,1) - E(1,0)$ si $E(1,1) - E(1,0) \geq 0$, ou un arc de n_2 vers t avec la capacité $E(1,0) - E(1,1)$ si $E(1,1) - E(1,0) < 0$.
- enfin un arc de n_1 vers n_2 avec la capacité $E(1,0) + E(0,1) - E(0,0) - E(1,1)$

Enfin Kolmogorov *et al.* montrent que la somme de termes représentables par graphe est elle même représentable par graphe. Il suffit d'appliquer les constructions ci-dessus et de sommer les capacités des arcs ainsi créés. La coupure minimale de source s et de puit t du graphe ainsi construit produit un minimiseur exact de l'énergie. La variable u_i associée au nœud n_i vaut 0 si après la coupure n_i est relié à la source s, et $u_i = 1$ si n_i est relié au puit t.

Annexe D

Un cadre générique pour le traitement des images

Nous présentons les motivations et l'intérêt de l'utilisation de la programmation générique dans le contexte du traitement des images et de la vision par ordinateur. L'accent est mis sur la manière dont la programmation générique gère les différents types d'images et comment écrire des algorithmes de manière générique.

En général, la plupart des personnes impliquées dans le traitement des images et la vision par ordinateur sont plutôt mathématiciennes qu'informaticiennes. Par conséquent, une bibliothèque de traitements des images se doit d'être facile à utiliser afin de permettre aux utilisateurs de se focaliser sur les algorithmes et les méthodes sans se soucier des détails de mise en œuvre. Cependant une telle bibliothèque ne doit pas sacrifier les performances au profit d'une utilisation plus aisée. Ces deux qualités ne sont pas réalisables en même temps de manière immédiate. En effet, les outils informatiques dont nous disposons à l'heure actuelle ne remplissent pas ces critères ; c'est-à-dire, l'utilisation d'un langage de haut niveau (proche du langage naturel) se traduit généralement par de piètres performances comparées à l'utilisation d'un langage de bas niveau (assembleur ou langage C). Nous proposons dans ce chapitre plusieurs solutions qui reposent sur l'utilisation de la programmation générique statique. Le langage de mise en œuvre utilisé est le C++ (170).

Il existe de nombreux types d'images. La diversité tient tout d'abord à la dimension des images : signaux, images en 2 ou 3 dimensions, graphes... Elle repose aussi sur le type des valeurs associées au pixel. Ces valeurs peuvent êtres encodées sous forme d'entiers, de valeurs flottantes, de vecteurs... Nous présentons, dans ce manuscrit, commnt gérer cette combinatoire en factorisant le code, en utilisant la programmation générique.

Nous montrons que, de la même manière, de nombreux algorithmes de traitement des images peuvent se mettre sous forme de canevas. En effet, beaucoup d'algorithmes ne diffèrent entre eux que par de petites variations. De nombreuses méthodes de traitement des images se décrivent comme la succession de différents traitements, et là encore nous désirons formaliser cette succession sous la forme d'un canevas. Comme pour la généricité sur les types de données, cela nous permettra d'être générique vis-à-vis des algorithmes et de leurs variantes. Nous proposons également un système qui permet de gérer ces cas et formulons le problème de l'écriture d'algorithme générique comme une composition de plusieurs morceaux d'algorithmes. Notre solution consiste alors en un système qui permet de composer/agencer plusieurs morceaux d'algorithmes.

Nous présentons dans la partie D.1 le contexte et les motivations pour la création d'une bibliothèque générique de traitement des images. Dans la section D.2, nous présentons les différents paradigmes reposant sur le langage C pour mettre en oeuvre des bibliothèques génériques. La partie D.3 est dédiée à la présentation des bibliothèques utilisant le langage C++. Dans la partie D.4, les différentes modalités de composition d'algorithmes sont présentées. Nous exhibons principalement deux types de composition. La première sert à gérer les variations entre différents algorithmes issus du même canevas. Le second type de composition permet de chaîner des algorithmes. Dans la partie D.5, nous présentons les principales solutions existantes pour résoudre le problème de composition. Nous présentons notre solution au problème de composition dans la section D.6. Enfin la section D.7 contient la description de quelques canevas classiques rencontrés en traitement des images.

Une partie de ce travail a été publiée dans les actes de *International Symposium on Mathematical Morphology (ISMM'02)* (50) et dans les actes électroniques du *Workshop for PhD Students in Object-Oriented Systems and Doctoral Symposium of ECOOP 2004* (49).

D.1 Introduction

L'écriture d'une bibliothèque de traitement des images et de vision par ordinateur n'est pas chose facile (67). Le code qui met en œuvre une méthode est d'une importance capitale. Dans son livre (143), Pitas présente plusieurs méthodes classiques de traitement des images ainsi que leur mise œuvre avec le langage C. Les domaines de traitement des images et de vision par ordinateur souffrent du manque de comparaisons de méthodes principalement à cause de la difficulté de mise en œuvre (110; 146). La notion de reproductibilité des résultats scientifiques est reprise par plusieurs auteurs (27; 43; 118; 146) ; i.e, un article scientifique devrait toujours être accompagné de sa mise en œuvre. C'est le cas par exemple du livre de Mallat (118) où le code source qui a servi à générer les résultats est disponible dans la bibliothèque Wavelab (66).

De nombreuses bibliothèques de traitement des images sont disponibles sur l'Internet mais elles n'offrent généralement que certains types de structure d'images et de types de données. En revanche, la plupart des algorithmes de traitement des images s'appliquent sur de nombreux types : signaux temporels, images 2D et 3D,... où les types des données sont variés : valeur binaire, valeur flottante, entier, valeur vectorielle... Du fait de cette diversité, il peut être difficile pour un praticien du domaine de trouver une bibliothèque qui lui convienne. En effet, les mises en œuvre des algorithmes sont généralement restreintes à certains types de données (84) (généralement les pixels sont des entiers non signés codés sur 8 bits). C'est le cas par exemple des bibliothèques *Megawave* (74), *Eikona* (71) et *ImageJ* (94).

Une des solutions pour gérer cette explosion est l'utilisation de la programmation générique. Cette approche a été proposée par Haney *et al.* dans (86) et par Jazayeri dans (98). Le concept de la programmation générique s'énonce ainsi : étant donnés T types de données, S structures de données et A algorithmes, le paradigme de la programmation générique fournit un cadre de travail pour réduire les $T \times S \times A$ mises en oeuvre à $T + S + A$ mises en oeuvre. Les bibliothèques qui utilisent ce paradigme ont été développées initialement avec le langage ADA ou Scheme ; c'est le cas de *UNM Scheme* (177) par exemple. En revanche, le langage C++ permet également d'utiliser la programmation générique grâce à la paramétrisation. Ce

type d'approche en C++ a été popularisé grâce à la bibliothèque standard du C++, *Standard Template Library* (133) qui utilise abondamment la paramétrisation.

La difficulté de mise en œuvre est encore plus grande quand on souhaite que l'algorithme reflète la théorie utilisée tout en préservant les performances.

D.1.1 Généricité vis-à-vis des types de données

Des routines de calculs qui sont capables de travailler sur différents types de valeurs en entrée est un véritable plus. De telles routines sont appelées "génériques". Autrement dit, une routine met en œuvre de manière générique un algorithme si elle n'est écrite qu'*une seule* fois et qu'elle accepte différents types en entrée. Köthe dans (111), et D'Ornellas et van den Boomgaard dans (69; 67) proposent d'utiliser la *programmation générique* pour mettre en œuvre les algorithmes de traitement des images.

D'un point de vue fonctionnel, une bibliothèque de traitement des images et de vision par ordinateur consiste en un ensemble d'algorithmes travaillant sur des structures de données. Rappelons qu'un algorithme travaille sur des entrées et produit le résultat escompté en un nombre fini d'instructions élémentaires. D'après cette définition, il semble naturel qu'un algorithme soit une fonction comme suggéré par Meyers dans (129). De plus, puisque nous désirons la généricité, ces algorithmes doivent être paramétrés par le type des entrées. Le type de retour doit être déduit par le compilateur. Cette déduction est possible une fois que les types en entrée sont connus. La description de ce processus est présentée sur la figure D.1.

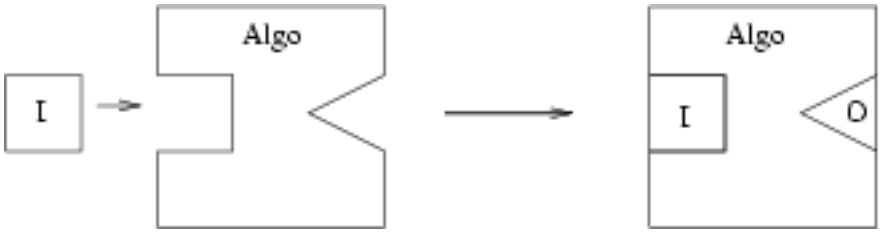


FIG. D.1 – Ce diagramme présente un algorithme générique. Un utilisateur manipule un algorithme "abstrait". Une fois que le type de l'entrée *I* (carré) est fourni, le compilateur infère le type de retour 0 (triangle) et spécialise l'algorithme pour cette entrée particulière.

Une bibliothèque de traitement des images et de vision par ordinateur doit donc remplir les critères suivants :

- Tout d'abord, les routines de calculs doivent être rapides (157). Les bibliothèques existantes remplissent généralement ce critère de performance.
- Puisque les personnes impliquées dans le traitement des images sont plutôt des mathématiciens que des informaticiens, la mise en œuvre des algorithmes doit être aussi expressive que possible (50). En d'autres termes, la mise en œuvre doit refléter la théorie et les notations.
- Les mises en œuvre doivent être générique (111; 69; 67) dans le but de ne pas céder à l'explosion combinatoire. Il faut éviter d'écrire une version particulière d'un algorithme pour chaque type d'images.
- Le langage utilisé doit être proche du langage C ou C++. En effet, la plupart des personnes impliquées dans le traitement des images ne connaissent que ces langages (143).

– Afin de diminuer le nombre de bogues (bugs) dans un programme, un système de typage fort doit être utilisé. Un tel système empêche d'effectuer des opérations sur des types incompatibles. En outre, si de telles opérations ont lieu alors elles sont détectées à la compilation.
– Enfin l'utilisation d'un algorithme disponible dans une bibliothèque doit être facile pour un utilisateur. De plus, la mise en œuvre d'un nouvel algorithme doit également être facile à écrire pour un développeur.

Ce dernier point met en évidence les deux catégories de personnes impliquées dans le traitement des images : d'une part les utilisateurs et d'autre part les développeurs.

D.2 Les paradigmes classiques

Dans cette partie nous présentons les paradigmes classiques utilisés pour mettre en œuvre une bibliothèque de calcul scientifique et plus particulièrement de traitement des images. Nous présentons ces paradigmes en utilisant le langage C. Nous décrivons tout d'abord l'approche utilisée par Matlab (120). Ensuite, nous présentons une approche en langage C où plusieurs types de données peuvent être utilisés. Enfin, nous présentons un paradigme où le langage C est toujours utilisé mais où les macros jouent un rôle important. Bien que ce dernier paradigme soit peu utilisé, c'est celui qui procure le plus de généricité.

D.2.1 Matlab et Le langage C avec un type de données universel

Une manière simple de gérer différents types de données est de considérer qu'un type unique et général est utilisé pour encoder toutes les valeurs. En utilisant le langage C le type atomique `double` (ou `float`) est largement utilisé dans ce but. En effet, une variable de type `double` peut représenter un entier signé ou non, une valeur binaire... C'est le choix qui est fait par le logiciel de calcul scientifique Matlab. Un type image est déclaré en C de la manière suivante :

```
1 typedef struct
2 {
3   unsigned nrows, ncols;
4   float** data;
5 }  image2d;
```

Le nombre de lignes et de colonnes de l'image est respectivement donné par `nrows` et `ncols`. Les valeurs des pixels sont stockées dans un tableau en deux dimensions. Pour instancier une image de taille $512x512$ et affecter la valeur du pixel (128,128) à 1.6180 nous écrivons les lignes suivantes :

```
1 image2d ima;
2 ima.nrows = ima.ncols = 512;
3 /*  ...  */
4 ima.data[128][128] = 1.6180;
```

Les commentaires correspondent à la réservation en mémoire de l'image. Nous occultons cette gestion dans notre exposé.

Un opérateur est défini comme une fonction. Par exemple un opérateur qui effectue la moyenne des pixels point à point de deux images s'écrit ainsi :

```
1  image2d* mean(const image2d* ima1, const image2d* ima2)
2  {
3    unsigned row, col;
4    image2d* ima;
5    assert(ima1->nrows == ima2->nrows && ima1->ncols == ima2->ncols);
6    /*  ...  Allocation de l'image ima */
7    ima = new_image2d(ima1->nrows, ima1->ncols);
8    for (row = 0; row < ima1->nrows; ++row)
9      for (col = 0; col < ima1->ncols; ++col)
10       ima->data[row][col] = (ima1->data[row][col] + ima2->data[row][col]) / 2;
11   return ima;
12 }
```

Le contenu de cette procédure est assez simple. Tout d'abord, deux entiers sont créés afin de parcourir les pixels de l'image (ligne 3). Un pointeur vers l'image résultat (ligne 4) est créé. Avant d'instancier l'image résultat (ligne 6), nous vérifions que la taille des deux images est identique (ligne 5). Pour chaque ligne et pour chaque colonne (autrement dit pour chaque pixel), l'image de sortie est affectée à la moyenne arithmétique des pixels des deux images en entrée. Enfin, le résultat est retourné. Un appel à cette fonction s'écrit comme suit :

```
1 /* ima1 et ima2 sont supposée instanciée */
2 ima = mean(ima1, ima2)
```

Cette manière de gérer la généricité présente plusieurs défauts. Tout d'abord, la manière de stocker l'image est gourmande en mémoire. En effet, sur un ordinateur 32 bits, un `double` est généralement encodé sur 8 octets, et un `float` sur 4. Si nous désirons traiter des images à niveaux de gris présentant 2^8 niveaux alors nous avons seulement besoin d'un octet pour coder chaque pixel. Ensuite, le choix du type `double` implique que nous ne gérons pas les données vectorielles. Cette approche n'est donc pas tout à fait générique. Une image vectorielle doit alors être vue comme plusieurs images scalaires. Enfin, le principal reproche que nous pouvons formuler envers cette approche est le manque de typage. En effet, à aucun moment nous précisons quel est le type des données que nous manipulons. Par exemple, si nous manipulons des images binaires alors nous pouvons appeler la procédure `mean` sur de telles images. Nous pouvons même l'appeler avec une image binaire mais qui stocke les valeurs sous forme de flottants.

Le dernier point est un sévère inconvénient pour une bibliothèque de calcul scientifique. En effet, le compilateur est incapable de vérifier la cohérence et la sémantique des calculs effectués, et ce, à cause du manque de typage.

Nous présentons maintenant une meilleure solution.

D.2.2 Le langage C avec différents types de données

Le manque de contrôle de types conduit à une perte de la sémantique et à l'introduction de bugs. Une solution consiste à renforcer la sécurité du code en forçant l'usage explicite de types. La solution dans le langage C consiste à utiliser le mot clef `enum` qui permet de définir une énumération des types qui peuvent être utilisés. Ainsi le programmeur sait toujours savoir quel type est utilisé pour encoder les données. Cette approche a été proposée par Dobbie *et al.* dans (64). La définition d'une image s'écrit dont ainsi :

```
1 typedef enum { INTU8, FLOAT, DOUBLE, BIN } data\_t;
```

```
2
3 typedef struct
4 {
5   unsigned nrows, ncols;
6   data_t type;
7   void** data;
8 }  image2d;
```

La ligne 1 définit l'énumération des types possibles pour encoder les données, et elle est appelée `data_t`. Quatre étiquettes sont utilisées : `INTU8` pour signifier que les données sont des entiers non signés encodés sur 8 bits, `float` (respectivement `double`) pour signifier que les données sont encodées par un nombre flottant sur 4 octets (respectivement 8 octets), et enfin `BIN` pour signifier que les données sont binaires. Un champ nommé `type` dans la structure `image2d` (ligne 6) permet de savoir quel type est réellement encodé. En outre, puisque la structure connaît effectivement le type utilisé pour encoder les valeurs, elle peut également l'utiliser pour les stocker. Par exemple, pour une image où les données sont des entiers non signés sur 8 bits, le type utilisé est `INTU8` et le champ `data` est donc un double pointeur qui pointe vers des données de type **unsigned char**. Pour que ce pointeur puisse pointer vers différents types réels, il doit être *universel* : il est donc de type `void **`.

Le mécanisme pour accéder à un pixel est caché par des accesseurs : par exemple `get_INTU8` et `set_INTU8` pour accéder et affecter des valeurs de type `INTU8`. Une demande de lecture du pixel (*i*, *j*) sur l'image `ima` est réalisée par l'appel suivent : `get_INTU8(ima, i, j)`. Cette fonction commence tout d'abord par vérifier que le type des données tenues par l'image `ima` est bien `INTU8` - c'est à dire que `ima->type` vaut `INTU8`. Puis elle vérifie que l'accès demandé est bien dans le support de l'image, puis elle retourne la valeur. En procédant ainsi, la sécurité vis-à-vis des types de données est renforcée.

Nous écrivons maintenant la procédure `mean` de la partie précédente en utilisant cette approche. Puisque maintenant le type des données tenues par l'image est connu, il faut utiliser les bons accesseurs, sinon une erreur se produira à l'exécution. Nous définissons donc une version de l'algorithme pour chaque type de donnée. Par exemple, pour le cas `INTU8`, nous avons le code suivant :

```
1 image2d* mean_INTU8(const image2d* ima1, const image2d* ima2)
2 {
3   unsigned row, col;
4   image2d* ima = new_image2d(ima1->nrows, ima1->ncols, INTU8);
5   for (row = 0; row < ima1->nrows; ++row)
6     for (col = 0; col < ima1->ncols; ++col)
7       *set_INTU8(ima,row,col) = (*get_INTU8(ima1,row,col) +
8                                  *get_INTU8(ima2,row,col)) / 2;
9   return ima;
10 }
```

Afin d'avoir une fonction unique, nous utilisons la structure de contrôle **switch/case** pour sélectionner la bonne fonction. Le code est alors le suivant :

```
1 image2d* mean(const image2d* ima1, const image2d* ima2)
2 {
3   image2d* ima;
4   assert(ima1->nrows == ima2->nrows && ima1->ncols == ima2->ncols);
5   assert(ima1->type == ima2->type);
6   switch (ima1->type)
```

```
7   {
8     case INTU8:
9       ima = mean_INTU8(ima1, ima2);
10      break;
11    /* other cases (FLOAT, BIN, etc.) */
12  }
13  return ima;
14 }
```

Bien que nous ayons ajouté plus de sûreté dans les types utilisés, le prix à payer pour l'écriture d'un algorithme est la duplication du code. En effet, il faut écrire une version de l'algorithme pour *chaque type* de donnée. Cela brise un des principes de la généricité. En outre, quand un utilisateur désire ajouter un nouveau type de données, il doit modifier la version en façade qui effectue l'appel vers la bonne mise œuvre pour gérer ce nouveau cas.

Nous présentons maintenant le dernier paradigme à partir du langage C.

D.2.3 Le langage C avec des macros

L'objectif de ce paradigme est de garder la bonne propriété de typage de la partie précédente tout en évitant la duplication du code. Pour cela nous utilisons le pré-processeur du langage C et donc les *macros*. Une *macro* est définie par le mot clef `#define` du langage C. Une *macro* peut être vue comme une fonction au sens où elle peut prendre un ou plusiurs arguments. En revanche, ce n'est pas une fonction au sens propre car elle ne peut pas s'appeler récursivement. La définition du type image est la suivante :

```
1 #define decl_image2d(T)       \
2 typedef struct                \
3 {                             \
4   unsigned nrows, ncols;      \
5   T** data;                   \
6 }    image2d_##T;
7
8 typedef unsigned char INTU8;
9 typedef float  FLOAT;
```

Nous pouvons voir la définition du type image comme une structure qui se nomme `image2d_##T` (ligne 6). Les deux caractères `##` sur la ligne 6 correspondent à l'opérateur de concaténation de chaînes de caractères du pré-processeur. Pour déclarer une image dont les données sont de type `INTU8`, nous écrivons `decl_image2d(INTU8)`. Au moment du passage du pré-processeur sur la ligne `decl_image2d(INTU8)`, le pré-processeur va chercher la définition de `decl_image2d` et remplace l'argument `T` par `INTU8`. Ainsi, le type `image2d_INTU8` est défini. Par conséquent, après le passage du pré-processeur le code généré est le suivant :

```
1 typedef struct
2 {
3   unsigned nrows, ncols;
4   INTU8** data;
5 }
6  image2d_INTU8;
```

La définition d'un type image qui contient des données de type `FLOAT`, s'écrit de manière similaire `decl_image2d(FLOAT)`. Nous avons donc réussi à générer une structure d'images

pour chaque type de données contenues dans les images. Le champ des données est maintenant bien typé, en comparaison avec la partie précédente, le type universel `void**` est remplacé par celui du type des données contenues dans les images.

L'écriture d'algorithmes utilise le même procédé de *macros*. L'algorithme `mean` s'écrit ainsi :

```
1  #define def_mean(T)                                                          \
2  image2d_##T* mean_##T(const image2d_##T* ima1, const image2d_##T* ima2)     \
3  {                                                                            \
4    unsigned row, col;                                                         \
5    image2d_##T* ima = new_image2d_##T(ima1->nrows, ima1->ncols);              \
6    for (row = 0; row < ima1->nrows; ++row)                                    \
7      for (col = 0; col < ima1->ncols; ++col)                                  \
8        ima->data[row][col] = (ima1->data[row][col] + ima2->data[row][col]) / 2; \
9    return ima;                                                                \
10 }
11
12 def_mean(INTU8)
13 def_mean(FLOAT)
```

Au moment du passage du pré-processeur sur les lignes 12 et 13, la version de l'algorithme `mean` est générée pour les images dont les données sont du type `INTU8` et `FLOAT`. Ceci résout le problème du paradigme de la partie précédente. En effet, le code n'est plus dupliqué manuellement mais à la demande du programmeur par l'intermédiare de *macros*.

Désormais, le compilateur est capable d'appliquer son système de typage et donc de détecter les erreurs à la compilation. C'est un avantage non négligeable comparé aux versions précédentes qui ne généraient l'erreur qu'à l'exécution du programme.

Nous quittons maintenant le langage C pour le langage C++ qui offre des solutions plus élégantes et efficaces pour résoudre le problème de la généricité dans le cadre du traitement des images.

D.3 Vers d'autres paradigmes en utilisant C++

Nous décrivons dans cette partie, les alternatives autorisées par l'utilisation du langage C++. Les *macros* sont gérées par le pré-processeur, alors que les patrons de conception ("templates" en anglais) sont directement gérés par le compilateur. Cette caractéristique nous permet d'obtenir une généricité totale. Nous décrivons brièvement ce mécanisme avant de décrire les bibliothèques Vigra (181) et Cimg (42) et Qgar (148). Les concepts utilisés par ces bibliothèques sont repris et adaptés dans les bibliothèques Horus (102; 104; 105; 106; 176) et Olena (140) que nous décrivons après. Toutes ces bibliothèques font un usage intensif des patrons de conception.

D.3.1 Le mécanisme des patrons de conception

Nous imaginons maintenant que la structure `image2d` est *méta* au sens où elle est paramétrée par un type `T` inconnu. La fonction `mean` est également paramétrée ainsi. La paramé-

trisation se déclare dans le langage C++ en utilisant le mot clef **template**. Nous obtenons le code suivant :

```
1  template<class T>
2  struct image2d
3  {
4    image2d(unsigned nrows, unsigned ncols);
5    unsigned nrows, ncols;
6    T** data;
7  };
8
9  typedef unsigned char INTU8;
10 typedef float FLOAT;
11
12 template<class T>
13 image2d<T>* mean(const image2d<T>* ima1, const image2d<T>* ima2)
14 {
15   unsigned row, col;
16   image2d<T>* ima = new image2d<T>(ima1->nrows, ima1->ncols);
17   for (row = 0; row < ima1->nrows; ++row)
18     for (col = 0; col < ima1->ncols; ++col)
19       ima->data[row][col] = (ima1->data[row][col] + ima2->data[row][col]) / 2;
20   return ima;
21 }
```

D'une manière simple, nous avons le même type de définition que pour le paradigme avec les *macros*. En revanche, contrairement à l'approche précédente, nous n'avons pas besoin d'instancier manuellement les types de données. Les types sont définis implicitement et à la volée[1] par le compilateur. Il en est de même pour les fonctions. Voici un exemple d'utilisation :

```
1  int main()
2  {
3    image2d<INTU8>* ima1 = new image2d<INTU8>(128, 128);
4    image2d<INTU8>* ima2 = new image2d<INTU8>(128, 128);
5    image2d<INTU8>* ima = mean(ima1, ima2);
6    return 0;
7  }
```

Le compilateur instancie donc les types, les fonctions et vérifie la cohérence du typage.

Les bibliothèques *Qgar*(148) et *CImg*(42) utilisent ce mécanisme.

Qgar

La bibliothèque *Qgar* (148; 173; 174) repose sur ces patrons de conceptions. Elle est spécialisée dans l'analyse de documents. Seules les images en dimension deux sont gérées. La structure des images est paramétrée par le type des données. Deux types de politiques sont proposées pour l'accès aux données : la première ne fait aucune vérification dans un souci de performance, tandis que la seconde vérifie que l'accès à un point de l'image est bien valide. L'itération sur les points de l'image se fait pour des boucles imbriquées.

[1] Notons qu'il est possible en C++ de forcer une instanciation manuelle des patrons de conception.

Cimg

La bibliothèque Cimg (42), pour "Cool Image", gère des images en dimension 4. La structure de ces images est paramétrée par le type des données qu'elle contient. Elle propose un système de macros qui permet d'éviter l'écriture des boucles imbriquées et de gérer les accès aux pixels

D.3.2 Le modèle à la STL : Vigra

Jusqu'à présent, nous avons cherché à être génériques vis-à-vis des types de données. Les exemples étaient limités aux images à deux dimensions. Nous présentons maintenant les approches inspirées de la "Standard Template Library"(STL) (133) qui fait partie maintenant de la bibliothèque standard du C++ (170). La STL a largement été inspirée par les travaux de Stepanov *et al.* décrits dans (166). Nous présentons brièvement l'approche prônée par la STL, puis nous décrivons brièvement la bibliothèque Vigra (181).

Codage à la STL

La première difficulté à résoudre concerne le parcours des pixels de l'image. En effet, selon la dimension de l'image (2 ou 3 dimensions), il faut 2 ou 3 boucles imbriquées (une pour chaque dimension). Pour obtenir la généricité vis-à-vis des dimensions de l'image, il faut que la mise en œuvre des structures d'images soient cachées pour celle ou celui qui écrit un algorithme. La solution, initialement proposée par Lawton *et al.* dans (113), réside dans l'utilisation d'*itérateurs*. Un *itérateur* est un objet qui possède les méthodes `begin()` et `end()`, qui renvoient respectivement le premier et le dernier pixel. A partir d'un itérateur, il est possible d'accéder à la valeur pointée en utilisant l'opérateur de déréférencement *. L'opérateur ++ permet de passer au point suivant. Pour chaque type d'image, la structure image doit fournir son type d'itérateur. Par exemple, la définition d'une image en dimension 2 avec itérateur est la suivante :

```
1  template<class T>
2  struct image2d
3  {
4    typedef image2d_iterator<T> iterator;
5    unsigned nrows, ncols;
6    T **data;
7    iterator begin();
8    iterator end();
9    // ...
10 };
```

Une définition similaire est écrite pour des images en dimension 3. Le code pour mettre à zéro tous les points d'une image est alors le suivant :

```
1  template<class Iter>
2  void set_zero(Iter first_, Iter last_)
3  {
4    for (Iter p = first_; p != last_; ++p)
5      *p = 0;
6  }
7
```

```
8 // l'image ima est supposée être déclarée
9 set_zero(ima.begin(), ima.end());
```

Quand la fonction `set_zero` est appelée, la fonction est instanciée par le compilateur et l'itérateur `p`, de type `image2d_iterator`, est utilisé pour parcourir les points de l'image. Une seule boucle est nécessaire pour effectuer le parcours. En effet, la paramétrisation des fonctions (en utilisant le mot clef `template`) et la déduction de types (en utilisant le mot clef `typedef`) est gérée par le compilateur - autrement dit, la gestion des types est dite statique.

Nous pouvons donc écrire des algorithmes qui sont complètement génériques, bien typés et aussi rapides que du code écrit en C (puisque tout le travail est réalisé par le compilateur, il n'y a pas de surcoût à l'exécution). Nous rappelons qu'en programmation orientée objet classique, la plupart du travail est réalisé à l'exécution (utilisation de l'héritage et du polymorphisme d'pérations grâce au mot clef `virtual`) et conduit à de piètres performances pour le calcul scientifique comme énoncé par Köethe dans (111).

La STL possède d'autres concepts que les itérateurs. Les quatre autres concepts importants pour obtenir de la généricité sont les suivants :

- Les **conteneurs** permettent de gérer les structures de données.
- Les **algorithmes** permettent de gérer les procédures de calculs.
- Les **objets fonctions** se comportent et peuvent s'utiliser comme des fonctions.
- Les **concepts** fournissent les propriétés que doivent remplir les types pour être utilisés par un algorithme.

Dans cette thèse nous reviendrons sur les concepts d'**algorithmes** et d'**objets fonctions** dans le chapitre suivant afin de définir des algorithmes génériques.

Nous décrivons maintenant la bibliothèque Vigra.

Vigra

La bibliothèque Vigra (181), pour "Vision with Generic Algorithm", est dédiée aux images en deux dimensions et repose sur une approche de type STL. La structure des images est paramétrée par le type des valeurs que les pixels contiennent. Elle utilise des itérateurs comme ceux de la STL pour parcourir les images. En revanche, ils sont utilisés à travers l'usage d'accesseurs. Ces accesseurs permettent, par exemple, de ne modifier qu'une composante d'un vecteur. Les fonctions qui mettent en œuvre les algorithmes sont donc paramétrées par le type des différents itérateurs. Le code pour la fonction `mean` est le suivant :

```
1  template <class SrcIterator1, class SrcAccessor1,
2            class SrcIterator2, class SrcAccessor2
3            class DestIter, class Dest Acc>
4  void mean(SrcIterator src1, SrcIterator srcend1, SrcAcc srcacc1,
5            SrcIterator src2, SrcIterator srcend2, SrcAcc srcacc2,
6            DestIter dest, DestIter desAcc)
7  {
8     for (; src != srcend; ++src1, ++src2, ++dest)
9       destacc.set((srcacc(src1) + srcacc(src2))/2, dest);
10 }
```

Cette approche est complètement générique. Bien que cette bibliothèque ne propose que des structures d'images à deux dimensions, une extension aux cas des signaux et des images en 3 dimensions ne présentent aucune difficulté majeure.

D.3.3 Horus

Cette bibliothèque gère les images 1D, 2D et 3D. Les structures utilisent le mécanisme de patrons de conception pour contenir différents type de données. Chaque type d'image possède une fonction membre `signature` qui donne une description des caractéristiques du type de l'image (dimensions de l'image, type des valeurs des pixels...) Ces informations sont utilisées par tous les algorithmes qui sont appliqués à une image. Ces algorithmes vérifient que l'image à traiter possède bien les bonnes propriétés avant d'effectuer les calculs. Contrairement aux autres bibliothèques, les algorithmes sont mis en œuvre sous forme de méthodes. C'est un sévère inconvénient. En effet, l'ajout d'un algorithme nécessite de modifier le type des images (car une méthode est ajoutée). Cette modification du noyau de la bibliothèque (modification des types élémentaires de la bibliothèque) pour ajouter une nouvelle fonctionnalité brise la modularité de la bibliothèque.

D.3.4 Olena

A notre connaissance, c'est la bibliothèque qui offre le plus de généricité. Elle gère les images en 1, 2 et 3 dimensions et elle est générique vis-à-vis du type des données des pixels. Chaque type d'image correspond à une mise en œuvre particulière. Les algorithmes sont écrits de manières génériques. Cela signifie que les algorithmes fonctionneront sur des images de dimensions supérieures à 3 ou des graphes quand ces types d'images seront mis en œuvre.

Comme la dimension fait partie du type de l'image, il est possible de spécialiser une fonction pour des images d'une dimension fixée. Les erreurs d'affectation tant au niveau du type des valeurs qu'au niveau de la dimension sont détectées à la compilation. Par exemple, le code suivant produit une erreur.

```
1 image2d ima;
2 ima(x,y,z) = 1;  // erreur
```

En outre, *Olena* dispose d'un système de types qui propose des types pour encoder les données de manières sûres. Ce système propose notamment un type `int8` pour encoder des valeurs entières non signées sur 8 bits. La principale différence avec les types prédéfinis par la langage C++ est la suivante : les opérations effectuées sur ces types sont sûres de produire le résultat escompté (pas de débordement de capacité, par exemple). Par exemple, si un dépassement de capacité est observé, l'utilisateur sera prévenu. C'est un avantage non négligeable pour le développement et la réduction d'erreurs. D'autres comportements pour gérer les dépassements de capacité sont proposés. Outre le cas de l'erreur, le comportement peut-être modulaire (pour modéliser $\mathbb{Z}/256\mathbb{Z}$) ou absorbant (quand la limite est atteinte, elle reste à cette valeur).

Olena fournit des itérateurs. En revanche, ils ne se comportent pas comme ceux de la STL. En effet, un itérateur dans *Olena* correspond à itérer sur l'ensemble de points de l'image. Autrement-dit, un itérateur ne fait pas référence à un *p*ixel mais uniquement à un site ; la valeur de l'image à ce site n'est pas prise en compte. Un tel itérateur est donc utilisé pour lire ou modifier la valeur d'un pixel à un site. Le principal avantage de ce type d'itérateur est que le même itérateur peut être utilisé pour parcourir des images différentes qui ont le même support. L'opérateur `[]` sert d'accesseur aux pixels de l'image.

Olena repose sur un système de hiérarchie en diamant appelé SCOOP, présenté par Burrus *et al.* dans (28). Ce système permet de définir les images par des propriétés. Il est alors

possible de spécialiser les algorithmes en fonction de ces propriétés. Par exemple, les propriétés suivantes sont définies : `vectorial_image` pour préciser que l'image est vectorielle, `image2d` pour signifier que l'image est de dimension 2. . . Dans la signature d'un algorithme apparaît donc la propriété que doit remplir une image pour que l'appel puisse être réalisé. Par exemple, le code pour l'algorithme `mean` qui n'accepte que des images dont les valeurs des pixels sont des scalaires, est le suivant :

```
1  template <class I>
2  image<I> mean(const non_vectorial_image<I>& ima1,
3                const non_vectorial_image<I>& ima2)
4  {
5    assert(ima1.size() == ima2.size());
6    oln_concrete_type(I) output(ima1.size())
7    oln_iter_type(I) p(input)
8    for_all(p)
9      output[p] = (ima1[p] + ima2[p])/2
10   return output
11 }
```

La signature de l'algorithme `mean` spécifie la propriété que doivent satisfaire les images `ima1` et `ima2`, mais le type de ces images (`ima1` et `ima2`) n'est pas le type réel. Un mécanisme à base de méta-programmation (46) (un méta-programme est un programme dont l'exécution est réalisée au moment de la compilation) permet de récupérer ce type réel. Un système de macros permet de cacher cette difficulté au programmeur. Ces macros sont utilisées dans les lignes 5 et 6. La macro `oln_concrete_type` (ligne 5) sert à définir le type de l'image de sortie (déduit à partir de du type de l'entrée), tandis que la macro `oln_iter_type` (ligne 6) définit le type de l'itérateur.

Nous avons présenté différentes approches qui permettent de prendre en compte, dans les algorithmes, la diversité des images en terme de structures et de valeurs associées aux pixels. Nous nous focalisons maintenant sur la manière d'écrire les algorithmes.

D.4 Composition d'algorithmes

Désormais l'accent sera mis sur les caractéristiques des algorithmes utilisés en traitement des images et vision par ordinateur, plutôt que sur les types de structure d'images utilisés par ces algorithmes.

C'est un problème qui a déjà été abordé par d'Ornellas dans (67), qui a ainsi montré que la majorité des algorithmes de morphologie mathématique pouvait se mettre sous forme de canevas. Intuitivement, un canevas est un patron qui permet de simplifier l'écriture d'un algorithme en exhibant l'essence d'une classe d'algorithmes.

Tout d'abord, il faut rappeler que d'un point de vue fonctionnel, un algorithme est une routine qui travaille sur des données, fournies en entrée, et qui produit un résultat donné en sortie. Notre objectif est ici de pouvoir manipuler de manière abstraite - c'est-à-dire sans préciser les types des entrées/sorties - des algorithmes : nous appellerons alors ces algorithmes des *algorithmes abstraits ou composants*. La première partie (sous-section D.4.1 et D.4.2) de cette partie sera consacrée à la présentation des concepts de base nécessaires à la fabrication et à la composition de ces éléments. La seconde partie décrit les différents types qualitatifs de compositions possibles entre ces briques élémentaires.

D.4.1 Contraintes du praticien et du développeur

Prenons un exemple simple qui illustre les besoins d'un utilisateur.

```
1 Image1_type ima1;
2 Image2_type ima2 = algo2((algo1(ima1)));
3 save(ima2);
4 save(algo3(ima2));
```

Cet exemple est représentatif de l'utilisation d'un traitement. Le point principal est présenté sur la ligne 2 où le type de l'image qui doit stocker le résultat est défini par `Image2_type`. En effet, la détermination du type de l'image retourné par le traitement `algo2((algo1(ima1)` appliqué à l'image `ima1` n'est pas immédiat.

Un langage gérant l'inférence de types (tel que OCaml (188)) résout très facilement ce problème car le langage est orienté pour gérer ces types d'affectation. Ceci est rendu possible car le compilateur intègre un système d'inférence de types. Notre but est de fournir un système en C++ qui permet d'écrire facilement ce type de lignes.

D.4.2 Chaîne de traitements

La plupart des algorithmes de traitement des images et de vision par ordinateur peuvent être naturellement décomposés en blocs. Prenons un exemple simple où trois blocs sont présents :

```
initialisation                // Premier bloc
pour chaque point de l'image  // Deuxième  bloc
   faire un travail               // Variation
finalisation                  // Troisième bloc
```

Comme nous pouvons le voir, cet algorithme se représente comme la composition (concaténation) de blocs simples (qui sont aussi des algorithmes) indépendants les uns des autres. En concaténant ces blocs nous avons construit un algorithme de plus haut niveau.

Outre la concaténation, cet exemple présente également un autre type de composition : la variation. Une variation est une partie d'un algorithme qui est spécifique à un problème donné. Ici, par exemple, la classe d'algorithme est celle des algorithmes qui travaillent sur chaque point de l'image. En revanche, le traitement effectué sur chaque point dépend du problème : c'est la variation.

D.4.3 Compositions d'algorithmes

Ces deux types de composition que nous venons de présenter ne sont pas nouveaux : en effet, Goguen les présentait déjà dans (79). La concaténation permet de composer les algorithmes au sens usuel (i.e mathématique) tandis que l'autre correspond à un affinage et permet de gérer les variations.

Composition horizontale

La composition d'algorithmes est fréquemment utilisée en traitement des images, au sens où des traitements sont appliqués successivement. La figure D.2 présente le principe d'une telle composition d'algorithmes. On peut voir une telle composition comme une fonction qui associe à un couple de traitements, un traitement.

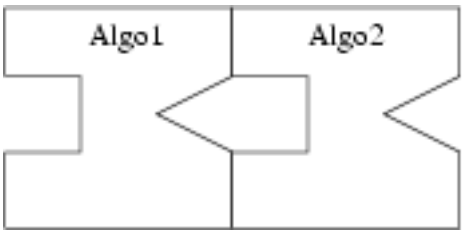


FIG. D.2 – Composition horizontale d'algorithmes.

Composition verticale

L'autre type de composition (la variation) permet de raffiner les traitements d'un algorithme de haut niveau. Ce principe permet de rendre un algorithme **adaptatable**. Prenons l'exemple d'un algorithme de tri : la variation pour ce dernier serait la relation d'ordre, ce qui permettrait de trier les éléments selon un critère donné par l'utilisateur. Le corps de l'algorithme demeure inchangé. La figure D.3 illustre le schéma de cette composition.

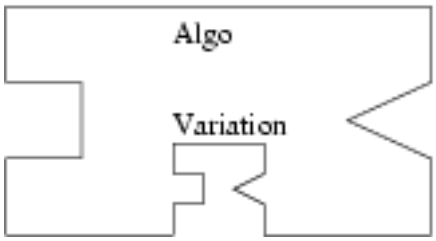


FIG. D.3 – Composition verticale d'algorithmes. L'algorithme est paramétré par sa variation.

Cette manière d'agencer ces algorithmes n'est pas sans avantages. Tout d'abord, ce type d'approche permet d'éviter le problème du passage à l'échelle avec ce type d'approche. En effet, le fait de pouvoir facilement concaténer les algorithmes et de pouvoir gérer facilement les variations que présente une classe d'algorithmes, permet d'enrichir une bibliothèque logicielle sans redondance de code. Cette manière de décrire un algorithme est expressive puisque ces compositions reflètent exactement la manière dont fonctionne cet algorithme.

En revanche, les différentes classes d'algorithmes et les variations associées doivent être exhibées en fonction du domaine scientifique abordé. En outre, une mise en œuvre doit être proposée pour réaliser en pratique ces compositions. Nous rappelons que le mécanisme de composition doit être facile à utiliser pour une personne non informaticienne.

Jusqu'à présent, nous n'avons pas évoqué le problème de typage des algorithmes et de leurs compositions. En effet, les types des données en entrée et des données en sortie doivent passer d'un algorithme à l'autre. Cette propagation de types doit être automatique, sans quoi l'intérêt de la composition deviendrait nulle. Cette manière de procéder a de plus l'avantage de rendre le code source plus sûr. Pour que la propagation des types puissent se faire, il faut que l'assemblage des compositions ait lieu durant la compilation du programme.

Nous présentons dans la partie suivante, les solutions classiques pour mettre en œuvre les principes évoqués ici.

D.5 Les solutions classiques

Outre la présentation des solutions classiques, nous discutons de leurs avantages et de leurs inconvénients. Dans toute la partie nous prendrons l'exemple de la composition de deux fonctions pour illustrer la composition d'algorithmes. Plus précisément, pour la clarté de l'exposé, seules les fonctions unaires seront considérées, et ce, sans perte de généralité ; en effet, nous pouvons toujours concaténer plusieurs variables en une seule. Nous verrons que ce cas simple permet d'exhiber des cas délicats. Nous notons `F` et `G` les deux fonctions unaires considérées. Le code source de cet exemple est donné en C++ avec la convention qu'un texte entre guillemet "..." indique qu'une partie du code source a été omise.

Dans la suite, nous supposons que nous disposons d'un système de *"traits"* originalement proposé par (134). Un "trait" fait référence à une caractéristique associée à un type. C'est un mécanisme qui peut être utilisé pour faire de la déduction de type. Pour notre exemple, nous supposons que nous mettons en place un tel mécanisme qui associe le type de sortie d'un algorithme en fonction du type de ses entrées. Dans la suite, nous notons le mécanisme de déduction `Output_Algo_Trait`. L'obtention du type de retour d'un algorithme en fonction de son entrée `Input` est notée `Output_Algo_Trait<Input>::Ret`.

Nous présentons maintenant les différentes solutions existantes pour le problème de la composition.

D.5.1 Les fonctions "templates"

Rappelons qu'il est naturel de représenter un algorithme par une fonction (129). Dans un souci de généricité, une telle fonction est paramétrée par le type des entrées de l'algorithme. Ce qui nous conduit à écrire le code suivant :

```
template <class Input>
typename Output_Twice_Trait<Input>::RET twice(const Input & x)
  { return 2*x }
```

Étant donné qu'il n'est nul besoin de préciser le type des paramètres pour utiliser une telle fonction, cette solution est particulièrement pratique car générique. En outre, le compilateur infère automatiquement le type des entrées et génère le type de sortie grâce au "trait" associé à l'algorithme.

Cependant, cette solution présente un inconvénient majeur puisqu'il est impossible de passer une fonction paramétrée à une autre fonction paramétrée par un type de donnée. En d'autres termes, cela signifie que le polymorphisme de fonction d'ordre supérieur ne peut pas être exprimé par ce procédé.

L'opération de composition s'écrit donc ainsi : `F((G(input)))` et le type de retour est donné par la ligne suivante :

```
typename Output_F_Trait< typename Output_G_Trait<Input>::RET>::RET
```

En d'autre termes, le type de retour de cette composition est le type de retour de `F` dont le type en entrée est celui donné par le type de retour de `G` ; et où le type d'entrée de `G` est `Input`. Un utilisateur doit écrire cette dernière ligne à chaque fois qu'il désire effectuer une composition. Remarquons que le type de retour de plusieurs compositions imbriquées devient assez vite

très pénible à écrire.

Nous présentons maintenant une solution qui repose sur la notion d'objets fonctions.

D.5.2 Le codage à la "STL"

Le langage C++ possède la capacité de créer des objets qui se comportent comme des fonctions. Cela est rendu possible en surchargeant l'opérateur "()". Ces objets particuliers sont appelés *objets fonctions*. La bibliothèque standard du C++, largement inspiré par la bibliothèque "Standard Template Library" (STL) (170), a popularisé ce type d'approche. La mise en œuvre est immédiate :

```
template <class Input, class Output>
struct Twice : public std::unary_function<Input, Output>{
   Typedef Output Result;
   Result operator()(const Input & x) { return 2*x; }
};
```

Dans ce cas, les types d'entrée et de sortie doivent être fournis *explicitement* par le programmeur avant l'appel de la fonction. La STL fournit un opérateur de composition `compose1` pour les fonctions unaires. La composition s'écrit ainsi :

```
typedef G<Input, Output_G_Trait<Input>::RET > G_T ;
compose1(F< G_T::Result, Output_F_Trait< G_T::Result> >(), G_T())
```

Encore une fois, la déduction du type de retour doit être précisée par l'utilisateur. Au final remarquons que cette méthode proposée par la STL requiert que le programmeur donne explicitement la signature de la fonction. Le fait de donner le type en entrée pour un algorithme est contraignant mais toujours possible. En revanche, le fait de devoir donner la signature de la fonction est une contrainte rédhibitoire dans le cadre qui nous intéresse, puisque le compilateur pourrait le faire automatiquement. Enfin, nous remarquons également que le type de retour de cette composition n'est pas une fonction paramétrée. Ceci empêche tout espoir de réutiliser le résultat de cette composition (qui est une fonction) dans une nouvelle composition.

Nous présentons maintenant une solution à notre problème beaucoup plus élégante et adéquate.

D.5.3 Les foncteurs polymorphes directs

Rappelons que nous sommes intéressés par la composition de fonctions en C++ afin que le résultat soit polymorphique, comme dans un langage fonctionnel. McNamara and Smaragdakis présentent dans (122; 163) une bibliothèque nommée FC++ qui fournit une solution au problème de la composition et qui offre la propriété de polymorphisme. Le but de cette bibliothèque est d'émuler un comportement fonctionnel en C++. La solution repose l'utilisation de classe paramétrée imbriquée. Afin de mettre évidence la sémantique de la fonction, la méthode déclare explicitement la signature d'une fonction. Ceci est atteint en paramétrant l'opérateur parenthèse, `operator()`, et en incluant un système de trait dans la classe. Le code pour la fonction `twice` est le suivant :

```
  struct Twice {
    template <class Input>
    struct Sig
    {
       typedef Input FirstArgType;
       typedef typename OutputTwiceTrait<Input>::RET   ResultType;
    };
    template <class Input>
    typename Sig<Input>::ResultType operator()(const Input & x)
    {  return 2*x; }
  } twice;
```

L'encodage de la signature est réalisé par l'utilisation de la classe imbriquée `Sig`. Cette dernière joue le rôle du "trait", et définit donc le type d'entrée et de retour. Ces informations permettent de déduire les types comme un langage fonctionnel le fait. Remarquons que l'opérateur "()", `operator()` , est défini comme pour les fonctions paramétrées.

Puisque l'objet fonction contient également sa signature, il est envisageable de passer un objet fonction à un objet fonction d'ordre supérieur. En effet, une fonction d'ordre supérieur doit être capable de déduire sa signature à partir des objets fonctions en entrée. Puisque qu'une fonction d'ordre supérieur doit être considérée comme une fonction classique, la *même* forme de construction doit être utilisée. Nous insistons sur le fait que le type de retour doit être une fonction. Voici le code pour effectuer une composition de deux fonctions :

```
struct Compose
{
  ... // Definition de la signature Sig

  template <class F, class G>
  Composer<F,G> operator()(const F &f, const G & g)
  { return Composer<F,G>(f,g); }
} mycompose;
```

Le type de retour de la composition est une instance d'un objet qui est construit à partir des fonctions f et g, données en entrée. L'opérateur parenthèse `operator()` de cet objet effectue le calcul de la composition $g(f(.))$. Le code pour cet objet est le suivant :

```
template <class F, class G>
struct Composer
{
  Composer( const F& ff, const G& gg ) : f(ff), g(gg){}
  const F &f;
  const G &g;

  ... // Definition of Sig

  template <class X> typename Sig<X>::ResultType
  operator()( const X& x ) const { return f( g(x) ); }
};
```

Remarquons que cette manière d'effectuer la composition implique *nécessairement* l'écriture de deux classes disjointes. En effet, la classe `Compose` est incapable de tenir les fonctions f et g puisqu'elle ne les connaît pas (les fonctions ne sont connues qu'au moment de l'appel de la fonction avec l'opérateur parenthèse).

Nous voyons maintenant comment utiliser cette approche. Pour effectuer l'appel de la fonction `twice` sur l'entier 3, l'utilisateur écrit simplement `twice(3)`. Pour effectuer la composition double de `twice` sur l'entier 3, il écrit `compose(twice, twice)(3)`. Cette approche a été conçue pour faciliter l'écriture des compositions pour les utilisateurs. Ce type d'écriture est celui prôné par l'approche fonctionnelle (language OCaml par exemple). En revanche, si l'utilisateur désire stocker le résultat dans une variable alors il doit écrire *explicitement* la demande de déduction de type, en respectant les étapes intermédiaires. Par exemple, pour une simple composition de deux fonctions, il doit écrire le code suivant pour stocker le résultat dans une variable i :

```
MyCompose::Sig<Twice, Twice>::ResultType::Sig<int>::ResultType i=
            mycompose(twice, twice)(3);
```

Il faut remarquer que le code à écrire pour récupérer le type de sortie après plusieurs composition est long. Plusieurs lignes sont nécessaires pour déterminer le type de retour de la composition `compose(compose(twice, twice),compose(twice, twice))`.

Nous allons présenter notre solution dans la partie suivante.

D.6 Notre solution à base de déduction de type et de classe imbriquée

Notre solution suit le travail de McNamara and Smaragdakis. En effet, nous gardons l'idée de leur solution pour améliorer la signature des fonctions en C++. Rappelons que notre but est de proposer une approche où le code à écrire est simple, à la fois pour un utilisateur et un programmeur. Remarquons qu'un utilisateur connaît toujours le type de l'image qu'il souhaite traiter. A partir de point d'entrée, la déduction des types de sorties et des types intermédiaires est automatique.

L'écriture et l'utilisation d'algorithmes sont à peu près du même ordre de difficulté que FC++. Notre solution repose également sur l'utilisation de classe imbriquée. Contrairement à la méthode de McNamara et Smaragdakis, une classe imbriquée définit à la fois le "trait" pour le type de retour et l'opérateur parenthèse. Cette classe imbriquée est paramétrée par le type d'entrée de l'algorithme. Une méthode similaire à celle de FC++ est utilisée pour effectuer la déduction de type.

Contrairement à ce qui est généralement fait, nous définissons un algorithme en deux étapes (d'où les classes imbriquées). La première étape gère une fonctionnalité "abstraite" d'un algorithme sans prendre en compte le type des données en entrées. Par exemple, *twice* est la notion abstraite ; c'est le fait de doubler une quantité et ce, peu importe le type de cette quantité. La deuxième étape sert à spécialiser un algorithme "abstrait" pour un type d'entrée particulier, par exemple, la fonction prend en entrée des entiers. Évidemment, seuls les algorithmes spécialisés sont capables d'effectuer les calculs. Les avantages d'une telle construction sont les suivants : d'une part, ces deux types d'algorithmes sont des *types*. Cela signifie qu'un utilisateur ou un programmeur peut les référencer à la compilation en utilisant

le mot clef **typedef** du C++. D'autre part, une fois qu'un algorithme est défini de manière abstraite - c'est-à-dire sans avoir précisé le type des entrées - et qu'il a été spécialisé pour une entrée particulière, l'obtention du type de retour est immédiate : il est donné par le membre `Output`. Remarquons que cette approche se situe entre le style de programmation de STL et celui de FC++.

Nous détaillons maintenant notre solution pour la composition. Nous montrons également les différences entre l'approche proposée par FC++ et la nôtre.

D.6.1 Le cas d'une fonction simple

Nous commençons avec l'exemple de la fonction `twice`. Rappelons-nous la figure D.1 où un algorithme est présenté comme une fonction qui prend des entrées et déduit son type de retour à partir du type de ces entrées une fois que celles-ci sont connues. Notre solution reflète ce diagramme en définissant le membre `Output` à l'intérieur d'une classe imbriquée qui est paramétrée par le type de son entrée. Nous obtenons le code suivant :

```
struct twice {
  template <class Input>
  struct toReal {
    typedef ... Output;
    Output operator()(const Input & x)
    { return 2*x; }
  };
};
```

Comme nous pouvons le voir, la seule différence entre notre solution et celle de FC++ est que la classe imbriquée définit à la fois le type de retour et l'opérateur parenthèse, `operator()`. Nous appelons cette classe imbriquée `toReal` puisqu'elle fait référence à l'action de spécialiser un algorithme "abstrait". Contrairement à FC++, cette solution impose d'écrire *explicitement* le type d'entrée de l'algorithme avant de pouvoir l'utiliser. En revanche, le type de retour est facilement obtenu par le membre `Output`. Voici un exemple d'utilisation :

```
typedef twice::toReal<int> algo;
std::cout << algo()(3) << std::endl;
algo::Output i = algo()(3);
std::cout << i << std::endl;
```

Comme nous pouvons le voir, l'appel à une fonction simple comme `twice` n'est pas aussi pratique que dans la bibliothèque FC++. En revanche, la classe d'algorithmes est désormais désignée par un type. Nous montrons maintenant l'avantage que cela apporte.

D.6.2 Composition et algorithmes avec variations

Nous présentons maintenant notre méthode pour la composition. Nous rappelons la figure D.3 qui présente le type de composition verticale, ce qui correspond au cas d'un algorithme présentant une variation. Remarquons que ce diagramme est récursif : en effet, la variation a le même diagramme que l'algorithme lui-même.

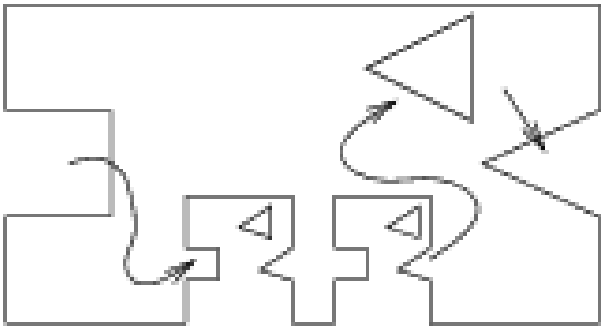


FIG. D.4 – Diagramme pour une composition en utilisant notre solution. Le type de retour est décrit par un triangle. Il est déduit et stocké par chaque sous algorithme. Les lignes fléchées représentent la propagation des types.

Le diagramme pour la composition de chaînage d'algorithmes devrait également présenter ce motif récursif ; mais le diagramme de la figure D.2, qui présente cette composition, n'est pas récursif. Nous présentons en figure D.4 un autre diagramme pour la composition avec la propagation des types. Comme nous pouvons le voir, la seule différence entre une composition horizontale (concaténation) et verticale (variation) est que deux sous-blocs sont présents dans la version horizontale alors qu'un seul est nécessaire pour la version verticale. Autrement dit, il n'y a pas de différence intrinsèque entre les deux versions.

Contrairement à une fonction simple, une fonction d'ordre supérieur est directement paramétrée par les fonctions internes qui la compose. Pour la composition des fonctions `F` et `G` on a donc le code suivant pour la première étape :

```
template<class F, class G>
struct compose {...};
```

Le résultat de la composition de deux fonctions unaires est donc un type. Nous écrivons `compose<twice,twice>` pour la composition de `twice` par `twice`. Ce type est maintenant utilisable comme une fonction simple (en fait, c'est même une fonction simple). La structure doit définir le type réel des fonction `F` et `G` et doit également définir le type de retour. Enfin, l'opérateur parenthèse `operator()` définit le calcul à effectuer. Le code correspondant à la deuxième étape est le suivant :

```
template<class Input>
struct toReal
{
  typedef typename G::toReal<Input>                    _G;
  typedef typename F::toReal<typename _G::Output>      _F;
  typedef typename _F::Output                          Output;
  Output operator()(const Input& x)
  { return _F()(_G()(x)); }
};
```

Contrairement à FC++, nous n'avons pas besoin de découper le code de la composition en deux parties. Ceci est possible car par abus de langage on peut dire que les algorithmes sont

des types. On notera que les instances des algorithmes ne sont créées que lorsque les calculs sont effectués.

Puisque tous les algorithmes sont des types, un utilisateur peut utiliser le mot clef **typedef** du C++ pour manipuler les algorithmes sans prendre en compte le type des entrées. Par exemple, nous pouvons écrire :

```
typedef compose<twice, twice> Algo1_;
typedef compose<twice, twice> Algo2_;
typedef compose<Algo2, Algo1_> FinalAlgo_;
```

Une fois qu'un algorithme est écrit, l'utilisateur précise pour quels types en entrée, l'algorithme doit s'instancier. Pour instancier l'algorithme de l'exemple précédant, `FinalAlgo_`, avec une entrée de type entier, nous écrivons alors :

```
typedef int Input;
typedef FinalAlgo_::toReal<Input> FinalAlgo;
FinalAlgo::Output i = FinalAlgo()(3);
```

Peu importe la définition de l'algorithme (i.e, le nombre de combinaisons qu'il a fallu faire pour le définit), le type de retour est toujours donné par le "trait" `Output`.

Nous avons donc réussi à définir un mode de composition qui permet de créer des fonctions d'ordre supérieur. Avec notre méthode, le type de retour des algorithmes est facile à obtenir. En revanche, les algorithmes doivent être instanciés manuellement pour le type d'entrée que l'on souhaite utiliser.

Enfin remarquons que nous n'utilisons pas le mot clef **virtuel** du langage C++. Par conséquent la résolution des appels de fonctions et méthodes est réalisée à la compilation. L'optimiseur du compilateur est donc capable d'enlever ces appels de fonctions (*inliner* en anglais), et donc de produire un code efficace (contrairement à une résolution dynamique des appels).

D.6.3 Objets pratiques dédiés au traitement des images

Contrairement à FC++, notre solution permet de définir des objets qui sont capables de *stocker* des variables. Par exemple, supposons que l'on désire calculer le minimum d'une séquence d'éléments. L'objet fonction, utilisant notre approche, qui met en œuvre cet algorithme est donné ci-dessous :

```
struct findMin {
  template <Input>
  struct toReal {
    toReal(): count(0)
    void reset() { count = 0;}
    void operator()(const Input & x)
    {
      if (count == 0) { min =  x;}
      else {
        if (x < min) min = x;
      }
```

```
    }
    int count;
    Input min;
  };
};
```

Nous rappelons que ce code est écrit une fois pour toute et est réutilisable.

D.7 Canevas classiques en traitement des images

Nous présentons maintenant quelques canevas utiles pour le traitement des images. Le code est disponible à l'adresse suivante :http://jerome.berbiqui.org/these/. Note mise en œuvre utilise une bibliothèque générique écrite en C++ par le Laboratoire de Recherche et Développement de l'Epita.

D.7.1 Canevas d'algorithmes parallèles

Le canevas d'algorithmes parallèles correspond à la classe d'algorithmes suivante : une fonction n-aire est appliquée sur chaque pixel des n images. L'ordre de parcours n'intervient pas dans le résultat. Le complément, la valeur absolue, le cosinus, la projection, la conversion sont quelques exemples d'algorithmes qui illustrent le canevas parallèle où la fonction est unaire. L'addition, le maximum, le produit scalaire illustrent le cas d'une fonction binaire. On suppose que l'on dispose d'un objet `init` qui initialise une image avec la politique suivante : `init(ima1, ima2)` initialise l'image `ima1` avec les dimensions de `ima2`.

Une mise en œuvre en C++ est la suivante :

```
template <class OP>
struct parallelPattern {

  template <class Input>
  struct toReal {

    typedef typename OP::template toReal<typename Input::Datatype> OPreal;
    typedef image<typename OPreal::Output> Output;

    Output operator()(const Input & ima)
    {
      Output output(ima.get_size());
      fwd_iter p(ima);
      for (p=begin; p<=end; ++p)
        output[p] = OPreal()(ima[p]);
      return output;
    }

    Output operator()(const Input & ima1, const Input & ima2)
    {
      Output output(ima1.get_size());
      fwd_iter p(ima1);
      for (p=begin; p<=end; ++p)
        output[p] = OPreal()(ima1[p], ima2[p]);
```

```
      return output;
    }

  };
};
```

Cette classe d'algorithmes est paramétrée par l'opération à effectuer sur les pixels `Operator`.

D.7.2 Canevas de convolution généralisée

Le canevas de convolution généralisée est définie de la manière suivante : les valeurs des pixels dans le voisinage d'un point p sont combinées avec les valeurs d'un noyau pour donner le résultat au point p. Le filtrage gaussien, l'érosion sont deux exemples de ce canevas.

On suppose que l'on dispose d'un objet `init` qui initialise une image avec la politique suivante : `init(ima1, ima2, kernel)` initialise l'image `ima1` avec les dimensions de `ima2` et ajoute un bord à `ima1` suffisamment grand pour que le résultat sur des points proches du bord soit correctement défini. Les valeurs des pixels sur ces points hors du domaine initial de l'image, dépendent du noyau utilisé. Le même objet `init` est utilisé ; ce nouveau type d'initialisation étant défini par surcharge de l'opérateur `()`. Le code est le suivant :

```
template <class OP, class INIT = init_ima_se_copy>
struct generalizedConvolutionPattern {

  template <class Input, class Kernel>
  struct toReal {

    typedef typename OP::template toReal<Input, Kernel> OPreal;
    typedef typename INIT::template toReal<Input, Kernel> Init;
    typedef image<typename OPreal::Output> Output;

    Output operator()(const Input & ima, const Kernel & kernel)
    {
      Output output(ima.get_size());
      Init()(ima, kernel);
      fwd_iter p(ima);
      for (p=begin; p<=end; ++p)
        output[p] = OPreal()(ima, point(p), kernel);
      return output;
    }
  };
};
```

D.7.3 Canevas d'opérations sur voisinage

Le canevas d'opérations sur voisinage est défini de la manière suivante : les valeurs des pixels dans le voisinage d'un point p sont utilisées pour donner le résultat au point p. Comme pour le canevas de convolution généralisée, on suppose que l'on dispose d'un objet `init` qui initialise correctement une image.Le canevas est le suivant :

```
template <class OP, class INIT = init_ima_se_copy>
struct neighborhoodPattern {
```

```
  template <class Input>
  struct toReal {

    typedef typename OP::template toReal<Input> OPreal;
    typedef typename INIT::template toReal<Input, neighb> Init;
    typedef image<typename OPreal::Output> Output;

    Output operator()(const Input & ima, const neighb & ngb)
    {
      Output output(ima.get_size());
      Init()(ima, ngb);
      fwd_iter p(ima);
      for (p=begin; p<=end; ++p)
        output[p] = OPreal()(ima, point(p), ngb);
      return output;
    }
  };
};
```

D.7.4 Canevas d'opérations séquentielles

Le canevas d'opérations séquentielles est défini de la manière suivante : comme pour le canevas des convolutions généralisées, les valeurs des pixels dans le voisinage d'un point p sont combinées avec les valeurs d'un noyau pour donner le résultat au point p. En revanche, le résultat dépend de l'ordre de parcours. Les filtres récursifs de Deriche (60) et les cartes de distances (18) sont quelques exemples qui utilisent ce canevas. Comme pour le canevas de convolution généralisée, on suppose que l'on dispose d'un objet `init` qui initialise correctement une image En outre, on suppose que l'on dispose de deux itérateurs sur l'image : le premier, `for_all_forward`, parcourt l'image dans l'ordre usuel (pour chaque ligne et pour chaque colonne), et le second, `for_all_backward`, qui parcourt l'image en sens inverse. On suppose de plus que l'on dispose d'une fonction `forward_neighb` qui, étant donné un voisinage, retourne les points du voisinage déjà traités dans un parcours en *forward*. Une telle fonction `backward_neighb` est définie d'une manière similaire pour le parcours en `backward`. Le canevas est le suivant :

```
neighb get_se_plus(const neighb& ngb)
{
  neighb output;
  neighb_iter n(ngb);
  for (n=begin; n<=end; ++n)
    if ((dpoint(n).get_irow()<0) ||
        ((dpoint(n).get_irow()==0) && (dpoint(n).get_icol()<=0)))
      output.add(n);
  return output;
}


neighb get_se_minus(const neighb& ngb)
{
  neighb output;
  neighb_iter n(ngb);
```

```
  for (n=begin; n<=end; ++n)
    if ((dpoint(n).get_irow()>0) ||
        ((dpoint(n).get_irow()==0) && (dpoint(n).get_icol()>=0)))
      output.add(n);
  return output;
}

template <class OP1, class OP2, class INIT = init_ima_se_copy_output_copy>
struct sequentialPattern {

  template <class Input>
  struct toReal {

    typedef typename OP1::template toReal<Input> OP1real;
    typedef typename OP2::template toReal<Input> OP2real;
    typedef typename INIT::template toReal<Input, neighb> Init;
    typedef image<typename OP1real::Output> Output;

    Output operator()(const Input & ima1,
                      const Input & ima2,
                      const neighb & ngb)
    {
      Output output(ima1.get_size());
      Init()(ima1, output, ngb);
      neighb ngb_plus = get_se_plus(ngb);
      neighb ngb_minus = get_se_minus(ngb);

      bool shall_continue = true;
      while (shall_continue == true)
        {
          shall_continue = false;
          output.set_border_copy();
          fwd_iter p(output);
          for (p=begin; p<=end; ++p)
            {
              typename Input::Datatype tmp = output[p];
              output[p] = OP1real()(output, ima2, point(p), ngb_plus);
              if ((shall_continue == false) && (tmp != output[p]))
                shall_continue = true;
            }
          bkd_iter bp(output);
          for (bp=begin; bp<=end; ++bp)
            {
              typename Input::Datatype tmp = output[bp];
              output[p] = OP2real()(output, ima2, point(bp), ngb_minus);
              if ((shall_continue == false) && (tmp != output[bp]))
                shall_continue = true;
            }
        }
      return output;
    }
  };
};
```

D.7.5 Canevas d'opérations à base de queue

Le canevas d'opérations à base de queue est défini de la manière suivante : une première passe sur l'image met dans une queue certains points. Ces points sont en suite utilisés comme point de départ de front de propagation. La propagation s'effectue grâce à l'utilisation de queues de priorité (186). Ce canevas est souvent utilisé pour mettre en œuvre des filtres morphologiques : les filtres morphologiques connectés (182), le calcul de la ligne de partage des eaux (186) et les opérateurs de reconstruction (184).

Nous supposons que nous disposons d'un type `queue` qui possède les méthodes suivantes : la méthode `end(p, val)` qui met dans la queue le pixel p avec la priorité *val*, la méthode `deq()` qui retourne le point de plus haute priorité, la méthode `pop()` qui enlève le point de plus haute priorité de la queue et la méthode `empty()` qui retourne une valeur binaire qui précise si la queue est vide ou non. Le canevas est le suivant :

```
template <
class COND1, class COND2,
class OP1, class OP2,
class INIT = init_watershed,
class QUEUE = a_priority_queue>
struct queuePattern {

  template <class Input1, class Input2>
  struct toReal {



    template <class T>
    struct cmp_queue_elt
    {
      bool
      operator()(const std::pair<T, point >& l,
                 const std::pair<T, point>& r) const
      {
        return l.first > r.first;
      }
    };

    typedef typename OP1::template toReal<Input1, Input2> OP1real;
    typedef typename OP2::template toReal<Input1, Input2> OP2real;
    typedef typename COND1::template toReal<Input1, Input2> Cond1real;
    typedef typename COND2::template toReal<Input1, Input2> Cond2real;
    typedef typename INIT::template toReal<Input1, Input2, Input2, neighb> Init;
    typedef Input2 Output;

    typedef std::pair<typename Input1::Datatype, point> queue_elt_t;
    typedef typename QUEUE::template toReal<queue_elt_t,
                                    cmp_queue_elt<typename Input1::Datatype> >
                                    QUEUEreal;


    Output operator()(const Input1 & ima1,
                      const Input2 & ima2,
```

```
                                const neighb & ngb)
    {
      Output output(ima1.get_size());
      Init()(ima1, ima2, output, ngb);

      QUEUEreal PQ;
      // Initialization
      {
        fwd_iter p(ima2);
        for (p=begin;p<=end;++p)
          if (Cond1real()(ima1, ima2, p, ngb))
            {
              OP1real()(ima1, output, p, ngb);
              PQ.push(queue_elt_t(ima1[p], p));
            }

      }

      // Propagation
      while (PQ.empty() == false)
        {
          point p = PQ.top().second;
          PQ.pop();
          neighb_iter dp(ngb);
          for (dp=begin;dp<=end;++dp)
            if (output.is_inner(p+dp))
              if (Cond2real()(ima1, output, p+dp, ngb))
                {
                  OP2real()(ima1, output, p, p+dp, ngb);
                  PQ.push(queue_elt_t(ima1[p+dp], p+dp));
                }
        }
      return output;
    }
  };
};
```

D.8 Conclusion

Nous avons proposé un système de mise en œuvre de canevas en C++. La mise en œuvre repose sur l'utilisation de la paramétrisation et de classes imbriquées. Le code généré est rapide puisque la composition est effectuée à la compilation. Un mise en œuvre est disponible à l'adresse suivante : http://jerome.berbiqui.org/these/.

Cette approche peut être étendue de plusieurs manières. Tout d'abord, un système de contrainte des paramètres tel que celui proposé par McNamara *et al.* dans (123). L'utilisation d'un langage dédié à l'écriture de ces canevas pourrait améliorer la lisibilité du code et donc des algorithmes mis en œuvre. L'utilisation d'un langage dédié au traitement des images est proposé par Cecchini *et al.* dans (32). Enfin, une mise en œuvre d'*expression templates*, comme par exemple celle de Veldhuizen (178), dédiée aux algorithmes de traitement des images permettrait d'améliorer les performances des algorithmes.

Bibliographie

[1] A. Acart et C. Vogel : Analysis of bounded variation penalty methods of ill-posed problems. *Inverse Problem*, pages 1217–1229, 1994.

[2] S.T. Acton et D.P. Mukherjee : Scale space classification using area morphology. *IEEE Transactions on Image Processing*, 9(4):623–635, April 2000.

[3] S. Alliney : Digital filters as absolute norms minimizers. *IEEE Transactions on Signal Processing*, 40(6):1548–1562, 1992.

[4] S. Alliney : An algorithm for the minimization of mixed l^1 and l^2 norms with application to Bayesian estimation. *IEEE Transactions on Signal Processing*, 42(3):618–627, 1994.

[5] S. Alliney : A property of the minimum vectors of a regularizing functional defined by means of the absolute norm. *IEEE Transactions on Signal Processing*, 45(4):913–917, 1997.

[6] F. Alter, S. Durand et J. Froment : Adapted total variation for artifact free decompression of JPEG images. *Journal of Mathematical Imaging and Vision*, 23(2):199–211, 2005.

[7] A. Amini, T. Weymouth et R. Jain : Using dynamic programming for solving variational problems in vision. *IEEE Transactions on Pattern Analysis and Machine Interaction*, 12(9): 855–867, 1990.

[8] J. Astola, P. Haavisto et Y. Neuvo : Vector median filters. *Proceedings of the IEEE*, 78(4):678–689, 1990.

[9] J.-F. Aujol, G. Aubert, L. Blanc-Feraud et A. Chambolle : Image decomposition into a bounded variation component and oscillating component. *Journal of Mathematical Imaging and Vision*, 22(1):71–88, 2005.

[10] R. Azencott : *Simulated Annealing : Parallelization Techniques*. John Wiley, 1992.

[11] J. A. Bangham, P. Chardaire, C. J. Pye et P. D. Ling : Multiscale nonlinear decomposition : The sieve decomposition theorem. *IEEE Transactions on Pattern Analysis and Machine Interaction*, 18(5):529–539, May 1996.

[12] J. A. Bangham et P.D. Ling d R. Harvey : Scale-space from nonlinear filter. *IEEE Transactions on Pattern Analysis and Machine Interaction*, 18:520–528, 1996.

[13] C. Berge : *The Theory of Graphs*. Dover, 1962.

[14] J. M. Bioucas-Dias et J. Leitão : Two-dimensional phase unwrapping. *IEEE Transactions on Image Processing*, 11(4):408–422, April 2002.

[15] J. M. Bioucas-Dias et G. Valadão : Phase unwrapping via graph-cuts. *In Proceedings of the Iberian Conference on Pattern Recognition and Image Analysis (IbPRIA)*, volume 3522, pages 360–367, Estoril, Portugal, June 2005. Springer-Verlag.

[16] A. Blake et A. Zisserman : *Visual Reconstruction*. The MIT Press, 1987.

[17] P. Blomgren et T.F. Chan : Color TV : Total variation methods for restoration of vector-valued images. *IEEE Transactions on Image Processing*, 7(3):304–309, 1998.

[18] D. Borgefors : Distance transformations in digital images. *CVGIP*, 34(3):344–371, 1986.

[19] C. Bouman et K. Sauer : A generalized Gaussian image model for edge-preserving map estimation. *IEEE Transactions on Signal Processing*, 2(3):296–310, july 1993.

[20] S. Boyd et L. Vandenberghe : *Convex Optimization*. Cambridge University Press, 2004.

[21] Y. Boykov et M.-P Jolly : Interactive graph cuts for optimal boundary and region segmentation of objects in n-D images. *In International Conference on Computer Vision*, pages 105–112, 2001.

[22] Y. Boykov et V. Kolmogorov : Computing geodesic and minimal surfaces via graph cuts. *In International Conference on Computer Vision*, volume 1, pages 26–33, 2003.

[23] Y. Boykov et V. Kolmogorov : An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Interaction*, 26(9):1124–1137, 2004.

[24] Y. Boykov, O. Veksler et R. Zabih : Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Interaction*, 23(11):1222–1239, 2001.

[25] E.J. Breen et R. Jones : Attribute openings, thinnings and granulometries. *Computer Vision and Image Understanding*, 64(3):377–389, 1996.

[26] X. Bresson, S. Esedoglu, P. Vandergheynst, J.P. Thiran et S. Osher : Global minimizers of the active contour/snake model. Rapport technique 05-04, UCLA CAM Report, 2005.

[27] J. Buckheit et D. L. Donoho : Wavelab and reproductible research. *In* Springer-Verlag, éditeur : *Wavelet and Statistics*, pages 53–81, Berlin, 1995.

[28] N. Burrus, A. Duret-Lutz, T. Géraud, D. Lesage et R. Poss : A static C++ object-oriented programming (SCOOP) paradigm mixing benefits of traditional OOP and generic programming. *In Workshop on multiple paradigm with OO languages. MPOOL'03*, 2003.

[29] J.L. Carter : *Dual Methods for Total Variation-Base Image restoration*. Thèse de doctorat, U.C.L.A, 2001.

[30] V. Caselles, B. Coll et J. Morel : Topographic maps and local contrast changes in natural images. *International Journal of Computer Vision*, 33(1):5–27, 1999.

[31] V. Caselles et P. Monasse : Grain filters. *J. of Math. Imaging and Vision*, 17(3):249–270, 2002.

[32] R. Cecchini et A. Del Bimbo : A programming environment for imaging applications. *Pattern Recognition Letters*, 14:817–824, October 1993.

[33] A. Chambolle : Partial differential equations and image processing. *In IEEE International Conference on Image Processing (ICIP 94)*, pages 16–20, 1994.

[34] A. Chambolle : An algorithm for Total Variation minimization and applications. *Journal of Mathematical Imaging and Vision*, 20:89–97, 2004.

[35] A. Chambolle : Total Variation minimization and a class of binary MRF. Rapport technique CMAP-578, Ecole Polytechnique - centre de mathématiques appliquees, june 2005.

[36] T. Chan, A. Marquina et P. Mulet : High-order Total Variation based image restoration. *SIAM Journal on Scientific Computing*, 22(2), 2000.

[37] T.F. Chan et S. Esedoğlu : Aspect of Total Variation regularized L^1 function approximation. Rapport technique 7, UCLA, 2004.

[38] T.F. Chan, S. Esedoglu et M. Nikolova : Algorithms for Finding Global Minimizers of Image Segmentation and Denoising Models. Rapport technique, UCLA, septembre 2004.

[39] T.F. Chan, G.H. Golub et P. Mulet : A nonlinear primal-dual method for total variation-based image restoration. *SIAM Journal Scientific Computing*, 20(6):1964–1977, 1999.

[40] T.F. Chan et L. Vese : Active contours without edges. *IEEE Transactions on Image Processing*, 10(2):266–277, 2002.

[41] F. Cheng et A. N. Venetsanopulos : An adaptive morphological filter for image processing. *IEEE Transactions on Image Processing*, 1:533–539, 1992.

[42] Cimg : C++ template image processing library. Web page. http ://cimg.sourceforge.net.

[43] J. Claerbout : Hypertext documents are reproductible research, 1994. http ://-sepwww.stanford.edu.

[44] T.H. Cormen, C.E. Leiserson, R.L. Rivest et C. Stein : *Introduction to Algorithms*. The MIT Press, 2001.

[45] J. Crespo, R. Schafer, J. Serra, C. Gratin et F. Meyer : The flat zone approach : A general low-level region merging segmentation method. *Signal Processing*, 62(1):37–60, 1998.

[46] K. Czarnecki et U. Eisenecker : *Generative programming : Methods, Tools, and Applications*. Addison-Wesley, 2000.

[47] J. Darbon : Total Variation minimization with L^1 data fidelity as a contrast invariant filter. *In IEEE International Symposium on Image and Signal Processing and Analysis (ISPA 2005)*, Zagreb, Croatia, September 2005.

[48] J. Darbon et C.B. Akgul : An efficient algorithm for attribute openings and closings. *In European Signal Processing Conference (EUSIPCO)*, Antalya, Turkey, September 2005.

[49] J. Darbon, T. Géraud et P. Bellot : Generic algorithmic blocks dedicated to image processing. *In ECOOP Workshop for PhD Students*, Oslo, Norway, June 2004.

[50] J. Darbon, T. Géraud et A. Duret-Lutz : Generic implementation of morphological image operators. *In International Symposium on Mathematical Morphology (ISMM)*, pages 175–184, Sydney, Australia, April 2002. CSIRO Publishing.

[51] J. Darbon et S. Peyronnet : A vectorial self-dual morphological filter based on Total Variation minimization. *In International Symposium on Visual Computing (ISVC 2005)*. Springer-Verlag, 2005.

[52] J. Darbon, B. Sankur et H. Maître : Error correcting code performance for watermark protection. *In Symposium SPIE on Electronic Imaging—-Security and Watermarking of Multimedia Contents III (EI27)*, volume 4314, pages 663–672, San Jose, CA, USA, January 2001.

[53] J. Darbon et M. Sigelle : Exact optimization of discrete constrained Total Variation minimization problems. *In International Workshop on Combinatorial Image Analysis (IWCIA)*, volume 3322 de *Lecture Notes in Computer Science Series*, pages 548–557, Auckland, New Zealand, December 2004. Springer-Verlag.

[54] J. Darbon et M. Sigelle : Exact optimization of discrete constrained Total Variation minimization problems. Rapport technique 2004C004, ENST, Paris, France, October 2004.

[55] J. Darbon et M. Sigelle : A fast and exact algorithm for Total Variation minimization. *In Iberian Conference on Pattern Recognition and Image Analysis (IbPRIA)*, volume 3522, pages 351–359, Estoril, Portugal, June 2005. Springer-Verlag.

[56] J. Darbon et M. Sigelle : A fast and exact algorithm for Total Variation minimization. Rapport technique 2005D002, ENST, Paris, France, January 2005.

[57] J. Darbon et M. Sigelle : Image restoration with discrete constrained Total Variation part i : Fast and exact optimization. *Journal of Mathematical Imaging and Vision*, 2005.

[58] J. Darbon et M. Sigelle : Image restoration with discrete constrained Total Variation part ii : Levelable functions, perfect sampling and extensions. *Journal of Mathematical Imaging and Vision*, 2005.

[59] G. David : *Singular Sets of Minimizers for the Mumford-Shah Functional*. Birkhauser, 2005.

[60] R. Deriche : Using Canny's criteria to derive a recursively implemented optimal edge detector. *International Journal of Computer Vision*, 1987.

[61] F. Dibos et G. Koepfler : Total Variation minimization by the fast level sets transform. *In IEEE Workshop on Variational and Level Sets Methods*, pages 179–185, 2001.

[62] F. Dibos, G. Koepfler et P. Monasse : *Geometric Level Sets Methods in Imaging, Vision, and Graphics*, chapitre 7- Total Variation Minimization for Scalar/Vector Regularization, pages 121–140. Springer-Verlag, 2003.

[63] P.M. Djurić, Y. Huang et T. Ghirmai : Perfect sampling : A review and applications to signal processing. *IEEE Transactions on Signal Processing*, 50(2):345–256, 2002.

[64] M.R. Dobie et P.H. Lewis : Data structures for image processing in C. *Pattern Recognition Letters*, 12(8):457–466, 1991.

[65] D.C. Dobson et C.R. Vogel : Recovery of blocky images from noisy and blurred data. *SIAM Journal on Applied Mathematics*, 56(4):1181–1199, 1996.

[66] D. Donoho, M.R. Duncan, X. Huo et O. Levi : Wavelab. http://www-stat.stanford.edu/~wavelab.

[67] M.C. d'Ornellas : *Algorithmic Patterns for Morphological Image Processing*. Thèse de doctorat, University of Amsterdam, 2001.

[68] M.C. d'Ornellas, R. Van Den Boomgaard et J.-M. Geusebroek : Morphological algorithms for color images based on a generic-programming approach. *In Proceedings to the Brazilian Conference on Image Processing and Computer Graphics (SIBGRAPI'98)*, pages 220–228, 1998.

[69] M.C. d'Ornellas et R. van den Boomgaard : Generic algorithms for morphological image operators — a case study using watersheds. *In* H.J.A.M. Heijmans et J. Roerdink, éditeurs : *Mathematical Morphology and its Applications to Image and Signal Processing*, pages 323–330, 1998.

[70] S. Durand, F. Malgouyres et B. Rougé : Image deblurring, spectrum interpolation and application to satellite imaging. *SIAM Journal on Applied Mathematics*, 56(4):1181–1199, 1996.

[71] EIKONA : http://www.alphatecltd.com.

[72] I. Ekeland et R. Temam : *Convex Analysis and Variational Problem*. Amsterdam : North Holland, 1976.

[73] L. Evans et R. Gariepy : *Measure Theory and Fine Properties of Functions*. CRC Press, 1992.

[74] J. Froment : *MegaWave2 User's Guide*. CMLA, École Normale Supérieure de Cachan, Cachan, France, February 2000.

[75] S. Geman et D. Geman : Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Interaction*, 6(6):721–741, 1984.

[76] T. Géraud, P.-Y. Strub et J. Darbon : Color image segmentation based on automatic morphological clustering. *In IEEE International Conference on Image Processing (ICIP)*, volume 3, pages 70–73, Thessaloniki, Greece, October 2001.

[77] T. Géraud, P.-Y. Strub et J. Darbon : Segmentation d'images en couleur par classification morphologique non supervisée. *In Proceedings of the International Conference on Image and Signal Processing (ICISP)*, pages 387–394, Agadir, Morocco, May 2001. Faculty of Sciences at Ibn Zohr University, Morocco. In French.

[78] G. Giorgi, A. Guerraggio et J. Thierfelder : *Mathematics of Optimization : Smooth and Nonsmooth Case*. Elsevier Science, 2004.

[79] J. A. Goguen : Parameterized programming and software architecture. *In Proceedings of the Fourth IEEE International Conference on Software Reuse*, pages 2–10, 1996.

[80] J. Goutsias, H.J.A.M. Heijman et K. Sivakumar : Morphological operators for image sequences. *Computer Vision and Image Understanding*, 63(2):326–346, 1995.

[81] D. Greig, B. Porteous et A. Seheult : Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistics Society*, 51(2):271–279, 1989.

[82] F. Guichard et J.-M. Morel : *Image Iterative Smoothing and PDE's.* disponible sur demande à fguichard@poseidon-tech.com, 2000.

[83] F. Guichard et J.M. Morel : Mathematical morphology "almost everywhere". *In Mathematical Morphology, Proceedings of the 6th International Symposium (ISMM)*, pages 293–303. CSIRO Publishing, April 2002.

[84] T. Géraud, Y. Fabre, D. Papadopoulos et J.F. Mangin : Vers une réutilisabilité totale des algorithmes de traitement des images. *In Symposium on Signal and Image Processing (GRETSI'99)*, pages 331–334, 1999.

[85] A.G. Hanbury et J. Serra : Morphological operators on the unit circle. *IEEE Transactions on Image Processing*, 10(12):1842–1850, December 2001.

[86] S. Haney et J. Crotinger : How templates enable high-performance scientific computing in C++. *IEEE Computing in Science and Engineering*, 1(4):260–268, 1999.

[87] H. Heijmans : Connected morphological operators for binary images. *Computer Vision and Image Understanding*, 77(1):99–120, 1999.

[88] W.H. Hesselink : Salembier's min-tree algorithm turned into breadth first search. *Information Processing Letters*, 88(5):225–229, December 2003.

[89] D. Hochbaum et J. Shanthikumar : Convex separable optimization is not muchharder than linear optimization. *Journal of the ACM*, 37:843–862, 1990.

[90] D. S. Hochbaum : An efficient algorithm for image segmentation, markov random fields and related problems. *Journal of the ACM*, 48(2):686–701, 2001.

[91] WW.C. Hoffman : *Statistical Methods in Radio Wave Propagation*. Pergamon Press, 1960.

[92] X. Huang, M. Fisher et D. Smith : An efficient implementation of max tree with linked list and hash table. *In Proc. of the Int. Conf. on Digital Image Computing : Techniques and Applications*, pages 299–308, 2003.

[93] M. Huber : Perfect sampling using bounding chains. *The Annals of Applied Probability*, 14(2):734–753, 2004.

[94] ImageJ : : Image processing library and analysis in java. disponible à http://rsb.info.nih.gov/ij/.

[95] H. Ishikawa : Exact optimization for Markov random fields with convex priors. *IEEE Transactions on Pattern Analysis and Machine Interaction*, 25(10):1333–1336, October 2003.

[96] H. Ishikawa et D. Geiger : Occlusions, discontinuities, and epipolar lines in stereo. *In European Conference on Computer Vision*, pages 232–258, 1998.

[97] P. T. Jackway et M. Deriche : Scale-space properties of the multiscale morphological dilation-erosion. *IEEE Transactions on Pattern Analysis and Machine Interaction*, 18(1):38–51, January 1996.

[98] M. Jazayeri, éditeur. *Component Programming - a fresh look at software components*. IEEE, 1995.

[99] R. Jones : Connected filtering and segmentation using component trees. *Computer Vision and Image Understanding*, 75(3):215–228, September 1999.

[100] D. Jungnickel : *Graphs, Networks and Algorithms*. Springer-Verlag, 2005.

[101] D. Kay et J.R. Levine : *Graphics File Formats*. MacGraw-Hill, 1995.

[102] M. Klupsch et W. Eckstein : Object-oriented image processing in Smalltalk : Using complex operator objects. *In* N.C. Callaos, éditeur : *International Conference on Information Systems Analysis and Synthesis (ISAS)*, Orlando, FL, USA, 1996.

[103] D.E. Knuth : *The Art of Computer Programming*, volume 1 Fundamental algorithms. Addison Wesley, third édition, 1997.

[104] D. Koelma, R. Van Balen et A. Smeulders : SCIL-VP : a multi-purpose visual programming environment. *In ACM/SIGAPP Symposium on Applied Computing*, pages 1188–1198, 1992.

[105] D. Koelma et A. Smeulders : A visual programming interface for an image processing environment. *Pattern Recognition Letters*, 15(11):1099–1109, November 1994.

[106] D. Koelma et A. Smeulders : An image processing library based on abstract image data-types in C++. *In International Conference on Image Analysis and Processing*, numéro 974 de Lecture Notes in Computer Science, pages 97–102. Springer-Verlag, 1995.

[107] G. Koepfler et F. Dibos : Global Total Variation minimization. *SIAM Journal of Numerical Analysis*, 37(2):646–664, 2000.

[108] V. Kolmogorov : Primal-dual algorithm for convex markov random fields. Rapport technique, Microsoft Research, 2005.

[109] V. Kolmogorov et R. Zabih : What energy can be minimized via graph cuts ? *IEEE Transactions on Pattern Analysis and Machine Interaction*, 26(2):147–159, 2004.

[110] U. Köthe : Reusable implementations are necessary to characterize and compare vision algorithms. *In DAGM-Workshop on Performance Characteristics and Quality of Computer Vision Algorithms*, Braunschweig, Allemagne, September 1997.

[111] U. Köthe : *Handbook on Computer Vision and Applications*, chapitre Chapter 3 : Reusable Software in Computer Vision. Acadamic Press, 1999.

[112] E. Lawler : *Combinatorial Optimization : Networks and Matroids*. Dover, 1976.

[113] D. Lawton et D. Mead : A modular object oriented image understanding environment. *In International Conference on Pattern Recognition*, volume 2, pages 611–616, Atlantic City, NJ, USA, June 1990.

[114] J. Lee : *A First course in Combinatorial Optimization*. Cambridge Press, 2004.

[115] R. Lukac : Adaptive vector median filter. *Pattern Recognition Letters*, 24(12):1889–1899, 2003.

[116] R. Lukac, B. Smolka, K.N. Plataniotis et A.N. Venetsanopoulos : Selection weighted vector directional filters. *Computer Vision and Image Understanding*, 94(1-3):140–167, 2004.

[117] Z. Ma et H.R. Wu : Classification based adaptive vector filter for color image restoration. *In IEEE International Conference on Acoustics, Speech and Sigenal Processing (ICASSP)*, 2005.

[118] S. Mallat : *A Wavelet Tour of Signal Processing*. Academic Press, 1999.

[119] G. Matheron : *Random Sets and Integral Geometry*. John Wiley, 1975.

[120] Matlab : The mathworks incorporation. http://www.mathworks.com/, 2003.

[121] H. Maître, éditeur. *Le traitement des Images de Radar RSO*. Hermès, 2001.

[122] B. McNamara et Y. Smaragdakis : Functional programming in C++. *In International Conference on Functional Programming (ICFP)*, Montreal, Canada, September 2000.

[123] B. McNamara et Y. Smaragdakis : Static interfaces in C++. *In First Workshop on C++ Template Programming, Erfurt, Germany*, Octobre 10 2000.

[124] Megawave2 : image processing software available at http://www.cmla.ens-cachan.fr/Cmla/Megawave/.

[125] A. Meijster et M.H.F. Wilkinson : Fast computation of morphological area pattern. *In IEEE ICIP'01*, pages 668–671, Thessaloniki, Greece, October 2001.

[126] A. Meijster et M.H.F. Wilkinson : A comparison of algorithms for connected set openings and closings. *IEEE Transactions on Pattern Analysis and Machine Interaction*, 24(4):484–494, Avril 2002.

[127] F. Meyer : Levelings, image simplification filters for segmentation. *Journal of Mathematical Imaging and Vision*, 20:59–72, 2004.

[128] Y. Meyer : *Oscillating Patterns in Image Processing and Nonlinear Evolution Equations*. American Mathematical, 2001.

[129] S. Meyers : How non-member functions improve encapsulation. *In C/C++ User Journal*, 2000.

[130] P. Monasse : *Représentation morphologique d'images numériques et application au recalage*. Thèse de doctorat, Université Paris-Dauphine, june 2000.

[131] P. Monasse et F. Guichard : Fast computation of a contrast-invariant image representation. *IEEE Transactions on Image Processing*, 9(5):860–872, 2000.

[132] K. Murota : *Discrete Convex Optimization*. SIAM Society for Industrial and Applied Mathematics, 2003.

[133] D.R. Musser et A. Staini : *STL Tutorial and Reference Guide : C++ Programming with the Standard Template Library*. Addison-Wesley, 1996.

[134] N.C. Myers : Traits : a new and useful template technique. *C++ Report*, 7(5):32–35, juin 1995.

[135] L. Najman et M. Couprie : Quasi-linear algorithm for the component tree. *In SPIE Symposium on Electronic Imaging*, pages 18–22, San Jose, USA, 2004.

[136] H.T. Nguyen, M. Worring et R. van den Boomgaard : Watersnakes : Energy-driven watershed segmentation. *IEEE Transactions on Pattern Analysis and Machine Interaction*, 23(3):330–342, 2003.

[137] M. Nikolova : Local strong homogeneity of a regularized extimator. *SIAM Journal on Applied Mathematics*, 61:633–658, 2000.

[138] M. Nikolova : Minimizers of cost-functions involving nonsmooth data-fidelity terms. application to the processing of outliers. *SIAM Journal on Numerical Analysis*, 40(3):965–994, 2002.

[139] M. Nikolova : A variational approach to remove outliers and impulse noise. *Journal of Mathematical Imaging and Vision*, 20:99–120, 2004.

[140] Olena : C++ template image processing library. page internet http://www.lrde.epita.fr/olena/.

[141] S. Osher, A. Solé et L. Vese : Image decomposition and restoration using Total Variation minimization and the $\mathbf{H}^{-1}$*norm*. *SIAM J. Multiscale Modeling and Simulation*, 1(3), 2003.

[142] E. Pechersky, A. Maruani et M. Sigelle : On Gibbs Fields in Image Processing. *Markov Processes and Related Fields*, 1(3):419–442, 1995.

[143] I. Pitas : *Digital Image Processing Algorithms and Applications*. Wiley, 2000.

[144] K.N. Plataniotis et A.N. Venetsanopoulos : *Color Image Processing and Application*. Springer, 2000.

[145] I. Pollak, A.S. Willsky et Y. Huang : Nonlinear evolution equations as fast and exact solvers of estimation problems. *IEEE Transactions on Signal Processing*, 53(2):484–498, 2005.

[146] K. Price : Anything you can do, I can do better (no you can't). *Computer Vision, Graphics, and Image Processing*, 36:387–391, 1986.

[147] J. G. Propp et D. B. Wilson : Exact sampling with coupled Markov chains and applications to statistical mechanics. *Random Structures and Algorithms*, 9(1):223–252, 1996.

[148] Qgar : bibliothèque de traitement des images disponibles à http://www.qgar.org/.

[149] M. Rousson et R. Deriche : A variational framework for active and adaptative segmentation of vector valued images. Rapport technique RR-4514, INRIA, 2002.

[150] S. Roy : Stereo without epipolar lines : A maximum-flow formulation. *International Journal of Computer Vision*, 34(2):147–162, 1999.

[151] L. Rudin, S. Osher et E. Fatemi : Nonlinear Total Variation based noise removal algorithms. *Physica D.*, 60:259–268, 1992.

[152] P. Salambier, A. Oliveras et L. Garrido : Antiextensive connected operators for image and sequence processing. *IEEE Transactions on Image Processing*, 7(4):555–570, April 1998.

[153] P. Salambier et J. Serra : Flat zones filtering, connected operators and filters by reconstruction. *IEEE Transactions on Image Processing*, 4:1153–1160, 1995.

[154] P. Salembier et L. Garrido : Binary partition tree as an efficient representation for image processing, segmentation, and information retrieval. *IEEE Transactions on Image Processing*, 9(4):561–576, April 2000.

[155] K. Sauer et C. Bouman : Bayesian estimation of transmission tomograms using segmentation based optimization. *IEEE Transactions on Nuclear Science*, 39(4):1144–1152, 1992.

[156] L. Schrijver : *Combinatorial Optimization - Polyhedra and Efficiency*. Springer, 2003.

[157] Scientific Computing In Object-Oriented Languages, 2000 : Page internet. http://oonumerics.org/.

[158] R. Seidel et M. Sharir : Top-down analysis of path compression. *SIAM Journal on Computing*, 34(3):515–525, 2005.

[159] J. Serra : *Image Analysis and Mathematical Morphology*. Academic Press, 1988.

[160] J. Serra : *Mathematical Morphology in Image Processing*, chapitre Anamorphoses and Function lattices, pages 483–523. Marcel-Dekker, 1992.

[161] J. Serra et P. Salembier : Connected operators and pyramids. *In Proc. SPIE Image Algebra Math. Morphology*, volume 2030, pages 65–76, 1993.

[162] M. Sigelle : *Champs de Markov en Traitement d'Images et Modèles de la Physique Statistique : Application à la Relaxation d'Images de Classification* http://www.tsi.enst.fr/~sigelle/tsi-these.html. Thèse de doctorat, ENST, 1993.

[163] Y. Smaragdakis et B. McNamara : Fc++ : Functional tools for object-oriented tasks. *Sotware : Practice and Experience*, 32:1015–1033, 2002.

[164] P. Soille : *Morphological Image Analysis Principles and Applications*. Springer-Verlag, 1999.

[165] M. Sonka, V. Hlavac et R. Boyle : *Image Processing, Analysis, and Machine Vision*. PWS, 1999.

[166] A. Stepanov, M. Lee et S.A. Smith : *The C++ Standard Template Library*. Prentice Hall, 2000.

[167] G. Strang : L^1 and L^∞ approcimation of vector fields in the plane. *In Partial Differential Equations in Applied Science (Tokyo, 1982)*, pages 273–278, 1983.

[168] G. Strang : Maximal flow through a domain. *Mathematical Programming*, 26(3):123–143, 1983.

[169] D. Strong et T.F. Chan : Edge preserving and scale-dependent properties of Total Variation regularization. *Inverse Problem*, 19:165–187, 2003.

[170] B. Stroustrup : *The C++ Programming Language*. Addison-Wesley, 1997.

[171] E. Tadmor, S. Nezzar et L. Lese : A multiscale image representation using hierarchical (BV,L^2) decompositions. Rapport technique, UCLA, 2003.

[172] R.E. Tarjan : Efficiency but of a good but not linear set union algorithm. *Journal of ACM*, 22:215–225, 1975.

[173] K. Tombre, C. Ah-Soon, Ph. Dosch, A. Habed et G. Masini : Stable, robust and off-the-shelf methods for graphics recognition. *In International Conference on Pattern Recognition*, pages 406–408, 1998.

[174] K. Tombre, C. Ah-Soon, Ph. Dosch, G. Masini et S. Tabbone : Stable and robust vectorization : How to make the right choices. *In* Springer-Verlag, éditeur : *Graphics recognition—Recent advances*, volume 1941, pages 3–18, 2000.

[175] R.J. Trudeau : *Introduction to Graph Theory*. Dover, 1976.

[176] University of Amsterdam, The Netherlands. *Horus User Guide*, 2002.

[177] UNM : bibliothèque de traitement des images disponibles à http://www.cs.unm.edu/~williams/image-ref.html.

[178] T. Veldhuizen : Expression templates. *C++ report*, 7(5):26–31, 1995.

[179] L. Vese et S. Osher : Modeling textures with Total Variation minimization and oscillating patterns in image processing. *Journal of Scientific Computing*, 19(1-3):553–572, 2003.

[180] L. Vese et S. Osher : Image denoising and decomposition with Total Variation and oscillatory functions. *Journal of Mathematical Imaging and Vision*, 20(1-2):7–18, January 2004.

[181] Vigra : bibliothèque de traitement des images disponibles à http://kogs-www.informatik.uni-hamburg.de/~koethe/vigra/.

[182] L. Vincent : Grayscale area openings and closings, their efficient implementation and applications. *In Proc. EURASIP Mathematical Morphology and Its Application to Signal Processing*, pages 22–27, 1993.

[183] L. Vincent : Morphological area openings and closings for gray-scale images. *In Shape in Picture : MAthematical Description of Shape in Grey level images*, pages 197–208. A. Toet and D. Foster and H.J.A.M. Heijmans and P. Meer, 1993.

[184] L. Vincent : Morphological grayscale reconstruction in image analysis : Applications and efficient algorithms. *IEEE Transactions on Image Processing*, 2(2):176–201, April 1993.

[185] L. Vincent : Granulometries and opening trees. *Fundamenta Informaticae*, 41:57–90, 2000.

[186] L. Vincent et P. Soille : Watersheds in digital spaces : an efficient algorithm based on immersion simulations. *IEEE Transactions on Pattern Analysis and Machine Interaction*, 13(6):583–598, June 1991.

[187] C. Vogel et M. Oman : Iterative method for Total Variation denoising. *SIAM J. Sci. Comput.*, 17:227–238, 1996.

[188] P. Weis et X. Leroy : *Le langage Caml*. InterEditions, 1993.

[189] M.H.F. Wilkinson et J.B.T.M. Roerdink : Fast morphological attribute operations using Tarjan's union-find algorithm. *In Proceedings of the ISMM2000*, pages 311–320, Palo Alto, CA, June 2000.

[190] G. Winkler : *Image Analysis, Random Fields and Dynamic Monte Carlo Methods. A Mathematical Introduction*. Applications of mathematics. Springer-Verlag, seconde édition, 2003.

[191] W. Yin, D. Goldfoarb et S. Osher : Total Variation based image cartoon-texture decomposition. Rapport technique, UCLA, 2005.

[192] E. Yoruk, E. Konukoglu, B. Sankur et J. Darbon : Person authentication based on hand shape. *In European Signal Processing Conference (EUSIPCO)*, Vienna, Austria, September 2004.

[193] E. Yoruk, E. Konukoglu, B. Sankur et J. Darbon : Shape-based hand recognition. *IEEE Transactions on Image Processing*, 2005.

[194] B. Zalesky : Network flow optimization for restoration of images. *J. Appl. Math.*, 2(4):199–218, 2002.

[195] B. Zalesky : Efficient determination of gibbs estimator with submodular energy functions. Rapport technique, United Institution of Information Problem, 2005.