

SORBONNE UNIVERSITÉ
TELECOM PARISTECH

SÉMINAIRES ET PRATIQUE EN IMAGE
PRAT

Morphologie mathématique sur des graphes

Auteur :

de Sousa Silva, Raul Alfredo

Encadrant :

Isabelle Bloch

Délivré le : 3 février 2020



Table des matières

1	Introduction	5
2	Les graphes dans le traitement d'images	7
2.1	Graphes	7
2.2	Extension aux images	8
2.3	Les graphes pour la représentation d'images	8
2.4	Graphcut	14
2.5	Exemples	15
3	La morphologie mathématique dans les traitement d'images	19
3.1	Morphologie mathématique	19
3.2	L'extension aux fonctions	20
3.3	L'extension aux images : binaire, niveaux de gris, couleur	21
3.4	Watershed	22
3.5	Exemples	24
4	Application	29
4.1	La combinaison des deux outils	29
4.2	Travail de référence	30
4.3	Application à la segmentation interactive d'images médicales	31
4.3.1	Première approche	33
4.3.2	Deuxième approche	34
4.3.3	Troisième approche	35
4.4	Mesures d'évaluation	35
4.5	Performance	36
5	Conclusion	43

Table des figures

2.1	Exemple d'une trame discrète	9
2.2	Exemples de graphes de k-plus proches voisins. Librement adapté de [1].	10
2.3	Exemple d'un graphe de voisins à distance d. Librement adapté de [2].	10
2.4	Exemple d'un graphe de voisinage barycentrique. Librement adapté de [2].	11
2.5	Exemple d'un graphe de Delaunay. Librement adapté de [1].	11
2.6	Exemple d'un graphe de Gabriel. Librement adapté de [1].	12
2.7	Exemple d'un graphe de voisinage relatif. Librement adapté de [1].	13
2.8	Exemple d'un graphe d'arbre couvrant le poids minimal. Librement adapté de [1].	13
2.9	Illustration de la technique <i>Graphcut</i> . Source [3].	14
2.10	(a) Image originale, (b) 639 zones (98% of reduction), (c) image reconstruite, (d)-(g) originale+étiquettes (h)-(k). Extrait de [4].	16
2.11	Segmentation automatique d'une image cytologique automatiquement colorée basée sur un graphe de pixels à 8-connexité et partitions d'énergie (a) Image originale. (b) Minima du gradient. (c)-(e) Noyau, fond et minima du cytoplasme classifié avec un clustering flou. (f) Segmentation obtenue de la fusion avec des contours imposées en rouge. Extrait de [4].	17
2.12	Filtre alternés séquentiels définis sur des graphes appliqués à deux exemples. Extrait de [5].	18
3.1	Comparatif entre deux couleurs du système RGB. Les deux premières couleurs sont séparées par une distance colorimétrique plus grande que les deux dernières couleurs, par contre la différence est bien plus évidente dans le deuxième cas.	22
3.2	Résultat de la ligne de partage des eaux appliqué à une image dans différentes conditions de pré et post-traitement	25
3.3	Application des filtres morphologiques basées sur EDPs à la débruitage. Extrait de [6].	25
3.4	Application des filtres morphologiques basées sur EDPs à la segmentation. Extrait de [6].	26
3.5	Detection de bords dans des images du poumon en CT : (a) Image originale avec du bruit <i>salt-and-pepper</i> . (b) Image traitée avec Laplacien de Gaussienne. (c) Image traitée avec le détecteur de Sobel. (d) Image traité avec le Gradient morphologique. (e) Image traitée avec un détecteur de bords par dilatation. (f) Image traitée avec un nouvel opérateur morphologique proposé. Extrait de [7].	27

4.1	Différents modes d'acquisition pour une même coupe du cerveau	32
4.2	Segmentation de référence de la segmentation de l'image 4.1	33
4.3	Segmentation de l'œdème par l'approche 1 : (a)-(c) Images originales avec les contours trouvé et référence, (d)-(f) Segmentation de référence (g)-(i) Segmentation obtenue, (h)-(k) Overlap des segmentations, référence en vert, obtenue en rouge, coïncidence en jaune	38
4.4	Segmentation de l'œdème par l'approche 2 : (a)-(c) Images originales avec les contours trouvé et référence, (d)-(f) Segmentation de référence (g)-(i) Segmentation obtenue, (h)-(k) Overlap des segmentations, référence en vert, obtenue en rouge, coïncidence en jaune	39
4.5	Segmentation de l'œdème par l'approche 3 : (a)-(c) Images originales avec les contours trouvé et référence, (d)-(f) Segmentation de référence (g)-(i) Segmentation obtenue, (h)-(k) Overlap des segmentations, référence en vert, obtenue en rouge, coïncidence en jaune	40

Liste d'équations

2.1	Définition du graphe de Gabriel	12
2.2	Définition du graphe de voisinage relatif	12
2.3	Fonction de perte du graphcut	14
2.4	Définition de la coupe minimale	15
2.5	Définition de coupe minimale multiclasses	15
3.1	Définition de dilatation	20
3.2	Définition d'érosion	20
3.3	Définition d'ouverture	20
3.4	Définition de fermeture	20
3.5	Fonction équivalente de la boule	21
3.6	Définition de dilatation fonctionnelle	21
3.7	Définition d'érosion fonctionnelle	21
3.8	Définition de plus forte pente	23
3.9	Définition de dénivelé	23
3.10	Définition de Cout	23
3.11	Définition de distance topographique	23
3.12	Définition de bassin versant	24
3.13	Définition de ligne de partage des eaux	24
4.1	Définition de longueur de Riemann	31
4.2	Fonction décroissante du gradient	31
4.3	Poids entre régions et le fond dans approche 1	34
4.4	Poids entre régions et la tumeur dans approche 1	34
4.5	Poids entre régions et classes dans approche 2	34
4.6	Poids entre régions et le fond dans approche 3	35
4.7	Poids entre régions et la tumeur dans approche 3	35
4.8	Mesure DICE	36
4.9	Mesure SD _{abs}	36
4.10	Mesure VD	36

Chapitre 1

Introduction

Les images sont un type de données très puissant qui permet aux êtres humains de facilement comprendre l'organisation spatiale d'une scène ou inférer plusieurs choses à propos d'un objet ou situation. En médecine, l'obtention d'images est aussi très importante puisqu'elle permet d'avoir un moyen d'étudier et évaluer l'intérieur du corps sans être invasif (ou très peu) et de réduire le risque d'une intervention invasive ou d'un mauvais diagnostic en ne le faisant pas.

En général les images sont représentées par des pixels/voxels dans des structures régulières en temps/espace, ce qui permet d'avoir une vision de la projection d'une scène où d'un volume de manière assez représentative.

Cependant, les caractéristiques de l'image ou de la méthode d'acquisition exigent que l'image soit traitée, à plusieurs niveaux, pour rendre l'image plus interprétable ou même pour permettre que le détail d'intérêt soit plus facilement trouvé ou dévoilé. Alors, toute une chaîne de traitement des images est proposée à partir de l'extension de concepts et outils mathématiques, entre autres les graphes et la morphologie mathématique.

Les représentations d'images par graphes permettent de structurer l'information contenue dans les images et de travailler sur des régions par exemple, plutôt que sur la représentation initiale par pixels ou voxels. Les opérations de morphologie mathématique peuvent être adaptées à ce type de structure, par exemple pour le filtrage d'images, ou la segmentation comme il sera fait ici.

Donc, ce travail propose une étude des graphes et de la morphologie mathématique, et l'application de ces deux outils pour la segmentation d'œdèmes/tumeurs dans des images de résonance magnétique du cerveau. Trois approches ont été proposées et au total 50 des images du jeu de données ont été utilisées pour faire l'évaluation des algorithmes. Ses performances ont été évaluées comparativement à d'autres méthodes de référence, en général basées sur le même jeu de données et en utilisant des techniques de *deep learning*.

La suite de ce rapport s'organise de la manière suivante : le chapitre 2 introduira la notions de graphe, et les extensions possibles au cas du traitement images et inclura quelques exemples. Le chapitre 3 traitera de la morphologie mathématique ainsi que son extension au traitement d'images et sera aussi finalisé par quelques exemples de travaux. Le chapitre 4 sera dédié à l'application proposée. Il commencera par une description du travail de base et passera à la proposition d'application à la segmentation des images IRM du cerveau, qui sera conclue avec la présentation et une discussion des résultats. Finalement, le chapitre 5 reprend tous ce qui a été

fait au long de ce projet et conclue de manière plus généralisée sur les résultats du projet.

Chapitre 2

Les graphes dans le traitement d'images

2.1 Graphes

Le graphe est un concept mathématique créé pour permettre une étude des objets et de leurs relations dans un ensemble. La première fois où ce concept a été évoqué a été en 1736 dans le travail *sept ponts de Königsberg* du mathématicien suisse Leonhard Euler [8].

Les graphes ont trouvé des applications dans plusieurs domaines depuis leur création, dont le domaine du traitement d'images où l'interprétation des images comme un ensemble de régions inter connectés par des relations de voisinage permet toute une gamme de traitements qui vont du débruitage à la reconnaissance d'objets [1, 9]. Tout cela, à cause de la justesse de ce modèle aux conditions pratiques de générations de la majorité images, qui normalement sont des structures fortement contextualisées.

Un graphe G est représenté par une structure de la forme $G = (V, E)$ où V est l'ensemble des objets, appelés sommets (*vertices* en anglais) et E est l'ensemble des relations entre ces objets, appelés arêtes (*edges* en anglais). Évidemment, tant l'ensemble V comme le E , sont finis et discrets.

Chaque sommet, ainsi que chaque arête, peut être caractérisé par une grandeur \vec{X} qui pourra être une valeur de niveau de gris (unidimensionnel) ou les valeurs des composantes RGB d'un pixel d'une image couleur, surtout pour les sommets, et des valeurs de distance physique ou colorimétrique ainsi que n'importe quelle autre mesure de distance pour les arêtes. De plus, les arêtes, qui sont dans les cas élémentaires, bidirectionnelles ($e_{x,y} = x \rightarrow y = y \rightarrow x = e_{y,x}$), pourront être unidirectionnelles ($e_{x,y} \neq e_{y,x}$, $x \rightarrow y \neq y \rightarrow x$), cas où le graphe sera appelé *graphe orienté*. Il y a aussi la possibilité d'accepter que les graphes aient des arêtes qui pointent d'un sommet à lui-même, ce que nous dénommons *boucle*.

D'autres concepts de graphes qui nous seront utiles pour la suite sont les concepts de clique et de composante connexe. Une clique d'un graphe non orienté est, en théorie des graphes, un sous-ensemble des sommets de ce graphe dont le sous-graphe induit est complet, c'est-à-dire que deux sommets quelconques de la clique sont toujours adjacents. Une composante connexe d'un graphe est un sous-graphe connexe de ce graphe. Un graphe connexe à n sommets possède au moins $n - 1$ arêtes.

Les applications des graphes dans le traitement d'images passent par le débrui-

tage, la segmentation, la détection de contours, la détection de mouvement, la compression de données, ainsi que la reconnaissance d'objets et le recalage d'objets par mise en correspondance de graphes. Souvent la théorie des graphes est utilisée sur des images avec d'autres outils mathématiques comme les champs de Markov, la décision bayésienne et la morphologie mathématique pour résoudre les problèmes quotidiens du traitement d'images.

2.2 Extension aux images

Puisque les graphes ont pour principe l'idée de représenter les objets et leurs relations, ils sont devenus un outil très puissant pour représenter les relations spatiales entre les parties de l'image. Notons que les outils traditionnels comme des histogrammes et les descripteurs de textures ne donnent aucune information par rapport aux relations spatiales entre les zones de l'image.

Alors les graphes seront utiles pour générer ces représentations spatiales à partir des arêtes qui lient deux points est où le poids ajouté à chaque arête correspondra à une espèce de distance (physique, colorimétrique ou autre) entre ces régions que servira à une segmentation, ou réunion ou suivi de ces régions.

À partir des définitions présenté dans la section 2.1 nous pourrons imaginer de multiples représentations des images sous la forme de graphes où les sommets pourront être les pixels, ou des régions (segmentés) de l'image, ou même des objets précis qui sont d'intérêt dans une image (ou dans les vidéos). Les arêtes sont évidemment les connexions entre ces régions, mais le fait d'avoir des arêtes pondérées pourra être intéressant ou pas selon le type de traitement souhaité. Par exemple, dans le cas des débruitages plus simples ou de la segmentation markovienne, où les arêtes ne font que représenter le voisinage entre les pixels, le poids des arêtes est peu important. Par contre dans les algorithmes de coupure de graphes (*Graphcut*) les arêtes pondérées sont fondamentales pour définir un ordre de préférence.

2.3 Les graphes pour la représentation d'images

Parmi tout une gamme de possibilités de graphes que nous pourrions imaginer pour représenter une image il y en a quelques-uns qui sont plus adaptés et couramment utilisés lors d'une chaîne de traitement ou de la mise en œuvre d'un algorithme [2]. Ils sont présentés ci-dessous.

Trame discrète et graphe de pixels

C'est le plus simple que nous pouvons faire en termes de graphe pour les images. À chaque pixel nous ajoutons une arête à tous ses voisins. Le voisinage se base sur le concept de connexité ce qui permet d'avoir 4 (bas, haut, droite et gauche) ou 8 voisins (les autres 4 des diagonales). La trame carrée est souvent utilisée à cause de la distribution de capteurs dans les caméras utilisées. Cette trame pourra être triangulaire ou hexagonale lorsque le système d'acquisition n'est plus composé de pixels carrés. Dans ces cas, le nombre de voisins s'adapte à la forme des capteurs du système d'acquisition (3 ou 6). Si nous passons à des images 3D, le système de voisinage pourra changer pour les trames cubiques selon la façon dont l'extension

est faite, pouvant, alors, varier entre 6 (voisinage de faces), 18 (voisinage d’arêtes) ou 26 (voisinage de sommets), tandis que pour les trames hexagonales le nombre de voisins est invariablement égal à 24.

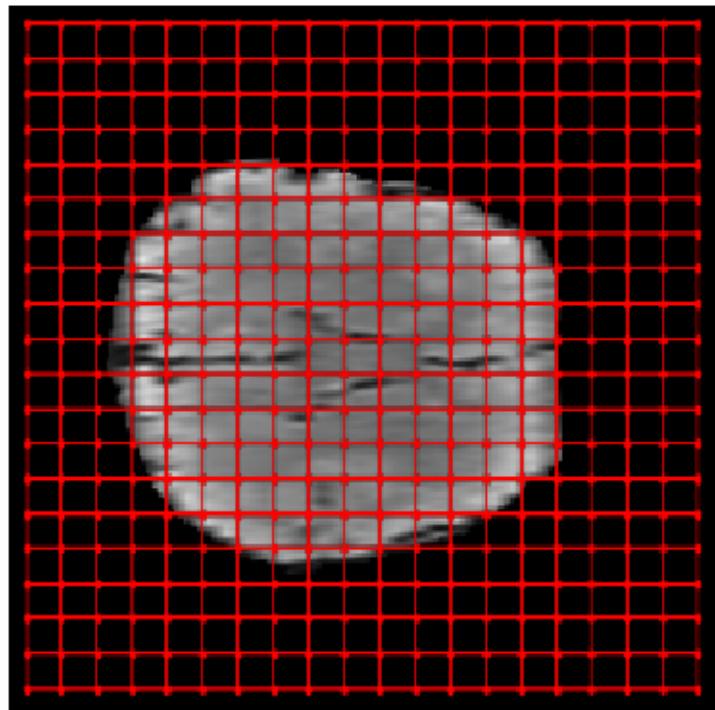


FIGURE 2.1 – Exemple d’une trame discrète

k-plus proches voisins (k-PPV)

Pour chaque objet $v \in \mathcal{V}$, nous définissons comme une arête du graphe chaque paire constituée de v et de l’un de ses k voisins les plus proches au sens d’une certaine distance. Cette procédure est l’une des plus simples et des plus classiques pour générer des graphes de voisinage. Cependant, le graphe obtenu n’est pas nécessairement connexe, pas nécessairement planaire, pas même défini dans certaines configurations.

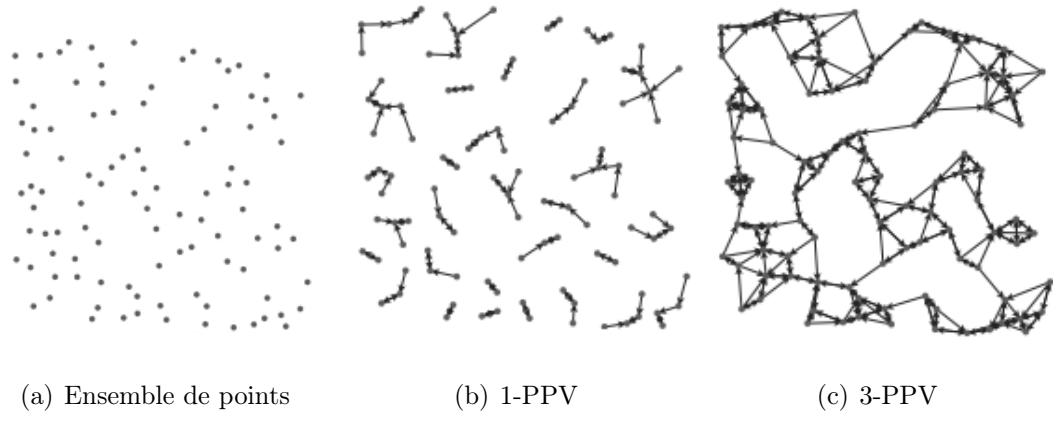


FIGURE 2.2 – Exemples de graphes de k -plus proches voisins. Librement adapté de [1].

Voisins à distance d

Une arête du graphe est définie par chaque paire d'objets dont la distance (euclidienne) l'un par rapport à l'autre n'est pas supérieure à une valeur donnée d . Là encore, ce graphhe n'est pas nécessairement connexe ou planaire. De plus, cela dépend fortement du choix du paramètre d . O'Callaghan [10] a peaufiné cette définition en donnant une condition supplémentaire. Selon lui, un objet w est voisin d'un autre objet v si la distance euclidienne entre v et w n'est pas supérieure à d ; et w n'est pas "caché" derrière un autre objet de \mathcal{V} , qui a déjà été classé comme voisin de v .

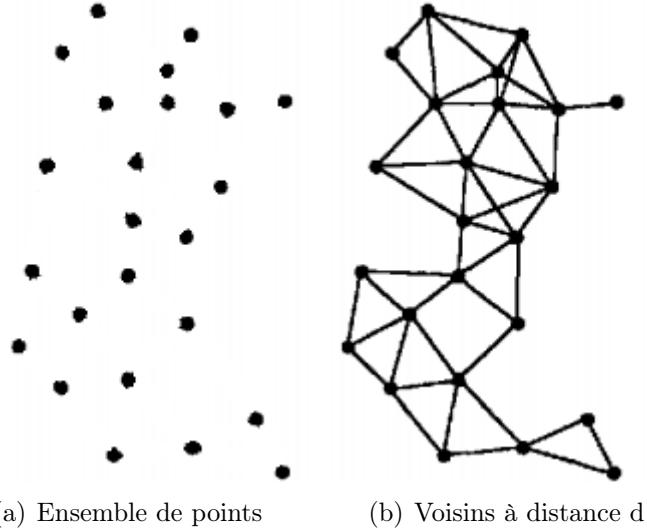


FIGURE 2.3 – Exemple d'un graphe de voisins à distance d . Librement adapté de [2].

Voisinage barycentrique

Ici, \mathcal{V} est supposé être un ensemble fini de points. Pour chaque $v \in \mathcal{V}$, nous définissons les voisins de p comme un ensemble $\{q, q_2, \dots, q_k\}$ tels que p soit inclus

dans la coque convexe de ces points. Cela nécessite l'utilisation d'un plus grand ensemble S^* afin que S soit inclus dans l'enveloppe convexe de S^* . L'inconvénient de ce type de graphe est qu'il n'est pas unique. Il faut ajouter d'autres conditions.

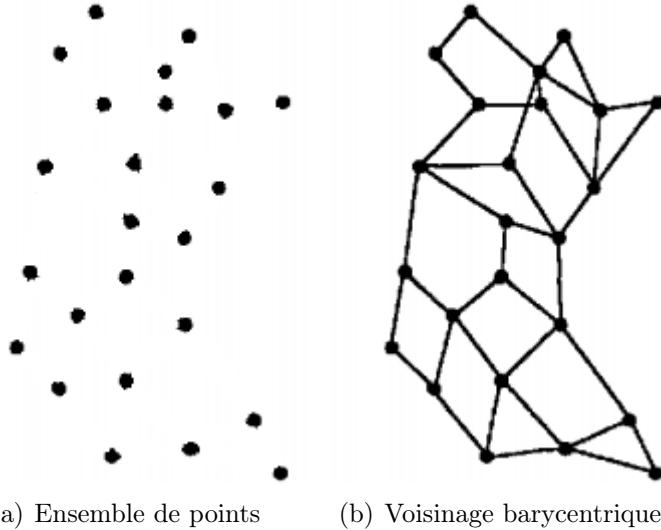


FIGURE 2.4 – Exemple d'un graphe de voisinage barycentrique. Librement adapté de [2].

Triangulation de Delaunay

Les arêtes de ce graphe sont définies par toutes les paires de points (p, q) dans \mathcal{V} , dont les polygones de Voronoï $Z(p)$ et $Z(q)$ sont adjacents, c'est-à-dire, partagent une arête. Ainsi, la triangulation de Delaunay est souvent appelée le graphe dual du diagramme de Voronoï.

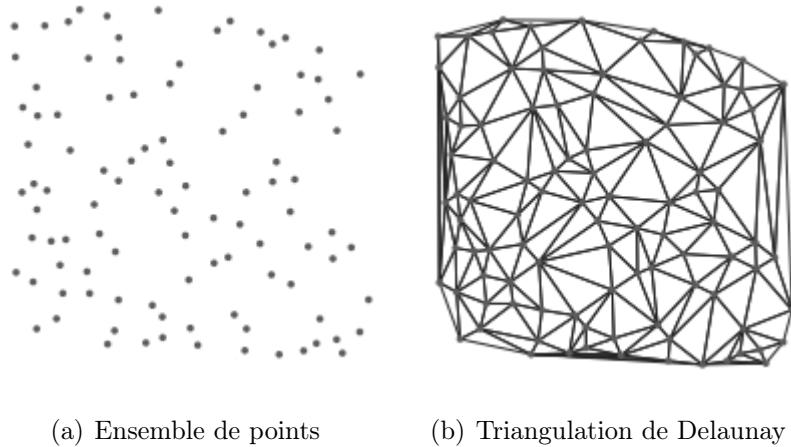


FIGURE 2.5 – Exemple d'un graphe de Delaunay. Librement adapté de [1].

Graphe de Gabriel

Pour chaque paire de points (A, B) dans le plan, définissons d'abord $D(A, B)$ comme étant le segment d'admission du disque fermé $[AB]$ comme l'un de ses dia-

mètres. Les arêtes du graphe de Gabriel sont alors définies par toutes les paires de points (p, q) dans \mathcal{V} , de sorte qu'il n'y a pas d'autre point de \mathcal{V} dans $D(p, q)$, c'est-à-dire :

$$\forall m \in \mathcal{V} \setminus \{p, q\}, \text{ dist}(m, p)^2 + \text{dist}(m, q)^2 \geq \text{dist}(p, q)^2 \quad (2.1)$$

où dist est ici la distance euclidienne dans le plan. Ce graphe a été introduit en 1969 par Gabriel. Dans le domaine du traitement d'images, il est parfois appelé graphe de perception.

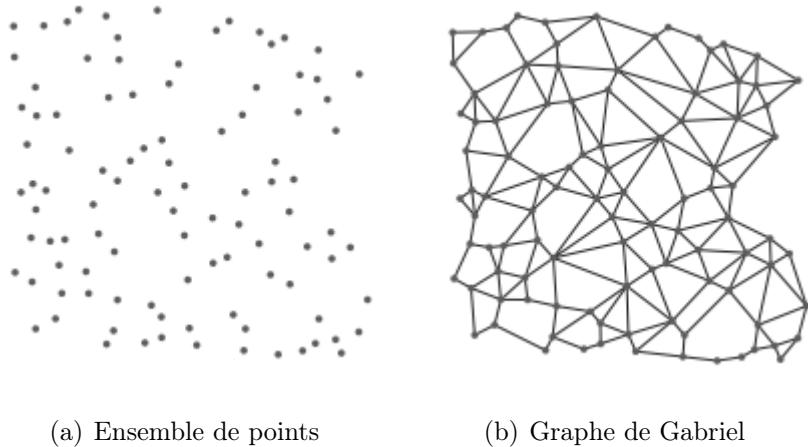


FIGURE 2.6 – Exemple d'un graphe de Gabriel. Librement adapté de [1].

Graphe de voisinage relatif

De même, définissons pour chaque paire (A, B) de points dans le plan $\text{lun}(A, B)$ l'intersection de disques ouverts ayant pour rayon $[AB]$ et centrés en A et B , respectivement. Les arêtes du graphe de voisinage relatif sont ensuite définies par toutes les paires de points (p, q) dans \mathcal{V} , de sorte qu'il n'y a pas d'autre point de \mathcal{V} dans $\text{lun}(p, q)$, c'est-à-dire :

$$\begin{aligned} \forall m \in \mathcal{V} \setminus \{p, q\}, \text{ dist}(m, p) &\geq \text{dist}(p, q) \\ \text{ou} \\ \text{dist}(m, q) &\geq \text{dist}(p, q). \end{aligned} \quad (2.2)$$

Ce graphe, introduit pour la première fois par Lankford, a été étudié, entre autres, par Toussaint et Supowit [11].



(a) Ensemble de points



(b) Graphe de voisinage relatif

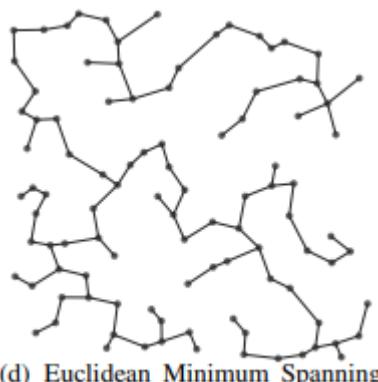
FIGURE 2.7 – Exemple d'un graphe de voisinage relatif. Librement adapté de [1].

Arbres couvrant de poids minimal

Un arbre couvrant de poids minimal d'un graphe est un arbre couvrant (sous-ensemble qui est un arbre et qui connecte tous les sommets ensemble) dont la somme des poids des arêtes est minimale (c'est-à-dire de poids inférieur ou égal à celui de tous les autres arbres couvrants du graphe). L'arbre couvrant minimum peut s'interpréter de différentes manières selon ce que représente le graphe. De manière générale si on considère un réseau où un ensemble d'objets doivent être reliés entre eux (par exemple un réseau électrique et des habitations), l'arbre couvrant minimum est la façon de construire un tel réseau en minimisant un coût représenté par le poids des arêtes [12].



(a) Ensemble de points



(b) Arbre couvrant le poids minimal

(d) Euclidean Minimum Spanning Tree.

FIGURE 2.8 – Exemple d'un graphe d'arbre couvrant le poids minimal. Librement adapté de [1].

2.4 Graphcut

L'un des outils les plus puissantes pour la manipulation des graphes est la coupure de graphes ou *Graphcut*. Introduit par Greig en 1989 [13], la coupure de graphes est une solution très rapide et efficace pour un problème NP-difficile (dans le cas multi-classes) qui est l'étiquetage au niveau pixellique d'une image. Cela peut servir à la segmentation, au calcul de profondeur dans des images, au calcul des cartes de disparités et d'autres problèmes formulés sous la minimisation d'une énergie.

Le principe derrière la coupure de graphes est relativement simple, pour le cas binaire (deux classes) et la solution donnée par *Graphcut* atteint un minimum global. Cependant, la technique n'est pas facilement étendue au cas multi-classes, ce qui laisse cas général encore sans solution optimale, bien que les approches proposées pour l'extension au cas multi-classes donne toujours une réponse proche du minimum global.

Un graphe quelconque $G = (V, E)$ peut recevoir autres deux sommets appelés s et t qui seront appelés respectivement source et puits et qui seront liés à tous les autres sommets du graphe par des arêtes e_{s,v_i} et $e_{v_i,t}$ (v_i sommet $\in V$). Ce nouveau graphe G' est coupé de manière à séparer le graphe entre ceux qui seront attachés à la source et ceux qui seront attachés au puits de manière à minimiser la valeur globale de la coupe, c'est-à-dire, la somme des poids des arêtes coupées. La figure 2.9 sert d'illustration à cette formulation théorique.

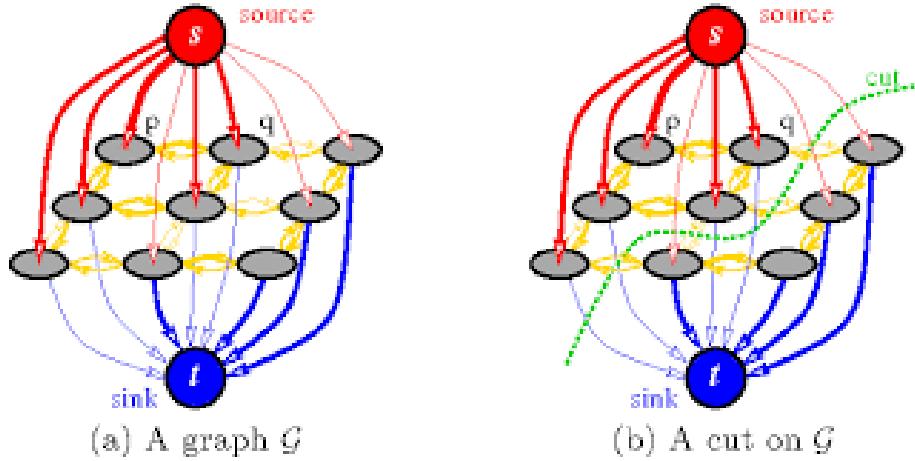


FIGURE 2.9 – Illustration de la technique *Graphcut*. Source [3].

Alors, nous pouvons définir une coupure comme l'ensemble C d'arêtes telles que, la suppression de C de E effacerait tout chemin reliant les terminaux s et t . À fin de minimiser toute la coupe d'un graphe, il faut définir une fonction de mesure de la coupe. De manière générale une coupe pourra être mesuré par la fonction 2.3.

$$\forall n \in \mathbb{N}^*, L_n(C) = \sqrt[n]{\sum_{e_{i,j} \in C} w_{i,j}^n} \quad (2.3)$$

À partir de cette mesure de poids de la coupe, nous définissons une coupe minimale de manière très intuitive comme étant la coupe C^* telle que n'importe quelle

autre coupe C aura un poids plus grand ou égal que C^* .

$$C^* = \operatorname{argmin}_{C \in C_{s,t}} (L_n(C)) \quad (2.4)$$

L'ensemble des poids sur les arêtes pourrait être vu comme des capacités de flux à travers le “tuyau” qui relie les points i et j ($e_{i,j}$). Dans ce cas, nous avons l'intérêt à maximiser le flux entre la source et le puits (d'où, les noms) en respectant une règle de conservation de flux sur tous les sommets (sauf la source et le puits), c'est-à-dire, que dans tous les sommets tout le flux arrivant est égal au flux sortant (le sommet ne cumule pas le matériel qui écoule dans les “tuyaux”).

Alors, un théorème fondamental pour l'implémentation de la coupure de graphes est celui de Ford and Fulkerson [14] qui fait le lien entre le flux maximal et la coupe minimale entre deux sommets. En effet le théorème prouve que pour un graphe G , le flux maximal entre deux sommets est égal à la capacité minimale de toutes les coupes qui séparent ces sommets.

L'extension de la coupure des graphes au cas multi-classes se fait avec l'ajout de non seulement deux terminaux, en l'occurrence, s et t , mais de k terminaux, désormais appelés $T = \{t_1, \dots, t_k\} \in V$. L'équation de poids d'une coupe est la même et une légère modification de la définition de coupe minimale se fait nécessaire, puisque dans ce nouveau cadre, il faut séparer tous les terminaux entre eux. L'équation 2.4 devient alors, l'équation 2.5.

$$C^* = \operatorname{argmin}_{C \in C_{t_1, \dots, t_k}} (L_n(C)) \quad (2.5)$$

Plusieurs approches ont été proposées au fil des années pour résoudre le problème multi-classes. L'une des approches, celui de Dahlhaus [15] est de computer k coupes C_i que séparent t_i et tous les autres terminaux $T \setminus \{t_i\}$. À la fin, les $k - 1$ plus petites coupes (au sens de L_n) sont gardées pour partitionner le graphe.

Une autre approche intéressante proposé par Boykov [16] est de construire une couche du graphe pour chaque classe et relier les sommets équivalents. À la fin, la classe est définie par la position de la coupe pour chaque sommet.

Chaque approche vise à la fois, réduire les ressources computationnelles comme temps de calcul, et de mémoire, et aussi s'approcher du minimum global. Certains approches peuvent garantir une distance maximale au minimum global, et d'autres qui garantiront une complexité la plus faible possible.

2.5 Exemples

Plusieurs articles sont présentés tous les ans avec des évolutions dans l'application des graphes aux images, soit en termes d'amélioration des algorithmes de calcul de ces graphes et des grandeurs associées, soit pour définir des modèles plus adaptés à certains contextes. Voici quelques exemples :

Le travail de Ta [4] montre des types de graphes et les exemples de mesures qui peuvent être extraites pour la segmentation des images cytologiques et histologiques. Les figures 2.10 et 2.11 sont des bons exemples.

En général sont utilisés les graphes d'adjacences de régions, les trames carrées et les graphes de k-PPV. Avec des régularisations discrètes, des partitions d'énergies

discrètes et les *clusterings* semi-supervisés, l'article de Ta montre des possibilités de simplifications des images comme segmentation ou classification.

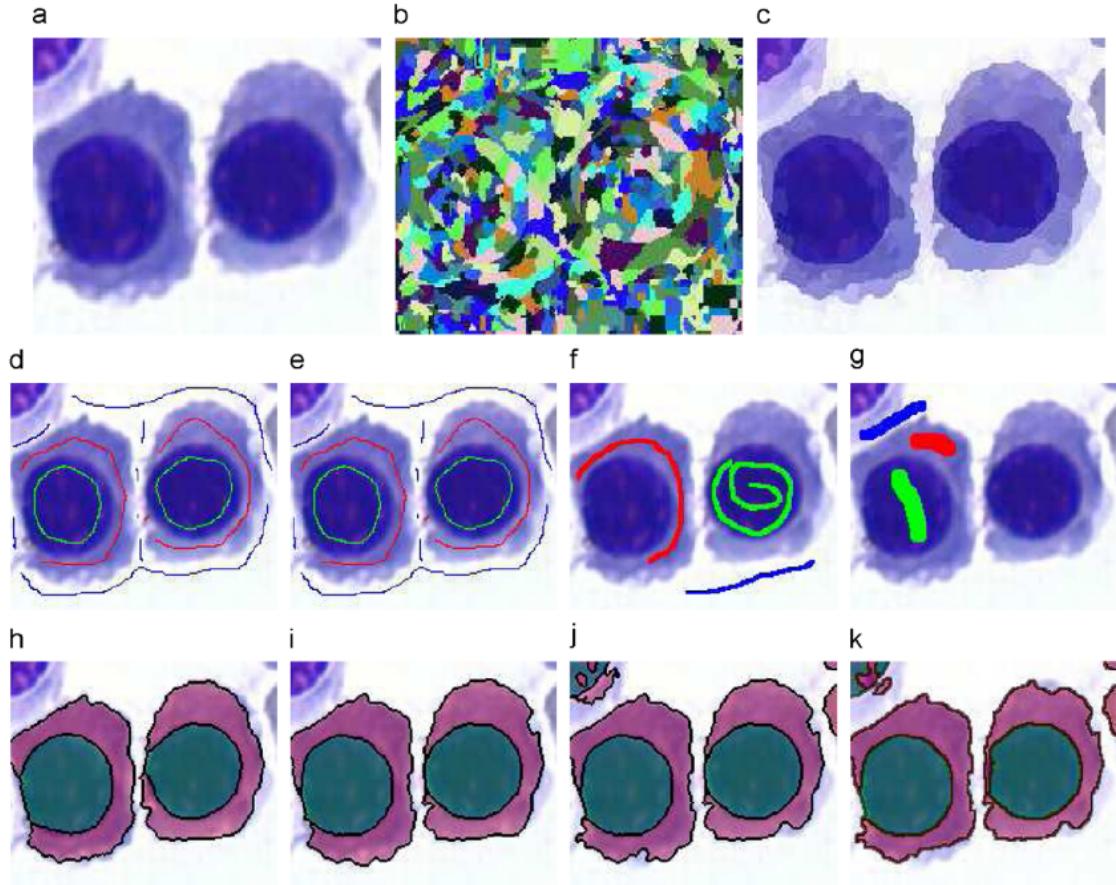


FIGURE 2.10 – (a) Image originale, (b) 639 zones (98% of reduction), (c) image reconstruite, (d)–(g) originale+étiquettes (h)–(k). Extrait de [4].

Dans [5], des filtres alternés séquentiels basés sur des dilatations sommet-arête et arête-sommet conduisent à des résultats comme ceux de la figure 2.12.

Le travail de Gunduz [17] se sert des graphes pour créer des attributs qui seront utiles pour la classification des cellules comme saines, inflammées ou cancéreuses.

Dans le travail de Gilboa [18], des opérateurs non-locaux sont définis pour traiter les textures et des zones répétitives. Dans ce cas, un lien entre la théorie spectrale des graphes et les opérateurs définis est tracé.

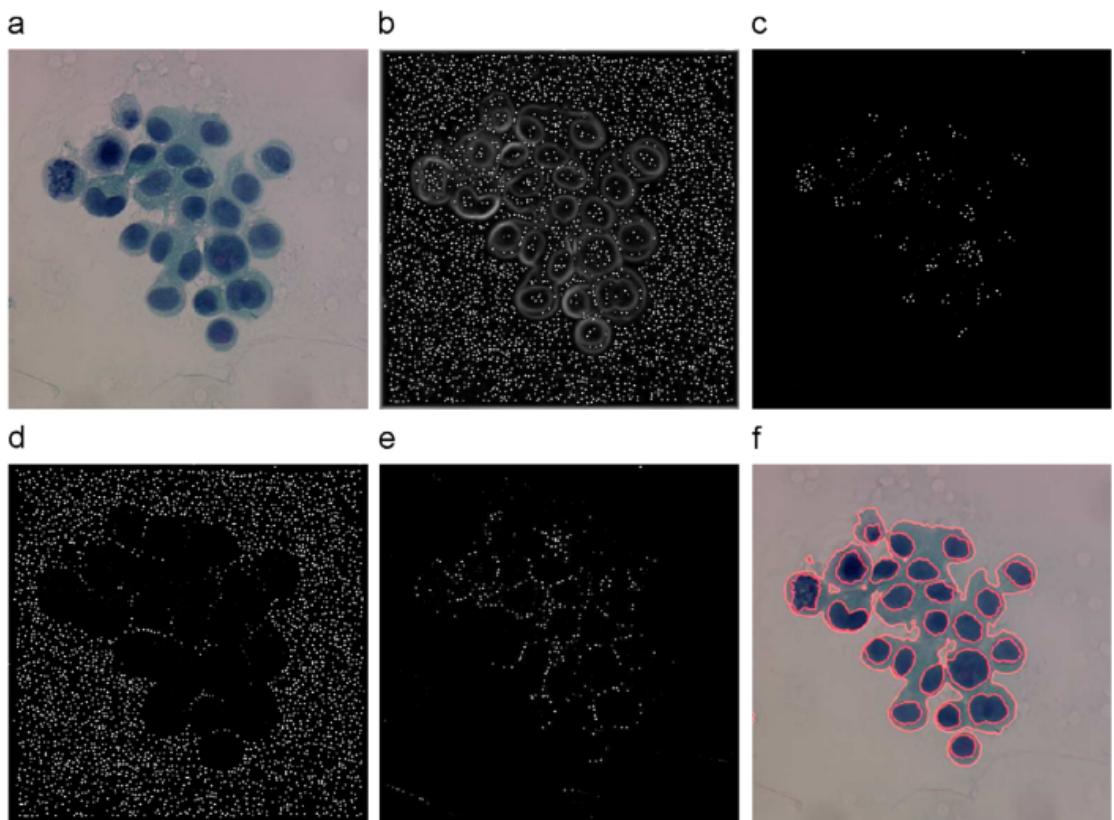


FIGURE 2.11 – Segmentation automatique d'une image cytologique automatiquement colorée basée sur un graphe de pixels à 8-connexité et partitions d'énergie (a) Image originale. (b) Minima du gradient. (c)–(e) Noyau, fond et minima du cytoplasme classifié avec un clustering flou. (f) Segmentation obtenue de la fusion avec des contours imposées en rouge. Extrait de [4].

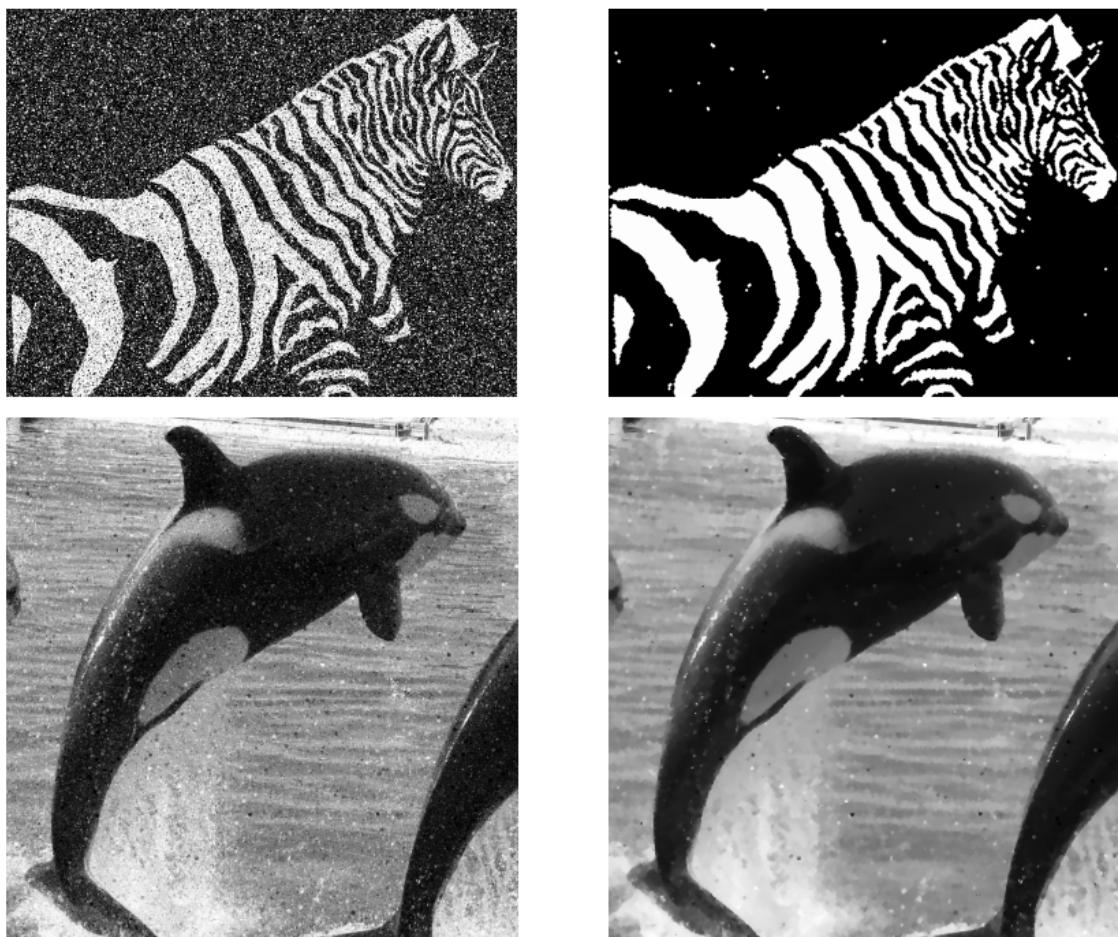


FIGURE 2.12 – Filtre alternés séquentiels définis sur des graphes appliqués à deux exemples. Extrait de [5].

Chapitre 3

La morphologie mathématique dans les traitement d'images

3.1 Morphologie mathématique

La morphologie mathématique est un concept créé en 1964 par Georges Matheron et Jean Serra dans les laboratoires de l’École des Mines de Paris [19, 20]. Développée à l’origine pour l’étude des matériaux poreux, la morphologie mathématique a été rapidement reconnue internationalement et plusieurs groupes de recherche ont été créés pour l’étudier. Conçue initialement pour répondre à des applications industrielles, la morphologie mathématique a été étendue au traitement d’images par Jean Serra à partir des années 1980 principalement.

La morphologie mathématique est en fait composée de plusieurs outils, surtout des opérations appliquées à des ensembles génériques d’où il peut être extrait une relation de hiérarchie comme les graphes où des régions discrètes du \mathbb{R}^n . Chacun de ces opérateurs possède des fortes propriétés, entre autres, la propriété de non réversibilité, mais aussi d’autres qui seront décrites ci-dessous.

- **Croissance** : Une opération appliquée à deux ensembles, un inclus dans l’autre, ne modifie pas cette relation. En termes mathématiques : $X \subseteq Y \iff Z(X, B) \subseteq Z(Y, B)$ (Pour Z opération générique). La propriété duale est la décroissance ;
- **Extensitivé** : Le résultat de l’opération inclut l’objet original. En termes mathématiques : $X \subseteq Z(X, B)$; Le contraire ($Z(X, B) \subseteq X$) est appelé anti-extensivité (Pour Z opération générique) ;
- **Commutativité** : Est la capacité à commuter avec une autre opération similaire. En termes mathématiques $Z(X \circ Y, B) = Z(X, B) \circ Z(Y, B)$ (Pour Z et \circ opérations génériques) ;
- **Invariance** : Est la capacité “de saturer” lors de plusieurs opérations de suite. En termes mathématiques : $Z(Z(X, B), B') = Z(X, \max(B, B'))$ (Pour Z opération générique). La propriété duale de l’invariance est l’itérativité qui est la capacité à cumuler suites d’opérations.
- **Idem-potence** : Est la capacité “ignorer” les suites d’opérations. En termes mathématiques : $Z(Z(X, B), B) = Z(X, B)$.

Ces propriétés, exprimés par des ensembles, s’étendent à des fonctions et plus généralement à des treillis. Les principaux opérateurs de la morphologie mathématique, et qui sont à la base de la création de tous les autres opérateurs sont la

dilatation et l'*érosion*. Les opérateurs sont définis par une addition de Minkowski (\oplus) qui correspond à une addition vectorielle. La dilatation est définie par :

$$D(X, B) = X \oplus B = \{x + b \mid x \in X, b \in B\} \quad (3.1)$$

On parle alors de la dilatation de l'ensemble X par l'ensemble B . La dilatation est une opération croissante par rapport à X et à B ($B \subseteq B' \Rightarrow D(X, B) \subseteq D(X, B')$), extensive (si et seulement si l'origine appartient à l'ensemble B), qui commute avec la réunion, et itérative.

De manière similaire l'*érosion* est définie comme un équivalent de la soustraction de Minkowski donné par :

$$E(X, B) = X \ominus B = \{x \in \mathbb{R}^n \mid \forall b \in B, x - b \in X\} \quad (3.2)$$

C'est l'*érosion* de l'ensemble X par l'ensemble B . L'*érosion* est une opération croissante par rapport à X et décroissante par rapport à B ($B \subseteq B' \Rightarrow D(X, B') \subseteq D(X, B)$), anti-extensive (si et seulement si l'origine appartient à l'ensemble B), qui commute avec l'intersection, et itérative.

Deux autres opérateurs immédiats très importants sont l'*ouverture* et la *fermeture*. Ils sont définis de la manière suivante. Une ouverture est définie comme une opération de d'*érosion* suivie d'une opération de dilatation :

$$O(X, B) = D(E(X, B), B) \quad (3.3)$$

On parle alors de l'*ouverture* de l'ensemble X par l'ensemble B . L'*ouverture* est une opération croissante par rapport à X , anti-extensive, invariante et idempotente.

Une fermeture est définie comme une opération de dilatation suivi d'une opération d'*érosion* :

$$C(X, B) = E(D(X, B), B) \quad (3.4)$$

C'est la fermeture de l'ensemble X par l'ensemble B . La fermeture est une opération croissante par rapport à X , extensive, invariante et idempotente.

3.2 L'extension aux fonctions

Pour étendre le concept de morphologie mathématique formulé aux ensembles ainsi que ces opérateurs, il nous faut définir préalablement des équivalents fonctionnels d'opérations binaires :

$\cup \rightarrow sup$	Réunion \rightarrow Supremum
$\cap \rightarrow inf$	Intersection \rightarrow Infimum
$\subseteq \rightarrow \leq$	Inclus \rightarrow Plus petit ou égal
$\supseteq \rightarrow \geq$	Inclut \rightarrow Plus grand ou égal

À partir de ces opérateurs nous pouvons redéfinir les opérateurs de dilatation d'érosion, d'ouverture et de fermeture appliqués aux fonctions comme il suit ($f : \mathbb{R}^n$ ou $\mathbb{Z}^n \rightarrow \mathbb{R}^+$ ou \mathbb{N}) :

$$\begin{aligned} \text{Dilatation : } D(f, B)(x) &= \sup\{f(y) | y \in \hat{B}_x\} & \text{Ouverture : } O(f, B) &= D[E(f, B), B] \\ \text{Érosion : } E(f, B)(x) &= \inf\{f(y) | y \in \hat{B}_x\} & \text{Fermeture : } C(f, B) &= E[D(f, B), B] \end{aligned}$$

Dans ce cas, les éléments structurants ne se limitent plus à des formes ou ensembles mais pourront être aussi des fonctions. Un exemple de fonction possible et équivalent à une boule serait :

$$g(x) = \begin{cases} 0, & \text{si } x \in B(O, r) \text{ (} O \text{ boule de centre } 0 \text{ et rayon } r \text{)} \\ -\infty, & \text{ailleurs} \end{cases} \quad (3.5)$$

Les opérateurs morphologiques sont définis comme :

$$D(f, B)(x) = \sup_y \{f(y) + g(x - y)\} \quad (3.6)$$

$$E(f, B)(x) = \inf_y \{f(y) - g(y - x)\} \quad (3.7)$$

Dans ce cas, les opérateurs morphologiques ont pour caractéristiques de couper des pics (érosion et ouverture) et boucher les vallées (dilatation et fermeture) d'autant plus que l'élément structurant est grand. Les propriétés trouvées pour les définitions sur des ensembles sont toujours valables (avec les adaptations appropriées) et elles ne seront pas redonnées ici.

3.3 L'extension aux images : binaire, niveaux de gris, couleur

Si nous considérons les images comme des fonctions du \mathbb{R}^3 , éventuellement du \mathbb{R}^4 ou même du \mathbb{R}^n , ce que nous avons sont des hypersurfaces qui sont représentées comme des fonctions dans \mathbb{R} des vecteurs du \mathbb{R}^n de manière plus générique.

Donc, nous pourrions penser que le même raisonnement de fonctions vectorielles à valeurs réelles (ou vectorielles) serait facilement étendu à partir des opérateurs unidimensionnels avec le changement de $x \in \mathbb{R}$ pour $\mathbf{x} \in \mathbb{R}^n$.

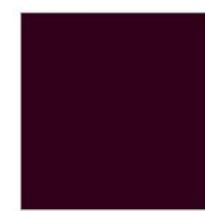
Par contre, l'extension aux images n'est pas immédiate, puisque il faut prendre en compte la corrélation entre les canaux de couleur. Le système de couleurs RGB n'est pas un système de couleurs absolu, ce qui signifie que la distance entre les coordonnées des couleurs n'est pas représentative de la différence entre les couleurs (en gros, cela revient à dire qu'il peut avoir deux coordonnées dans le système RGB qui donnent la même couleur). Cet aspect peut être observé dans la figure 3.1.

Hex: #	331900
Red:	51
Green:	25
Blue:	0



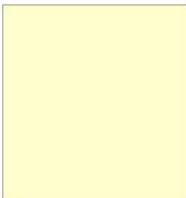
(a) Couleur 1

Hex: #	330019
Red:	51
Green:	0
Blue:	25



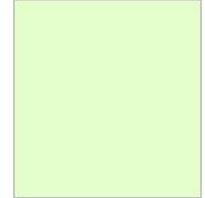
(b) Couleur 2

Hex: #	FFFFCC
Red:	255
Green:	255
Blue:	204



(c) Couleur 3

Hex: #	E5FFCC
Red:	229
Green:	255
Blue:	204



(d) Couleur 4

FIGURE 3.1 – Comparatif entre deux couleurs du système RGB. Les deux premières couleurs sont séparées par une distance colorimétrique plus grande que les deux dernières couleurs, par contre la différence est bien plus évidente dans le deuxième cas.

Alors, quand nous appliquons des opérateurs de morphologie mathématique, nous pouvons augmenter/réduire l'intensité d'un canal de manière plus importante que dans les autres. Cela générera des fausses couleurs, qui pourraient même ne pas exister dans l'image de départ et qui risquent ne pas avoir de sens pour le contexte.

Alors, même si nous définissons un ordre parmi les couleurs avant d'appliquer la morphologie mathématique, il faudra être conscient que cela signifie donner une préférence à une couleur, ce qui n'est pas forcément désirable. Le travail d'Angulo [21] essaye de réunir plusieurs approches et méthodes d'extension de la morphologie mathématique et d'étudier les précautions que doivent être prises en compte pour que la généralisation soit pertinente. Un travail encore plus détaillé sur des images couleurs et hyperspectrales pourra être trouvé en [22].

3.4 Watershed

L'un des opérateurs de morphologie mathématique plus avancés et plus puissants issus de la morphologie mathématique est la ligne de partage des eaux (LPE), ou *Watershed*. En effet ce nom et le principe intuitif derrière la technique sont fortement liés au contexte de création de la morphologie mathématique. On rappelle que la morphologie mathématique a été conçue pour l'étude des matériaux poreux, alors, le premier modèle de ces pores était des cratères rondes sur la surface des matériaux.

En partant de ce principe, nous pourrions imaginer des images comme un paysage composé de plusieurs pics et vallées dans un espace du \mathbb{R}^2 avec l'auteur étant le niveau de gris dans chaque point.

Les pics sont les lignes de partage des eaux et les vallées sont les bassins versants. L'intuition nous donne une bonne piste sur la signification de ces concepts. Les bassins versants sont les régions de l'espace associés à un minimum local. En termes physiques, c'est une région telle que tout volume d'eau partant d'un point dans cette

région écoulera vers le centre de la région, soit le minimum local. La ligne de partage des eaux est, donc, le limite entre ces bassins versants.

Pour définir ces termes de façon plus formelle, il nous faudra d'abord définir certains concepts préliminaires. Avec eux, nous serons capables de clarifier une définition mathématique de ces deux concepts clés.

Plus grande pente

Une pente est simplement une version large d'un gradient. Il n'est pas nécessaire de le définir comme le coefficient angulaire entre deux points séparés par une distance infinitésimale, mais simplement comme le coefficient angulaire entre deux points quelconques de l'espace. La plus forte pente est, bien sûr, le maximum de la pente dans une région donnée. C'est ce qui l'équation 3.8 représente.

$$\text{Desc}(x) = \max\left\{\frac{f(x) - f(y)}{d(x, y)}, y \in \mathcal{V}(x)\right\} \quad (3.8)$$

Où, $f(x)$ est la valeur de l'image dans le point x et $d(x, y)$ est une mesure de distance arbitraire entre deux points.

Dénivelé d'un chemin

Un chemin π n'est rien de plus qu'une suite de points x_0, x_1, \dots, x_n . Le dénivelé d'un chemin est simplement la somme de toutes les pentes pondérées par la distance entre deux points consécutifs.

$$T_f(\pi) = \sum_{i=1}^2 d(x_{i-1}, x_i) \text{Cout}(x_{i-1}, x_i) \quad (3.9)$$

Où :

$$\text{Cout}(x_{i-1}, x_i) = \begin{cases} \text{Desc}(x_{i-1}) & , \text{if } f(x_{i-1}) > f(x_i) \\ \text{Desc}(x_i) & , \text{if } f(x_{i-1}) < f(x_i) \\ \frac{\text{Desc}(x_{i-1}) + \text{Desc}(x_i)}{2} & , \text{if } f(x_{i-1}) = f(x_i) \end{cases} \quad (3.10)$$

Distance topographique

La distance topographique est définie comme le plus petit dénivelé possible pour un chemin qui relie les points x et y . Il est intéressant de s'apercevoir que cette distance vaudra 0 sur un plateau.

$$T_f(x, y) = \inf\{T_f(\pi), \pi = (x = x_0, x_1, \dots, x_n = y)\} \quad (3.11)$$

Bassin versant

Le bassin versant associé au minimum local M_i est finalement défini comme l'ensemble des points x tels que le minimum local associé est celui que possède une distance topographique plus petite que n'importe quel autre minimum local.

$$BV(M_i) = \{x \mid \forall j \neq i, T_f(x, M_i) + f(M_i) < T_f(x, M_j) + f(M_j)\} \quad (3.12)$$

Ligne de partage des eaux

La ligne de partage des eaux est enfin définie comme la courbe de frontière entre tous les bassins versants de l'image :

$$LPE(f) = [\cup_i BV(M_i)]^C \quad (3.13)$$

L'objectif de la ligne de partage des eaux est de segmenter l'image à partir des maxima du gradient de l'image, c'est-à-dire, les contours. Cela est assez intuitif comme bonne segmentation, même s'il peut être un peu naïve comme solution. Le principe de la LPE est, alors, de remplir des bassins versants de l'image jusqu'à que deux régions se rejoignent. Quand les régions se rejoignent, c'est un signe qu'un maximum du gradient a été atteint.

Malgré l'efficacité de la méthode, la ligne de partage des eaux exige un pré-traitement et post-traitement. La LPE applique à une image sans pré-traitement nous donnera un résultat peu satisfaisant, comme celui de la figure 3.2. Si au contraire, une reconstruction morphologique est appliquée avant de calculer le gradient, le résultat est fortement amélioré. Encore, si une contrainte sur la taille des régions est appliquée, le résultat peut être encore mieux. Après une jonction des régions pourrait être utile pour améliorer le résultat final.

3.5 Exemples

Il est aussi facile de trouver des exemples où la morphologie mathématique est utilisée pour le traitement d'images surtout pour la segmentation d'images, la reconstruction d'objets et la construction de représentations compactes des objets. Voici quelques exemples.

Dans [23] une étude pour la différentiation des hyperplasies folliculaires et des lymphomes folliculaires à petites cellules clivées en utilisant des granulométries et des transformations de distance.

Un autre exemple est celui de [6] où des opérateurs de morphologie mathématique non-locaux basées sur les équations différentielles partielles sont définis pour le traitement et la segmentation des images. Des exemples d'application pour le débruitage et pour la segmentation sont présentés dans les figures 3.3 et 3.4.

Un dernier exemple est celui de [7] où la morphologie mathématique est utilisé pour la détection de bords dans des images médicales (voir la figure 3.5).

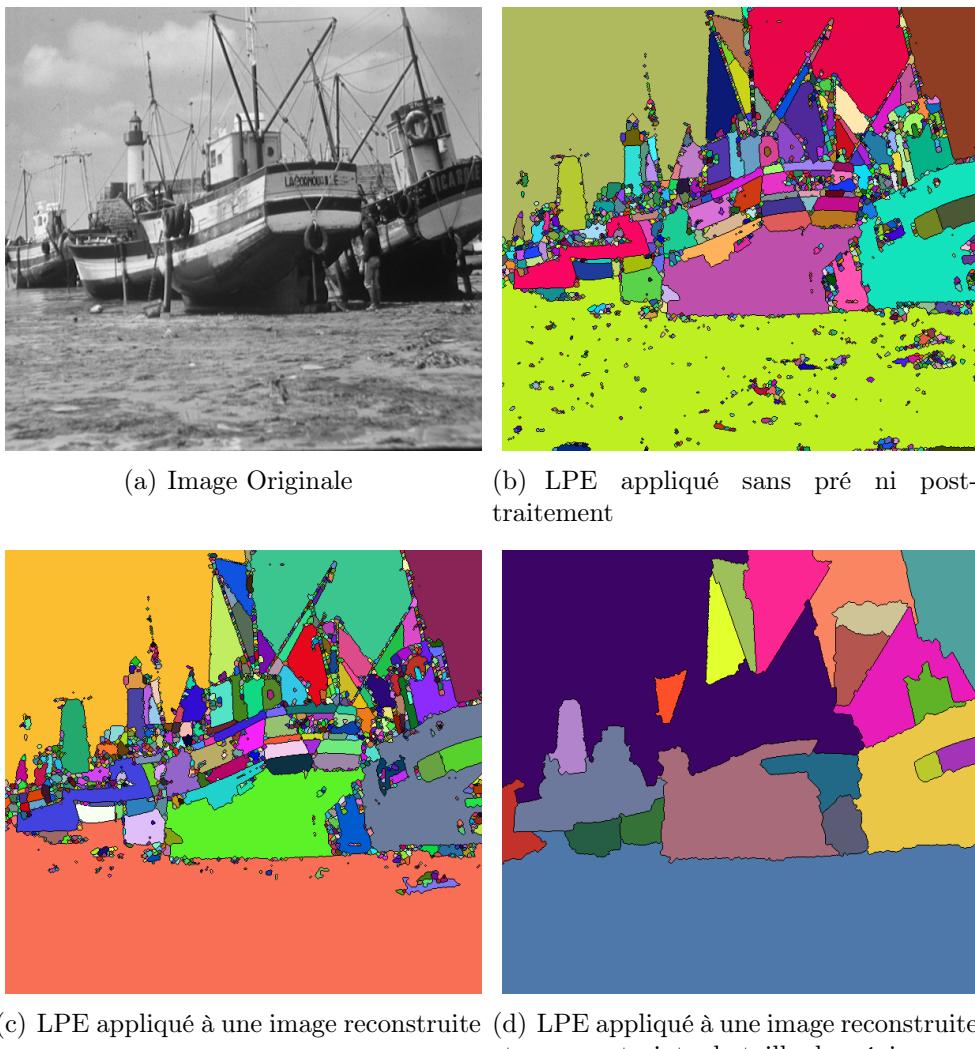


FIGURE 3.2 – R sultat de la ligne de partage des eaux appliqu    une image dans diff rentes conditions de pr  et post-traitement

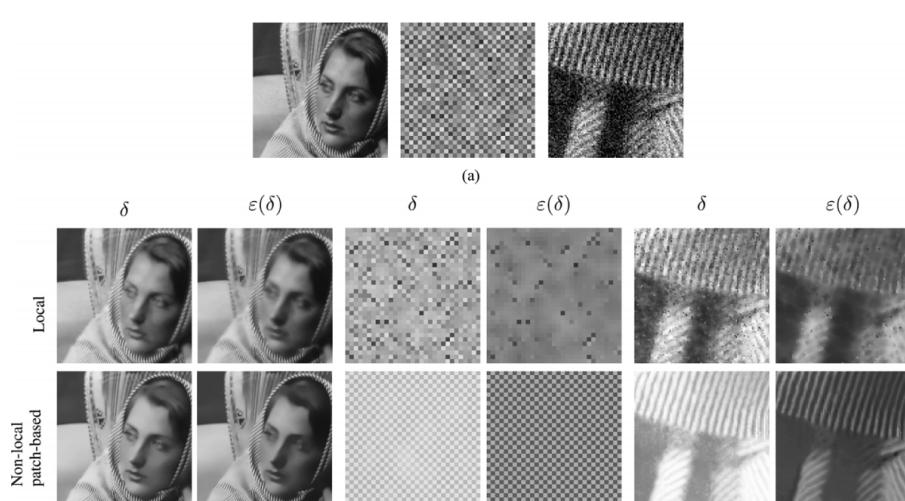


FIGURE 3.3 – Application des filtres morphologiques bas es sur EDPs   la d bruitage. Extrait de [6].

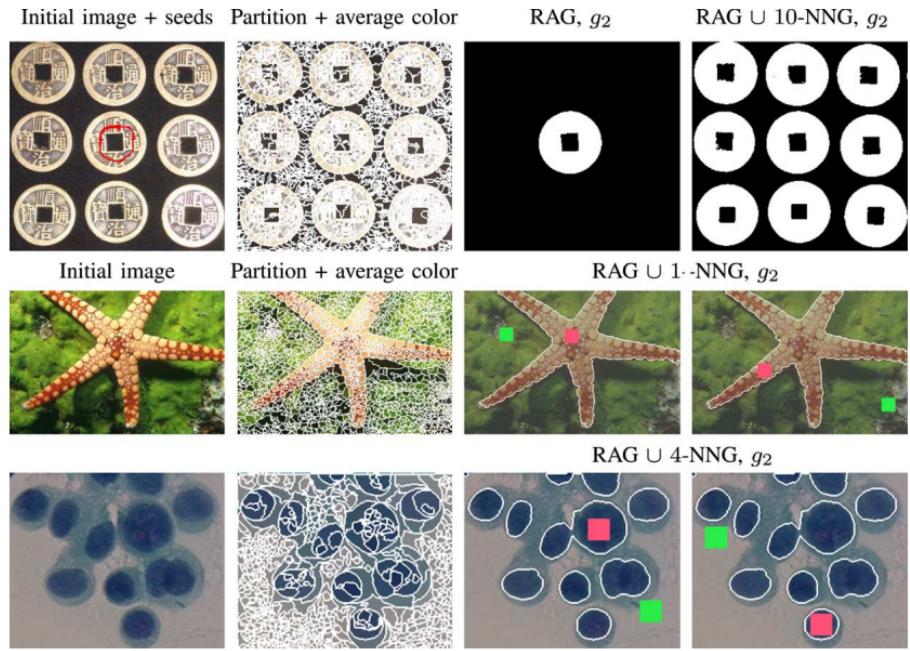


FIGURE 3.4 – Application des filtres morphologiques basées sur EDPs à la segmentation. Extrait de [6].

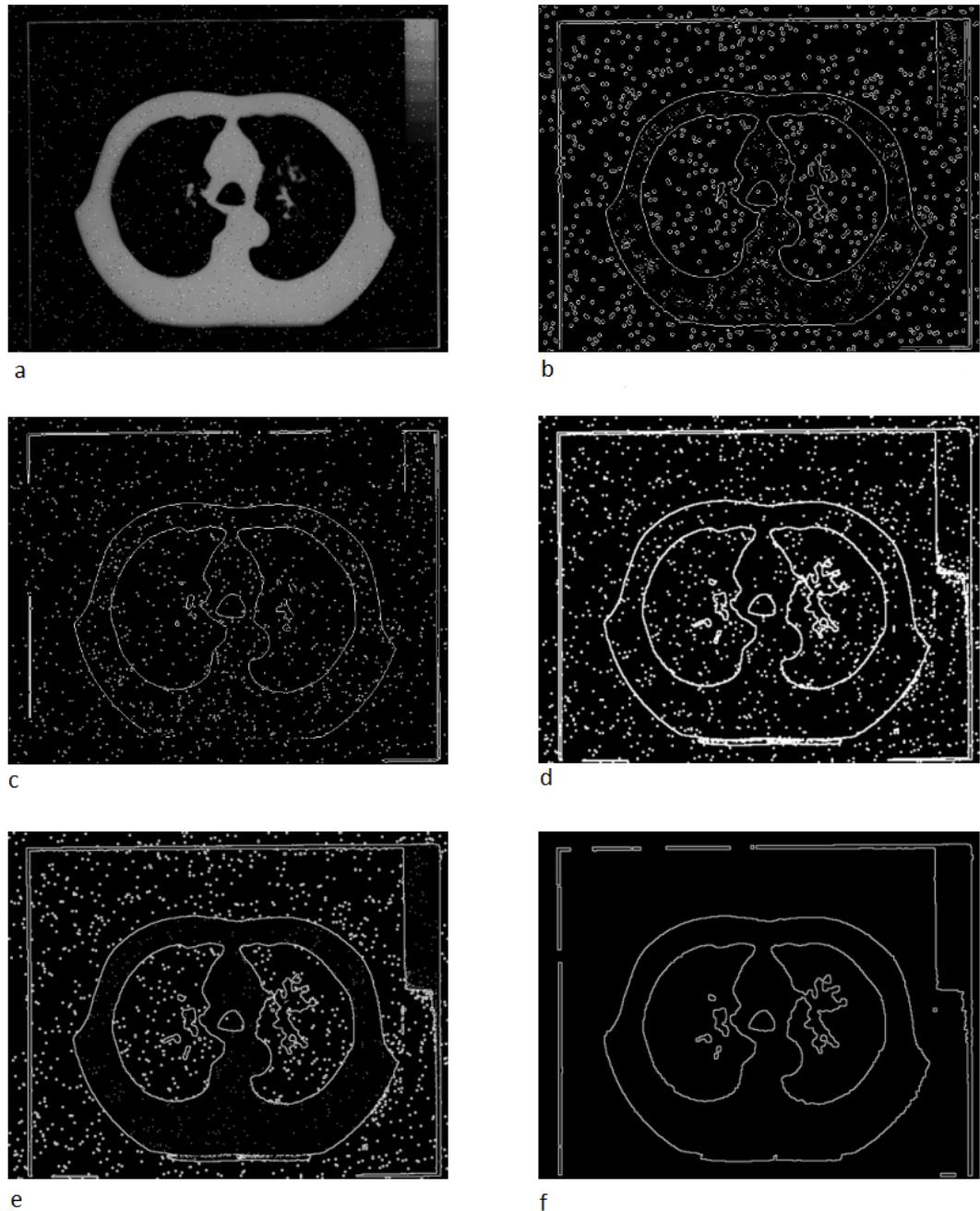


FIGURE 3.5 – Détection de bords dans des images du poumon en CT : (a) Image originale avec du bruit *salt-and-pepper*. (b) Image traitée avec Laplacien de Gaussienne. (c) Image traitée avec le détecteur de Sobel. (d) Image traitée avec le Gradient morphologique. (e) Image traitée avec un détecteur de bords par dilatation. (f) Image traitée avec un nouvel opérateur morphologique proposé. Extrait de [7].

Chapitre 4

Application

4.1 La combinaison des deux outils

Quand nous appliquons une représentation sous forme de graphes aux images, nous pouvons aussi définir des opérateurs de morphologie mathématique qui pourront se montrer utiles pour les différentes tâches de traitement d'images. La morphologie mathématique est étendue aux graphes dans plusieurs articles où des considérations à propos de formulations sont faites ainsi que différentes définitions des opérateurs ou même de nouveaux opérateurs sont proposés pour la morphologie mathématique.

L'une des premières abstractions de la morphologie mathématique aux graphes a été proposée en 1988 par Vincent [2]. Dans ce travail il présente plusieurs structures de graphes potentiellement utiles ainsi que leurs algorithmes de construction. De plus les principaux opérateurs morphologiques (érosions et dilatations, filtres morphologiques, fonction distance, squelettes, opérateurs géodésiques, reconstruction, étiquetage, etc.) sont définis et étudiés pour des graphes. Il est présenté également des algorithmes rapides pour construire les graphes transformés.

Le travail d'Heijmans [24] présente une théorie systématique pour la construction d'opérateurs morphologiques sur des graphes à partir d'une analyse critique du travail antérieur de Vincent [2]. Il définit un graphe structurant, qui permet de générer des nouveaux opérateurs de manière similaire au principe des éléments structurants. Les opérateurs créés sont invariants par rapport à la symétrie des graphes. Malgré la ressemblance entre la morphologie traditionnelle et celle des graphes, il montre que tous les résultats de la morphologie n'ont pas d'extension aux graphes puisque nous n'avons pas de garantie d'équivalence de structures à chaque sommet (les graphes ne sont pas forcément des structures régulières spatialement).

Dans le travail de Cousty [25], des opérateurs morphologiques sur une trame pour tous les sous-graphes d'un graphe G sont étudiés. Deux adjonctions duales ont été considérées à partir de ce principe. Plusieurs ouvertures, fermetures, granulométries et filtres alternés séquentiels qui agissent sur le graphe G et ses sous-graphes ont été définis. L'approche proposée est évaluée en comparaison avec l'approche typique d'éléments structurants et le résultat obtenu se montre supérieur dans la réduction du bruit par rapport à la mesure du PNSR de l'image.

Un article plus récent de Stell [26] présente une nouvelle formulation de la morphologie mathématique sur des graphes pour développer une extension aux ensembles approximatifs. Une approche de la morphologie mathématique en termes

de relations est formulée et les nouveaux opérateurs sont donnés.

Ce sont des travaux plus théoriques sur la morphologie mathématique qui ont permis l'application de ces deux outils au traitement des images de sorte à modéliser une prise en compte des relations de proximité dans un espace choisi (distances physiques, colorimétriques, taille, distribution, etc.). À partir de ces travaux des applications plus concrètes ont été développées et ce sont ces applications qui seront présentées, étudiées et comparées dans la suite de ce travail.

4.2 Travail de référence

Le travail de Stawiaski [27] est l'inspiration de ce projet. Son travail de segmentation sur des images médicales avec *low-level watershed* sera décrit, reproduit et des propositions d'amélioration seront faites.

Dans un premier temps, Stawiaski fait des rappels sur la théorie des graphes, où il définit les concepts d'arbre et forêt, forêt couvrante de poids maximal (*maximal spanning forests*), arbre couvrant de poids minimal (*minimal spanning trees*) ainsi que la coupe minimale (*minimal cuts*) et le problème du plus court chemin (*shortest path forest cuts*). Stawiaski prouve aussi le lien entre ces algorithmes (coupe minimale et plus court chemin) qui, pour une moyenne d'ordre $n \rightarrow \infty$, sont équivalents à la forêt couvrante de poids maximal.

Après ces définitions, il fait une étude des graphes d'adjacence de pixels et d'adjacence de régions. Une étude des segmentations dans le cas à deux et à plusieurs classes ainsi qu'avec l'utilisation de marqueurs est menée. Le prochain pas est d'ajouter la segmentation morphologique à la coupure de graphes.

Enfin, les concepts définis jusqu'à ce point sont utilisés pour la segmentation de différents types d'images avec une méthode similaire pour chacun, et pourtant adaptée à chaque jeu de données. Sont utilisées des images cardiaques, pulmonaires et du foie. Pour chaque image un graphe d'adjacence de régions est construit et des marqueurs sont introduits par l'utilisateur (griffonnage). Ces deux données sont utilisées pour appliquer une méthode de coupure de graphes qui pourra être améliorée avec des nouvelles annotations faites par l'utilisateur.

Ce qui est appelé *low-level watershed* est en fait une application de la ligne de partage des eaux sans aucun pré-traitement comme indiqué dans la section sur le sujet. L'une des motivations pour l'appliquer de cette manière est l'hypothèse (assez raisonnable) que tous les contours importants de l'image sont préservés. De plus, le graphe formé par les régions générées par la LPE est sensiblement plus léger et viable pour une implémentation dans un ordinateur commun surtout pour les images 3D, qui sont l'objectif du travail de Stawiaski.

Alors, le premier pas de la solution proposée par Stawiaski c'est de transformer l'image 3D en une image de régions où les pixels deviennent indistinguables dans chaque région. Avec cette manœuvre nous passons de presque 9 millions de pixels à autour de 25 mille régions (pour une image 240x240x155).

Un graphe d'adjacence de régions est, enfin, construit en reliant toutes les régions distinctes deux à deux telles que, deux pixels adjacents dans un graphe de pixels pré-défini appartiennent à ces régions (l'un à r_i et l'autre à r_j). En effet, les poids du graphe seront la somme de tous les poids calculés sur toute paire de pixels adjacents appartenant aux deux régions r_i et r_j .

Pour définir les poids utilisés dans la liaison entre les régions adjacentes du graphe Stawiaski fait appel à la géométrie riemannienne pour formuler la longueur d'un contour $|C_{(r_i, r_j)}|_R$ entre les régions r_i et r_j , qui est définie par l'équation suivante :

$$w_{r_i, r_j} = |C_{(r_i, r_j)}|_R = \sum_{\substack{m \in r_i, \\ n \in r_j, \\ m \in F_{(r_i, r_j)}, \\ n \in F_{(r_i, r_j)}}} g(\max\{||\nabla I(m)||, ||\nabla I(n)||\}) \quad (4.1)$$

où, r_i , r_j , qui sont deux régions de l'image I , $F_{(r_i, r_j)} = \{e_{m,n} \in E \mid m \in r_i, n \in r_j\}$, E est l'ensemble d'arêtes du graphe de pixels prédéfini sur l'image I , et

$$g(||\nabla I(p)||) = \left(\frac{1}{1 + ||\nabla I(p)||} \right)^k \quad (4.2)$$

Pour appliquer la coupure des graphes l'algorithme attend que l'utilisateur griffonne plusieurs coupes pour indiquer les régions de l'œdème et/ou tumeur. Les régions dont les pixels ont été griffonnés sont attachées avec un poids infini à la source ou au puits selon la convention de l'algorithme. Les autres régions seront alors classées selon la force des poids avec les voisins lors du *Graphcut*.

4.3 Application à la segmentation interactive d'images médicales

Comme le travail de Stawiaski utilise une base de données d'un challenge de MICCAI 2008 qui n'a pas été trouvé sur internet, une autre base de données d'imagerie médicale sera utilisé. C'est la base de données du Medical Segmentation Decathlon [28], qui possède plusieurs types d'acquisitions d'images de plusieurs organes. On se concentrera sur les images IRM de tumeur dans le cerveau (Task 1).

La base de données est composé de 750 images 3D MRI du cerveau de taille 240 x 240 x 155 chacune obtenu sur 4 modes d'acquisition différents à savoir :

- FLAIR
- Gadolinium-enhanced T₁-weighted
- T₁-weighted
- T₂-weighted

Les quatre modes d'acquisition permettent d'avoir une meilleure perception des structures étant donné que chacun des modes est caractérisé par un meilleur contraste entre différentes régions du cerveau. Voici un exemple d'une coupe de la même région du cerveau acquise en différentes modalités.

Les modes d'acquisition se réfèrent aux temps caractéristiques de l'acquisition des images. T₁ est un paramètre caractéristique de la dynamique du vecteur de magnétisation dans la direction parallèle au champ magnétique principal. T₂ est un paramètre caractéristique de la dynamique du vecteur de magnétisation dans le plan orthogonal au champ magnétique principal. Une pondération de ces paramètres revient à augmenter ou réduire le temps de récupération (TR) et le temps d'écho (TE) de manière à favoriser la visualisation de contraste entre structures de différentes valeurs de T₁ ou T₂.

Les images FLAIR ne sont qu'un T₂ "superpondéré". Tandis que les temps TR et TE sont de l'ordre de centaines et dizaines de millisecondes respectivement pour

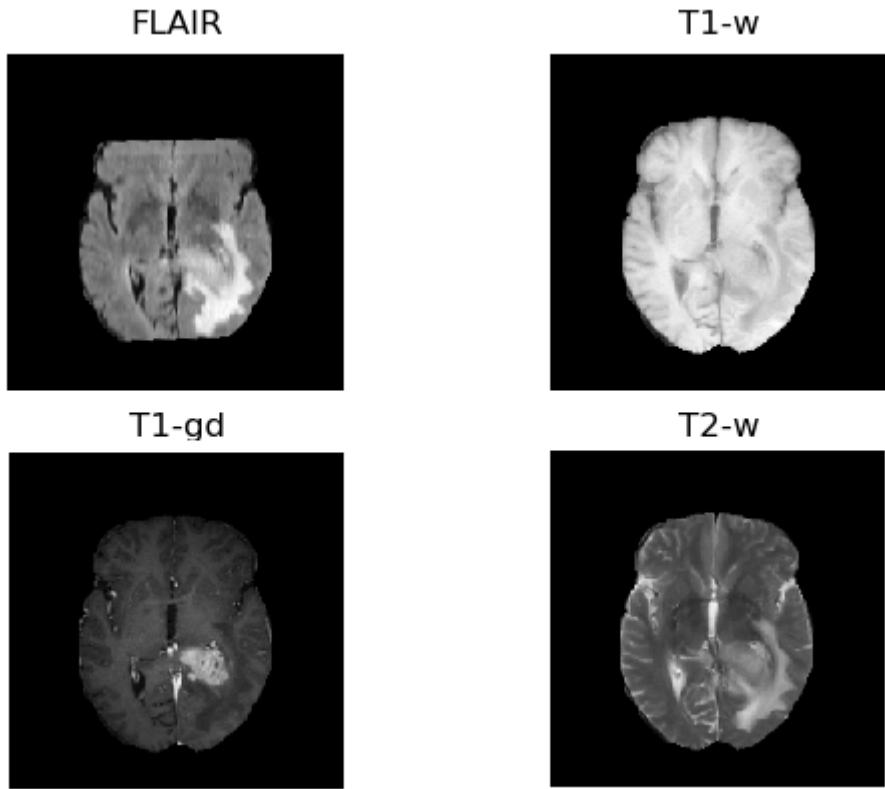


FIGURE 4.1 – Différents modes d’acquisition pour une même coupe du cerveau

un T_2 -weighted, les images FLAIR sont obtenues avec TR et TE de l’ordre des secondes et centaines de millisecondes respectivement. Cela permet d’avoir un meilleur contraste entre les structures solides du cerveau et le liquide cérébrospinal, ce qui est utile pour reconnaître des maladies comme la sclérose en plaques.

Les images sont divisées entre des ensembles d’apprentissage (484) et de test (266). Pour chaque ensemble d’images de l’ensemble d’apprentissage sur les différents modes d’acquisition, une image d’étiquettes est donné comme référence de la segmentation. Pourtant, la lésion n’est pas simplement segmentée comme appartenant à l’objet ou au fond (segmentation binaire) mais divisée en 4 classes qui représentent :

- Fond
- Tumeur non-rehaussée
- (Edème
- Tumeur rehaussée

Les classes 2 et 3 (Tumeur non-rehaussée et Tumeur rehaussée) sont des classes différencier par l’acquisition en Gadolinium-enhanced T_1 -weighted. En fait des zones actives des tumeurs ont une tendance à cumuler les agents de contraste, dans ce cas, le gadolinium, ce qui génère des images assez claires et donc, contrastées par rapport aux autres parties de l’image. Alors, cela dit quelles sont les régions actives de la tumeur et quelles sont les régions à traiter avec plus d’attention lors d’une chirurgie ou d’une procédure de chimiothérapie. La segmentation de référence donnée par la segmentation de la coupe présentée dans l’image 4.1 est présentée dans l’image 4.2.

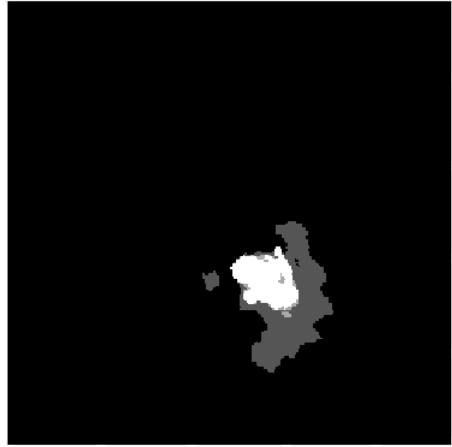


FIGURE 4.2 – Segmentation de référence de la segmentation de l'image 4.1

Le travail consistera à reproduire la segmentation par surfaces minimales à partir du *low-level watershed*, et en plus évaluer les caractéristiques de la fonction d'énergie utilisée. Après une discussion sur des améliorations possibles sera faite et ces améliorations seront implémentées à titre de comparaison.

4.3.1 Première approche

Une première approche a consisté à faire une segmentation binaire à partir des images FLAIR de tout l'œdème/tumeur. Ce choix a été fait parce que les images FLAIR sont les plus adaptées pour une détection facile de l'œdème, puis que l'œdème/tumeur apparaît relativement plus clair dans ces images et se distinguent du liquide cérébrospinal, qui apparaît noir dans ces images. Avec cette première segmentation nous pourrions avancer plus dans la segmentation de la tumeur en elle-même. L'approche initiale construit un graphe binaire où les arêtes qui lient les régions entre elles reçoivent les poids décrits par Stawiaski et présentés dans l'équation 4.1.

Les arêtes entre les régions et les étiquettes (source et puits) sont données par le marquage. Les zones marquées comme appartenant à l'œdème/tumeur sont attachées à l'étiquette de zone tumorale avec un poids infini, tandis qu'une zone marquée au fond, sera attachée à l'étiquette de fond avec un poids infini.

Pour obtenir les zones caractéristiques de chaque classe (œdème et cerveau pour l'instant) nous attendons de l'utilisateur le marquage des zones considérées comme œdème/tumeur et comme cerveau, comme proposé initialement par Stawiaski.

Pour faire le marquage des zones de manière simple et rapide, une coupe de l'image est présentée à l'utilisateur qui doit à son tour indiquer les extrémités d'une boîte englobante des zones appartenant au fond et à des zones tumorales (coins supérieur gauche et inférieur droit). Alors, toutes les zones telles qu'au moins un pixel appartienne à la boîte indiquée sont contraintes (à partir de l'affectation d'un poids infini) à l'une des classes.

Nous pourrions écrire de manière plus simple que les poids entre les régions et les noeuds terminaux valent :

$$w_{r_i,s} = \begin{cases} \infty, & \text{si } \exists p \in r_i \mid p \in M_t \\ 0, & \text{sinon} \end{cases} \quad (4.3)$$

$$w_{r_i,t} = \begin{cases} \infty, & \text{si } \exists p \in r_i \mid p \in M_s \\ 0, & \text{sinon} \end{cases} \quad (4.4)$$

où r_i est une région de la LPE, p un pixel de l'image, M_s est l'ensemble de pixels marquées comme appartenant à l'œdème/tumeur et M_t l'ensemble de pixels appartenant au fond/cerveau sain.

Un algorithme est proposé pour faire le marquage des zones et est présenté dans les annexes et quelques résultats sont dans la figure 4.3. À la fin de l'exécution de ce code, un fichier ‘.csv’ avec les coins respectifs est généré pour que l'utilisateur puisse s'en servir pour les autres étapes.

4.3.2 Deuxième approche

Puisque le choix d'appliquer la méthode proposée sur des images FLAIR a été justifié par son contraste par rapport au reste de l'image, un changement trivial serait de considérer la possibilité d'ajouter des arêtes entre source et puits qui prennent en compte le niveau de gris de chaque zone. Donc, les arêtes entre les régions et les labels (source et puits) sont données par l'équation 4.5.

$$w_{r_i,\{s,t\}} = \frac{(\bar{I}_{r_i} - \mu_{x_{\{s,t\}}})^2}{\sigma^2} \quad (4.5)$$

où, \bar{I}_{r_i} c'est le niveau de gris du pixel p , μ_{x_s} est le niveau de gris moyen des pixels appartenant à chaque classe (ici s ou t), et σ est l'écart-type du niveau de gris moyen des classes, supposé égal pour toutes les classes (en pratique nous choisissons le plus petit).

Pour obtenir la moyenne et la variance caractéristique de chaque classe nous garderons le système de marquage présenté dans la sous-section antérieure, les grandes différences par rapport à ce qui est proposé par Stawiaski sont les suivantes : (1) l'utilisateur pourra marquer une seule coupe au lieu de parcourir l'image en marquant plusieurs zones ; (2) au lieu de simplement contraindre les régions à appartenir à la classe indiquée, nous avons deux poids reliant la région aux deux classes selon la distance entre le niveau de gris moyen de chaque région et la moyenne estimée basée sur le marquage ; (3) le marquage ne consistera pas à griffonner l'image mais à indiquer les coins supérieur gauche et inférieur droit d'une zone appartenant entièrement à la classe indiquée.

La segmentation est effectivement sensible à ce marquage, alors il sera mieux d'avoir un marquage de la zone la plus claire du cerveau et la plus sombre de l'œdème/tumeur pour que la segmentation soit la plus précise possible. De plus, pour garantir un résultat plus satisfaisant il est appliqué une contrainte sur la variance des images : les variances calculées dans les zones marquées comme œdème/tumeur et fond ne doivent pas être plus que le double l'une de l'autre. Cela nous permet de faire la considération que les variances sont égales et alors nous n'utilisons que la plus petite parmi les deux variances calculées.

Le code qui permet d'exécuter cette segmentation est dans les annexes et quelques résultats sont dans la figure 4.4.

4.3.3 Troisième approche

Puisque nous proposons les deux premières approches, qui sont *a priori* indépendants, la question qui se pose naturellement c'est : pourquoi ne pas unifier les deux approches dans un seul ?

Alors, la troisième approche se repose sur l'évaluation de l'image à partir de *Graphcuts* mais en utilisant à la fois les zones du marquage et le niveau de gris entre les zones.

Le marquage fait pour les deux premières approches et les paramètres associés sont recyclés pour permettre une comparaison juste. Les zones marquées dans la première approche sont fixés avec poids infini à la classe du marqueur, les autres zones sont affectées à chacune des classes par rapport à la différence entre la moyenne de la région et de la classe, comme il est proposé initialement dans la deuxième approche.

Le changement apporté dans cette approche c'est la suppression de la variance. Au lieu de diviser la différence au carré entre la moyenne de la région et de la classe par la variance, comme indiqué dans l'équation 4.5, on divisera par une constante fixée pour toutes les images (β) afin que ces arêtes soient beaucoup plus petites que celles du marquage forcé à l'infini et plus proche des valeurs des poids de régularisation qui relie les régions entre elles, permettant une régularisation plus efficace.

Nous pourrions écrire de manière plus simple que les poids entre les régions et les noeuds terminaux valent :

$$w_{r_i,s} = \begin{cases} \infty, & \text{si } \exists p \in r_i \mid p \in M_t \\ 0, & \text{si } \exists p \in r_i \mid p \in M_s \\ \frac{(\bar{I}_{r_i} - \mu_{x_s})^2}{\beta}, & \text{sinon} \end{cases} \quad (4.6)$$

$$w_{r_i,t} = \begin{cases} \infty, & \text{si } \exists p \in r_i \mid p \in M_s \\ 0, & \text{si } \exists p \in r_i \mid p \in M_t \\ \frac{(\bar{I}_{r_i} - \mu_{x_t})^2}{\beta}, & \text{sinon} \end{cases} \quad (4.7)$$

où r_i est une région de la LPE, \bar{I}_{r_i} est la moyenne de niveau de gris dans la région de la LPE, p un pixel de l'image, $\mu_{x_{\{s,t\}}}$ la moyenne de niveau de gris de la classe estimé par le marquage, M_s est l'ensemble de pixels marquées comme appartenant à l'œdème/tumeur et M_t l'ensemble de pixels appartenant au fond/cerveau sain.

Le code qui permet d'exécuter cette segmentation est dans les annexes et quelques résultats sont dans la figure 4.5.

4.4 Mesures d'évaluation

Puisque nous avons les premiers résultats, il faut définir au moins une mesure qui puisse nous permettre d'évaluer le résultat obtenu et de comparer à des résultats atteints par Stawiaski dans son travail.

Pour cela nous allons, donc, définir trois mesures d'évaluation, qui ont aussi été utilisés dans le travail de référence et qui sont bien adaptées au contexte d'images tridimensionnelles.

- Critère de Dice (*DICE*) : Ce score est calculé comme deux fois le nombre de voxels à l'intersection de la segmentation et de la référence, divisé par la somme du nombre de voxels dans chacun des volumes. Cette valeur est égale à 1 pour une segmentation parfaite et égale à 0 comme valeur la plus basse possible, lorsqu'il n'y a aucun chevauchement entre la segmentation et la référence. L'erreur du critère de Dice est alors donnée comme 1 moins le score.

$$DICE = \frac{2|V_{seg} \cap V_{ref}|}{|V_{seg}| + |V_{ref}|} \quad (4.8)$$

- Distance de surface absolue symétrique moyenne (*Average symmetric absolute surface distance* ou SD_{abs}), en mm . Les voxels à la frontière de la segmentation et de la référence sont déterminés. Ceux-ci sont définis comme les voxels de l'objet qui ont au moins un voisin (parmi les 26 voisins les plus proches) qui n'appartient pas à l'objet. Pour chaque voxel dans ces ensembles, le voxel le plus proche dans l'autre ensemble est déterminé (en utilisant la distance euclidienne et les distances du monde réel, en prenant donc en compte les résolutions généralement différentes dans les différentes directions de balayage). Toutes ces distances sont stockées, pour les voxels limites à la fois de référence et de segmentation. La moyenne de toutes ces distances donne les moyennes de la distance de surface absolue symétrique. Cette valeur est 0 pour une segmentation parfaite. Il est indiqué par les distributeurs du jeu de données que les images ont été rééchantillonées pour que chaque voxel représente 1 mm^3 .

$$SD_{abs} = \frac{1}{|C_{seg}|} \sum_{s \in C_{seg}} |d(s, C_{ref})| + \frac{1}{|C_{ref}|} \sum_{r \in C_{ref}} |d(r, C_{seg})| \quad (4.9)$$

- Différence de volume absolu relatif (*Relative absolute volume difference* ou VD), en %. Le volume total de la segmentation est divisé par le volume total de la référence. De ce nombre, 1 est soustrait, la valeur absolue est prise et le résultat est multiplié par 100. Cette valeur est 0 pour une segmentation parfaite et supérieure à zéro sinon. Notez que la valeur parfaite de 0 peut également être obtenue pour une segmentation non parfaite, tant que le volume de cette segmentation est égal au volume de la référence.

$$VD = 100 * \left| 1 - \frac{V_{seg}}{V_{ref}} \right| \quad (4.10)$$

4.5 Performance

Nous appliquons les approches proposées sur une partie de l'ensemble d'apprentissage pour avoir les résultats présentés dans les figures ci-dessus. Nous voyons bien que les contours trouvés sont assez conservateurs dans le sens qu'ils sont souvent dans la segmentation de référence.

Ce que nous pouvons apprendre à partir de la visualisation de ces échantillons et en ajoutant l'expérience de l'observation des autres résultats, c'est que l'approche

1 garantit qu'au moins la zone marquée sera bien gardée dans la classe indiquée, cependant, il peut se passer que l'échantillon ne soit pas suffisant pour garantir une bonne propagation de la boîte marquée à toute la partie de l'image appartenant à cette classe.

De l'autre côté, l'utilisation de la moyenne tout seule risque de ne pas être suffisante aussi pour propager la zone jusqu'aux bords, qui sont en général moins clairs que le reste de l'œdème/tumeur.

En faisant l'application des deux approches proposées pour 50 des images de la base de données utilisée, les valeurs moyennes obtenues pour les critères de performance proposés sont indiquées dans la table 4.1.

Trois références sont indiquées pour permettre une comparaison avec ce que nous proposons. Ces références sont basées sur ce même jeu de données qui est largement utilisé (avec incrémentations) depuis 2012 [29]. Les références indiquées sont d'autres travaux proposés sur ce même jeu de données pour résoudre le problème de la segmentation avec des réseaux de neurones.

En [30], Isensee *et al.* propose un cadre nnU-Net («no-new-Net»). Il réside sur un ensemble de trois modèles U-Net relativement simples qui ne contiennent que faibles des modifications du U-Net d'origine. De plus Le nnU-Net adapte automatiquement ses architectures à la géométrie d'image donnée. Malgré la grande capacité d'adaptation de ce modèle, sa performance sur la tâche de segmentation des zones tumorales, par rapport aux autres tâches proposées pour le Medical Segmentation Decathlon est un peu affaiblie.

En [31], Perslev *et al.* propose un réseau *deep learning* simple et soigneusement évalué pour la segmentation de volumes d'images médicales arbitraires. Le système ne nécessite aucune information spécifique à la tâche, aucune interaction humaine et est basé sur une topologie de modèle fixe et un ensemble d'hyperparamètres fixes, éliminant le processus de sélection du modèle et sa tendance inhérente à entraîner un sur-ajustement au niveau de la méthode. Comme dans le premier cas, ce réseau a été proposé pour être généralisé à d'autres applications et cela explique sa performance peu accrue.

En [32], Yuan *et al.* présente un nouveau réseau antagoniste génératif attentionnel (UAGAN) à deux flux pour la segmentation d'images médicales non appariées multimodales. Ce modèle est flexible et peut utiliser plus de deux modalités en raison de la structure unifiée et a moins de paramètres que le même modèle non unifié. Pour capturer les fonctionnalités invariantes à la modalité bénéfiques à la segmentation, il est possible de fusionner les fonctionnalités des flux de segmentation et de transfert. Dans ce cas nous parlons d'une méthode optimisée pour la tâche de segmentation des zones tumorales du cerveau, alors, les mesures d'évaluation sont nettement plus importantes.

Les résultats présentés montrent que l'algorithme proposé semble être peu performant surtout comparativement au critère de Dice disponible pour tous les algorithmes. Pourtant, une analyse un peu plus judicieuse des résultats montrent que l'algorithme a des résultats très significatifs dans la majorité des cas, et très mauvais dans les autres, et cela dans les deux approches proposées.

En appliquant un seuillage pour supprimer les résultats de la première approche qui ont eu un critère de Dice inférieur à 0.25 (moins d'un quart de la tumeur trouvée), les mesures d'évaluation s'élèvent à 0.692 pour le critère de Dice et la distance moyenne est réduite à 2.25 mm. Cela nous montre que non seulement nos résultats

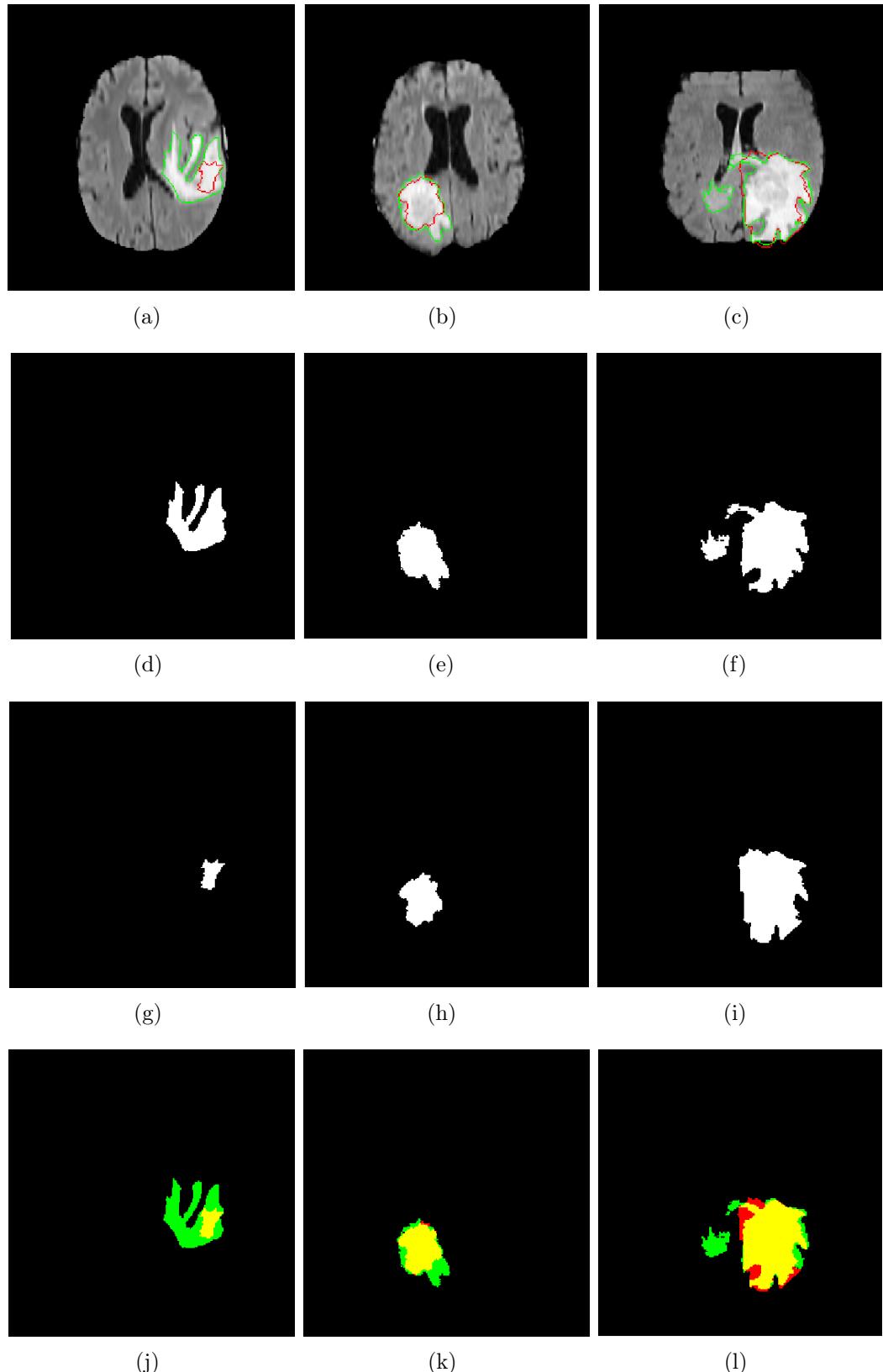


FIGURE 4.3 – Segmentation de l'œdème par l'approche 1 : (a)-(c) Images originales avec les contours trouvé et référence, (d)-(f) Segmentation de référence (g)-(i) Segmentation obtenue, (h)-(k) Overlap des segmentations, référence en vert, obtenue en rouge, coïncidence en jaune

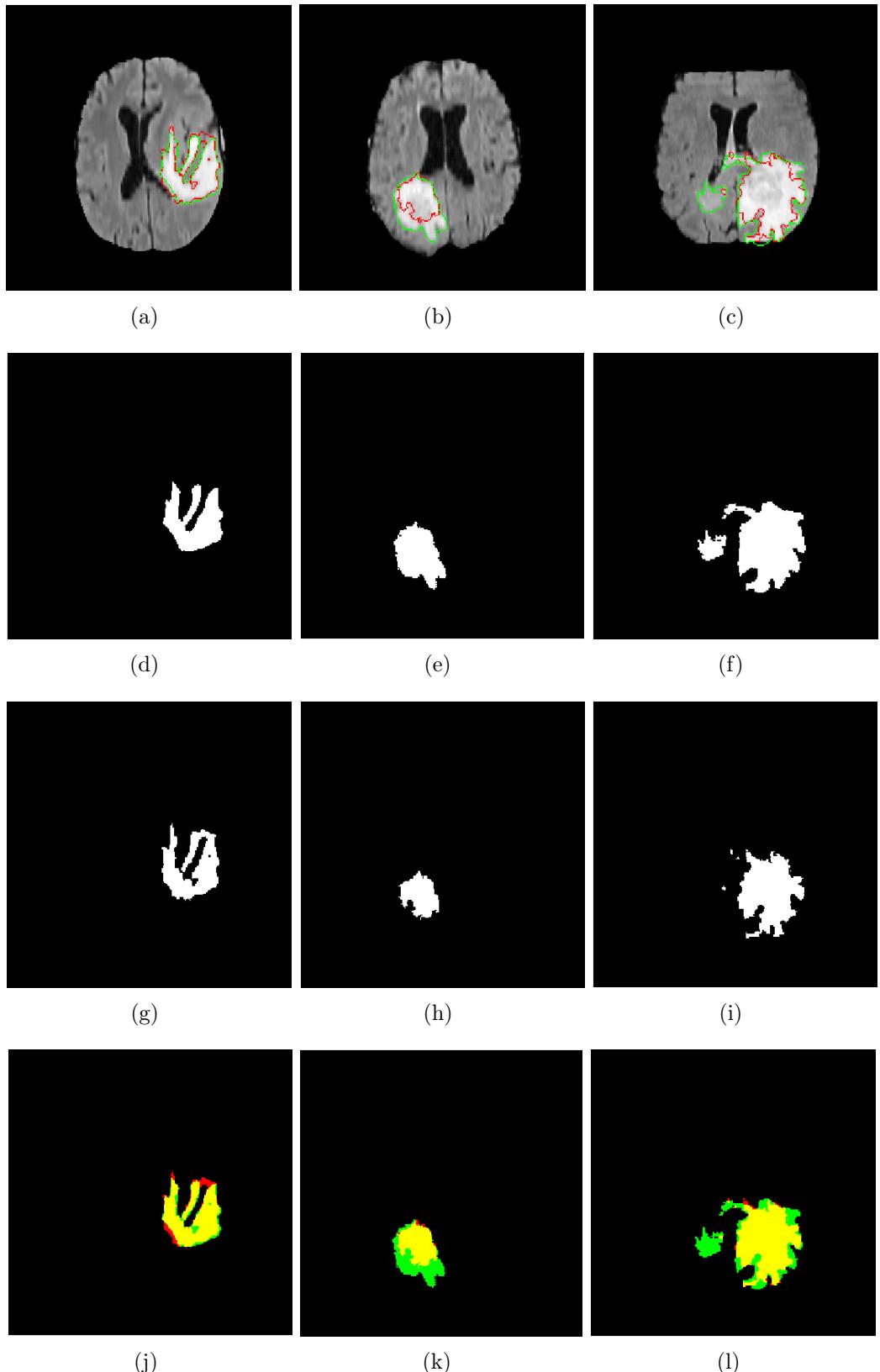


FIGURE 4.4 – Segmentation de l’œdème par l’approche 2 : (a)-(c) Images originales avec les contours trouvé et référence, (d)-(f) Segmentation de référence (g)-(i) Segmentation obtenue, (h)-(k) Overlap des segmentations, référence en vert, obtenue en rouge, coïncidence en jaune

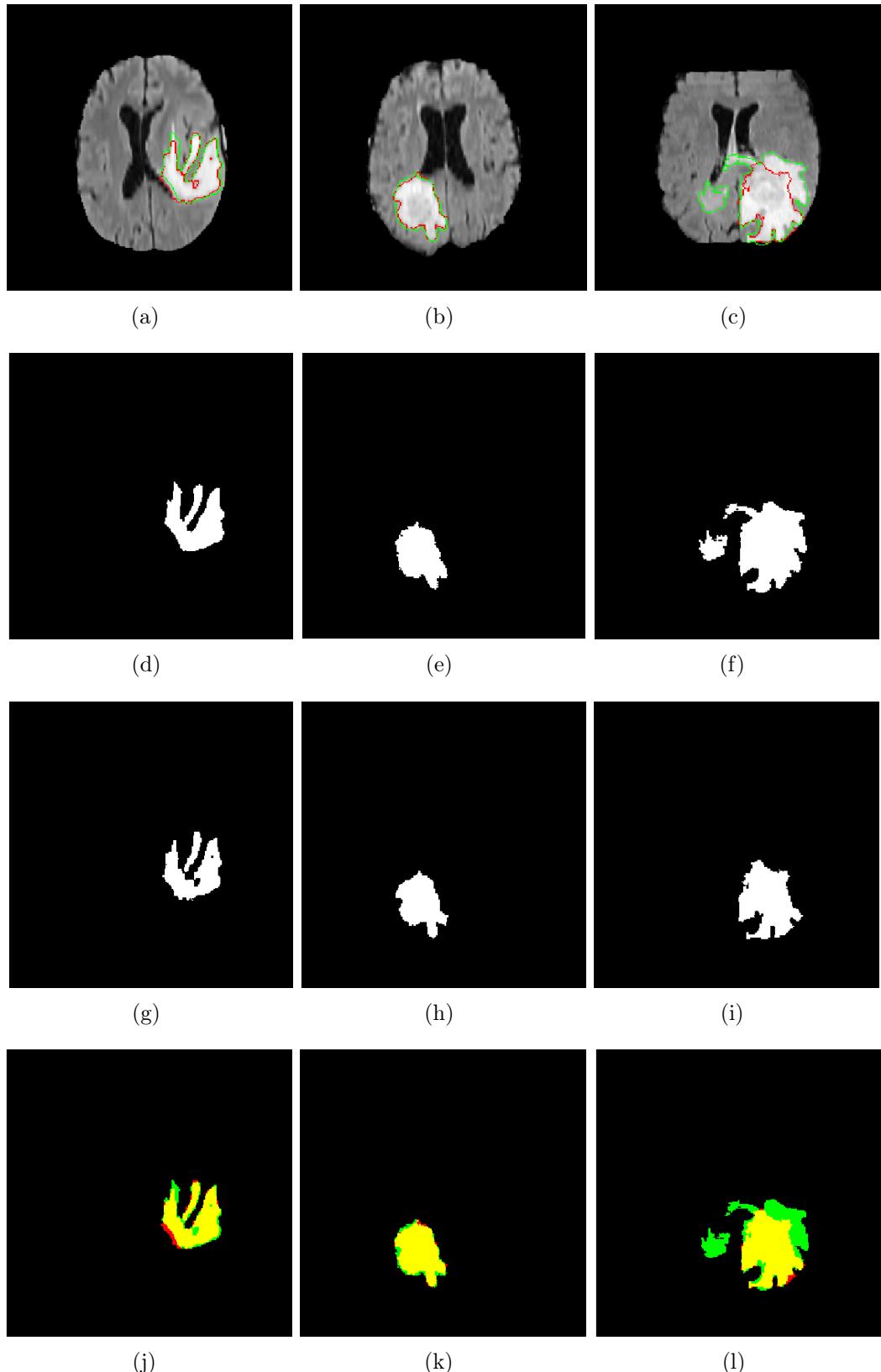


FIGURE 4.5 – Segmentation de l'œdème par l'approche 3 : (a)-(c) Images originales avec les contours trouvé et référence, (d)-(f) Segmentation de référence (g)-(i) Segmentation obtenue, (h)-(k) Overlap des segmentations, référence en vert, obtenue en rouge, coïncidence en jaune

Approche	Dice	VD [%]	SD _{abs} [mm]
1° approche	0.453	59.45	10.65
2° approche	0.304	66.96	15.68
3° approche	0.44	60.43	9.53
Isensee[30]	0.677	-	-
Perslev[31]	0.60	-	-
Yuan[32]	0.815	-	2.53

TABLE 4.1 – Valeurs moyennes des mesures évaluées sur 50 images de la base de données comparé à des résultats d’autres articles.

sont biaisés par les mauvais échantillons mais aussi que l’algorithme peut être encore plus performant que ceux qui ont servi à titre de comparaison.

La conclusion à laquelle nous arrivons c’est que l’algorithme proposé manque de robustesse par rapport à l’image donnée. Alors, pour résoudre ce problème de la robustesse nous pourrions penser à trois choses : (1) l’algorithme est évidemment sensible au choix des régions, soit pour le marquage lui-même, soit pour le biais potentiel dans les estimations des moyennes de niveau de gris. (2) le nombre de régions marqués est trop petit par rapport à la dynamique de toute l’image, certaines zones de la tumeur sont trop claires par rapport au reste et le risque de tomber sur une de ces coupes n’est pas négligeable. (3) les variances ne sont pas trop importantes, puisque dans la troisième proposition nous l’avons supprimé et les résultats ont resté aussi performants.

Cela nous amène à quelques propositions de changement pour garantir la robustesse :

1. Passer l’image par un pré-traitement : limitation du niveau de gris à la tranche 0-255, lissage gaussien pour essayer de réduire le nombre de régions.
2. Éliminer le calcul de la variance lors du marquage des images.
3. Marquer plus d’une coupe.
4. Faire en même temps le calcul par les 3 approches et choisir celui avec le plus grand volume (ou même rejeter toutes si elles ne dépassent pas un majorant inférieur de volume).

La première proposition se justifie par une possibilité d’homogénéiser les images, le pré-traitement, ignoré jusqu’ici peut être une façon de faciliter à comprimer le nombre de régions dans l’image, réduisant le temps de calcul et permettant de comparer les niveaux de gris dans plusieurs images.

La variance s’est montrée peu utile pour restreindre une fausse propagation ou un autre effet quelconque que l’image puisse, éventuellement, subir sans elle. Il pourra être plutôt être remplacé par un terme d’échelle que puisse mettre les poids de régularisation et de liaison aux étiquettes dans une proportion convenable.

Puisque tous les cas ont montré des résultats très conservateur par rapport à la segmentation de référence, nous n’avons, au moins à principe, rien à perdre en construisant trois graphes et en choisissant celui de plus grand volume. Malgré la forte corrélation observe entre les bons résultats des deux approches, il arrive parfois d’avoir une méthode beaucoup plus puissante que l’autre dans une image (c’est l’exemple du cerveau à gauche des images 4.3 à 4.5).

À propos du temps de calcul de chaque image, il y a une différence significative entre les approches 1, 2 et 3. Tandis que le calcul comptant seulement avec les poids entre les régions prends environ 1400 s (+-200) par image, le calcul avec la prise en compte du niveau de gris prends autour de 2400 s (+-310 s) par image, ce qui signifie une augmentation de plus de 60 % du temps de calcul. La troisième approche est aussi significative en termes de temps de calcul. Le temps nécessaire pour calculer une image est aussi en moyenne de 2400 s (+-310 s).

Chapitre 5

Conclusion

Pendant ce semestre une recherche plus approfondie sur la théorie de graphes et la morphologie mathématique m'a permis de mener à terme ce travail. Au-delà de l'objectif de trouver un bon algorithme de segmentation pour les oedèmes qui puisse à la fois être précis et rapide pour permettre aux professionnels d'avoir rapidement une idée de l'extension de l'oedème/tumeur, et planifier des interventions chirurgicales et/ou thérapeutiques, l'objectif de ce travail est d'apprendre à mener une recherche bibliographie et mettre en évidence l'apprentissage et les compétences acquises au cours de ce Master.

Pour faire cela, une étude bibliographique a été faite et les informations recueillies lors de cette recherche sont résumées dans les premiers chapitres de ce travail. Enfin, inspirés par un travail de doctorat dans cette même ligne, nous avons proposé une solution au problème mentionné.

Trois approches sont proposées initialement pour attaquer le problème posé et des résultats intéressants ont été obtenus. Malgré la performance basse de l'algorithme proposé vis-à-vis des références indiquées sur le même jeu de données, une analyse un peu plus approfondie montre que la performance est en fait plutôt bonne, au moins meilleure que deux des algorithmes de comparaison.

Cela est très intéressant puisque les références obtenues sont basées sur des approches par *deep learning*, qui sont coûteuses en temps d'entraînement, sont lourdes en mémoire et exigent une entrée de taille fixe, choses qui ne sont pas nécessaires dans l'approche proposée.

D'ailleurs, les quelques propositions d'améliorations indiquées à la fin du chapitre précédent devraient permettre d'augmenter significativement la qualité de la segmentation de sortie, et d'améliorer la performance de l'algorithme. De plus, une parallélisation du code est possible, étant donné que chaque région est indépendante et la construction du graphe peut être faite en parallèle. Cela aussi pourra permettre une amélioration significative du temps de calcul.

Bibliographie

- [1] Olivier Lézoray and Leo Grady. *Image processing and analysis with graphs : theory and practice*. CRC Press, 2012.
- [2] Luc Vincent. Graphs and mathematical morphology. *Signal Processing*, 16 :365–388, 1989.
- [3] Yuri Boykov and Olga Veksler. Graph cuts in vision and graphics : Theories and applications. In *Handbook of mathematical models in computer vision*, pages 79–96. Springer, 2006.
- [4] Vinh-Thong Ta, Olivier Lézoray, Abderrahim Elmoataz, and Sophie Schüpp. Graph-based tools for microscopic cellular image segmentation. *Pattern Recognition*, 42 :1113–1125, 2009.
- [5] Laurent Najman and Jean Cousty. A graph-based mathematical morphology reader. *Pattern Recognition Letters*, 47 :3–17, 2014.
- [6] Vinh-Thong Ta, Abderrahim Elmoataz, and Olivier Lézoray. Nonlocal pdes-based morphology on weighted graphs for image and data processing. *IEEE transactions on Image Processing*, 20 :1504–1516, 2010.
- [7] Zhao Yu-Qian, Gui Wei-Hua, Chen Zhen-Cheng, Tang Jing-Tian, and Li Ling-Yun. Medical images edge detection based on mathematical morphology. In *2005 IEEE engineering in medicine and biology 27th annual conference*, pages 6492–6495, 2005.
- [8] N. Biggs, E. K. Lloyd, and R. J. Wilson. *Graph Theory, 1736-1936*. Clarendon Press, New York, NY, USA, 1986.
- [9] Salim Jouili and Salvatore Tabbone. Applications des graphes en traitement d’images. In *International Conference on Relations, Orders and Graphs : Interaction with Computer Science - ROGICS’08*, 2008.
- [10] John F. O’Callaghan. An alternative definition for " neighborhood of a point ". *IEEE Transactions on Computers*, 100 :1121–1125, 1975.
- [11] Kenneth J Supowit. The relative neighborhood graph, with an application to minimum spanning trees. *Journal of the ACM (JACM)*, 30 :428–448, 1983.
- [12] Ronald L Graham and Pavol Hell. On the history of the minimum spanning tree problem. *Annals of the History of Computing*, 7 :43–57, 1985.
- [13] Dorothy M Greig, Bruce T Porteous, and Allan H Seheult. Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society : Series B (Methodological)*, 51 :271–279, 1989.
- [14] LR Ford and DR Fulkerson. Flows in networks, princeton university press, 1962. *Ford Flows in Networks 1962*, 1996.

- [15] Elias Dahlhaus, David S. Johnson, Christos H. Papadimitriou, Paul D. Seymour, and Mihalis Yannakakis. The complexity of multiterminal cuts. *SIAM Journal on Computing*, 23 :864–894, 1994.
- [16] Yuri Boykov, Olga Veksler, and Ramin Zabih. Markov random fields with efficient approximations. In *Proceedings. 1998 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No. 98CB36231)*, pages 648–655. IEEE, 1998.
- [17] Cigdem Gunduz, Bülent Yener, and S Humayun Gultekin. The cell graphs of cancer. *Bioinformatics*, 20 :i145–i151, 2004.
- [18] Guy Gilboa and Stanley Osher. Nonlocal operators with applications to image processing. *Multiscale Modeling & Simulation*, 7 :1005–1028, 2008.
- [19] Christian Ronse, Laurent Najman, and Etienne Decencière. *Mathematical morphology : 40 years on*. Springer, 2005.
- [20] Edward Dougherty. *Mathematical morphology in image processing*. CRC press, 2018.
- [21] J Angulo. Apports de la morphologie mathématique couleur au filtrage et à la segmentation, 2007.
- [22] Jesus Angulo. *Morphologie mathématique pour des images multi-variées*. PhD thesis, Université Paris-Est Marne la Vallée, 2012.
- [23] Eric Raymond, Martine Raphael, Michel Grimaud, Luc Vincent, Jacques Louis Binet, and Fernand Meyer. Germinal center analysis with the tools of mathematical morphology on graphs. *Cytometry : The Journal of the International Society for Analytical Cytology*, 14 :848–861, 1993.
- [24] H.J.A.M Heijmans, P Nacken, Alexander Toet, and Luc Vincent. Graph morphology. *Journal of Visual Communication and Image Representation*, 3 :24–38, 1992.
- [25] Jean Cousty, Laurent Najman, Fabio Dias, and Jean Serra. Morphological filtering on graphs. *Computer Vision and Image Understanding*, 117 :370–385, 2013.
- [26] John G Stell. Relations in mathematical morphology with applications to graphs and rough sets. In *International Conference on Spatial Information Theory*, pages 438–454. Springer, 2007.
- [27] Jean Stawiaski. *Mathematical morphology and graphs : Application to interactive medical image segmentation*. PhD thesis, Ecole Nationale Supérieure des Mines de Paris, 2008.
- [28] Amber L Simpson, Michela Antonelli, Spyridon Bakas, Michel Bilello, Keyvan Farahani, Bram van Ginneken, Annette Kopp-Schneider, Bennett A Landman, Geert Litjens, Bjoern Menze, et al. A large annotated medical image dataset for the development and evaluation of segmentation algorithms. *arXiv preprint arXiv :1902.09063*, 2019.
- [29] Mina Ghaffari, Arcot Sowmya, and Ruth Oliver. Automated brain tumour segmentation using multimodal brain scans, a survey based on models submitted to the brats 2012-18 challenges. *IEEE Reviews in Biomedical Engineering*, 2019.
- [30] Fabian Isensee, Jens Petersen, Andre Klein, David Zimmerer, Paul F Jaeger, Simon Kohl, Jakob Wasserthal, Gregor Koehler, Tobias Norajittra, Sebastian

- Wirkert, et al. nnu-net : Self-adapting framework for u-net-based medical image segmentation. *arXiv preprint arXiv :1809.10486*, 2018.
- [31] Mathias Perslev, Erik Bjørnager Dam, Akshay Pai, and Christian Igel. One network to segment them all : A general, lightweight system for accurate 3d medical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 30–38. Springer, 2019.
 - [32] Wenguang Yuan, Jia Wei, Jiabing Wang, Qianli Ma, and Tolga Tasdizen. Unified attentional generative adversarial network for brain tumor segmentation from multimodal unpaired images. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 229–237. Springer, 2019.

Annexes

```

1 # -*- coding: utf-8 -*-
2 """
3 Created on Sun Nov 10 15:11:07 2019
4 @author: Raul Alfredo de Sousa Silva
5
6 Description:
7     This code is designed to reproduce the work of the PhD thesis
8     of Jean Stawiaski (reference above) especially
9     concerning the segmentation of medical images with a minimal
10    surface over a region adjacency graph.
11    In a nutshell what is made here is:
12        - Charging images from database
13        - Computing morphological gradient over a 3D image (We
14        chose to work firstly with the FLAIR image)
15        - Computing the low level watershed of the image
16        - Creating the region adjacency graph (computing weights,
17        and affecting them to respective nodes)
18        - Computing the Grathcut
19        - Plotting results
20        - Saving segmentation
21
22 What we must do in next steps:
23     - Provide ability to insert markers (scribbles, as in
24     Stawiaski's work)
25     - To be able to separate into all 4 classes the tumor
26
27 Reference:
28     Stawiaski, J. Mathematical morphology and graphs: Application
29     to interactive medical image segmentation
30     (Doctoral dissertation). Ecole Nationale Superieure des Mines
31     de Paris, 2008
32 """
33
34 # Charging libraries
35
36
37 from nilearn import image as img                         # To charge images
38 import numpy as np                                       # For general manipulation
39 import skimage.morphology as skm                         # Many morphological
40     operations including watershed

```

```

40 from skimage.feature import peak_local_max          # Some
   complementary functions to labelise regions
41 from scipy.ndimage import label      # Complementary function to
   labelaize regions
42 import maxflow                      # Computing minimal surface
   with a graphcut approach
43 from nibabel import Nifti1Image        # Create output file
   with the result
44 import time                         # Measure processing time
45 import os
46
47 def segmentation_brain(datafname,mu_0,mu_1,var=1,resultfname="/
   results_tr/result_XXX.ni.gz"):
48
49     tic = time.clock()
50     # Part 1 - Preprocessing (charging file computing watershed)
51     # Extracting actual data
52     data = img.load_img(datafname)
53     image = data.get_data()
54
55     # Computing morphological gradient over a 3D image
56     flair = image[:,:,:,:0]
57     strel = skm.ball(1)
58     grad= skm.dilation(flair,strel)-skm.erosion(flair,strel)
59     # Computing the low level watershed of the image
60     local_mini = peak_local_max(grad.max()-grad, indices=False)
61     markers = label(local_mini)[0]
62     labels = skm.watershed(grad, markers, watershed_line=False) #
   To pick regions
63
64     M = labels.max()
65     print("")
66     print("Regions founded: ",M)
67
68     # Part 2 - Creating the region adjacency graph (computing
   weights, and affecting them to respective nodes)
69     print("Creating the graph...")
70     nodesmax = M                                # One node for each
   region and for each possible label
71     edgesmax = nodesmax*10                        # Every node has
   the mean number of neighbors
72     G = maxflow.Graph[float](nodesmax,edgesmax)
73     for i in range(M):
74         if (round((i/M)+.05,1) < round((i+1)/M+.05,1)):
75             print("Progress: ",100*round((i/M),2),"%")
76             ind = np.nonzero((labels == i+1))
77             mu = np.mean(image[ind])
78             tw = (mu-mu_0)**2/(2*var)
79             sw = (mu-mu_1)**2/(2*var)
80             G.add_nodes(1)
81             G.add_tedge(i,sw,tw)
82
83         if (i==0):           # There's nothing else to do (
   points in 1 are useless and computationally costful)
84             continue
85
86         indx,indy,indz = np.nonzero((labels == i+1))
87         ind = []

```

```

88     weight = []
89     for n in range(len(indx)):
90         point = (indx[n],indy[n],indz[n])
91         neighs = [(indx[n]+1,indy[n],indz[n])]
92         neighs.append((indx[n]-1,indy[n],indz[n]))
93         neighs.append((indx[n],indy[n]+1,indz[n]))
94         neighs.append((indx[n],indy[n]-1,indz[n]))
95         neighs.append((indx[n],indy[n],indz[n]+1))
96         neighs.append((indx[n],indy[n],indz[n]-1))
97         for neigh in neighs:
98             j = labels[neigh]-1
99             if (j < i):
100                 if (j,i) not in ind:
101                     ind.append((j,i))
102                     weight.append( (1+max(grad[point],grad[
neigh]))**(-2) )
103             else:
104                 index = ind.index((j,i))
105                 weight[index] += (1+max(grad[point],grad[
neigh]))**(-2)
106             for w in range(len(ind)):
107                 p,q = ind[w]
108                 G.add_edge(p,q,weight[w],weight[w])
109             print("Progress: 100 %")
110
111 # Part 3 - Computing the Graph-cut
112 print("Finding best cut...")
113 cost = G.maxflow()
114 print("Cost of this cut: ",cost)
115 print("Recomposing image...")
116
117 seg = np.zeros(image.shape[0:3])
118 for i in range(M):
119     if (round((i/M)+.05,1) < round((i+1)/M+.05,1)):
120         print("Progress: ",100*round((i/M),2),"%")
121     indices = np.nonzero(labels == i+1)
122     seg[indices] = G.get_segment(i)
123     print("Progress: 100 %")
124
125 toc = time.clock()
126 tm = toc - tic
127 print("Processing time to one image: ",tm)
128 print("Image segmented! Manipulate the variable 'seg' to see
the results.")
129
130 # Saving segmentation obtained
131 t = resultfname.rfind("/")
132 try:
133     os.makedirs(resultfname[0:t+1])
134 except:
135     print("Folder already exists")
136 else:
137     print("Folder created")
138
139 result = Nifti1Image(seg, affine=np.eye(4))
140 result.to_filename(resultfname)
141
142 return resultfname,tm

```

```

143
144 def segmentation_brain_inf(datafname ,c0 ,c1 ,p ,resultfname="/
145   results_tr/result_XXX.ni.gz"):
146
147   tic = time.clock()
148   # Part 1 - Preprocessing (charging file computing watershed)
149   data = img.load_img(datafname)
150   image = data.get_data()
151
152   # Computing morphological gradient over a 3D image
153   flair = image[:,:,:,:,0]
154   strel = skm.ball(1)
155   grad= skm.dilation(flair,strel)-skm.erosion(flair,strel)
156   # Computing the low level watershed of the image
157   local_mini = peak_local_max(grad.max()-grad, indices=False)
158   markers = label(local_mini)[0]
159   labels = skm.watershed(grad, markers, watershed_line=False) #
160   To pick regions
161
162   M = labels.max()
163   print("")
164   print("Regions founded: ",M)
165
166   # Part 2 - Creating the region adjacency graph (computing
167   # weights, and affecting them to respective nodes)
168   print("Creating the graph...")
169   nodesmax = M                                     # One node for each
170   region and for each possible label
171   edgesmax = nodesmax*10                           # Every node has
172   the mean number of neighbors
173   G = maxflow.Graph[float](nodesmax,edgesmax)
174
175   lab0 = np.unique(labels[c0[1]:c0[3],c0[0]:c0[2],p])
176   lab1 = np.unique(labels[c1[1]:c1[3],c1[0]:c1[2],p])
177
178   for i in range(M):
179     if (round((i/M)+.05,1) < round((i+1)/M+.05,1)):
180       print("Progress: ",100*round((i/M),2),"%")
181
182     G.add_nodes(1)
183     if (i==0):                                     # There's nothing else to do (
184       points in 1 are useless and computationally costful)
185       continue
186
187     if(i+1 in lab1):
188       G.add_tedge(i,0,1000)
189     if(i+1 in lab0):
190       G.add_tedge(i,1000,0)
191
192     idx,indy,indz = np.nonzero((labels == i+1))
193     ind = []
194     weight = []
195     for n in range(len(idx)):
196       point = (idx[n],indy[n],indz[n])
197       neigs = [(idx[n]+1,indy[n],indz[n])]
198       neigs.append((idx[n]-1,indy[n],indz[n]))
199       neigs.append((idx[n],indy[n]+1,indz[n]))
200       neigs.append((idx[n],indy[n]-1,indz[n]))

```

```

195     neighs.append((indx[n],indy[n],indz[n]+1))
196     neighs.append((indx[n],indy[n],indz[n]-1))
197     for neigh in neighs:
198         j = labels[neigh]-1
199         if (j < i):
200             if (j,i) not in ind:
201                 ind.append((j,i))
202                 weight.append( (1+max(grad[point],grad[
neigh]))**(-2) )
203             else:
204                 index = ind.index((j,i))
205                 weight[index] += (1+max(grad[point],grad[
neigh]))**(-2)
206             for w in range(len(ind)):
207                 p,q = ind[w]
208                 G.add_edge(p,q,weight[w],weight[w])
209             print("Progress: 100 %")
210
211 # Part 3 - Computing the Graph-cut
212 print("Finding best cut...")
213 cost = G.maxflow()
214 print("Cost of this cut: ",cost)
215
216 print("Recomposing image...")
217
218 seg = np.zeros(image.shape[0:3])
219 for i in range(M):
220     if (round((i/M)+.05,1) < round((i+1)/M+.05,1)):
221         print("Progress: ",100*round((i/M),2),"%")
222     indices = np.nonzero(labels == i+1)
223     seg[indices] = G.get_segment(i)
224     print("Progress: 100 %")
225
226 toc = time.clock()
227 tm = toc-tic
228 print("Processing time to one image: ",tm)
229 print("Image segmented! Manipulate the variable 'seg' to see
the results.")
230
231 # Saving segmentation obtained
232 t = resultfname.rfind("/")
233 try:
234     os.makedirs(resultfname[0:t+1])
235 except:
236     print("Folder already exists")
237 else:
238     print("Folder created")
239
240 result = Nifti1Image(seg, affine=np.eye(4)) # It's compulsory
241 the passage of a affine matrix as an argument
242 result.to_filename(resultfname)
243
244 return resultfname,tm
245 def segmentation_brain_comb(datafname,c0,c1,p,mu_0,mu_1,var=1e9,
resultfname="/results_tr/result_XXX.nii.gz"):
246     tic = time.clock()

```

```

248
249     # Part 1 - Preprocessing (charging file computing watershed)
250     data = img.load_img(datafname)
251     image = data.get_data()
252
253     # Computing morphological gradient over a 3D image
254     flair = image[:, :, :, 0]
255     strel = skm.ball(1)
256     grad= skm.dilation(flair,strel)-skm.erosion(flair,strel)
257     # Computing the low level watershed of the image
258     local_mini = peak_local_max(grad.max()-grad, indices=False)
259     markers = label(local_mini)[0]
260     labels = skm.watershed(grad, markers, watershed_line=False) #
261     To pick regions
262
263     M = labels.max()
264     print("")
265     print("Regions founded: ",M)
266
267     # Part 2 - Creating the region adjacency graph (computing
268     # weights, and affecting them to respective nodes)
269     print("Creating the graph...")
270     nodesmax = M                                     # One node for each
271     region and for each possible label
272     edgesmax = nodesmax*10                           # Every node has
273     the mean number of neighbors
274     G = maxflow.Graph[float](nodesmax,edgesmax)
275     G.add_nodes(M)
276
277     lab0 = np.unique(labels[c0[1]:c0[3],c0[0]:c0[2],p])
278     lab1 = np.unique(labels[c1[1]:c1[3],c1[0]:c1[2],p])
279     if 1 in lab1:
280         lab1 = lab1[1:-1]
281
282     for i in range(M):
283         if (round((i/M)+.05,1) < round((i+1)/M+.05,1)):
284             print("Progress: ",100*round((i/M),2),"%")
285
286         if (i==0):                                     # There's nothing else to do (
287             points in 1 are useless and computationally costful)
288             continue
289
290         if(i+1 in lab1):
291             G.add_tedge(i,0,1000)
292         elif(i+1 in lab0):
293             G.add_tedge(i,1000,0)
294         else:
295             ind = np.nonzero((labels == i+1))
296             mu = np.mean(image[ind])
297             tw = (mu-mu_0)**2/(2*var)
298             sw = (mu-mu_1)**2/(2*var)
299             G.add_tedge(i,sw,tw)
300
301         indx,indy,indz = np.nonzero((labels == i+1))
302         ind = []
303         weight = []
304         for n in range(len(indx)):
305             point = (indx[n],indy[n],indz[n])

```

```

301     neighs = [(indx[n]+1,indy[n],indz[n])]
302     neighs.append((indx[n]-1,indy[n],indz[n]))
303     neighs.append((indx[n],indy[n]+1,indz[n]))
304     neighs.append((indx[n],indy[n]-1,indz[n]))
305     neighs.append((indx[n],indy[n],indz[n]+1))
306     neighs.append((indx[n],indy[n],indz[n]-1))
307     for neigh in neighs:
308         j = labels[neigh]-1
309         if (j < i):
310             if (j,i) not in ind:
311                 ind.append((j,i))
312                 weight.append( (1+max(grad[point],grad[
313         neigh]))**(-2) )
314             else:
315                 index = ind.index((j,i))
316                 weight[index] += (1+max(grad[point],grad[
317         neigh]))**(-2)
318                 for w in range(len(ind)):
319                     p,q = ind[w]
320                     G.add_edge(p,q,weight[w],weight[w])
321     print("Progress: 100 %")
322
323     # Part 3 - Computing the Graph-cut
324     print("Finding best cut...")
325     cost = G.maxflow()
326     print("Cost of this cut: ",cost)
327     print("Recomposing image...")
328
329     seg = np.zeros(image.shape[0:3])
330     for i in range(M):
331         if (round((i/M)+.05,1) < round((i+1)/M+.05,1)):
332             print("Progress: ",100*round((i/M),2),"%")
333             indices = np.nonzero(labels == i+1)
334             seg[indices] = G.get_segment(i)
335     print("Progress: 100 %")
336
337     toc = time.clock()
338     tm = toc - tic
339     print("Processing time to one image: ",tm)
340     print("Image segmented! Manipulate the variable 'seg' to see
the results.")
341
342     # Saving segmentation obtained
343     t = resultfname.rfind("/")
344     try:
345         os.makedirs(resultfname[0:t+1])
346     except:
347         print("Folder already exists")
348     else:
349         print("Folder created")
350
351     result = Nifti1Image(seg, affine=np.eye(4))
352     result.to_filename(resultfname)
353
354     return resultfname,tm

```

Algorithme 5.1 – Bibliothèque avec les fonctions proposées par les trois approches

```

1 # -*- coding: utf-8 -*-
2 """
3 Created on Mon Dec  2 10:15:03 2019
4
5 @author: Raul Alfredo de Sousa Silva
6
7 Observation: Data used in this project is a part of the BRATS
8 dataset used in
9 the Decathlon Medical Segmentation and can be found at:
10     https://drive.google.com/drive/folders/1
11     HqEgzS8BV2c7xYNrZdEAnrHk7osJJ--2
12     Please read the instructions to use this code.
13 """
14 """
15 INSTRUCTIONS:
16     You can easily generate the filenames which you will
17     manipulate by running
18         this part of the code with the path to the images on your
19         computer completed
20         in the variable datapath below.
21 """
22
23 # Data file names
24
25 datapath = "C:/Users/raul-/Documents/PRAT_projet/Task01_BrainTumour
26     /imagesTr/"
27 f=open('datafnames.txt','w')
28 for number in range(1,51):
29     datafname = datapath + "BRATS_" + format(number,"03d") + ".nii.
30     gz"
31     f.write(datafname)
32     f.write('\n')
33 f.close()
34
35 # Labels file names
36
37 datapath = "C:/Users/raul-/Documents/PRAT_projet/Task01_BrainTumour
38     /labelsTr/"
39 f=open('labelfnames.txt','w')
40 for number in range(1,51):
41     labelfname = datapath + "BRATS_" + format(number,"03d") + ".nii.
42     gz"
43     f.write(labelfname)
44     f.write('\n')
45 f.close()
46
47 # Results file names
48
49 datapath = "C:/Users/raul-/Documents/PRAT_projet/Task01_BrainTumour
50     /resultsTr/"
51 f=open('resultfnames.txt','w')
52 for number in range(1,51):
53     resultfname = datapath + "BRATS_" + format(number,"03d") + ".nii
54     .gz"
55     f.write(resultfname)

```

```

49         f.write('\n')
50 f.close()
51
52 #%%
53
54 # Computing paramaters
55 ,,
56 INSTRUCTIONS:
57     Code: computing automatically parameters as mean, variance and
58     enclosing
59     box over the two regions with a simple indication from user of
60     what are the
61     regions composing the healthy brain and the edema.
62     Results are saved in the parameters.csv file
63     The second part of this script read this '.csv' file and apply
64     the segmentation.
65
66     Images will be presented one at a time to you. What you are
67     expected to do
68     is to define a bounding box around a part of the image
69     representative of
70     the brain (class 0) and the edema/tumor (class 1) by clicking
71     with the left
72     button in the image in the top left corner and bottom right
73     corner of the
74     bounding box.
75
76     At the end you must click with the right button of the mouse to
77     validate
78     your choices. You can simply continue with left clicks if you
79     want to
80     redefine your regions (each 4 clicks are considered as the top
81     and bottom
82     corners of each class), only the last 4 will be considered.
83
84     If the image presented to you doesn't seem to have any edema/
85     tumor, or
86     doesn't seem to be a good example, you can simply click with
87     center button
88     of your mouse and type in the console the number of the slice
89     for which you
90     would like to go (default is 90), by this you can look for the
91     slice of the
92     image that seems to fit the best to estimate the parameters to
93     apply the
94     algorithm.
95 ,,
96
97 import nilearn as nl                      # To charge images
98 import matplotlib.pyplot as plt            # To plot graphics
99 import numpy as np                         # For general manipulation
100 import pandas as pd                        # Create and manipulate files
101    with important data
102
103 mu_0 = []
104 mu_1 = []
105 var0 = []
106 vari1 = []

```

```

91 c00 = []
92 c01 = []
93 c02 = []
94 c03 = []
95 c10 = []
96 c11 = []
97 c12 = []
98 c13 = []
99 p = []
100 def onclick(event):
101     x = event.xdata
102     y = event.ydata
103     global but
104     but = event.button
105     global coords
106     coords = [x,y]
107     return coords, but
108
109 def choice_region(image,pos):
110     c = np.zeros(8).astype(np.uint8)
111     fig, ax = plt.subplots()
112     ax.imshow(image[:, :, pos],cmap ='gray')
113     ax.axis("off")
114     cid = fig.canvas.mpl_connect('button_press_event', onclick)
115
116     i=0
117     while(but != 3 and but != 2):
118         if(i%4==0):
119             print("Chose the top of the box representing region 0")
120             test=plt.waitforbuttonpress()
121             if (but==1):
122                 c[0] = int(coords[0])
123                 c[1] = int(coords[1])
124             elif(i%4==1):
125                 print("Chose the bottom of the box representing region
126 0")
127                 test=plt.waitforbuttonpress()
128                 if (but==1):
129                     c[2] = int(coords[0])
130                     c[3] = int(coords[1])
131             elif(i%4==2):
132                 print("Chose the top of the box representing region 1")
133                 test=plt.waitforbuttonpress()
134                 if (but==1):
135                     c[4] = int(coords[0])
136                     c[5] = int(coords[1])
137             else:
138                 print("Chose the bottom of the box representing region
139 1")
140                 test=plt.waitforbuttonpress()
141                 if (but==1):
142                     c[6] = int(coords[0])
143                     c[7] = int(coords[1])
144             i+=1
145
146             print("Done!!!")
147             fig.canvas.mpl_disconnect(cid)
148             plt.close('all')

```

```

147
148
149     return c
150
151 # Choose one of the two approaches above to manipulate files
152
153 # 1 - Doing by list of filenames
154
155 f1=open('datafnames.txt','r')
156 number = 0
157 #fn = f1.readlines()[:10]
158 for datafname in f1:
159     number += 1
160     datafname = datafname[:-1]
161 # 2 - Doing by numbering
162 #for number in range(1,51):
163 #    datafname = "C:/Users/raul-/Documents/PRAT_projet/
Task01_BrainTumour/results_tr_comb/result_"+ format(number,"03d")
"") + ".nii.gz"
164
165     print("Treating image ",number)
166     c = np.zeros(8).astype(np.uint8)
167
168     data = nl.image.load_img(datafname)
169     image = data.get_data()
170
171     # Preprocessing: Taking a cut of the image
172     # Function to take positions
173     mu_0n = 0
174     var_0n = 0
175     mu_1n = 0
176     var_1n = 0
177     but = 0
178     image = image[:, :, :, 0]
179     pos = 90
180     while (but !=3):
181         coords = []
182         but = 0
183         c = choice_region(image, pos)
184         if (but == 2):
185             print("It seems that layer {} didn't fit fo you".
format(pos))
186             pos = int(input("For which layer would you like to go ?
[0-155]"))
187             continue
188
189         # Chosing regions to take parameters
190         u = np.sort(np.reshape(image[c[1]:c[3], c[0]:c[2], pos], -1))
191         mu_0n = np.mean(u[int(3*len(u)/4):-1])
192         var_0n = np.var(u[int(3*len(u)/4):-1])
193         u = np.sort(np.reshape(image[c[5]:c[7], c[4]:c[6], pos], -1))
194         mu_1n = np.mean(u[0:int(len(u)/4)])
195         var_1n = np.var(u[0:int(len(u)/4)])
196
197         del data, image
198         mu_0.append(mu_0n)
199         mu_1.append(mu_1n)
200         var0.append(var_0n)

```

```

201     var1.append(var_1n)
202     c00.append(c[0])
203     c01.append(c[1])
204     c02.append(c[2])
205     c03.append(c[3])
206     c10.append(c[4])
207     c11.append(c[5])
208     c12.append(c[6])
209     c13.append(c[7])
210     p.append(pos)
211
212 parameters = pd.DataFrame({ 'mu_0':mu_0 , 'mu_1':mu_1 , 'sigma0':var0 ,
213                             'sigma1':var1 , 'c00':c00 , 'c01':c01 , 'c02':c02 , 'c03':c03 , 'c10'
214                             :c10 , 'c11':c11 , 'c12':c12 , 'c13':c13 , 'pos':p})
213 parameters.to_csv('parameters.csv')
214 f1.close()
215
216 #%% Computing minimal surfaces
217 ''
218 INSTRUCTIONS:
219     This part of the script takes the '.csv' file generated from
220     the first part
221     and generates the segmentation of the edema, which is saved in
222     the file
223     indicated by the variable resultfname.
224     The code is made in a manner that you can't load a '.txt' file
225     with the address
226     of the file (image) you want to manipulate. This can easily
227     automatize the
228     task.
229     Choose between one of the three approaches presented in the
230     work that are
231     implemented in the ms file:
232     Approach 1: ms.segmentation_brain(datafname,mu_0,mu_1,sigma,
233                                         resultfname)
234     Approach 2: ms.segmentation_brain_inf(datafname,c0,c1,p,
235                                         resultfname)
236     Approach 3: ms.segmentation_brain_comb(datafname, c0, c1, p,
237                                         mu_0, mu_1,
238                                         sigma = 1e9, resultfname
239                                         )
240
241 ''
242 # Extracting actual data
243 import ms                      # Library with the main algorithm
244 import pandas as pd              # Create and manipulate files with
245 import important data
246 import numpy as np               # For general manipulation
247
248 times = []
249 parameters = pd.read_csv('parameters.csv',index_col = 0)
250
251 # Choose one of the two approaches above to manipulate files
252
253 # 1 - Doing by list of filenames
254
255 f1=open('datafnames.txt','r')
256 f2=open('resultfnames.txt','r')
257 number = 0

```

```

247 #fn = f1.readlines()[:10]
248 for datafname, resultfname in zip(f1,f2):
249     number += 1
250     datafname = datafname[:-1]
251     resultfname = resultfname[:-1]
252
253 # 2 - Doing by numbering
254 #for number in range(1,51):
255 #    number = 1
256 #    datafname = "C:/Users/raul-/Documents/PRAT_projet/
257 #        Task01_BrainTumour/imagesTr/BRATS_"+ format(number,"03d") + "."
258 #        nii.gz"
257 #    resultfname = "C:/Users/raul-/Documents/PRAT_projet/
258 #        Task01_BrainTumour/resultsTr/result_"+ format(number,"03d") + "."
259 #        nii.gz"
260
261     print("Treating image {}".format(number))
262
263     mu_0 = parameters.mu_0[number-1]
264     mu_1 = parameters.mu_1[number-1]
265     sigma = min(parameters.sigma0[number-1], parameters.sigma1[
266     number-1])
266     c0 = parameters.loc[number-1,'c00':'c03'].astype(np.uint8)
267     c1 = parameters.loc[number-1,'c10':'c13'].astype(np.uint8)
268     p = parameters.loc[number-1,'pos'].astype(np.uint8)
269     resultfname,tm = ms.segmentation_brain(datafname,mu_0,mu_1,
270     sigma,resultfname)
271     resultfname,tm = ms.segmentation_brain_inf(datafname,c0,c1,p,
272     resultfname)
273     resultfname,tm = ms.segmentation_brain_comb(datafname,c0,c1,p,
274     mu_0,mu_1,sigma = 1e9, resultfname)
275     times.append(tm)
276
277 texecution = pd.DataFrame({'time': times})
278 texecution.to_csv('times.csv')
279 f1.close()
280 f2.close()
281 times = pd.read_csv('times.csv',index_col = 0)
282 #%%
283 ,,
284 INSTRUCTIONS:
285 This part of the code compute the metrics defined in the work to
286     measure the
287 quality of the segmentation.
288 You can't simply pass the corresponding file names of both
289     reference and segmentation
290 and metrics will be computed.
291 It's made in such a way that the metrics are computed over all
292     tumor.
293 If you would like to evaluate just specific labels such as edema or
294     tumor, please
295 change lines just between the hashes.
296 ,,
297
298 import nilearn as nl                      # To charge images
299 import numpy as np                         # For general manipulation
300 import metrics as m                        # Functions to compute metrics

```

```

293 import pandas as pd # Create and manipulate files
294     with important data
295 dice = []
296 vd = []
297 sd = []
298 p = []
299 s = []
300
301 # Choose one of the two approaches above to manipulate files
302
303 # 1 - Doing by list of filenames
304
305 f1=open('labelfnames.txt','r')
306 f2=open('resultfnames.txt','r')
307 number = 0
308
309 for labelfname, resultfname in zip(f1,f2):
310     # Eliminate \n
311     number += 1
312     labelfname=labelfname[:-1]
313     resultfname=resultfname[:-1]
314
315
316 # 2 - Doing by numbering
317
318 #for number in range(1,51):
319 #    labelfname = "C:/Users/raul-/Documents/PRAT_projet/
320 #        Task01_BrainTumour/labelsTr/BRATS_"+ format(number , "03d") + ".nii.gz"
321 #    resultfname = "C:/Users/raul-/Documents/PRAT_projet/
322 #        Task01_BrainTumour/resultsTr/result_"+ format(number , "03d") + ".nii.gz"
323
324     # Loading files
325     label = nl.image.load_img(labelfname)
326     result = nl.image.load_img(resultfname)
327     seg = result.get_data()
328     ref = label.get_data()
329     del label, result
330
331     #####
332     #
333     ref = (ref>0).astype(np.uint8)
334     seg = (seg>0).astype(np.uint8)
335     #
336     #####
337     met = m.compute_metrics(seg,ref)
338     print("Metrics for image {0}".format(number))
339     print("DICE: {0}, VD: {1}, SD_avg: {2}".format(met[0],met[1],
340           met[2]))
341     dice.append(met[0])
342     vd.append(met[1])
343     sd.append(met[2])
344     p.append(met[3])
345     s.append(met[4])

```

```

341
342 metrics_comb_8 = pd.DataFrame({'dice':dice, 'voldif':vd, 'sumdif':
343                                     sd,
344                                     'precision':p, 'sensitivity':s})
345 metrics_comb_8.to_csv('metrics_comb_8.csv')
346 f1.close()
347 f2.close()
348 #%%
349 ,,
350 INSTRUCTIONS:
351 This final part of this script is just an easy way to generate
352     images just like
353 those presented on the report:
354     The original image
355     The proposed segmentation
356     The actual segmentation
357     The superposition of the two images (green to ref, red to
358     proposed)
359     The contour of the two segmentation over the original image.
360     (green to ref, red to proposed)
361 ,,
362 import nilearn as nl                      # To charge images
363 import matplotlib.pyplot as plt            # To plot graphics
364 import numpy as np
365 import metrics as m
366
367 plt.close("all")
368
369 # Choose one of the two approaches above to manipulate files
370
371 # 1 - Doing by list of filenames
372
373 f1=open('datafnames.txt','r')
374 f2=open('labelfnames.txt','r')
375 f3=open('resultfnames_comb.txt','r')
376 number = 0
377
378 for datafname, labelfname, resultfname in zip(f1,f2,f3):
379
380     number +=1
381     # Eliminate \n
382     datafname=datafname[:-1]
383     labelfname=labelfname[:-1]
384     resultfname=resultfname[:-1]
385
386 # 2 - Doing by numbering
387
388 #for number in range(1,51):
389 #    datafname = "C:/Users/raul-/Documents/PRAT_projet/
390 #        Task01_BrainTumour/imagesTr/BRATS_"+ format(number,"03d") + "."
391 #        nii.gz"
392 #    labelfname = "C:/Users/raul-/Documents/PRAT_projet/
393 #        Task01_BrainTumour/labelsTr/BRATS_"+ format(number,"03d") + "."
394 #        nii.gz"
395 #    resultfname = "C:/Users/raul-/Documents/PRAT_projet/

```

```

Task01_BrainTumour/results_tr_comb/result_ "+ format(number , "03d
") + ".nii.gz"

392
393     # Loading files
394     data = nl.image.load_img(datafname)
395     label = nl.image.load_img(labelfname)
396     result = nl.image.load_img(resultfname)
397     seg = result.get_data()
398     image = data.get_data()
399     ref = label.get_data()
400     del data, label, result

401
402     # Choosing cut
403     pos = 90

404
405     # Postprocessing: Taking a cut of the image
406     plt.figure(1)
407     plt.imshow(image[:, :, pos, 0], cmap = 'gray')
408     plt.axis("off")
409     plt.savefig('images/image_{0}cut_{1}.png'.format(number, pos))

410
411     # Postprocessing: Obtained segmentation
412     plt.figure(2)
413     plt.imshow(seg[:, :, pos], cmap = 'gray')
414     plt.axis("off")
415     plt.savefig('images/segmentation_{0}cut_{1}.png'.format(number,
pos))

416
417     # Postprocessing: Reference segmentation
418     plt.figure(3)
419     plt.imshow((ref[:, :, pos]>0), cmap = 'gray')
420     plt.axis("off")
421     plt.savefig('images/reference_{0}cut_{1}.png'.format(number, pos))

422
423     # Postprocessing: comparing segmentations
424     conf = np.zeros([image.shape[0], image.shape[1], 3])
425     conf[:, :, 0] = 255*(seg[:, :, pos])
426     conf[:, :, 1] = ref[:, :, pos]

427
428     plt.figure(4)
429     plt.imshow(conf)
430     plt.axis("off")
431     plt.savefig('images/comparison_{0}cut_{1}.png'.format(number,
pos))

432
433     m.draw_contour(seg[:, :, pos], ref[:, :, pos], image[:, :, pos, 0],
number, pos)

434 f1.close()
435 f2.close()
436 f3.close()
437
438 #%%

```

Algorithme 5.2 – Code principal pour faire toutes les tâches

```

1 # -*- coding: utf-8 -*-
2 """
3 Created on Sun Dec 15 23:25:10 2019
4
5 @author: Raul Alfredo de Sousa Silva
6
7 Library to compute the metrics defined on the work:
8     Dice
9     Relative absolute volumetric difference
10    Average symmetric absolute surface distance
11    Precision
12    Sensitivity
13 """
14 import numpy as np
15
16 # Function draw_contour
17 def draw_contour(seg, segmentation, image, number=0, pos=0, n_im=5):
18     """
19     Draw the contour over the given image.
20     seg: contour image (2D) obtained by your method
21     segmentation: reference contour image (2D)
22     image: image over which the contour will be drawn
23     There's no output, image with contour is automatically plotted.
24     """
25     from skimage.morphology import dilation, disk
26     from matplotlib.pyplot import figure, imshow, axis, savefig
27
28     strel = disk(1)
29     cont = dilation(seg, strel)-seg
30     u = (segmentation>0).astype(np.int8)
31     cont_r = dilation(u, strel)-u
32
33     imres = np.zeros((cont.shape[0], cont.shape[1], 4))
34     imref = np.zeros((cont.shape[0], cont.shape[1], 4))
35     imres[:, :, 0] = cont[:, :]
36     imref[:, :, 1] = cont_r[:, :]
37     imres[:, :, 3] = cont[:, :]
38     imref[:, :, 3] = cont_r[:, :]
39
40     figure(n_im)
41     imshow(image, cmap = 'gray')
42     imshow(imres)
43     imshow(imref)
44     axis("off")
45     savefig('images/contour_{0}cut_{1}.png'.format(number, pos))
46
47 def dice(seg, ref):
48     return round(2*np.sum((seg>0) & (ref>0))/(np.sum(seg>0) + np.sum(ref>0)), 3)
49
50 def volume_difference(seg, ref):
51     return 100*round(abs(1-np.sum(seg)/np.sum(ref)), 3)
52
53 def precision(seg, ref):
54     return round(np.sum((seg>0) & (ref>0))/np.sum((seg>0)), 3)
55
56 def sensitivity(seg, ref):
57     return round(np.sum((seg>0) & (ref>0))/np.sum((ref>0)), 3)

```

```

58
59 def surface_distance_abs(seg,ref):
60     import skimage.morphology as skm
61     if (np.sum(seg)==0):
62         return 0
63     se = skm.cube(3)
64     seg_surface = seg - skm.erosion(seg,se)
65     ref_surface = ref - skm.erosion(ref,se)
66     dist = []
67     ix,iy,iz = np.where(ref_surface)
68     p_ref = np.array([ix,iy,iz])
69     indx,indy,indz = np.where(seg_surface)
70     for i in range(len(indx)):
71         p = np.array([[indx[i]],[indy[i]],[indz[i]]])
72         dists = np.sqrt(np.sum((np.repeat(p,p_ref.shape[1],1)-p_ref
73 )**2,0))
74         dist.append(np.min(dists))
75     ix,iy,iz = np.where(seg_surface)
76     p_seg = np.array([ix,iy,iz])
77     indx,indy,indz = np.where(ref_surface)
78     for i in range(len(indx)):
79         p = np.array([[indx[i]],[indy[i]],[indz[i]]])
80         dists = np.sqrt(np.sum((np.repeat(p,p_seg.shape[1],1)-p_seg
81 )**2,0))
82         dist.append(np.min(dists))
83
84 def compute_metrics(seg,ref):
85     D = dice(seg,ref)
86     VD = volume_difference(seg,ref)
87     SD = surface_distance_abs(seg,ref)
88     P = precision(seg,ref)
89     S = sensitivity(seg,ref)
90     return [D,VD,SD,P,S]

```

Algorithme 5.3 – Bibliothèque avec les fonctions proposées pour mesurer la performance de segmentation