SEGMENTATION MULTI-AGENTS EN IMAGERIE BIOLOGIQUE ET MÉDICALE : APPLICATION AUX IRM 3D

Richard Moussa

N° d'ordre : 0000

THÈSE

PRÉSENTÉE À

L'UNIVERSITÉ BORDEAUX I

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET D'INFORMATIQUE

Par **Richard MOUSSA**

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ : INFORMATIQUE

SEGMENTATION MULTI-AGENTS EN IMAGERIE BIOLOGIQUE ET MÉDICALE : APPLICATION AUX IRM 3D

Soutenue le : 12 décembre 2011

Après avis des rapporteurs :

Vincent CHEVRIER	Maître de Conférences/HDR
Philippe JOLY	Professeur

Devant la commission d'examen composée de :

Vincent CHEVRIER	Maître de Conférences/HDR	Rapporteur
Philippe JOLY	Professeur	Rapporteur
Achille BRAQUELAIRE .	Professeur	Examinateur
Pascal DESBARATS	Professeur	Examinateur
Marie BEURTON-AIMAR	Maître de Conférences	Examinateur
Mohamed MOSBAH	Professeur	Examinateur
Guillaume HUTZLER	Maître de Conférences/HDR	Examinateur
Pascal BALLET	Maître de Conférences	Examinateur

– 2011 –

Remerciements

Résumé

La segmentation d'images est une opération cruciale pour le traitement d'images. Elle est toujours le point de départ des processus d'analyse de formes, de détection de mouvement, de visualisation, des estimations quantitatives de distances linéaires, de surfaces et de volumes. À ces fins, la segmentation consiste à catégoriser les voxels en des classes basées sur leurs intensités locales, leur localisation spatiale et leurs caractéristiques de forme ou de voisinage. La difficulté de la stabilité des résultats des méthodes de segmentation pour les images médicales provient des différents types de bruit présents. Dans ces images, le bruit prend deux formes : un bruit physique dû au système d'acquisition, dans notre cas l'IRM (Imagerie par Résonance Magnétique), et le bruit physiologique dû au patient. Ces bruits doivent être pris en compte pour toutes les méthodes de segmentation d'images. Durant cette thèse, nous nous sommes focalisés sur des modèles Multi-Agents basés sur les comportements biologiques des araignées et des fourmis pour effectuer la tâche de segmentation. Pour les araignées, nous avons proposé une approche semi-automatique utilisant l'histogramme de l'image pour déterminer le nombre d'objets à détecter. Tandis que pour les fourmis, nous avons proposé deux approches : la première dite classique qui utilise le gradient de l'image et la deuxième, plus originale, qui utilise une partition intervoxel de l'image. Nous avons également proposé un moyen pour accélérer le processus de segmentation grâce à l'utilisation des GPU (Graphics Processing Unit). Finalement, ces deux méthodes ont été évaluées sur des images d'IRM de cerveau et elles ont été comparées aux méthodes classiques de segmentation : croissance de régions et Otsu pour le modèle des araignées et le gradient de Sobel pour les fourmis.

Mots-clés : Systèmes Multi-Agents, agents sociaux, GPU, segmentation d'images, IRM, imagerie médicale.

Table des matières

Remerciements i

Résumé ii

Introduction 1

I Problématique 3

1 Systèmes Multi-Agents 5

1.1 Introduction . . . 5

1.2 Les entités d'un SMA . . . 6

1.2.1 Définition d'un agent . . . 7

1.2.2 Les types d'agents . . . 8

1.2.3 Définition de l'environnement . . . 10

1.2.4 Interaction entre agents . . . 10

1.3 Comportement du système . . . 11

1.3.1 Émergence . . . 11

1.3.2 Auto-Organisation . . . 12

1.3.3 Stigmergie . . . 13

1.4 Les Méta-heuristiques . . . 14

1.4.1 Les colonies d'araignées . . . 15

1.4.2 Les colonies de fourmis . . . 16

1.4.3 Autres méta-heuristiques . . . 18

1.5 Conclusion . . . 19

2 La segmentation d'IRM médicales 20

2.1 IRM cérébrale . . . 20

2.1.1 Principe de l'IRM . . . 20

2.1.2 Image anatomique du cerveau . . . 21

2.1.3 Qualité de l'image acquise par IRM . . . 22

2.1.3.1 Le bruit . . . 22

2.1.3.2 Le volume partiel . . . 23

2.1.3.3 Les inhomogénéités d'intensité . . . 23
2.2 Segmentation d'IRM de cerveau . . . 24
2.2.1 Segmentation d'images . . . 24
2.2.2 Méthodes classiques de segmentation . . . 25
2.2.2.1 Méthodes basées sur les points . . . 26
2.2.2.2 Méthodes basées sur les contours . . . 29
2.2.2.3 Méthodes basées sur les régions . . . 31
2.2.3 Méthodes utilisant les SMA . . . 33
2.2.3.1 Approches points . . . 33
2.2.3.2 Approche contours . . . 34
2.2.3.3 Approche régions . . . 35
2.3 Conclusion . . . 38

II Modélisation et expérimentations 39

3 IPSiMAS 41
3.1 Segmentation régions : les araignées . . . 41
3.1.1 Création de l'environnement . . . 42
3.1.2 Définition des colonies . . . 42
3.1.3 Description du cycle de vie . . . 42
3.1.4 Identification des paramètres . . . 44
3.1.4.1 Des problèmes . . . 45
3.1.4.2 Des solutions . . . 45
3.1.5 Instanciation de l'environnement . . . 49
3.1.6 Processus de segmentation . . . 51
3.1.7 Construction de l'image segmentée . . . 55
3.2 Segmentation contours : les fourmis . . . 57
3.2.1 Création de l'environnement . . . 57
3.2.2 Description du cycle de vie . . . 58
3.2.3 Identification des paramètres . . . 59
3.2.3.1 Des problèmes . . . 59
3.2.3.2 Des solutions . . . 59
3.2.4 Instanciation de l'environnement . . . 60
3.2.5 Processus de segmentation . . . 61
3.2.6 Construction de l'image segmentée . . . 67
3.2.7 Amélioration de la recherche des contours . . . 69
3.3 Ordonnanceur . . . 69
3.3.1 Exécution séquentielle . . . 69
3.3.2 Exécution parallèle . . . 69
3.4 Vers une modélisation Orienté Objet . . . 71

3.4.1 Agent . . . 71
3.4.2 Environnement . . . 73
3.4.3 Simulation . . . 74
3.5 Conclusion . . . 74

4 Expérimentations 76
4.1 Les images de test . . . 76
4.2 Les méthodes de comparaison . . . 79
4.2.1 Otsu multi-niveaux . . . 79
4.2.2 Croissance de régions . . . 80
4.2.3 Opérateur de Sobel . . . 83
4.3 Critères de comparaisons . . . 85
4.3.1 Pour la segmentation en régions . . . 85
4.3.2 Pour l'extraction de contours . . . 86
4.4 Résultats sur la segmentation en régions : méthode des araignées . . . 87
4.5 Résultats sur la segmentation contours : modèles des fourmis . . . 91
4.6 Conclusion . . . 95

Conclusion 97

Bibliographie 100

Table des figures 108

Liste des tableaux 110

A Implémentation 111
A.1 Catégories de Parallélisation . . . 111
A.2 Modèles de parallélisation . . . 112
A.3 Programmation GPGPU . . . 113
A.4 Mise en œuvre . . . 114
A.4.1 Différents types de mémoire . . . 116
A.4.2 Le noyau . . . 117
A.4.3 Performance . . . 121

Introduction

Quelle que soit son origine, une image constitue une représentation d'un univers composé d'entités, *i.e.* des objets dans une scène (cellules, organes du corps humain, *etc.*), et l'identification de ces entités est souvent la première étape d'un processus d'analyse d'image, automatisé ou non. Le but de toute méthode de segmentation est l'extraction d'informations caractérisant ces entités. Ces informations correspondent à des points d'intérêt ou à des zones caractéristiques de l'image. En imagerie médicale par exemple, les images de cerveau (IRM, scanner, ...) sont utilisées pour étudier les pathologies qui touchent cet organe. En effet, les changements structurels dans le cerveau peuvent être le résultat de ces pathologies et la quantification de ces changements, par la mesure de caractéristiques de régions d'intérêt, peut être utilisée pour caractériser la gravité des maladies ou leur évolution.

Une multitude d'algorithmes de segmentation d'images existe à ce jour. La diversité des images, les origines variées des chercheurs, l'évolution de la puissance de calcul des ordinateurs, et un certain empirisme dans l'évaluation des résultats ont conduit à la production d'autant de procédures spécialisées et souvent difficiles à calibrer ou à réutiliser. Une segmentation robuste nécessite l'incorporation et l'utilisation efficace des connaissances contextuelles globales. De nombreux éléments à prendre en compte comme la variabilité de l'arrière-plan, les propriétés polyvalentes des partitions cibles qui ont des caractéristiques précises, ou la présence de bruit, font qu'il est difficile d'accomplir cette segmentation automatiquement.

Notre travail porte sur la segmentation d'images IRM (Imagerie par Résonance Magnétique) de cerveau et dans ce contexte le bruit est un des phénomènes majeurs à prendre en compte. En effet, d'une part ce bruit est la somme de sources de bruit diverses (bruit thermique, bruit physiologique, *etc.*) et d'autre part, les petites structures à extraire des images (notamment dans le cerveau) sont de la taille des zones de bruit. Pour appréhender ce problème, nous avons développé une méthodologie basée sur les Systèmes Multi-Agents capables de rechercher les contours d'une image et d'en extraire les régions. De tels systèmes sont fréquemment rencontrés en simulation de comportement de robots, de phénomènes écologiques ou encore pour résoudre des problèmes d'optimisation mais leur utilisation en traitement d'images reste marginale. Nous avons étudié des modèles basés sur des agents réactifs. Le type d'agents réactifs que nous avons sélectionné, est appelé *"agents sociaux"* car leurs comportements s'inspirent de comportements observés chez les insectes sociaux comme les fourmis ou comme certains types d'araignées capables de réaliser une tâche collective : la construction de toiles gigantesques. En ce qui nous concerne, nous voulons concevoir une méthode de segmentation d'images résistante au bruit, où les acteurs du processus de segmentation seront des agents sociaux, l'image segmentée

devant émerger de l'action collective de tous les agents.

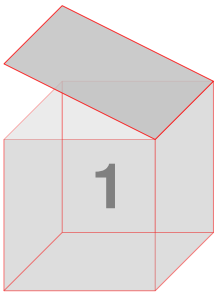
Un des reproches majeurs fait aux Systèmes Multi-Agents est la taille du système à implémenter, le nombre d'agents nécessaires pouvant être de l'ordre de plusieurs milliers. Le temps de simulation est lui aussi souvent un frein à l'utilisation des Systèmes Multi-Agents pour des applications de taille réelle. Dans notre cas, nous traitons des images de plusieurs millions de voxels, il est donc nécessaire de prévoir un très grand nombre d'agents et comme la tâche de segmentation est complexe, on ne peut espérer atteindre l'objectif avant un grand nombre d'itérations. Dans le cas de procédures séquentielles, la complexité des algorithmes est de l'ordre du nombre d'itérations de la simulation multiplié par le nombre d'agents. Nous avons donc étudié la possibilité de paralléliser nos simulations en utilisant le modèle de mémoire partagée disponible sur les cartes graphiques (GPU), espérant ainsi améliorer nos performances.

Ce manuscrit est composé de deux parties. La première partie présente la problématique de notre travail tant du coté des Systèmes Multi-Agents que de la segmentation d'images médicales. La seconde partie est consacrée à la présentation de notre modèle et des deux types d'agents que nous avons choisis pour segmenter les images. Nous y donnons aussi les résultats des expérimentations que nous avons effectuées sur des images de cerveau. Une annexe donne les détails d'implémentation de notre modèle sur GPU.

Première partie

Problématique

Chapitre



Systèmes Multi-Agents

La thématique des Systèmes Multi-Agents (SMA) est actuellement un champ de recherche très actif pour de nombreux types d'applications. Sa particularité provient de sa connexion avec plusieurs domaines de recherche comme l'Intelligence Artificielle Distribuée (IAD) et la Vie Artificielle (VA). C'est une discipline qui s'intéresse aux comportements collectifs produits par les interactions de plusieurs entités autonomes appelées agents. Dans ce chapitre, nous allons exposer l'essentiel des concepts utilisés dans le domaine du SMA. Tout d'abord, nous décrirons le principe général des SMA tout en exposant les différentes parties qui les composent en partant des notions d'agents, en passant par la définition d'un environnement jusqu'à l'exploration des caractéristiques d'un SMA. Puis nous présenterons les méta-heuristiques couramment utilisées et nous terminerons par un aperçu des différentes approches SMA appliquées en imagerie.

1.1 Introduction

Les SMA sont apparus au carrefour des recherches traitant l'IAD et la VA. L'une des sources d'inspiration pour ces systèmes est le comportement des sociétés d'animaux. Celles généralement étudiées sont les familles d'insectes telles que les abeilles, les fourmis ou les termites. D'autres sources d'inspiration se trouvent également dans la sociologie, la psychologie sociale, les sciences cognitives, *etc* (Ferber, 1995). Ces systèmes ont repris les modes de communication et de concertation (organisation, négociation, combinaison) entre les agents provenant de l'IAD et les idées d'autonomie et d'émergence de solutions à partir des interactions individuelles de la VA.

D'une part, les SMA se placent au sein des sciences cognitives, des sciences sociales et des sciences naturelles pour modéliser, expliquer et simuler des phénomènes naturels, et susciter des modèles d'auto-organisation. D'autre part, ils se présentent comme une pratique et une technique dont l'objectif est de réaliser des systèmes complexes à partir du concept *Agent* et de leurs interactions par l'intermédiaire des processus de communication, de coopération et de coordination d'actions

(Ferber, 1995). Le principe d'un modèle IAD réside dans le fait de découper le problème à résoudre en une multitude de sous-problèmes qui vont être à leur tour résolus par des agents afin de fournir des solutions partielles. Ces solutions partielles vont ensuite être regroupées pour construire la solution globale du problème. De plus, les systèmes SMA définissent les agents comme des entités ayant leur propre autonomie et flexibilité. L'objectif de la modélisation d'un SMA est alors de déterminer comment les agents vont se coordonner pour échanger leurs connaissances, leurs buts et leurs stratégies pour agir et résoudre des problèmes complexes. Généralement, les types d'application des SMA sont décomposés en trois grandes classes (Boissier et al., 2004) :

1. la **modélisation de phénomènes du monde réel** afin de les simuler pour mieux comprendre leur comportements. Ceci est réalisé par l'observation, la compréhension et l'explication de leur comportement et de leur évolution. Ce sont par exemple, des applications de simulation de phénomènes sociaux, environnementaux, éthologiques, *etc* ;
2. la **conception de programmes** dans lesquels des **agents** qualifiés d'*assistants* jouent le rôle des **êtres humains**. La notion d'agent simplifie la conception de ces programmes et amène de nouvelles problématiques centrées *utilisateurs* telles que la communication, la sécurité, *etc.* (Hayzelden and Bigham, 1999; Lespérance et al., 1994). On trouve un exemple de tel approche dans les systèmes de ventes aux enchères dans lesquels les agents jouent les rôles de commissaires priseurs et d'acheteurs (Sandholm, 1999).
3. la **résolution distribuée de problèmes** : il s'agit de concevoir des systèmes artificiels intégrant un ensemble d'agents et une dynamique organisationnelle permettant l'accomplissement collectif d'une tâche. L'hypothèse de base est que des agents, de par leurs comportements et leurs interactions locales, peuvent donner lieu, à un niveau global, à un comportement d'ensemble cohérent et fonctionnel (Steels, 1990). Ce sont par exemple les fourmis qui partent de leur fourmilière à la recherche de la nourriture en créant le plus court chemin (Dorigo et al., 1996). C'est plutôt ce type d'application qui nous concerne.

À l'heure actuelle, la programmation de robots, de jeux ou encore la simulation de processus biologiques sont autant d'applications des SMA qui montrent que ce domaine est très actif aussi bien en recherche fondamentale qu'en développement.

Nous allons maintenant décrire les concepts des catégories d'agents, de notion d'environnement, de modes d'interaction, d'auto-organisation et d'émergence.

1.2 Les entités d'un SMA

À première vue, un SMA peut être simplement considéré comme un ensemble d'agents partageant un environnement commun. Toutefois, il est difficile de trouver une définition unique du concept d'*Agent*, d'où il s'avère difficile de caractériser un SMA. Ferber (Ferber, 1995) a exposé quelques aspects fondamentaux qui permettent de réaliser cette caractéristisation à partir des systèmes rencontrés dans la nature:

– **un environnement E**, disposant en général d'une métrique ;

- **un ensemble d'objets O**, auxquels on peut associer une position dans E à un moment donné. Ces objets sont passifs : les agents peuvent les percevoir, les créer, les détruire et les modifier ;
- **un ensemble d'agents A**, lesquels représentent les entités actives du système ;
- **un ensemble de relations R**, qui unissent les objets entre eux ainsi que les agents ;
- **un ensemble d'opérateurs Op** permettant aux agents de A de percevoir, produire, consommer, transformer et manipuler des objets de O ;
- **des opérateurs** chargés de représenter l'application de ces opérations et la réaction du monde à cette tentative de modification, que l'on appellera les lois de l'univers.

D'après cette description, nous pouvons définir un SMA comme un système qui s'intéresse aux comportements collectifs produits par les interactions de plusieurs agents autonomes et flexibles et que ces interactions sont définies comme la coopération et la compétition entre ces agents qui peuvent être hétérogènes (figure 1.1). Dans ce qui suit, nous allons définir la notion d'agent ainsi que les catégories d'agents pouvant être représentées dans un SMA.

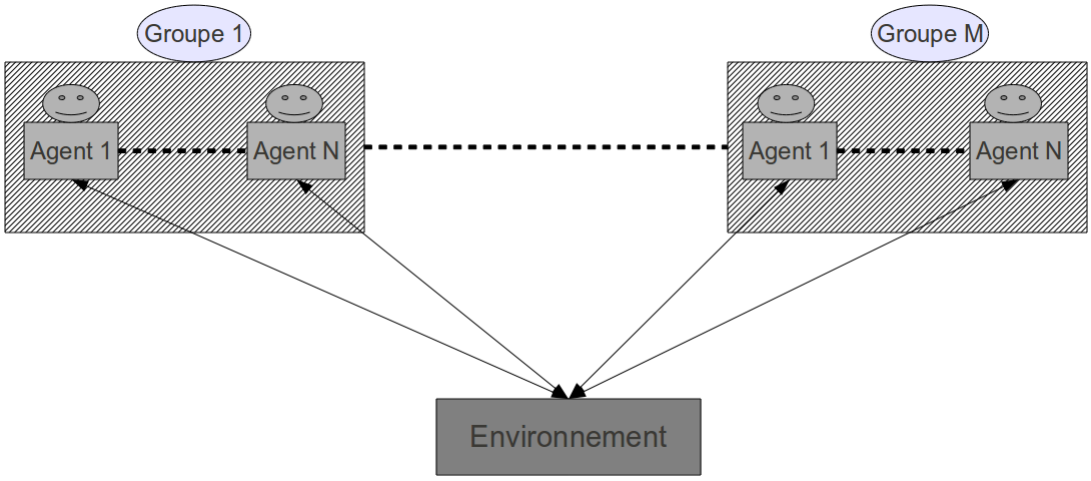


FIGURE 1.1: Relation unissant l'environnement et l'ensemble des agents.

1.2.1 Définition d'un agent

Dans la littérature, nous trouvons plusieurs définitions du concept d'**Agent**. Elles présentent certaines similitudes et dépendent du type d'application pour laquelle est conçu l'agent (Ferber, 1995; Jennings et al., 1998). Les définitions respectives de Ferber et de Jennings et al. sont généralement les plus communément utilisées au sein de la communauté Multi-Agents:

Définition 1 *(Ferber, 1995) Un agent est une entité autonome, réelle ou abstraite, qui est capable d'agir sur elle-même et sur son environnement, qui dans un univers Multi-Agents, peut communiquer avec d'autres agents, et dont le comportement est la conséquence de ses observations, de ses connaissances et des interactions avec les autres agents.*

Définition 2 *(Jennings et al., 1998) Un agent est un système informatique, situé dans un environnement, et qui agit d'une façon autonome et flexible pour atteindre les objectifs pour lesquels il a été conçu.*

Les définitions exposées ci-dessus présentent plusieurs propriétés clés tels que l'autonomie, l'action, la communication et la flexibilité. L'autonomie est un élément du comportement de l'agent qui est fortement liée aux notions d'apprentissage et d'adaptation. Si ceci est déterminé a priori, l'agent manque alors d'autonomie. Les agents sont considérés comme autonomes s'ils ne sont pas guidés par des commandes venant de l'utilisateur ou d'un autre agent, mais par un ensemble de tendances qui peuvent prendre la forme de buts individuels à satisfaire ou de fonctions de satisfaction ou de survie que l'agent cherche à optimiser. Maes définit les agents autonomes comme suit :

Définition 3 *(Maes, 1995) Les agents autonomes sont des systèmes de calcul qui se trouvent dans un environnement complexe et dynamique, perçoivent et agissent d'une manière autonome dans cet environnement, et ce faisant réalisent les objectifs et les tâches pour lesquelles ils ont été conçus.*

L'action est un concept fondamental pour les SMA reposant sur le fait que les agents accomplissent des actions qui vont modifier leur environnement et donc leurs prises de décisions futures (Ferber, 1995).

Les interactions entre agents dépendent du mode de communication. Une communication entre 2 agents est considérée comme une action pour celui qui la transmet et comme une perception pour celui qui la reçoit (Drogoul, 1994).

1.2.2 Les types d'agents

Nous distinguons deux grandes familles d'agents : les **agents réactifs** et les **agents cognitifs** (Wooldridge and Jennings, 1995). Cette distinction tient essentiellement au processus décisionnel chez l'agent et à la représentation de l'environnement dont il dispose. Si l'agent est doté d'une représentation symbolique de l'environnement à partir duquel il est capable de formuler des raisonnements, nous disons qu'il est cognitif tandis que s'il ne dispose que d'une représentation limitée à ses perceptions, nous disons qu'il est réactif (Ferber, 1995). Qu'il soit réactif ou cognitif, un agent évolue toujours selon un cycle en trois étapes :

1. **Perception** : ses capteurs lui fournissent une vision locale de son environnement ;
2. **Décision** : suivant ses intentions, son état interne et sa perception, l'agent choisi une action à effectuer ;
3. **Action** : il modifie l'environnement par son action.

Nous allons maintenant présenter les différents types d'agents les plus couramment utilisés dans les SMA.

Agents cognitifs Les agents cognitifs disposent d'une base de connaissances comprenant l'ensemble des informations disponibles et un raisonnement nécessaire à la réalisation de leur tâche ainsi qu'à la gestion des interactions avec les autres agents et avec l'environnement. De plus, ils possèdent des buts et des plans pour décider de leurs actions. Les groupes d'agents peuvent coordonner leur activité et négocier entre eux pour résoudre leurs conflits. Ces comportements peuvent être assimilés à des comportements sociaux et les recherches dans ce domaine s'appuient sur les travaux de sociologie

des organisations des groupes d'individus (Ferber, 1995). L'une des architectures cognitives les plus connues est l'architecture BDI : Belief (Croyance), Desire (Désir), Intention (Intention) (Rao and Georgeff, 1992). Les croyances correspondent aux informations dont dispose l'agent sur son environnement. Les désirs correspondent aux états de l'environnement que l'agent souhaiterait voir réalisés. Les intentions correspondent aux projets de l'agent pour satisfaire ses désirs.

Agents réactifs Contrairement aux agents cognitifs, les agents réactifs ne sont pas nécessairement assez intelligents individuellement pour que le système ait un comportement global que l'on puisse qualifier d'intelligent (Drogoul, 1994). Des mécanismes de réactions aux évènements et aux différentes perceptions, ne tenant pas compte ni d'une explication des buts, ni des mécanismes de planification, peut alors découler la résolution de problèmes complexes: le comportement complexe du système émerge alors de la coexistence et de la coopération des comportements simples de chaque agent (*cf.* figure 1.2).

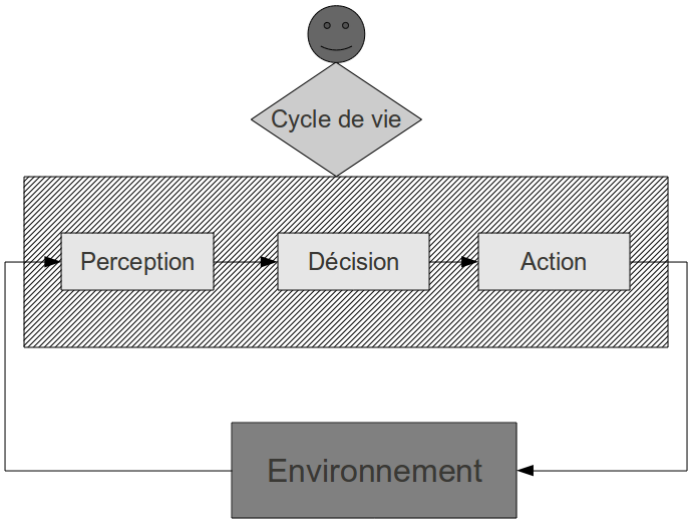


FIGURE 1.2: Principe de la réaction d'un agent réactif sur une environnement à partir de son cycle de vie.

L'organisation des fourmilières ou des termitières est généralement citée en exemple de tels systèmes qualifiés d'émergents. Dans ce cas, les buts sont la recherche de la nourriture, les soins apportés aux œufs ou encore la construction des nids. Les SMA réactifs présentent des intérêts en terme de fiabilité assurée par le grand nombre d'agents du système et leur simplicité ainsi que le passage à l'échelle par l'adaptation à l'augmentation du nombre d'agents (Parunak, 1997).

Agents hybrides Les agents hybrides sont conçus pour combiner des capacités réactives à des capacités cognitives, ce qui leur permet d'adapter leur comportement en temps réel à l'évolution de l'environnement (Fisher, 1985). Dans le modèle hybride, un agent est composé de plusieurs couches, rangées selon une hiérarchie en trois couches (Müller and Pischel, 1994) : la couche purement réactive, la couche intermédiaire et la couche sociale. Dans ces trois couches, on va de la simple capture d'informations à partir de senseurs jusqu'à la gestion de la coopération entre agents en passant par la construction d'abstraction à partir des connaissances.

1.2.3 Définition de l'environnement

L'environnement est l'endroit d'immersion dans lequel évoluent les agents. On distingue le plus souvent deux sortes : l'**environnement social** qui est composé des autres agents du système et l'**environnement physique** au sein duquel les agents évoluent.Un agent peut se placer dans l'un et/ou dans l'autre. L'**environnement physique** dépend largement de l'application traitée alors que l'**environnement social** dépend souvent des choix de conception des SMA. L'environnement est défini comme suit :

Définition 4 *(Weyns and Holvoet, 2007) L'environnement est une première classe d'abstraction fournissant les conditions environnementales aux agents pour exister et sert d'intermédiaire pour les interactions entre les agents ainsi que pour l'accès aux ressources.*

Voici les propriétés essentielles de l'environnement (Russell et al., 1996) :

- **accessible/inaccessible** : le système peut obtenir une information complète, exacte et à jour sur l'état de son environnement. Dans un environnement inaccessible, une information partielle est uniquement disponible ;
- **déterministe/non déterministe** : une action a un effet unique et certain. Si le système agit dans son environnement, il n'y a aucune incertitude sur l'effet de son action sur l'état de l'environnement. L'état suivant de l'environnement est complètement déterminé par l'état courant. Dans un environnement non déterministe, une action n'a pas un effet unique garanti ;
- **dynamique/statique** : l'état de l'environnement dynamique dépend des actions du système qui se trouve dans cet environnement, mais aussi des actions d'autres processus. Ainsi, les changements ne peuvent pas être prédits par le système. Un environnement statique ne peut changer sans que le système agisse ;
- **continu/discret** : le nombre d'actions et de perceptions possibles dans cet environnement est infini et indénombrable. Dans le cas discret, l'environnement est défini par un nombre d'états fini et selon un découpage régulier tel qu'une grille. Par exemple, dans le jeu de la poursuite (qualifié comme un problème de proies et de prédateurs) dont le but est de parvenir à ce que les prédateurs capturent les proies en les entourant (Ferber, 1995), les agents proies et prédateurs se déplacent dans une grille et ont chacun des coordonnées/positions à l'intérieur de chaque case).

L'environnement est défini à la fois comme lieu d'inscription permettant de partager le résultat des activités de chaque agent et comme ensemble de contraintes sur la dynamique des SMA (Müller and Pischel, 1994). L'environnement peut être très élémentaire et se réduire à l'ensemble des canaux de communication entre les agents. Toutefois, dans les SMA à base d'inspiration biologique ou physique, l'environnement qui possède généralement une métrique, est plus présent et a un rôle primordial (Ferber, 1995).

1.2.4 Interaction entre agents

L'interaction entre agents dans un SMA peut être définie comme la mise en relation dynamique de deux ou plusieurs agents par le biais d'un ensemble d'actions. Ces interactions prennent deux formes :

directes et indirectes.

Les **interactions directes** sont en général des caractéristiques des agents cognitifs qui, travaillant pour un but précis, sont capables de communiquer intentionnellement. Ce type d'interaction se rapproche d'un acte de langage. Tandis que les **interactions indirectes** sont les caractéristiques des agents réactifs qui répondent à des stimulis de l'environnement par le dépôt d'information relative à des critères environnementaux (Chevrier, 2002).

Dans le cas des agents réactifs, Les interactions entre les agents sont modélisées comme suit :

1. **coopération** : travailler ensemble dans un but commun, ceci est fait au travers de l'environnement sans aucune communication directe entre les agents ;
2. **coordination** : organiser la solution d'un problème de telle sorte que les interactions nuisibles soient évitées ou que les interactions bénéfiques soient exploitées.

Nous allons maintenant présenter comment les différentes interaction entre les agents peuvent conduire à la réalisation de tâches complexes par le SMA dans son ensemble.

1.3 Comportement du système

Dans les SMA, on parle souvent de phénomènes émergents ou d'auto-organisation des agents. L'**émergence** et l'**auto-organisation** soulignent des caractéristiques différentes du comportement d'un système (De Wolf and Holvoet, 2004). Les deux phénomènes peuvent exister séparément, comme ils peuvent coexister dans un système dynamique. Le but de cette section est la différenciation des caractéristiques importantes des deux concepts au travers d'une vue d'ensemble de l'historique de l'utilisation de chaque concept.

1.3.1 Émergence

Historique (Ali and Zimmer, 1998) L'origine de l'émergence pourrait bien être le postulat datant de la Grèce antique : *« le tout est plus que la somme de ses parties »*. Des traces du concept d'émergence et d'auto-organisation sont retrouvées dans des écrits de Thalès (625 av. J.-C. vers 547 av. J.-C.) et Anaximandre (610 av. J.-C. vers 546 av. J.-C.). De même, Aristote (384 av. J.-C., 322 av. J.-C.) parle du *« tout avant les parties »*. Il se réfère ainsi à l'explication admise de la précédence de l'entité représentant le tout par rapport aux parties sur lesquelles le tout est construit et il se sert de cette notion pour expliquer certains phénomènes que l'on n'arrive pas à décomposer. Au fil du temps, d'autres définitions ont été données (J.W. Von Goethe (1749 - 1832), Lewis (Lewes, 1875)).

Le concept Une des définitions les plus récentes de De Wolf et Holvoet correspond bien aux systèmes d'agents réactifs :

Définition 5 *(De Wolf and Holvoet, 2004) Un système exhibe de l'émergence quand il y a des émergents cohérents au niveau macro qui apparaissent dynamiquement à partir des interactions entre les parties au niveau micro. De tels émergents sont nouveaux par rapport aux parties du système prises individuellement en compte.*

Dans cette définition, les **émergents** désignent içi le résultat du processus d'émergence : propriétés, comportements, structures, modèles, *etc*. Le **niveau macro** considère le système dans son intégralité tandis que le **niveau micro** considère le système du point de vue des différentes entités qui le composent. J. Odell a identifié comme éléments essentiels des phénomènes émergents : le contrôle décentralisé, le lien bi-directionnel entre les deux niveaux micro et macro (figure 1.3), la robustesse du système et la flexibilité dûes à l'insensibilité aux perturbations locales et le fait que le comportement global soit nouveau par rapport aux comportements individuels du niveau macroscopique (James Odell, 2002).

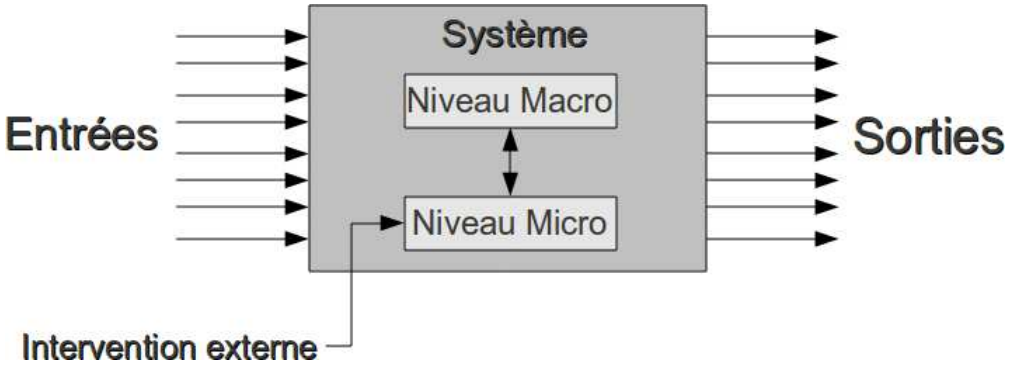


FIGURE 1.3: L'émergence.

De nombreux cas de systèmes émergents ont été étudiés en biologie et en écologie. En écologie, de nombreux systèmes naturels composés d'individus autonomes exhibent des aptitudes à effectuer des tâches qualifiées de complexes sans contrôle global. C'est le cas des colonies d'insectes sociaux tels que les termites ou les fourmis (Bonabeau and Theraulaz, 1997). En biologie, le système immunitaire est représentatif du fonctionnement d'un système complexe composé d'un ensemble d'agents autonomes. L'immunité du système étant le phénomène émergent obtenu.

1.3.2 Auto-Organisation

Historique Descartes fut le premier à parler d'organisation produite de manière spontanée (Descartes, 1856) et Asby fut pour sa part le premier à utiliser le mot d'auto-organisation (Ashby, 1947). Si nous nous référons aux origines de l'auto-organisation issues de la physique, une définition minimaliste basée sur l'ordre peut être donnée : le désordre dans un système physique est l'état homogène obtenu après avoir perturbé le système, alors que l'ordre est un système particulier et hétérogène (Atlan, 1986).

Le concept Le concept d'auto-organisation est utilisé depuis les dernières décennies en biologie (Thompson, 1992) pour expliquer la diversité organisée des motifs chez les vivants, par exemple les rayures et d'autres motifs chez les animaux ou les structures géométriques de coquillages (Heylighen, 1999). Toutefois, c'est un cas particulier d'auto-organisation qui ne fait pas apparaître tout l'aspect dynamique de l'auto-organisation. En effet, l'auto-organisation ne tend pas forcément vers un état d'équilibre stable (Nicolis and Prigogine, 1977). A l'heure actuelle, l'idée d'auto-organisation est

maintenant employée dans beaucoup de disciplines et d'applications telles que les travaux liés à la simulation, les réseaux et la robotique (Mostefaoui et al., 2003).

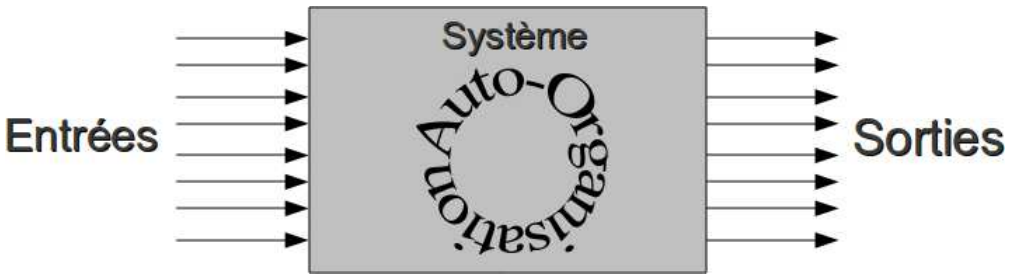


FIGURE 1.4: L'auto-organisation.

Avant d'étudier les propriétés de l'auto-organisation (figure 1.4), nous en donnons les deux définitions suivantes extraites de la littérature :

Définition 6 *(Dempster et al., 1998) L'auto-organisation réfère à ce qui est exactement suggéré : des systèmes qui s'organisent sans direction, manipulation ou contrôle externes.*

Définition 7 *(De Wolf and Holvoet, 2004) L'auto-organisation est un processus dynamique et adaptatif où les systèmes acquièrent et maintiennent une structure sans contrôle externe.*

Les propriétés les plus importantes de l'auto-organisation sont :

- **Mise en ordre croissante :** On peut définir l'organisation comme une augmentation de l'ordre dans le comportement d'un système qui permet à ce dernier d'acquérir une structure spatiale, temporelle ou fonctionnelle. Cependant, il faut signaler que tout système exhibant une mise en ordre croissante ne peut être dit auto-organisé que s'il est autonome.
- **Autonomie :** Toute augmentation d'ordre ne peut être appelée auto-organisation. La deuxième caractéristique importante de l'auto-organisation est l'absence de contrôle externe, d'où provient le terme "auto". Un système doit alors s'organiser sans interférence avec l'extérieur.
- **Adaptation et robustesse face aux changements :** Dans les systèmes auto-organisés, le terme robustesse est employé dans le sens de l'adaptabilité aux perturbations et aux changements. Un système auto-organisé doit pouvoir faire face à ces changements et maintenir son organisation de manière autonome.
- **Dynamique :** Les changements et les perturbations affectent les structures organisées. Pour répondre à ces changements, une structure auto-organisée devra savoir s'adapter. On parlera alors d'organisation dynamique.

Les SMA sont actuellement un des outils les plus utilisés pour modéliser les systèmes auto-organisés.

1.3.3 Stigmergie

On considère généralement que la combinaison de l'émergence et de l'auto-organisation constitue une approche prometteuse pour proposer des solutions aux problèmes complexes et fortement dynamiques (figure 1.5). Cette combinaison a suscité notre intérêt pour traiter la segmentation d'images,

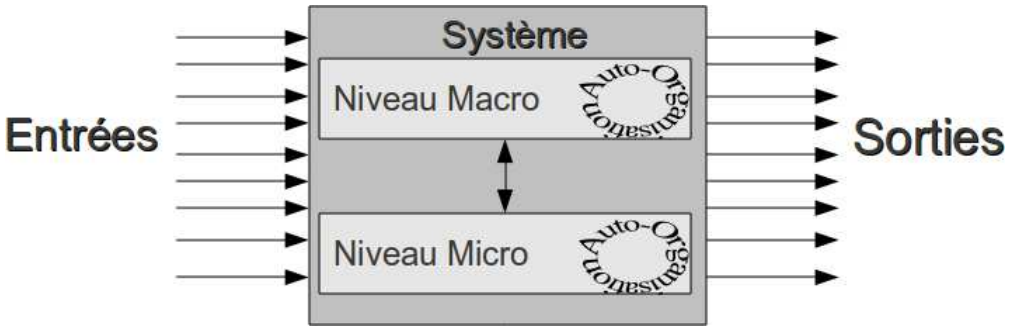


FIGURE 1.5: La stigmergie (De Wolf, 2007).

activité qualifiée de complexe et pour laquelle aucune méthodologie générique n'a pas encore été élaborée.

La combinaison des concepts entre l'émergence et l'auto-organisation a été définie par Grassé (Grassé, 1959). Il définit le terme de stigmergie qui provient des mots grecs *stigma* "marque, signe" et *ergon* "travail, action", exprimant la notion que les actions d'un agent laissent des signes dans l'environnement, signes perçus par lui-même et par les autres agents et qui déterminent leurs prochaines actions. Grassé a étudié le comportement des termites et a décrit les modifications de l'état de l'environnement par l'intermédiaire des gradients de phéromones déposés, les termites arrivent, sans régulation centrale, à communiquer et à coordonner leur action.

La structuration de l'environnement a un effet sur le comportement des agents qui agissent à leur tour sur la structure de l'environnement, et ainsi de suite. Nous assistons alors à un phénomène de couplage entre le système et son environnement, générant l'auto-organisation, à un niveau macro, de l'ensemble (Bonabeau and Theraulaz, 1997). Ce couplage permet au système de s'auto-produire comme un collectif auto-organisé. En d'autres termes, la stigmergie fournit un moyen de coordination entre les agents, émanant du système lui-même. C'est donc une méthode de communication indirecte dans un environnement émergent auto-organisé, où les individus communiquent entre eux en modifiant leur environnement. Plusieurs travaux concernent l'exploitation de l'environnement pour la résolution et l'optimisation de problèmes à travers des mécanismes stigmergiques : des phéromones, des toiles, des champs de forces, etc. Parmi les approches d'optimisation, nous citons l'algorithme d'optimisation par des colonies de fourmis qui miment la capacité de véritables fourmis à résoudre des problèmes complexes. Par exemple, partant de leur nid à la recherche de nourriture, les fourmis déposent des phéromones sur leur chemin afin de guider les autres vers un chemin qui finira par être le plus court chemin pour accéder à la nourriture. La perturbation des phéromones gérée par le phénomène de l'évaporation induit une auto-organisation des fourmis sur le chemin le plus utilisé avec une émergence de la solution optimale.

1.4 Les Méta-heuristiques

Une méta-heuristique est un processus de génération itérative qui permet de guider une heuristique subordonnée par la combinaison de divers concepts tels que l'exploitation et l'exploration

de l'espace de recherche, ainsi que des stratégies d'apprentissage qui sont utilisées pour structurer efficacement l'information afin de déterminer des solutions d'excellente qualité (College, 2002). Les méta-heuristiques sont en général non-déterministes et ne donnent aucune garantie d'optimalité mais peuvent faire usage de l'expérience accumulée durant la recherche de l'optimum, pour mieux guider la suite de la procédure de recherche. Nous distinguons les méta-heuristiques à solution unique parmi lesquelles on retrouve de nombreux algorithmes de recherche locale, et les méta-heuristiques à base de population. Notre intérêt se porte sur les méta-heuristiques à base de population qui travaillent sur une population d'agents communiquant indirectement via l'environnement.

1.4.1 Les colonies d'araignées

La recherche de solutions optimales grâce à des colonies d'agent-araignées est inspirée de l'activité de tissage de soies des araignées sociales (Chevrier, 2006). Tout comme les araignées sociales collaborent pour créer de gigantesque toiles, les agent-araignées tisseront une toile sur l'image dans le cas de l'analyse d'image. Cette toile fournira alors les éléments de segmentation de l'image. Ces araignées évoluent dans un environnement créé à partir d'un espace bien défini. Elles possèdent un cycle de vie décrivant les actions qu'elles doivent exécuter. Le cycle du système consiste à exécuter le cycle de vie de chaque araignée jusqu'à satisfaction d'un critère d'arrêt. Chaque point de l'espace est une position potentielle pour une araignée et permettra à cette dernière d'atteindre d'autres points grâce à un voisinage calculé sous deux formes :

1. un voisinage statique calculé localement à partir de la position de l'araignée ;
2. un voisinage dynamique obtenu à partir des soies entre le point courant et les autres points de l'environnement qui pourra s'agrandir au fur et à mesure que les araignées tisseront de nouveaux fils.

Chaque araignée est constituée d'un état interne et d'un ensemble de fonctionnalités. L'état interne comprend une position courante et un historique de la dernière position où l'araignée a déjà tissé. Les fonctionnalités dont dispose l'araignée sont les suivantes :

1. mouvement ;
2. tissage d'une soie entre deux points ;
3. retour à la dernière position tissée et restauration de son historique.

L'ensemble des araignées est partitionné en sous-ensembles nommés colonies. Chaque colonie a une tâche bien définie à accomplir et peut entrer en concurrence avec les autres colonies (s'il y en a) présentes dans l'environnement. La fonction de mouvement permet à l'araignée de se déplacer dans l'environnement. Le choix de la nouvelle position dépend entre autre des paramètres de la colonie à laquelle appartient l'araignée. Suivant qu'il y ait une ou plusieurs colonies, le calcul du choix de la position future sera différent.

Dans le cas où une seule colonie est présente, l'araignée a seulement le choix entre se déplacer vers un point du voisinage local non tissé ou vers un point déjà tissé dans l'environnement. Ensuite, chaque point du voisinage extrait aura une probabilité d'être choisi :

1. si l'ensemble appartient au voisinage local non tissé, alors tous les points auront la même chance d'être la nouvelle position ;
2. sinon, chaque point à une probabilité relative au nombre de soies présentes sur le point pour favoriser la sélection du prochain déplacement.

Dans le cas où plusieurs colonies sont présentes, un poids w(p) est associé à chaque point. Plus le poids d'un point est grand, plus la probabilité que ce point soit choisi sera grande. Pour chaque point p, la probabilité P(p) du déplacement est :

$$P(p) = \frac{w(p)}{\sum_{a \in Voisinage(p)} w(a)} \tag{1.1}$$

Cette fonction de poids va permettre de favoriser les pixels en fonction de leur nombre de soies. Si p n'a pas encore été tissé, alors le poids de ce point est grand (supérieure à une valeur de saturation pour la convergence de l'algorithme). Dans le cas contraire, le poids est défini selon les paramètres d'attraction des soies de la même colonie ou des autres colonies comme suit:

$$Attraction(MaColonie) + Attraction(AutresColonies) = 1 \tag{1.2}$$

D'où le poids de chaque point tissé devient :

$$\begin{aligned} w(p) \quad = \quad & Attraction(MaColonie) \times \#Soies(MaColonie) \\ + \quad & Attraction(AutresColonies) \times \#Soies(AutresColonie) \end{aligned} \tag{1.3}$$

Après le déplacement de l'araignée vers le nouveau point, une tentative de tissage est réalisée si le point satisfait un ensemble de contraintes (définies en fonction du problème à résoudre) et ce, afin de créer un lien entre la position courante et le dernier point où il y a eu un tissage. Dans le cas où le tissage est impossible, un retour en arrière peut être effectué en fonction d'une probabilité tirée au sort et de la probabilité de retour en arrière fixée pour chaque colonie.

L'algorithme 1 commence par placer les araignées dans l'environnement et le cycle de vie des araignées est lancé par le tissage des soies comme décrit ci-dessus jusqu'à satisfaction du critère d'arrêt. À la fin, le nombre de soie déposé sur les points est analysé pour extraire la solution optimale.

1.4.2 Les colonies de fourmis

L'heuristique des colonies de fourmis (ACO [1]) est inspirée par le comportement de recherche de nourriture d'une colonie de fourmis réelles pour trouver le plus court chemin entre sa fourmilière et la nourriture. Ceci est réalisé au moyen d'une substance chimique appelée *phéromone* déposée et accumulée par les fourmis qui se dirigent vers la nourriture. Durant la recherche de nourriture, la fourmi utilise ses propres connaissances de l'endroit où l'odeur de la nourriture provient (les informations heuristiques) et le fait que d'autres fourmis aient choisi le même chemin (l'information phéromone). Elle prend ensuite la décision de son propre chemin en confirmant la voie par le dépôt de phéromones, rendant ainsi plus dense leur concentration, et augmentant ainsi la probabilité du chemin

1. Ant Colony Optimization.

Algorithme 1 Colonies d'araignées

Entrées: E : Environnement , A : Liste d'araignées, C: Critère de terminaison.
Sorties: S : Solution optimale.

```
Placer les araignées A Dans E
t ← 0.
TantQue C non satisfait Faire
    t ← t + 1.
    Pour Chaque araignée a_i de A Faire
        Mouvement(a_i,E).
        Tissage(a_i,E).
        RetourEnArrière(a_i,E).
    Fin Pour
Fin TantQue
S = ExtraireSolution(E).
```

d'être choisi par les autres fourmis. On parlera alors d'émergence du meilleur chemin (le plus court) à partir des choix successifs des fourmis et avec une quantité de phéromone déposée inversement proportionnelle à la longueur du chemin (Dorigo et al., 1996).

Dorigo et al ont choisi une colonie de fourmis d'agents coopérants pour résoudre le problème du voyageur de commerce défini comme suit : supposons pour toute paire de nœuds V_i et V_j du graphe G, un poids de connexion attaché à l'arrête (V_i, V_j), une concentration de phéromone et des informations heuristiques. Soit une fourmi f_k présente sur le nœud V_i au temps, cette fourmi choisira de suivre l'arête conduisant au nœud V_j avec la probabilité :

$$P_{i\to j}^{f_k}(t) = \begin{cases} \frac{\phi_{i\to j}^{\alpha}(t)\times\eta_{i\to j}^{\beta}(t)}{\sum_{k\in CandidatsPotentiels(f_k)}\phi_{i\to k}^{\alpha}(t)\times\eta_{i\to k}^{\beta}(t)} & \text{Si} \quad j \in CandidatsPotentiels(f_k) \\ 0 & \text{Sinon} \end{cases} \tag{1.4}$$

Où les paramètres $\phi_{i\to j}$ et $\eta_{i\to j}$ sont respectivement la concentration de phéromones et les informations heuristiques sur l'arrête (V_i, V_j), α et β sont des constantes entre 0 et 1 qui déterminent respectivement le poids accordé à la concentration de phéromones et aux informations heuristiques, et CandidatsPotentiels(f_k) sont l'ensemble des sommets autorisés à être visités selon les contraintes du problème. Après avoir choisi une destination, chaque fourmi f_k se déplace et dépose sur l'arête une quantité de phéromone sur le chemin définie comme suit :

$$\Delta\phi_{i\to j}^{f_k}(t) = \frac{Q}{L_{f_k}} \tag{1.5}$$

Où Q est une constante positive et L_{f_k} est le coût de la voie utilisée par la fourmi f_k. Après que toutes les fourmis aient achevé leur recherche de chemins, la concentration de phéromone sur chaque voie est mise à jour selon la formule suivante:

$$\phi_{i\to j}^{f_k}(t) = (1-\rho)\times\phi_{i\to j}^{f_k}(t) + \sum_{f'_{k=1}}^{f'_{k=N}}\Delta\phi_{i\to j}^{f'_k} \tag{1.6}$$

Où ρ est un facteur d'évaporation ($0 \le \rho \le 1$). Cette évaporation permet de renforcer le poids des phéromones récentes alors que les anciennes concentrations, qui correspondent plutôt aux solutions

initiales indésirables trouvées sont petit à petit oubliées. L'algorithme 2 donne le traitement global qui est appliqué lors de la simulation.

Algorithme 2 Colonies de fourmis

Entrées: G : Graphe , V_1 : nœud source, V_2 : nœud destination, F : Liste de fourmis, C: Critère de terminaison.
Sorties: S : Solution optimale.

```
Placer les fourmis F sur V1.
t ← 0.
TantQue C non satisfait Faire
    t ← t + 1.
    Pour Chaque fourmi f_k de F Faire
        Se déplacer selon l'équation 1.4.
        Ajout de phéromones selon l'équation 1.5.
    Fin Pour
    Mettre à jour les phéromones selon l'équation 1.6.
Fin TantQue
S = ExtraireSolution(G).
```

1.4.3 Autres méta-heuristiques

Les méta-heuristiques ne se limitent pas seulement à celles décrites précédemment. Deux autres méthodes basées sur l'**algorithmique génétique** ou sur les **essaims de particules** ont aussi été utilisées par différents auteurs.

L'**algorithmique génétique** s'inspire d'une analogie entre les processus d'optimisation et l'évolution des êtres vivants. La simulation des mécanismes de variation et de sélection présente dans les processus évolutifs naturels est exploitée pour résoudre des problèmes artificiels d'optimisation, comme l'ordonnancement d'un atelier pour utiliser au mieux les ressources humaines et matérielles disponibles (Croce et al., 1995). Cette analogie avec l'optimisation consiste à considérer les solutions potentielles au problème comme des chromosomes. Ceux-ci sont manipulés par des opérateurs de sélection, de mutation ou encore de croisement. Le principe de l'opérateur de sélection consiste à favoriser la propagation des meilleures solutions parmi la population, tout en préservant la diversité génétique, pour explorer de nouvelles régions de l'espace de recherche (Holland, 1992). La mutation consiste à tirer aléatoirement un gène dans le chromosome et à le remplacer par une valeur aléatoire apportant ainsi la stochasticité nécessaire à une exploration efficace de l'espace de recherche. Cet opérateur garantit la possibilité d'atteindre tout l'espace d'état (mais dans un temps infini). L'opérateur de croisement concerne la phase de recombinaison, il maintient la diversité en manipulant les composantes des individus (chromosomes). On crée ainsi des générations de solutions depuis des solutions parentes vers des solutions enfants plus optimales en évaluant la qualité de chaque solution grâce à une fonction de *fitness* [2] de chaque chromosome.

2. Le choix de la fonction de fitness reste un des problèmes majeurs à sélectionner pour ce type de méthodes.

L'optimisation par **essaim de particules** (PSO [3]) a été introduit en 1995 par Kennedy et Eberhart comme une alternative à l'algorithme génétique standard (Kennedy and Eberhart, 2002). Cette optimisation est inspirée des essaims d'insectes, des bancs de poissons et des nuées d'oiseaux et de leurs mouvements coordonnés. Les individus de l'algorithme sont appelés particules et la population est appelée essaim. L'algorithme PSO se compose d'un essaim de particules "volant" dans l'espace de recherche. La position de chaque particule est une solution potentielle au problème. La vitesse de chaque particule est modifiée en fonction de sa distance à sa meilleure position personnelle et la meilleure position globale de l'essaim. En d'autres termes, les particules se déplacent en fonction de leur propre expérience, qui est dans ce cas la mémoire de la meilleure position qu'elles ont rencontrée, et en fonction de leur meilleur voisin (Clerc and Kennedy, 2002).

Ces méthodes ont été testées par d'autres auteurs dans le contexte de la segmentation d'images, on pourra se reporter aux travaux de Bhanu et al. pour l'utilisation des algorithmes génétiques (Bhanu et al., 1995) et aux travaux de Omran et al. pour l'utilisation des essaims de particules (Omran et al., 2006).

1.5 Conclusion

Ce chapitre a présenté les quelques éléments fondamentaux de la définition des SMA. Ils permettent la modélisation distribuée des traitements et des connaissances, différents modes de coopération, etc. Nous avons vu qu'il existait deux types d'agents, réactifs et cognitifs. Nous avons montré qu'il existe différents algorithmes pour modéliser l'activité d'un SMA. Dans le cadre de notre travail, ce sont les méta-heuristiques des araignées et des fourmis qui ont été choisies pour effectuer respectivement une détection de régions et de contours. Il vont représenter les agents dans l'environnement et le comportement approprié. Reste à définir le contexte dans lequel ce travail se situe. Ainsi, le prochain chapitre sera consacré à la présentation de ce contexte : l'analyse d'images IRM 3D de cerveau. Nous présenterons les différentes parties du cerveau à détecter et les méthodes classiques de l'opération que nous appliquons à ces images : la segmentation, qui est un des points fondamentaux du traitement et de l'analyse d'images.

3. Particle Swarm Optmization.

Chapitre 2

La segmentation d'IRM médicales

L'objectif de notre travail est la segmentation d'images IRM de cerveau. Nous allons présenter dans ce chapitre les spécificités des IRM cérébrales, puis la problématique de la segmentation elle-même et enfin différentes méthodes de segmentation s'appliquant à cette problématique.

2.1 IRM cérébrale

2.1.1 Principe de l'IRM

Le principe de l'IRM est basé sur la détection des propriétés magnétiques des protons contenus dans les molécules d'eau du corps. En effet, la concentration en eau des tissus varie d'un organe à l'autre ou avec son état physiologique. Afin d'estimer la quantité de protons dans les tissus, le corps est placé dans un champ magnétique de haute intensité $\overrightarrow{B_0}$. Soumis à ce champ magnétique, les protons sont alignés le long de la direction de $\overrightarrow{B_0}$. Cet alignement est alors perturbé par une courte durée d'un champ magnétique $\overrightarrow{B_1}$ perpendiculaire au champ $\overrightarrow{B_0}$ et oscillant à une fréquence donnée appelée fréquence de Larmor (42.56MHz / Tesla pour le proton). Lorsque $\overrightarrow{B_1}$ est supprimé, les protons reviennent à leur état précédent dans un mouvement d'oscillation. Le mouvement de ces dipôles électriques induit un petit champ magnétique qui est enregistré par l'antenne d'IRM. Cette RMN (Résonance Magnétique Nucléaire) du signal est la somme S de toutes les contributions individuelles des protons. Si $\overrightarrow{B_0}$ est homogène, tous les protons oscillent à la même fréquence et il est donc impossible de discriminer les protons dans l'espace 3D. Afin de fournir des informations d'imagerie, il est nécessaire de superposer un gradient magnétique $\overrightarrow{G}$(x, y, z) à $\overrightarrow{B_0}$ suivant les trois coordonnées spatiales. Ainsi, chaque proton, à un ensemble donné de coordonnées (x, y, z), oscille à une fréquence différente après l'extinction du champ $\overrightarrow{B_1}$. En présence de $\overrightarrow{G}$, le signal S recueilli par

l'antenne IRM peut être décrit par la formule suivante:

$$S(t) = \int_{-\infty}^{+\infty} M(x,y,z) \cdot e^{i \cdot \gamma \cdot (G_x x + G_y y + G_z z)} \cdot dx \cdot dy \cdot dz \tag{2.1}$$

où G_x, G_y et G_z sont les valeurs de $\overrightarrow{G}$ au point (x, y, z), γ est le rapport gyromagnétique et $\overrightarrow{M}$ (x, y, z) est la densité magnétique au même point. Cette expression étant une transformée de Fourier spatiale, on peut obtenir directement $\overrightarrow{M}$(x, y, z) en appliquant une transformée de Fourier inverse. Pour des coupes en 2D, $\overrightarrow{M}$(x, y) est représenté par le nombre complexe M.$e^{i\phi}$ où M est l'amplitude et ϕ est la phase. Le niveau de gris correspondant à l'image anatomique est proportionnel à M. ϕ peut être également utilisé pour construire une image de phase (servant entre autres à estimer le flux sanguin, la température, *etc*) (Denis De Senneville et al., 2003).

En modifiant les paramètres d'acquisition, en particulier le temps de répétition entre deux excitations (TR) ou le temps entre le signal d'excitation et la réception de l'écho (TE), on peut modifier la pondération de l'image. En effet, les tissus ont des temps de relaxation longitudinale (temps T_1) et transversale (temps T_2) caractéristiques (figure 2.1). Les écarts de temps T_1 et T_2 mesurés permettent de caractériser les tissus en chaque élément de l'échantillonnage discret.

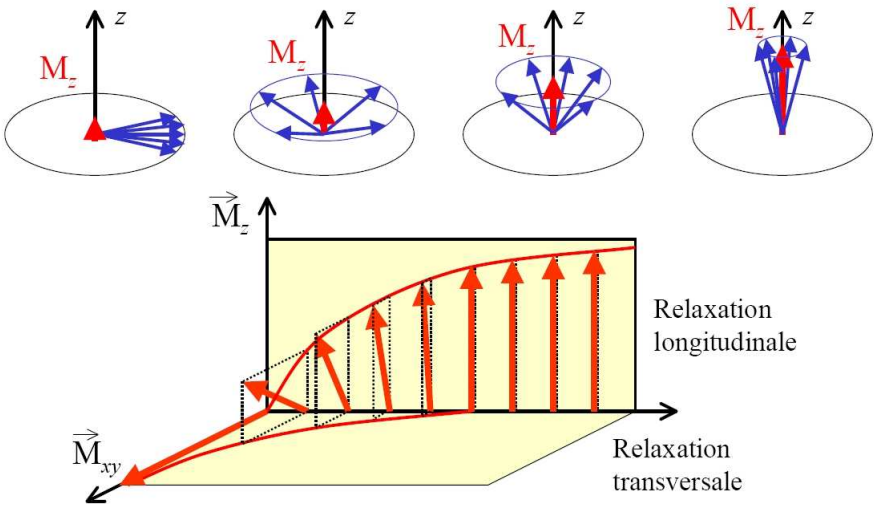


FIGURE 2.1: Le principe de la caractérisation des tissus de l'image.

2.1.2 Image anatomique du cerveau

Le cerveau est la partie la plus volumineuse du système nerveux central. Il est constitué essentiellement de deux hémisphères, séparés par la scissure inter-hémisphère, et relié par différentes structures telles que le corps calleux, le thalamus et l'hypothalamus. Comme le montre la figure 2.2, le cerveau est composé d' :

1. une matière grise (MG) ;
2. une matière blanche (MB) ;
3. un liquide céphalo-rachidien (LCR) ;

4. une peau ;
5. un muscle ;
6. une graisse ;
7. une peau / un muscle ;
8. une conjonction.

Lors de l'acquisition d'une image de cerveau, deux objets supplémentaires sont détectés : le fond et la crâne.

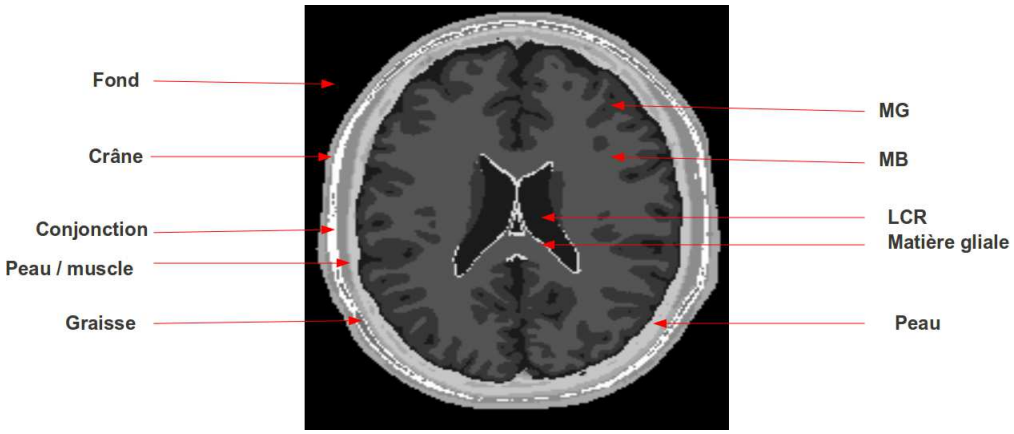


FIGURE 2.2: Coupe du cerveau.

2.1.3 Qualité de l'image acquise par IRM

Les sources d'artéfacts en IRM sont nombreuses. Ces perturbations sont responsables d'erreur dans l'encodage de l'image, de perte ou de rehaussement artificiel du signal. Les sections suivantes pointent les principales sources que l'on peut corriger en partie en post-traitement. Connaître leur origine permet de mieux les appréhender, de les réduire, voire de les supprimer.

2.1.3.1 Le bruit

Comme tout dispositif de mesure physique, les données sont entachées de bruit. Il provient à la fois du patient (agitation thermique des protons à l'origine d'émissions parasites) et de la chaine de mesure (convertisseurs analogique-numérique, antenne, *etc*). La perturbation par le bruit est généralement quantifiée par le rapport signal sur bruit (RSB), fonction de l'amplitude du signal observé par rapport à l'importance de la variation du bruit. Ce rapport, et donc la qualité de l'image, peut être amélioré en considérant différentes antennes, un champ magnétique $\overrightarrow{B_0}$ plus intense, une matrice de résolution moins fine, ou encore une multiplication des mesures. Néanmoins, il restera toujours un bruit dans l'image reconstruite finale. On peut considérer que le bruit dans l'image suit une distribution Ricienne qui, avec un rapport signal sur bruit suffisant (typiquement RSB > 3), peut être approximé par un bruit gaussien (Sijbers et al., 1998).

2.1.3.2 Le volume partiel

L'effet de volume partiel est lié à la discrétisation de l'espace : lorsque la surface entre plusieurs objets se trouve dans un même élément de volume discret (le point), la mesure dans ce point résulte d'un mélange des contributions des différents objets. Cet effet se manifeste principalement à l'interface entre les matières (MG, MB, LCR, *etc*) ou lors de la présence de structures trop fines pour être visibles à la résolution de l'image : vaisseaux sanguins, structures fines de la MG, *etc*. L'effet de volume partiel est particulièrement présent à l'interface LCR-MG dans les replis du cortex, car l'épaisseur des sillons corticaux est généralement inférieure à la résolution des images.

2.1.3.3 Les inhomogénéités d'intensité

Les inhomogénéités d'intensité sont des variations de l'intensité observées pour un même tissu. Leurs sources sont multiples :

- Les inhomogénéités liées aux imperfections de l'imageur, dues :
  - à l'hétérogénéité du champ statique $\overrightarrow{B_0}$ et du champ d'excitation $\overrightarrow{B_1}$. Les champs magnétiques produits ne peuvent en effet être parfaitement uniformes, provoquant des plages d'ombre dans l'image;
  - à la qualité de l'antenne de réception, et particulièrement aux variations spatiales de sa sensibilité. Lors des examens en IRM anatomique, on privilégie la plupart du temps des antennes avec une sensibilité spatialement stable.

    Ces sources d'inhomogénéités sont responsables de lents déphasages et décalages de fréquence dans le volume étudié. Elles produisent de lentes variations spatiales d'intensité dans l'image reconstruite nommé "champ de biais" [1] (Juntu et al., 2005).
- Les inhomogénéités liées à des propriétés biologiques des tissus, dues :
  - à des compositions histologiques différentes des tissus. En effet, les temps de relaxation T_1 et T_2 de la matière blanche et de la matière grise dépendent des régions anatomiques (Wansapura et al., 1999) et de l'âge (Cho et al., 1997). Par exemple le cortex et les noyaux gris centraux comme le putamen sont tous composés de MG, mais présentent des intensités légèrement différentes sur une acquisition pondérée T_1. Cet artefact est expliqué par des études histologiques, qui révèlent que le putamen est traversé par un grand nombre de faisceaux de fibres myélinisées. Ces fibres sont trop fines pour être visibles à l'IRM : l'intensité observée résulte d'un mélange de MG et de MB du fait de l'effet de volume partiel. De la même manière, la MB est plus claire dans le corps calleux que dans les autres régions, car les fibres myélinisées y sont plus concentrées et orientées dans une direction commune;
  - à l'artéfact de susceptibilité magnétique. Chaque tissu possède une susceptibilité magnétique qui s'exprime par une aimantation interne propre induite par le champ magnétique statique $\overrightarrow{B_0}$. La différence de susceptibilité magnétique à l'interface entre deux tissus provoque une distorsion du champ $\overrightarrow{B_0}$. Ces hétérogénéités locales sont responsables de déphasages et de

1. Bias field.

décalages de fréquences localisés, à l'origine d'une perte de signal, et d'hétérogénéité d'intensité. Elles sont principalement localisées aux interfaces air-tissu et cortical-tissu, et très marquées en présence de matériel métallique. En particulier, cet artéfact est responsable de perturbations dues à la seule présence du patient dans l'imageur.

Ces inhomogénéités sont des artéfacts dont la fréquence spatiale est plus importante que celle du champ de biais.

2.2 Segmentation d'IRM de cerveau

La présence simultanée de tous ces artefacts dans les images IRM anatomique du cerveau va constituer un difficulté supplémentaire pour leur segmentation.

Nous allons passer en revue différentes techniques de segmentation qui ont été proposées sur ces images, ainsi que leurs limites après avoir défini formellement la problématique de la segmentation d'images.

2.2.1 Segmentation d'images

Une image discrète 3D est composée d'un grille d'éléments appelés **voxels**. Un **voxel** correspondant à un volume élémentaire de cette grille.

À chaque voxel d'une image 3D est associée une valeur, qui correspond le plus souvent à un niveau de gris ou à une étiquette. Le type de la valeur associée à un voxel dépend du type d'acquisition utilisée pour obtenir l'image, mais n'entre pas en considération pour définir la segmentation d'image 3D.

L'opération de segmentation consiste à créer un partitionnement en régions de l'image. Ces régions doivent correspondre aux différentes régions recherchées par la tâche de segmentation.

Définition 8 *Le **partitionnement** d'une image **I** est la division de **I** en un ensemble de N sous-ensembles connexes $R_1, \ldots, R_N$, appelées régions de l'image, tel que :*

1. $\forall_i, R_i \neq \emptyset$;
2. $\forall_{i,j|i\neq j}, R_i \cap R_i = \emptyset$;
3. $\boldsymbol{I} = \bigcup_i R_i$

Définition 9 *Une région est définie comme un ensemble de voxels connexes maximal par rapport aux critères de segmentation utilisés.*

Cette maximalité doit être observée sur la totalité des régions simultanément. Pour une région donnée, le regroupement de quelques voxels pourrait permettre d'obtenir un meilleur résultat par rapport aux critères de segmentation. L'effet de bord de ce regroupement est qu'il supprime l'appartenance de voxels à d'autres régions, qui peuvent obtenir un résultat moins bon pour les mêmes critères de segmentation. Ainsi les régions sont des ensembles de voxels dits maximaux pour des critères de

segmentation donnés, si la fusion de deux régions voisines donne une partition moins bonne au regard de ces mêmes critères.

Pour modéliser les critères de segmentation vérifiés par le partitionnement effectué, on a besoin de définir un prédicat qui doit être satisfait pour les voxels appartenant à une même région. Ainsi, La segmentation d'une image devient le partitionnement de l'image conformément au prédicat choisi :

Définition 10 *Une image segmentée* $\boldsymbol{I}$*, utilisant un prédicat P, est l'ensemble E de N régions* $R_1; \ldots; R_N$ *tel que :*

1. $\boldsymbol{E}$ *est une partition de* $\boldsymbol{I}$*;*
2. $\forall_{R \in E}$*, R vérifie le prédicat P;*
3. $\forall_{(R_i,R_j) \in E^2 | R_i et R_j sont voisines}$*,* $R_i \cup R_j$ *ne vérifie pas le prédicat P.*

Dans cette définition, le prédicat correspond aux critères choisis afin de déterminer les voxels appartenant à la même région. Il faut remarquer que, étant donné un prédicat, plusieurs partitions satisferont la définition 10. Le résultat de la segmentation dépend alors :

1. du prédicat choisi;
2. de l'algorithme de segmentation.

Quelle que soit la définition de segmentation considérée, l'unicité de la partition satisfaisant la définition ne peut être garantie. Il est cependant possible de définir des critères de segmentation tels que une seule partition soit solution du problème. Selon la définition 10, le prédicat, qui est vrai si tous les voxels d'une même région ont la même valeur, permet de construire de manière déterministe l'unique partition. Cependant, utiliser de tels prédicats ne fournit que rarement une segmentation satisfaisante de l'image pour des applications concrètes. Ainsi, le processus de segmentation doit non seulement chercher une solution, mais aussi être un processus d'optimisation visant à trouver la meilleure partition parmi celles satisfaisant la définition 10. La segmentation d'image consiste alors à :

1. définir un ensemble de critères permettant de différencier les régions à segmenter dans l'image;
2. trouver la meilleure partition vérifiant un ensemble de critères prédéfinis.

Dans la suite, on parlera de points pour des pixels en 2D et pour des voxels en 3D.

2.2.2 Méthodes classiques de segmentation

Il est possible de regrouper les approches de segmentation de différentes façons (Haralick and Shapiro, 1985; Pham et al., 2000). La manière de classifier ces approches dépend des caractéristiques du domaine d'application et du contexte pour lequel elles sont différenciées. Nous avons décomposé les approches de segmentation en fonction des éléments de l'image sur lesquelles elles se basent. Cette classification se fait en trois groupes :

1. les méthodes basées sur les **points** : ce sont les méthodes qui cherchent une répartition des points en classes;

2. les méthodes basées sur les **contours** : ce sont les méthodes qui cherchent à placer correctement les frontières entre les régions de l'image;
3. les méthodes basées sur les **régions** : ce sont les méthodes qui cherchent à regrouper les points par région de l'image. Ainsi, les contours de l'image sont implicitement définis.

2.2.2.1 Méthodes basées sur les points

Les méthodes basées sur les points sont celles qui s'appliquent directement sur des points sans aucune considération pour la structure de la région. L'information localisée par point est utilisée pour décider si oui ou non le point appartient à la région désirée.

Seuillage

Cette méthode est probablement la méthode la plus simple parmi les méthodes de segmentation. Un seuil est choisi pour créer une partition binaire des intensités des points. Tous les points ayant une intensité supérieure au seuil sont regroupés dans une classe et ceux ayant une intensité inférieure au seuil sont placés dans une autre classe. Une multitude de méthodes ont été mises en œuvre à partir de cette simple définition mais elles souffrent de limitations sur les images IRM de cerveaux à cause des artefacts présentés dans la section 2.1.3.

En effet, certaines images sont biaisées, représentant des objets dont l'intensité n'est pas homogène, ou avec des parties de l'image plus contrastées que d'autres. Il peut être intéressant dans ce cas de recourir à un seuillage non-uniforme : on partage l'image en différentes régions, sur lesquelles on effectue un seuillage avec un seuil différent. De la même manière, on peut estimer la variation d'intensité entre deux zones en interpolant le seuil entre différentes régions.

Les seuils peuvent être fixes ou calculés en fonction de l'image. Dans ce cas, la méthode la plus courante consiste à calculer les seuils à partir de l'histogramme de l'image (Ridler and Calvard, 1978; Trier and Jain, 1995). L'histogramme des intensités d'une image représente, pour chaque intensité le nombre de points de l'image ayant cette intensité (figure 2.3). L'utilisation de cette méthode part de l'hypothèse que chaque pic, ou mode, de l'histogramme correspond à un certain objet de l'image. La recherche du seuil optimal consiste à partager l'histogramme suivant ses différents modes, c'est-à-dire à trouver les zones d'intensité minimale entre deux modes.

Bien que simple, cette méthode est très efficace pour la segmentation d'images ayant un très bon contraste entre les régions. Elle est généralement utilisée comme une première étape vers une segmentation.

Le principal inconvénient de cette technique est que les résultats dépendent des seuils utilisés. Toute modification aux valeurs des seuils peut donner une autre région segmentée. Les seuils sont généralement générés d'une manière interactive en utilisant un retour visuel sur l'histogramme de l'image. Un autre inconvénient est que cette technique est très sensible au bruit et à la non-homogénéité des intensités.

Classification

Les méthodes de classification par apprentissage sont des méthodes de reconnaissance de forme

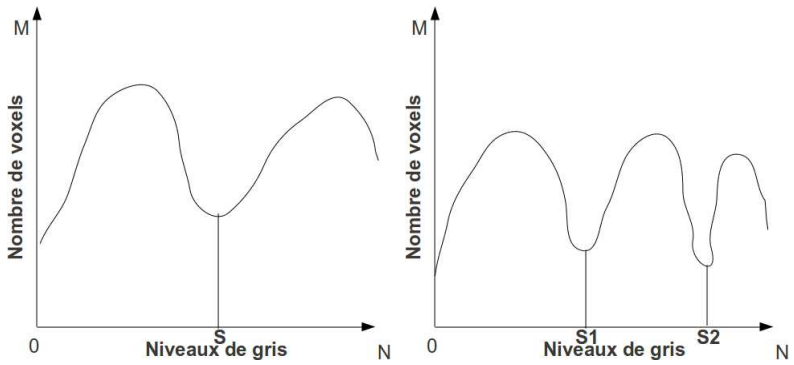


FIGURE 2.3: Exemples de seuillage simple et multiple sur deux histogrammes.

qui cherchent à diviser un espace d'éléments provenant de l'image en utilisant des données ayant des étiquettes connues. L'espace des éléments est un groupe de vecteurs d'attributs de dimension N représentant des caractéristiques à chaque point. Ces caractéristiques peuvent être les intensités des points, les gradients des points, les distances des points aux bords de l'image, *etc*.

Ces classifieurs sont de la catégorie *supervisée* car ils ont besoin de données pré-segmentées (soit manuellement ou par toute autre méthode). Les données pré-segmentées sont ensuite utilisées comme références pour procéder à la segmentation automatique sur de nouvelles données.

La forme la plus simple d'un classifieur est le modèle du plus proche voisin, où chaque point est classé dans la même classe que celle des données ayant des intensités proches. Le classifieur **k plus proches voisins** est la généralisation de cette approche, où le point est classé en fonction de la majorité des k plus proches données (Denœux, 1995). Un autre exemple similaire d'un classifieur est la fenêtre de Parzen, où la classification est faite en fonction de la majorité des couples de points au sein d'une fenêtre prédéfinie de l'espace des éléments centrés sur les points non-étiquetés (Breiman et al., 1977). Ces deux classifieurs sont non-paramétriques car ils ne font aucune hypothèse sur la structure statistique des données. Un autre classifieur utilisé est celui de Bayes ou maximum de vraisemblance. L'hypothèse de base est que les intensités des points sont échantillonnées indépendamment à partir d'un mélange de distributions de probabilités, généralement gaussiennes. La classification des points est obtenue en assignant chaque point selon la valeur maximale de sa probabilité d'appartenance à chaque classe.

Lorsque les données suivent une loi correspondant à un mélange de distributions gaussiennes, le classifieur du maximum de vraisemblance peut donner de bons résultats et est capable de fournir une segmentation dont les étiquettes sont les probabilités d'appartenance à chaque région. Les classifieurs standard exigent que la structure à segmenter possède des caractéristiques distinctes quantifiables. Parce que les données peuvent être étiquetées, les classifieurs permettent de transférer ces étiquettes à des nouvelles données tant que l'espace des éléments distingue suffisamment chaque étiquette. Étant non-itératif, ils sont relativement efficaces en temps de calcul et, à la différence du seuillage, ils peuvent être appliquées à des images multi-canaux.

L'inconvénient majeur de cette méthode est la constitution d'une base de données d'apprentissage, processus qui peut être long et laborieux et qui peut ne pas être exhaustif (ce qui peut conduire à des

résultats biaisés).

Dans cette même catégorie de méthodes de classification, il existe des algorithmes *non supervisés* tels que les algorithmes **k-moyennes** et **c-moyennes floues**.

k-moyennes

L'algorithme des k-moyennes classifie les objets selon leurs attributs en k classes prédéfinies en supposant que les attributs des objets forment un espace vectoriel (Macqueen, 1967). L'objectif est de minimiser la variance intra-classe :

$$V = \sum_{i=1}^{k} \sum_{x_i \in S_i} |x_j - \mu_i|^2 \tag{2.2}$$

où S_i, i = 1, 2, . . ., k sont les k classes et μ_i est le centroïde des points $x_j \in S_i$.

L'algorithme commence par partitionner les points en k ensembles initiaux, soit au hasard, soit en utilisant une heuristique. Il calcule ensuite le centroïde de chaque ensemble et construit une nouvelle partition en associant chaque point avec le centroïde le plus proche. L'algorithme se déroule suivant une alternance entre calcul des centroïdes des nouvelles classes et appariement des points avec le centroïde le plus proche jusqu'à convergence. Celle-ci est obtenue quand aucun point ne change de groupe ou lorsque les centroïdes ne changent plus.

Cet algorithme est très populaire car il est extrêmement rapide en pratique. En effet, le nombre d'itérations est typiquement inférieur au nombre de points. En terme de performance, cet algorithme ne garantit pas un optimum global. La qualité de la solution dépend grandement des ensembles initiaux et peut, en pratique, être bien en déçà de l'optimum global. Comme l'algorithme est très rapide, une méthode courante est de le lancer plusieurs fois et de retourner la meilleur partition. Un autre problème est qu'il est nécessaire de donner à priori le nombre de classes (i.e. k) à trouver ; cela n'est toutefois pas réellement gênant dans le cas de la segmentation d'IRM cérébrale car le nombre de classes est le plus souvent connu.

c-moyennes floues

L'algorithme des c-moyennes floues (FCM [2]) généralise l'algorithme des k-moyennes en permettant la classification floue basée sur la théorie des ensembles flous (Dunn, 1973). Dans ce cas, la fonction à minimiser est (Bezdek et al., 1999):

$$J_p(x, v : y) = \sum_{k=1}^{n} \sum_{i=1}^{c} (x_{ik})^p \|y_k - v_i\|_A^2 \tag{2.3}$$

où :

1. n est le nombre de points à traiter, c le nombre de classes désirées, p ∈ [1, +∞[est le poids de floutage ;
2. v = (v_1, . . ., v_c) est le vecteur des centres de classes ;
3. $\|\Delta\|_A$ est un produit scalaire où A est une matrice définie positive ;

2. Fuzzy C-Means.

4. x = $[x_{ik}] \in \mathbb{R}^{c.n}$, avec $x_{ik} \in [0, 1]$ $\forall$ 1 ≤ i ≤ c et 1 ≤ k ≤ n ; x_{ik} est la c-partition floue de y et doit vérifier :

$$\sum_{i=1}^{c} x_{ik} = 1 \ pour \ 1 \leq k \leq n, \ \sum_{i=1}^{n} x_{ik} > 0 \ pour \ 1 \leq i \leq c \tag{2.4}$$

Xue et al. utilisent les algorithmes c-moyennes floues pour combiner le filtre moyen au filtre médian local afin de réaliser la segmentation locale de volumes IRM de cerveaux (Xue et al., 2001). Pham utilise les inhomogéneités d'intensité des volumes IRM dans la fonctionnelle J_p et obtient ainsi un algorithme c-moyennes floue adaptatif qui permet une meilleure segmentation(Pham, 2001).

Les limites majeures de ces méthodes proviennent des données elles mêmes. En effet, ces méthodes sont sensibles au bruit qui malheureusement comme nous l'avons vu au début de ce chapitre est toujours présent dans les images IRM.

Champs aléatoires de Markov

Les champs aléatoires de Markov modélisent les interactions entre un point et son voisinage. Le formalisme des champs de Markov permet d'effectuer une segmentation de l'image en prenant en compte les interactions avec les points voisins. Nous considérons que les k régions que nous souhaitons segmenter forment une partition de l'image. Chaque région est représentée par une fonction caractéristique et identifiée par une étiquette dans 1, . . ., k. Le but de la segmentation est d'estimer le champ des étiquettes X à partir d'une réalisation bruitée de l'image Y . La démarche de la segmentation peut se formaliser comme un problème d'estimation bayésienne.

Une estimation du champ des étiquettes peut se faire suivant le critère du maximum à postériori (Zenglin et al., 2007). Cela se fait en minimisant l'énergie grâce à l'algorithme des modes conditionnels itérés [3] (Besag, 1986) ou du recuit simulé [4] (Laarhoven and Aarts, 1987).

Dans le cadre de la segmentation d'IRM cérébrale, plusieurs auteurs se sont intéressés à cette méthode pour réduire les artéfacts de l'image. On cite à titre exemple la modélisation des inhomogénéités par un bruit blanc gaussien additif à prendre en compte lors de la segmentation des 3 principaux tissus du cerveau (Rajapakse et al., 1997). Une autre forme de l'utilisation de cette méthode consiste à prendre en considération les effets du volume partiel en commençant par détecter 2 composants supplémentaires (mélanges de gris et de blanc et mélanges de gris et de LCR) afin de les replacer dans les 3 principaux tissus (Ruan et al., 2000).

La modélisation de ces artéfacts et leur combinaison restent à nos jours le problème majeur pour établir une connaissance de ces systèmes.

2.2.2.2 Méthodes basées sur les contours

Ces méthodes visent à délimiter les objets selon leurs contours. Elles ne se basent généralement pas sur les intensités mais sur les variations d'intensité dans l'image, variations qui sont normalement significatives aux frontières entre les régions.

3. ICM en anglais: Iterated Conditionnal Mode.
4. SA en anglais: Simulated Annealing.

Techniques de détection de bords

Les bords sont formés par l'intersection de deux régions avec des intensités différentes. Ils sont l'un des principaux indices de distinction visuelle de deux régions. L'algorithme correspondant se déroule en deux étapes :

1. les bords sont détectés en utilisant une certaine forme de différenciation ;
2. ces bords sont ensuite regroupés pour former des frontières séparant les points de la région désirée des autres points de l'image.

Un certain nombre d'opérateurs de détection de bords a été proposé à cet effet. Liu a proposé un algorithme de détection de surface 3D qui étend en 3D l'opérateur classique de Roberts (Liu, 1977). Zucker et Hummel ont mis au point un opérateur optimal de détection de bords, qui est essentiellement un opérateur de Sobel(Zucker and Hummel, 1981).

Ce sont des opérateurs se basant sur des dérivées premières ou secondes. Pour cette raison, ils sont connus pour être très sensibles aux bruits.

Morphologie mathématique

La morphologie mathématique propose un cadre théorique très cohérent pour le traitement et l'analyse d'images (Serra, 1983). Fondée sur des notions ensemblistes, elle étudie les caractéristiques morphologiques (forme, taille, ...) des objets par des transformations non linéaires associées à un ou plusieurs objet(s) de référence (élément(s) structurant(s)). La forme et la taille des éléments structurants permettent d'agir localement sur les objets d'une image. Les opérations morphologiques sont généralement simples à comprendre et à appliquer. Mais ceux-ci sont généralement difficiles à contrôler. Par exemple, il est difficile de contrôler les opérations de dilatation à moins de donner la limite supérieure du nombre de fois à dilater. Ces algorithmes exigent donc, en général, certains critères externes pour les contrôler. Ces opérations risquent également de changer la géométrie et la topologie des régions. Il est bien connu qu'une série de dilatations suivi par des érosions conduit à la perte de hautes fréquences et au remplissage des trous. De même, une série d'érosions suivie par des dilatations peuvent introduire des trous et des hautes fréquences. À partir de ces deux exemples simples, nous pouvons construire des opérateurs de plus en plus complexes, jusqu'à aboutir à des outils de haut niveau (ligne de partage des eaux, transformée en tout-ou-rien à niveau de gris). Ces algorithmes devraient être évités lorsque la précision est la première préoccupation et qu'il y a un risque de perte de données importantes. Comme pour les détecteurs de bords, on notera que les opérateurs morphologiques ne sont pas des algorithmes de segmentation par eux-mêmes, mais ils sont généralement une partie intégrante d'un pipeline de segmentation.

Modèles déformables

Les modèles déformables ont été introduits dans le cadre de la segmentation d'images par Terzopoulos et al. (Terzopoulos et al., 1988). Kass et al. ont introduit les modèles déformables les plus populaires, les *contours actifs* (Kass et al., 1988) . Les modèles déformables offrent une alternative robuste à la segmentation d'images en introduisant une information sur la forme de l'objet à segmenter, en étant peu sensible au bruit et aux contours irréguliers. De plus, certains modèles déformables sont

développés dans un espace continu (le contour obtenu a ainsi une précision supérieure à la résolution de l'image). Le principe de ces méthodes est de partir d'un modèle géométrique (courbe, surface, *etc*) et de plonger ce modèle dans l'image. Le modèle déformable peut ensuite se mouvoir sous l'action de forces internes (liées aux propriétés du modèle) et externes (liées à l'image) jusqu'à ce qu'il vienne se plaquer sur la structure d'intérêt. Les forces internes sont des contraintes qui régularisent la forme du modèle et l'empêchent de se déformer de façon anarchique. Les forces externes tendent à faire coïncider le modèle sur un contour d'objet à segmenter ou toute autre caractéristique de l'image.

Le principal avantage de cette méthode est son efficacité lorsque tous les paramètres sont bien choisis, ainsi que la finesse de la segmentation. Les inconvénients sont la sensibilité de la segmentation par rapport à la qualité de son initialisation, ainsi que le nombre généralement important de paramètres à régler, qui ne garantissent, au final, ni la terminaison ni la convergence vers le résultat souhaité. Même si les modèles déformables permettent l'obtention de contours fermés, ces contours sont lisses et induisent une variabilité anatomique dans le cas de la segmentation.

Isocontours

La méthode d'isocontours a été introduite par Osher et Sethian (Osher and Sethian, 1987). Elle est un cadre de travail analytique travaillant sur l'évolution géométrique d'objets. Au lieu d'employer une représentation Lagrangienne classique pour décrire les géométries, la méthode des isocontours les décrit à travers une fonction scalaire ϕ définie sur une grille fixe. L'équation d'évolution, formalisée sous forme d'une EDP [5], est contrainte par un champ de vitesse imposé dans le sens de la normale au contour. Ce champ est construit de manière à attirer le modèle vers les objets à extraire dans l'image sous contraintes de régularisation géométrique. La normale et la courbure du contour en chaque point sont facilement définies à partir des propriétés différentielles géométriques de cette représentation.

Un des principaux avantages de la description implicite d'un contour est qu'elle gère intrinsèquement les changements topologiques en cours de convergence. Par exemple, si un processus de segmentation est initialisé par des germes multiples, les collisions et fusions des composantes connexes sont gérées sans modification de l'algorithme, ce qui n'est pas le cas par exemple avec des contours actifs classiques. Le principal avantage de cette méthode est un meilleur résultat que celui obtenu par les modèles déformables produit quelque soit la forme de l'objet en question. Un inconvénient de cette méthode est le coût de calcul important dans sa formulation basique. Pour résoudre ce problème, Lefohn et al. ont utilisé des GPUs pour la détection d'une tumeur dans une image IRM de cerveau(Lefohn et al., 2003). Mais son inconvénient majeur est son processus d'initialisation (initialisation éloignée de la solution) ainsi que son paramétrage (connaissance à priori du nombre de types de région).

2.2.2.3 Méthodes basées sur les régions

Les méthodes basées sur les régions se focalisent sur l'extraction de régions en considérant leur homogénéité vis à vis de caractéristiques pertinentes (intensité, texture, ...) au niveau des points. Nous décrivons dans cette section les principales approches considérées dans la littérature.

5. Équation à Dérivées Partielles.

Croissance de régions

La croissance de régions est probablement la technique la plus simple en ce qui concerne cette approche. Elle consiste à extraire une zone d'intérêt d'une image en faisant croître une région à partir d'un ou de plusieurs germes constituant un sous-ensemble de la zone recherchée. La croissance est régie par un critère de propagation pouvant être basé sur des propriétés colorimétries, mais également géométriques ou topologiques. La croissance de région est rarement utilisée seule, mais généralement dans le cadre d'un ensemble d'opérations de traitement d'images. La qualité de la segmentation obtenue par ce type d'approche est liée à la pertinence du critère de propagation et au choix des germes, qui requièrent donc un temps de paramétrisation important.

Comme pour le seuillage, cette technique est simple, mais est souvent utilisée pour la segmentation. Le plus souvent, cette technique forme une partie du *"pipeline"* de segmentation pour une approche particulière. Elle est souvent utilisée comme méthode principale pour étudier les données de l'image avant l'application d'une technique de segmentation plus complexe. Cette technique est également sensible au bruit et à l'effet de volume partiel. Un autre inconvénient majeur de cette méthode est la nécessité du choix d'un point de départ (germe) pour construire la région d'intérêt.

Division/Fusion

La segmentation par division fournit une structure hiérarchisée qui permet d'établir des relations de proximité entre les régions, mais qui peut fractionner une même région en plusieurs ensembles distincts.

La segmentation par fusion produit un nombre minimal de régions connexes, mais fournit une structure horizontale qui n'indique pas de relation de proximité.

L'objectif de la méthode par division/fusion est de rassembler, à partir de la division grossière obtenue par une première division, les différents blocs adjacents de l'image.

L'algorithme standard de division/fusion a été proposé par Horowitz et Pavlidis (Horowitz and Pavlidis, 1974). Le processus est décomposé en deux étapes. L'image initiale peut être une première partition résultant d'une analyse grossière ou bien l'image brute.

Dans la première étape, ou division, nous analysons individuellement chaque région X_i. Si celle-ci ne vérifie pas le critère d'homogénéité, alors nous divisons cette région en blocs (le plus généralement en 4 quadrants) et nous réitérons le processus sur chaque sous-région prise individuellement. Nous pouvons tout à fait initier le processus en considérant que la première région est composée de toute l'image.

Dans la deuxième étape, ou fusion, nous étudions tous les couples de régions voisines (X_i,X_j). Si l'union de ces deux régions vérifie le critère d'homogénéité choisi pour la fusion, alors, nous fusionnons les régions.

Le principal avantage de cette méthode par rapport à la méthode de croissance de régions est qu'aucun point de départ est nécessaire et par conséquent, aucune interaction manuelle n'est nécessaire. Une difficulté de cette approche réside dans le parcours de l'ensemble de tous les couples de régions voisines. Des évaluations pour l'extraction des principaux tissus d'images 3D cérébrales ont été proposées(Manousakas et al., 1998). Ces méthodes ont l'inconvénient d'être dépendant de la qualité

de la décomposition de l'image originale (par exemple en *quadtree*).

Approches guidées par un atlas

Ces approches utilisent un atlas standard ou un modèle pour effectuer la segmentation. L'atlas est généré par la compilation d'informations sur la partie qui requiert la segmentation. L'atlas standard traite la segmentation comme un problème d'enregistrement. L'algorithme commence par rechercher une transformation qui envoie une image d'atlas pré-segmenté à l'image cible à segmenter. Ce processus est souvent nommé une déformation d'atlas. La déformation peut être réalisée à l'aide de transformations linéaires.

Ces approches ont été appliquées principalement dans l'imagerie IRM de cerveau. Un avantage de ces approches est que les étiquettes de l'image sont transférées ainsi que la segmentation. Le principal défaut de ces approches est dû à la grande variabilité de l'anatomie de cerveau. Pour résoudre ce problème, de nombreux chercheurs ont tenté d'appliquer une séquence de transformations linéaires et non-linéaires. La segmentation de structures complexes reste très difficile. Thompson et Toga ont introduit un atlas probabiliste pour modéliser la variabilité anatomique, mais leur méthode exige plus de temps et d'interaction pour accumuler des données (Thompson and Arthur Toga, 1997). Par conséquent, ces approches sont mieux adaptées à la segmentation des structures qui sont stables dans la population d'étude.

2.2.3 Méthodes utilisant les SMA

Dans ce qui suit, nous allons exposer brièvement quelques travaux utilisant les SMA dans le domaine du traitement d'images et plus particulièrement dans le but de segmenter ces images. Il est à noter que la plupart de ces travaux sont effectués dans le cas de la 2D.

2.2.3.1 Approches points

Haroun et al. ont proposé une approche de segmentation des principaux tissus d'une image IRM du cerveau (Haroun et al., 2005). Les auteurs ont présenté une approche coopérative mettant en œuvre deux algorithmes de segmentation d'image : la classification FCM pour gérer l'incertitude et l'imprécision, et la croissance de régions pour agir localement sur l'image. Un système Multi-Agents est introduit dans la phase de croissance de régions afin d'améliorer la qualité de la segmentation. Il existe deux types d'agents, un agent de type *"Contrôleur*" et plusieurs agents de type *"Croissance"* déployés sur l'image. Le processus est répété plusieurs fois jusqu'à ce qu'il ne reste aucun pixel susceptible d'être membre d'une région. Cette méthode utilise un processus de classification floue pour séparer les 3 principaux tissus du reste du cerveau afin de les segmenter (figure 2.4). Malgré le fait que le nombre de régions est petit, leur système n'arrive pas à détecter tous les pixels de l'image.

Scherrer et al. ont considéré la segmentation des tissus et des structures (la matière grise, la matière blanche et le liquide céphalo-rachidien) par les champs de Markov sur des images 3D de scanner comme deux procédures locales et coopératives immergées dans un cadre Multi-Agents (Scherrer

(a) (b)

FIGURE 2.4: Segmentation extraite de (Haroun et al., 2005) : (a) image originale et (b) image segmentée en MB, MG et LCR.

et al., 2007). La segmentation des tissus est effectuée en partitionnant le volume en sous-volumes où les agents estiment les modèles locaux des champs de Markov en coopération avec leurs voisins pour assurer la cohérence des modèles locaux. Ces modèles reflètent mieux la distribution d'intensité locale. La segmentation des structures est réalisée via des agents dynamiquement localisés qui intègrent des contraintes anatomiques et spatiales fournies par une description floue à priori de l'anatomie du cerveau (figure 2.5). Cette segmentation ne se réduit pas à une étape de post-traitement : les agents des structures coopèrent avec les agents des tissus pour rendre les modèles progressivement plus précis. Le processus s'arrête lorsque le système converge. L'inconvénient de cette approche est la taille de la fenêtre du voisinage qui diffère d'une image à l'autre pour estimer l'appartenance de chaque voxel à chaque classe de structure (dans le cas de Scherrer et al., il s'agit d'un voisinage de 20x20x20 voxels).

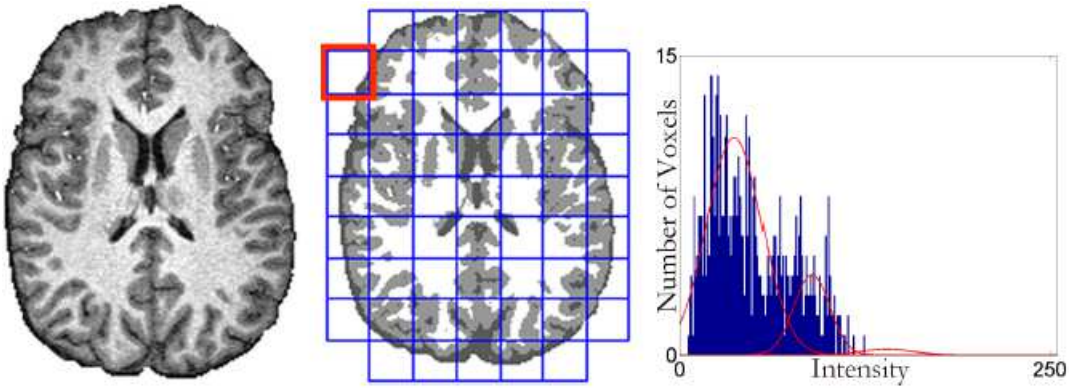
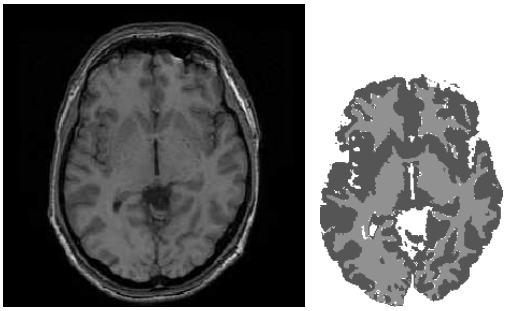


FIGURE 2.5: Segmentation extraite de (Scherrer et al., 2007) : (a) image originale, (b) image segmentée décomposée en blocs et (c) l'histogramme de la représentation des tissus dans le bloc rouge.

2.2.3.2 Approche contours

Pour segmenter des images 2D d'IRM du cerveau, Germond et al. ont proposé une méthode de segmentation coopérative basée sur la segmentation en régions de la MG et de la MB. Cette segmentation est effectuée par des agents spécialisés à partir d'un modèle déformable (McInerney

and Terzopoulos, 1996) du cerveau construit au préalable (Germond et al., 2000). À l'issue de la segmentation en régions de ces tissus, le contour du cerveau plus précis est obtenu par des agents contours qui utilisent la méthode des contours actifs (figure 2.6). A la fin de l'algorithme, un retour complet sur le processus de segmentation est considéré, initialisé par le contour reconstruit lors de la phase précédente. Ce processus est répété plusieurs fois jusqu'à satisfaction d'un critère d'arrêt.
L'inconvénient de cette approche est la propagation d'erreur induite par le passage d'information d'une coupe à l'autre.

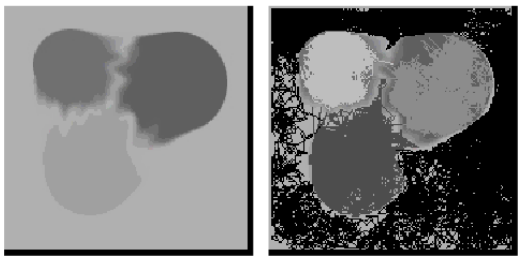


(a) (b)

FIGURE 2.6: Segmentation extraite de (Germond et al., 2000) : (a) image originale et (b) image segmentée en MB, MG et LCR et les contours les englobant.

2.2.3.3 Approche régions

Bourjot et al. ont développé un SMA basé sur les araignées sociales (Bourjot et al., 2003). Ils ont séparé la segmentation en deux parties : la segmentation d'une seule région et la segmentation de plusieurs régions. À l'étape initiale, les agents sont distribués aléatoirement sur l'image. Chaque agent commence à tisser des toiles selon une fonction de transition et l'objectif de la segmentation. Après un nombre d'itérations fixé par l'utilisateur, une analyse des toiles est faite pour extraire les régions de l'image (figure 2.7). L'inconvénient de cette méthode est la sélection empirique des informations couleurs de chaque colonie.

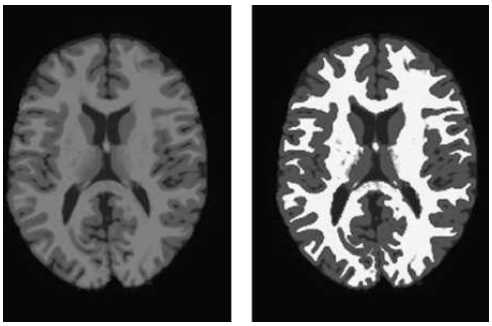


(a) (b)

FIGURE 2.7: Segmentation extraite de (Bourjot et al., 2003) : (a) image originale et (b) image segmentée en 4 régions.

Pour segmenter des images IRM de cerveau, Huang et al ont développé une approche de segmentation basés sur les colonies de fourmis(Huang et al., 2008; Cao et al., 2008). Leur approche considère

que chaque fourmi a la capacité de mémoriser un objet de référence qui sera mis à jour quand elle trouve une nouvelle cible. Une mesure floue de la connexité est adoptée pour évaluer la similarité entre la cible et l'objet de référence. Le comportement des fourmis est affecté par le voisinage direct et la coopération entre les fourmis est effectuée de façon empirique par le partage d'informations par le biais de la mise à jour de phéromones (figure 2.8). L'algorithme s'arrête après un nombre d'itérations fixé à l'avance par l'utilisateur. L'inconvénient de cette méthode est le nombre important de paramètres à déterminer (7 paramètres).

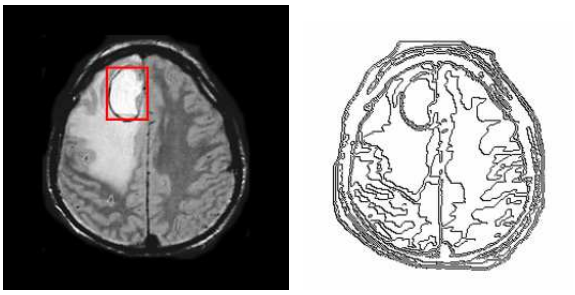


(a) (b)

FIGURE 2.8: Segmentation extraite de (Huang et al., 2008) : (a) image originale et (b) image segmentée en MB, MG et LCR.

Pour segmenter des images scanner CT du sein, Duchesnay et al. ont réalisé une plateforme Multi-Agents en plusieurs étapes (Duchesnay et al., 2003; Jacquelet et al., 2002; Duchesnay et al., 2001). Comme pré-classification, Duchesnay et al. utilisent la décomposition en « division/fusion » avec des agents de type *"région"*, ils obtiennent alors une segmentation qui ne donne que les zones homogènes dans l'image. Ils utilisent ensuite des agents de type *"contour"*, sans initialisation dont l'objectif est de détecter les hautes fréquences. Chacun de ces deux types d'agents partagent via l'environnement certaines caractéristiques liées à l'image. Le système interne est modélisé par un réseau de Pétri (Petri, 1975) et implémenté par sept règles. Un agent est ensuite placé dans chaque zone homogène et le processus de la segmentation basé sur les SMA commence. Les pyramides irrégulières sont adoptées comme élément organisationnel de la population d'agents ; la progression du processus se faisant de la base de la pyramide jusqu'au sommet. Ce processus s'arrête après un certain nombre d'itérations lorsqu'il y a convergence. Dans le même contexte, Benamrane et Nassane ont utilisé la plateforme JADE (Bellifemine et al., 2007) pour effectuer la segmentation d'images IRM de cerveau (figure 2.9) en changeant le système interne par la méthode de croissance de régions (Benamrane and Nassane, 2007). L'inconvénient de cette méthode repose sur le choix de l'initialisation des régions initiales.et aussi la complexité des règles d'interaction qui dépendent de l'image à segmenter.

Pour segmenter des images IRM de cerveau, Richard et al. ont proposé une architecture hiérarchique d'agents situés et coopératifs (Richard et al., 2004). Trois types d'agents ont été utilisés : un agent de contrôle global, un agent de contrôle local, et un agent dédié au tissu. Le rôle de l'agent de contrôle global est de partitionner le volume de données en territoires adjacents et d'affecter à



(a) (b)

FIGURE 2.9: Segmentation extraite de (Benamrane and Nassane, 2007) : (a) image originale et (b) extraction de contours par croissance de régions.

chaque territoire un agent de contrôle local. Le rôle de ce dernier est de créer les agents dédiés au tissu, dont l'objectif est d'effectuer une croissance de régions à l'intérieur du volume local selon des critères topologiques et colorimétriques. Les paramètres de distribution des données sont mis à jour par coopération entre les agents voisins. Par l'utilisation de différents types d'agents, le système proposé a pris en compte le traitement d'images de bas niveau, ainsi que le contrôle sur les tâches de segmentation (figure 2.10). Cependant, il résulte des structures complexes d'interaction dont la gestion est coûteuse. Dans le même contexte, ils proposent d'utiliser les champs de Markov cachés comme un critère de fusion (Richard et al., 2007). Cet algorithme présente l'inconvénient de dépendre largement du nombre de partitions qu'une image peut avoir. Ainsi, un mauvais choix à priori de ce nombre conduit inévitablement à un mauvais résultat.

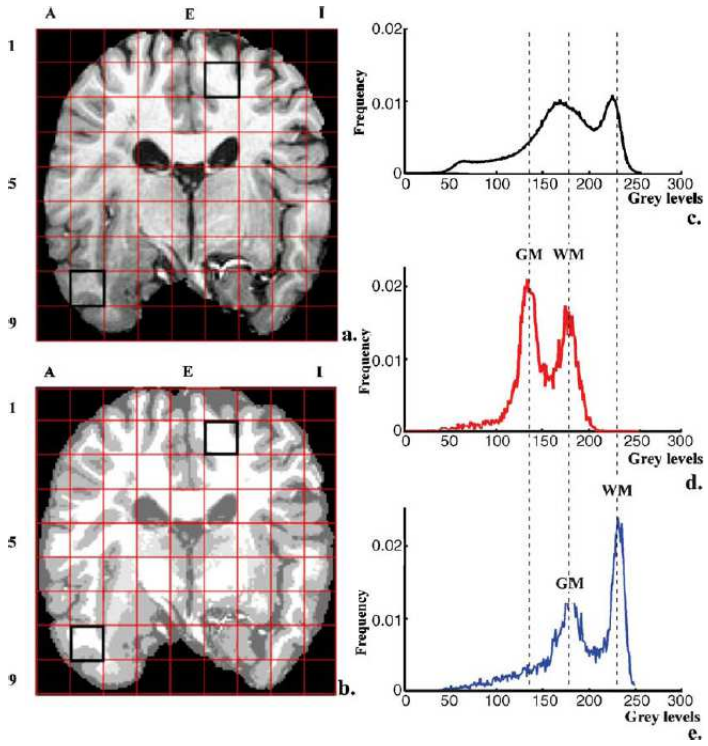


FIGURE 2.10: Segmentation extraite de (Richard et al., 2004) : (a) image originale décomposée en blocs, (b) image segmentée décomposée en blocs, (c) l'histogramme de l'image, (d) l'histogramme du bloc noir de l'image originale et (e) l'histogramme du bloc noir de l'image segmentée.

2.3 Conclusion

Ce chapitre a expliqué les différentes méthodes classiques de segmentation d'IRM basées points, contours ou régions et aussi certaines approches utilisant les SMA. Les différents points à retenir sont soit de posséder à priori une connaissance de l'image à segmenter soit la définition parfois fastidieuse, d'atlas, de référence, *etc*. Enfin, le bruit inhérent d'une part aux méthodes d'acquisition et d'autre part à la nature même de l'image : imagerie médicale sur des sujets vivants, est une élément important dans la recherche d'une segmentation de qualité. Dans le chapitre suivant, nous allons présenter notre plateforme de segmentation au moyen d'un SMA basé sur les agents sociaux avec pour objectif de réaliser des segmentations résistantes au bruit.

Deuxième partie

Modélisation et expérimentations

Chapitre 3

IPSiMAS

Nous venons de décrire dans la partie I les généralités sur les systèmes Multi-Agents et la segmentation d'images IRM cérébrales. Dans l'optique de pouvoir segmenter une image en régions et en contours (Moussa et al., 2009a), nous allons présenter les modèles utilisés dans notre plateforme **I**mage **P**rocessing **Si**mulated by **M**ulti-**A**gent **S**ystem (IPSiMAS). Ces modèles répondent à la problématique de la segmentation d'images bruitées. Ils utilisent des heuristiques sur le comportement d'agents sociaux, araignées et fourmis, tels que ceux vus au chapitre 1. Ce chapitre se terminera par une présentation du modèle Orienté Objet qui a été réalisé pour l'implémentation d'IPSiMAS.

3.1 Segmentation régions : les araignées

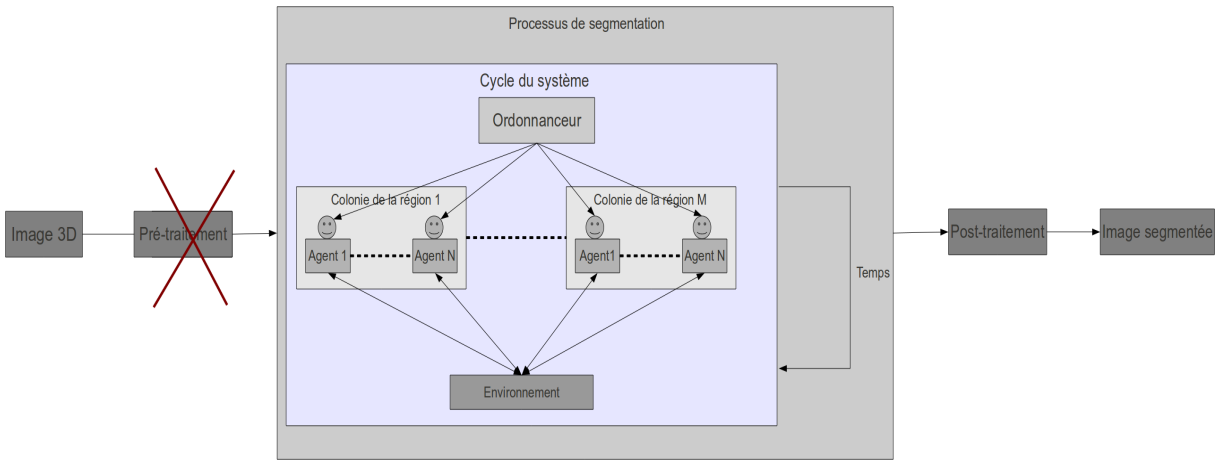


FIGURE 3.1: Chaine de segmentation en régions.

Le schéma 3.1 montre le principe de segmentation d'images avec des agents de types araignées suivant le principe d'une segmentation en régions tel que définit dans le chapitre précédent (Moussa

et al., 2008).

Les éléments principaux de ce modèle sont :

1. la définition d'un environnement pour les agents-araignées ;
2. la définition de colonie ;
3. le cycle de vie affecté à chaque agent-araignée ;
4. la description d'un ordonnanceur du système ;
5. la description d'éventuels post-traitements pour évaluer les résultats de segmentation.

Pour ce premier modèle de segmentation, nous n'utiliserons pas de pré-traitement de l'image. Les sections suivantes vont maintenant expliquer le modèle de segmentation d'images par les agents-araignées.

3.1.1 Création de l'environnement

À l'étape initiale, l'image 3D est chargée dans l'environnement. Chaque voxel est alors valué avec une quantité représentant son niveau de gris. C'est cette information qui sera utilisée par l'agent-araignée lors de l'étape de décision quant au choix du prochain voxel à taguer.

Au cours de la simulation, les voxels, *i.e.* l'environnement, contiendront également les informations relatives au tissage de la toile.

3.1.2 Définition des colonies

La segmentation d'images IRM suppose la détection de plusieurs niveaux de gris différents. Par exemple, dans le cas d'IRM cérébrales, on peut chercher à détecter 3 types de matières (LCR, MB, MG) caractérisées par un intervalle de niveaux de gris. Comme nous le verrons dans le chapitre suivant, des applications comme BrainWeb [1] propose des images IRM pour lesquels 10 régions, niveaux de gris, sont à détecter : LCR, MB, MG, crâne, fond, graisse, peau/muscle, peau, matière gliale, conjonction. Il est donc possible de définir de façon générale, une colonie d'agents-araignées comme une entité dont la tâche consiste à découvrir une région spécifique pour laquelle elle détient les caractéristiques de couleurs (niveaux de gris dans notre cas).

Dans un souci de meilleur prise en compte de l'image, ces intervalles de niveau de gris ne sont pas disjoints introduisant ainsi une compétition entre les colonies. À la fin du processus de segmentation, les voxels ayant été tagués par plusieurs colonies constituent une sorte de frontière des différents régions. Toutefois, il faut noter que ce multi-tag de voxels peut aussi être le reflet de différents artéfacts d'acquisition (bruit, mouvement du patient, *etc*).

3.1.3 Description du cycle de vie

La position sur l'image de chaque agent-araignée est initialisée aléatoirement. Ensuite, il exécutera son cycle de vie et ainsi partira à la recherche de voxels à tisser qui correspondent aux critères de

1. http://www.bic.mni.mcgill.ca/brainweb/

segmentation de sa propre colonie. La figure 3.2 reprend les étapes principale de ce cycle de vie tel qu'il a déjà été évoqué dans la section 1.2.2 pour les agents réactifs. Nous allons maintenant détailler chacune de ces étapes.

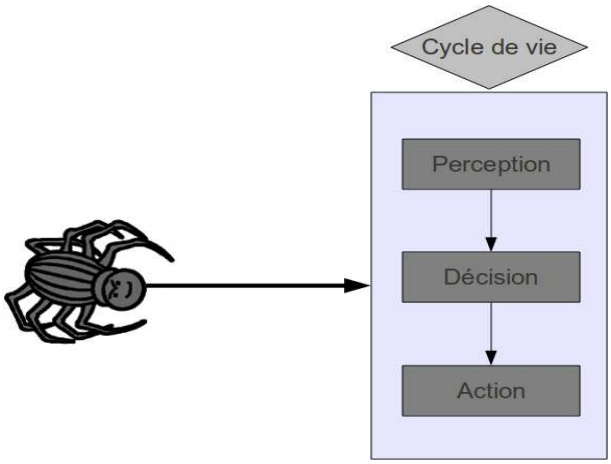


FIGURE 3.2: Cycle de vie des agents-araignées.

Perception

La perception vise principalement les candidats voxels potentiels. Lors de cette étape de recherche, une sélection de voisinage est effectuée, 3 catégories de voisinages sont distinguées :

1. au moins 1 voxel du voisinage n'est pas tissé ;
2. au moins 1 voxel du voisinage n'est pas saturé ;
3. l'ensemble du voisinage est saturé.

Décision

Dans le cas 1 : le niveau de gris du ou des voxels non tissés sont examinés. Si ce niveau de gris satisfait les contraintes d'intervalles données par la colonie, un fil est tissé, si plusieurs sont candidats un est tiré au sort. Dans le cas où aucun voxel ne remplit les conditions de couleur alors l'agent-araignée se déplacera aléatoirement sur un des voxels non tissés sans tisser de fil.

Dans le cas 2 : comme nous le détaillerons dans la section 3.1.4.2, par souci d'efficacité, un maximum de nombre de fil est défini pour chaque voxel. Quand ce maximum est atteint, le voxel est déclaré saturé. Lors du processus de décision de tissage, si tous les voxels du voisinage sont tissés, on considère alors comme candidats potentiels au tissage, les voxels non saturés. Une fonction de transition prenant en compte les caractéristiques des fils permet de choisir le voxel à tisser.

Dans le cas 3 : si le voisinage de l'agent-araignée est totalement saturé, il se déplace aléatoirement dans le voisinage, évidement sans tisser.

Action

Lorsque le choix de la nouvelle destination de l'agent-araignée est prise, cet agent applique la fonction de tissage et met à jour sa position dans l'image. Chaque agent garde en mémoire le précédent voxel

tissé afin de pouvoir éventuellement y retourner en cas de blocage. Tous les détails de ces procédures seront donnés dans la section 3.1.6.

Un ordonnanceur définit si les différents cycles de vie des agents sont exécutés en parallèle ou en séquence. Nous avons réalisé les deux implémentations. Tous ces points sont développés dans la section 3.3.

L'algorithme 3 présente le processus de segmentation.

Algorithme 3 Segmentation par les araignées

Entrées: Image : Matrice de Voxels $\in \mathbb{N}^3$, Cond : Condition d'arrêt, NbRégions $\in \mathbb{N}$, NbAgent $\in \mathbb{N}$, et Conf: paramètres de configuration

```
1: Extraire les NbRégions informations couleurs de Image.
2: A ← Placer les NbAgent dans Image et les répartir dans NbRégions colonies.
3: t ← 0.
4: TantQue Cond != Vrai Faire
5:    t ← t+1 ;
6:    Ordonnancer(A) ;
7:    Pour Chaque Colonie C_i Faire
8:       Pour Chaque Araignée A_ij Faire
9:          Perception(t, A_ij).
10:         Décision(t, A_ij, Conf).
11:         Action(t, A_ij, Image).
12:      Fin Pour
13:   Fin Pour
14: Fin TantQue
15: ConstruireImage(Image).
```

La complexité de cet algorithme est de $O(C_{lissage} + C_{InitAgent} + C_{Cond} \times (C_{Ordonnancer} + NbAgent \times C_{Cycle}) + C_{ConstruireImage})$ où :

1. $C_{lissage}$ est la complexité de la sélection automatique des informations couleurs pour les colonies ;
2. $C_{InitAgent}$ est la complexité de l'initialisation des agents-araignées ;
3. C_{Cond} est la complexité de la condition à satisfaire qui est liée au nombre de pas de temps ;
4. $C_{Ordonnancer}$ est la complexité de réorganiser les agents ;
5. $NbAgent$ le nombre d'agents placé dans l'environnement ;
6. C_{Cycle} la complexité du cycle de vie de chaque agent-araignée qui est à son tour dépendant du nombre de candidats potentiels ;
7. $C_{ConstruireImage}$ la complexité de la construction de l'image segmentée

3.1.4 Identification des paramètres

La simulation de la méthode des araignées est basée sur la stigmergie: chaque agent-araignée dépose des informations sur l'environnement, qui sont ensuite utilisées par d'autres agents-araignées ou par lui-même dans un prochain pas de temps système. L'image segmentée émerge alors de la toile tissée par tous les agents-araignées.

3.1.4.1 Des problèmes

La méthode des araignées possède l'avantage d'être une méthode reposant sur des mécanismes simples facilitant sa compréhension et son implémentation. Toute méthode de segmentation suppose un certain nombre de paramètres en entrée : colorimétrie, critère de choix, *etc*. Il est important de pouvoir définir une procédure automatique ou semi-automatique de valuation de ces paramètres, ceci afin de permettre la comparaison de la qualité des résultats obtenus pour différentes images. En minimisant le nombre de ces paramètres, la tâche s'en trouve d'autant simplifiée. Le deuxième point crucial des simulations SMA est la détermination soit du nombre de pas de temps soit du critère d'arrêt des simulations. Trop fréquemment, de premiers résultats sont invalidés par d'autres simulations dont on a juste augmenté le temps. Les différentes solutions à ces problèmes vont être maintenant présentées.

3.1.4.2 Des solutions

Tout d'abord, nous définissons l'ensemble des paramètres qui doivent être fixés et leur utilisation dans le programme. Ensuite, nous proposons des pistes pour réduire leur nombre. Enfin, nous exposerons les solutions retenues. Le tableau 3.1 décrit cinq des paramètres (paramètres de `Config`) utilisés dans l'algorithme 3 dans sa version initiale.

Paramètres	**Description**
Saturation	Le nombre maximal de fils autorisés à être présents dans un voxel
Couleur	La couleur de référence
Sélectivité	L'intervalle pour considérer qu'un voxel est "intéressant"
$Attraction_{MaColonie}$	La probabilité d'être attirée par les fils de la même colonie
$Attraction_{AutresColonies}$	La probabilité d'être attirée par les fils des autres colonies

TABLE 3.1: Les paramètres de chaque colonie.

Nous avons introduit dans la section précédente le concept de voxel saturé. Le choix du paramètre `Saturation` qui fixe le nombre maximal de nombre de fils acceptés par un voxel joue un rôle primordial dans la convergence du système. Ainsi, une valeur de `Saturation` très élevée nécessitera un nombre important de pas de simulation pour atteindre la convergence.

Il existe aussi un lien quantitatif entre le nombre de colonies et la valeur de `Saturation`, plus le nombre de colonies est grand plus il est nécessaire d'avoir un nombre de fils maximum grand.
N.B : une valeur de `Saturation` à 1 reflète un processus de segmentation équivalant à une croissance de région dynamique.

Nous avons vu que le nombre de colonies était lié au nombre de régions à détecter. Les deux paramètres `Couleur` et `Sélectivité` définissent respectivement pour chaque colonie la valeur

moyenne de la couleur définissant sa région et la plage de couleur acceptée autour de cette couleur de référence. Le choix des deux paramètres d'attractions ($Attraction_{MaColonie}$ et $Attraction_{AutresColonies}$) sera discuté par la suite.

Dans les paragraphes suivants, nous allons présenter les méthodes semi-automatiques de détermination des 3 premiers paramètres du tableau 3.1. Ces méthodes seront utilisées par la suite dans tous les processus de segmentation.

Détection semi-automatique des paramètres

Rappelons que chaque colonie a un niveau de gris (ou intensité) de référence qui sera utilisé par les agents-araignées de la colonie pour déterminer si elles doivent fixer ou non un fil entre deux voxels. Dans ce cas, il existe deux possibilités :

1. choisir "manuellement" les intervalles d'intensité, ceci peut se révéler fastidieux et de plus il faut s'attendre à des variations importantes dans ces choix en fonction des différents utilisateurs ;
2. utiliser une méthode pour déterminer automatiquement ou semi-automatiquement ces intervalles.

Une détection semi-automatique va nous permettre d'obtenir des paramètres optimaux sans qu'aucune connaissance de l'utilisateur sur l'image à segmenter ne soit nécessaire. Pour cela, nous allons utiliser l'histogramme de l'image. En effet, une région dans une image implique généralement un pic plus ou moins important dans son histogramme. Toutefois, nous pouvons noter que cet histogramme est particulièrement sensible au bruit. Par conséquent, nous allons introduire une méthode pour réduire les effets du bruit sur l'histogramme en le lissant et en permettant d'obtenir un nombre de représentants égal au nombre de régions à détecter.

Détection des maxima et minima locaux

Le lissage de l'histogramme permet de réduire les pics provoqués par le bruit. La méthode proposée ici est utilisée à chaque pas de temps. Pour chaque intensité i, on calcule le degré moyen n_i des degrés n des intensités [i-1, i+1]. Ce calcul est itéré jusqu'à l'obtention du nombre de pics (régions) attendu. Les valeurs obtenues sont des maxima locaux qui respectent la définition 11 avec $Seuil_1$ la taille minimale acceptée en nombre de voxels, pour un pic significatif.

Définition 11 *M_i est un maximum local s'il répond aux inégalités suivantes : $n_i > n_{i-1}$, $n_i > n_{i+1}$ et $n_i > Seuil_1$.*

Ce lissage de l'histogramme permet de détecter dynamiquement les réglages optimaux pour la segmentation de l'image. On notera que le risque d'effacer un maxima intéressant n'est pas nul.

L'algorithme 4 présente l'essentiel d'extraction d'un nombre de maxima locaux d'un histogramme.

À partir des maxima locaux, nous allons pouvoir calculer la moyenne de référence et l'intervalle de confiance ou sélectivité pour chaque colonie. On considère que les valeurs issues de ce calcul des

Algorithme 4 Détection des maxima locaux

Entrées: H: Histogramme de l'image ∈ ℕ, NbRégions ∈ ℕ, $Seuil_1 \in \mathbb{N}$ (nécessaire pour la définition 11)
Sorties: L: Liste de maxima ∈ ℕ

```
1:  L ← ListeVide().
2:  TantQue Taille(L) != NbRégions Faire
3:      L ← ListeVide().
4:      Pour i allant de 1 à Taille (H) Faire
5:          Si  H[i] répond à la définition 11 Alors
6:              L ← Ajouter(i).
7:          FinSi
8:      Fin Pour
9:      MiseAJour(H).
10: Fin TantQue
```

maxima locaux suit une loi normale N(m_i, σ_i), où m_i représente la coloration moyenne à détecter et σ_i l'écart type de la distribution de couleurs potentielles. En pratique, des effets de "surlissage" peuvent apparaître, en général causés par le bruit présent dans toute image "réelle". En effet, dans les images bruitées, une région ne sera pas représentée par un seul maximum mais plutôt par une ensemble de valeurs proches. Dans ce cas le maximum finalement retenu peut être décalé sur l'histogramme. Pour corriger cela, on calcule des minima locaux m_i entre chaque maximum local M_i suivant la définition 12 où le $Seuil_2$ est la valeur minimale de l'effectif des voxels de chaque minimum.

Définition 12 *m_i est un minimum local s'il répond aux inégalités suivantes : $n_i < n_{i-1}$, $n_i < n_{i+1}$ et $n_i > Seuil_2$.*

L'algorithme 5 calcul le minimum local entre deux maxima locaux voisins :

Algorithme 5 Détection d'un minimum local

Entrées: H: Histogramme de l'image ∈ ℕ, $IndexM_1 \in \mathbb{N}$, $IndexM_2 \in \mathbb{N}$, $Seuil_2 \in \mathbb{N}$ (nécessaire pour la définition 12)
Sorties: l: minima ∈ ℕ

```
1:  l ← ListeVide().
2:  TantQue Taille(l) != 1 Faire
3:      l ← ListeVide().
4:      Pour i allant de IndexM1 à IndexM2 Faire
5:          Si H[i] répond à la définition 12 Alors
6:              l ← Ajouter(i).
7:          FinSi
8:      Fin Pour
9:      MiseAJour(H).
10: Fin TantQue
```

La complexité des algorithmes 4 et 5 est de l'ordre $O(NbGris \times NbLissage)$ où :

1. *NbGris* reflète la plage des valeurs de niveaux de gris de l'intervalle de recherche ;
2. *NbLissage* décrivant le nombre de lissages effectués avant l'obtention de la solution optimale.

La figure 3.3 résume les étapes de la recherche des paramètres des colonies pour une image composée de 10 régions. La figure 3.3(a) est l'histogramme de l'image avec la distribution des intensités. La figure 3.3(b) est l'histogramme lissé où les pics en rouges représentent les maxima locaux et les pics en vert sont les minima locaux. La figure 3.3(c) représente la fonction de répartition des différentes couleurs de référence pour chaque colonie. Les chevauchements des courbes observées représentent la compétition entre des colonies pouvant éventuellement taguer le même voxel.

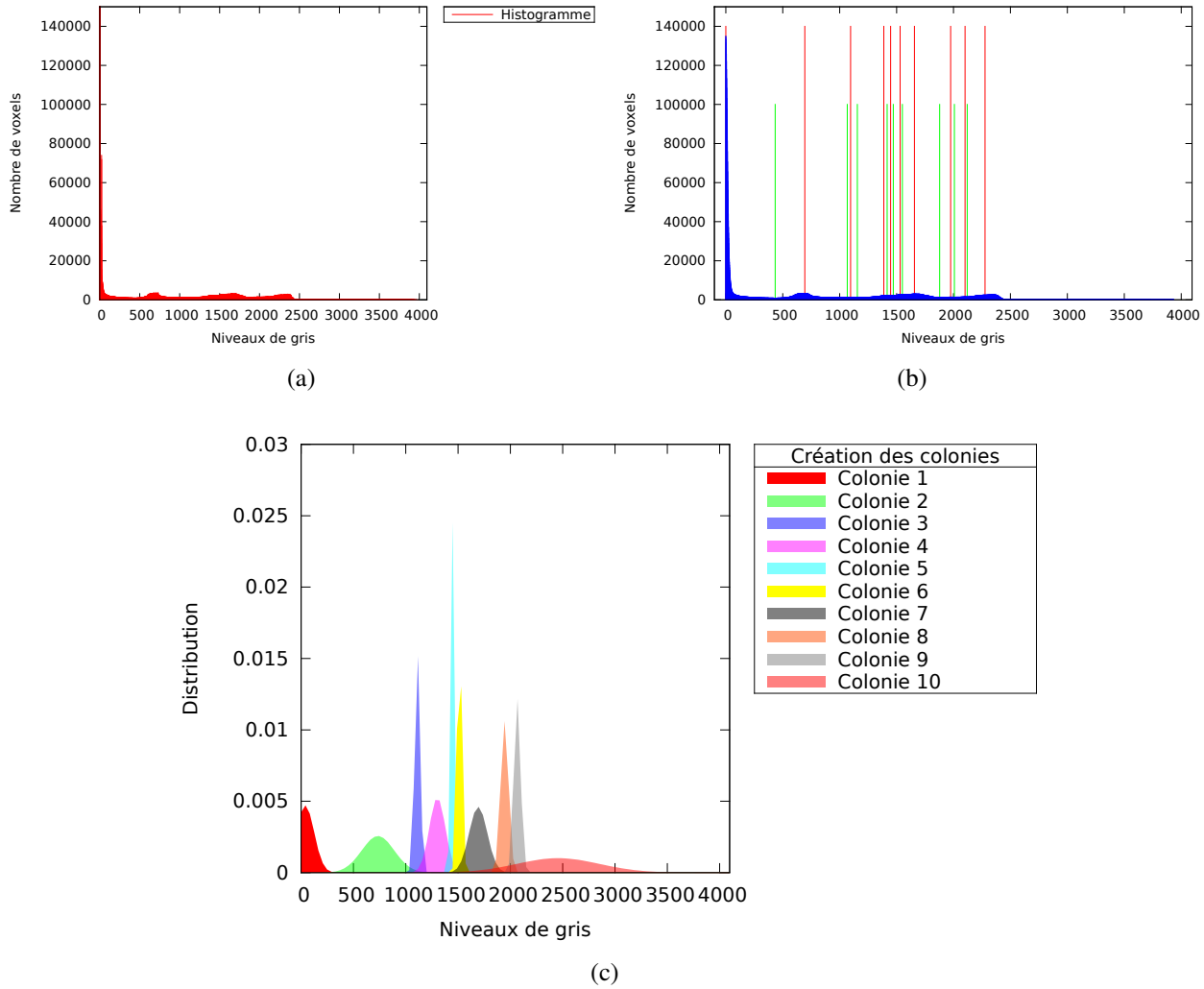


FIGURE 3.3: Sélection des informations colorimétries pour les colonies d'araignées : (a) Histogramme de l'image, (b) Histogramme lissé et sélection des maxima et minima locaux, (c) intervalles de recherche pour les différentes colonies.

Après la détermination des 3 premiers paramètres du tableau 3.1, il reste à déterminer l'attraction des fils pour les colonies. Les paramètres $Attraction_{MaColonie}$ et $Attraction_{AutresColonies}$ sont des probabilités dont la somme est égale à 1. Plus la valeur $Attraction_{AutresColonies}$ est supérieure à la valeur $Attraction_{MaColonie}$ plus on autorise une colonie à taguer des voxels ayant été attirés par d'autres colonies. Ceci s'avère intéressant dans le cas où l'on veut corriger la détection des voxels bruités.

Condition d'arrêt

Nous avons vu dans la figure 3.2 que la méthode des araignées possède un cycle de vie qui est répété

plusieurs fois jusqu'à ce que l'image soit segmentée. Le nombre de pas de temps de la simulation va influencer le résultat de la segmentation sur deux points importants :

1. la qualité de la segmentation ;
2. le temps d'exécution nécessaire pour atteindre ce résultat.

À chaque pas de temps, les agents-araignées tissent des fils entre les voxels, ces fils étant typés par la colonie de l'araignée, les informations relatives à ces fils seront utilisées pour déterminer à quelle région appartient chaque voxel. Si le nombre de pas de temps n'est pas suffisant, le nombre de voxels non tagués sera important et le résultat sera de mauvaise qualité. Au contraire, si le nombre de pas de temps est très grand, les agents-araignées ne feront qu'augmenter, à partir d'un certain temps, les fils déjà existant, sans fournir aucune nouvelle information. Ce dernier point aura comme effet un temps d'exécution long pour un résultat de qualité approximativement identique. Une solution évidente consiste à définir un critère d'arrêt de la simulation.

L'objectif de saturer ou même de taguer tous les voxels n'est pas atteignable avec une méthode comme celle des araignées. On définira donc un paramètre β dont la valeur est le nombre de nouveaux fils tissés pour un pas de temps. On ne prendra en compte que les fils tissés entre deux voxels de degré 0 *i.e.* non encore tagués. Lorsque β tend vers 0, nous pouvons considérer que le système se stabilise et les agents-araignées ne font que renforcer les fils existants. Il est possible de détecter quand β reste à 0. Ce moment détermine la possibilité d'arrêter la simulation. Nous pouvons améliorer cette condition en ajoutant deux nouveaux paramètres :

1. un seuil qui détermine quand β peut être considéré comme invalide ;
2. le nombre de fois où on autorise β à valoir consécutivement zéro avant d'arrêter la simulation.

Le seuil peut être déterminé par le nombre d'agent-araignées. En effet, à chaque pas de temps, chaque agent-araignée a la possibilité de fixer un fil entre les voxels. Le nombre maximum de fils tissés durant un pas de temps système est limité par le nombre d'agent-araignées.

3.1.5 Instanciation de l'environnement

Comme nous l'avons vu au chapitre 1, les agents ont besoin d'un environnement pour exécuter leur cycle de vie. Dans notre modèle, l'environnement est construit à partir d'un maillage cubique régulier stocké dans une matrice de niveaux de gris. Chaque voxel est une position potentielle pour un agent-araignée et permettra à ce dernier d'atteindre d'autres voxels grâce à un voisinage bien défini. On trouve dans la littérature la description de trois types de connexité entre des voisins :

Définition 13 *Deux voxels v_1 et v_2 sont dits 6-connexes si ils satisfont l'équation suivante :*

$$|x_{v_1} - x_{v_2}| + |y_{v_1} - y_{v_2}| + |z_{v_1} - z_{v_2}| = 1 \tag{3.1}$$

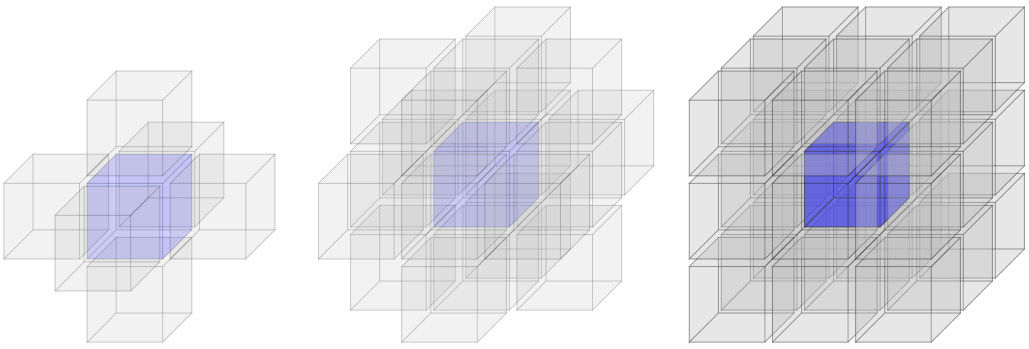
Définition 14 *Deux voxels v_1 et v_2 sont dits 18-connexes si ils satisfont l'équation suivante :*

$$|x_{v_1} - x_{v_2}| + |y_{v_1} - y_{v_2}| + |z_{v_1} - z_{v_2}| \leq 2 \text{ et } max(|x_{v_1} - x_{v_2}|,\ |y_{v_1} - y_{v_2}|,\ |z_{v_1} - z_{v_2}|) = 1 \tag{3.2}$$

Définition 15 *Deux voxels v_1 et v_2 sont dits 26-connexes si ils satisfont l'équation suivante :*

$$|x_{v_1} - x_{v_2}| + |y_{v_1} - y_{v_2}| + |z_{v_1} - z_{v_2}| \leq 3 \text{ et } max(|x_{v_1} - x_{v_2}|,\ |y_{v_1} - y_{v_2}|,\ |z_{v_1} - z_{v_2}|) = 1 \qquad (3.3)$$

La figure 3.4 montre le voisinage du voxel central (en bleu) pour la 6-connexité (a), la 18-connexité (b) et la 26-connexité (c).



(a) (b) (c)

FIGURE 3.4: Le voisinage d'un voxel : (a) 6-voisins, (b) 18-voisins et (c) 26-voisins.

Dans le modèle des araignées, pour une image 3D, nous avons retenu un voisinage en 26-connexité. Nous distinguons ensuite deux sortes de voisinage :

1. le $Voisinage_{direct}$: voisinage 26-connexe composé d'un ensemble statique des voxels ;
2. le $Voisinage_{tissé}$: un ensemble dynamique de voxels pouvant s'agrandir au fur et à mesure que les agents-araignées tisseront des fils entres les voxels. Ainsi tous voxels atteignables grâce à un fil seront considérés comme étant à une distance de 1 et ce quelque soit le nombre de voxels parcourus par le fil.

L'union de ces deux ensembles sera noté $Voisinage_{total}$. La figure 3.1.5 montre les différents voisinages du voxel sur lequel est positionné l'agent-araignée (voxel bleu). Le voisinage 26-connexe correspond à l'ensemble des voxels transparents comme expliqué précédemment. Ce voisinage est donc constitué des extrémités des fils tissés avec le voxel courant. Cet ensemble évolue dynamiquement au cours de la simulation, il est vide à l'instant *zéro*, et sera augmenté au fur et à mesure par le tissage des agent-araignées à chaque pas de temps.

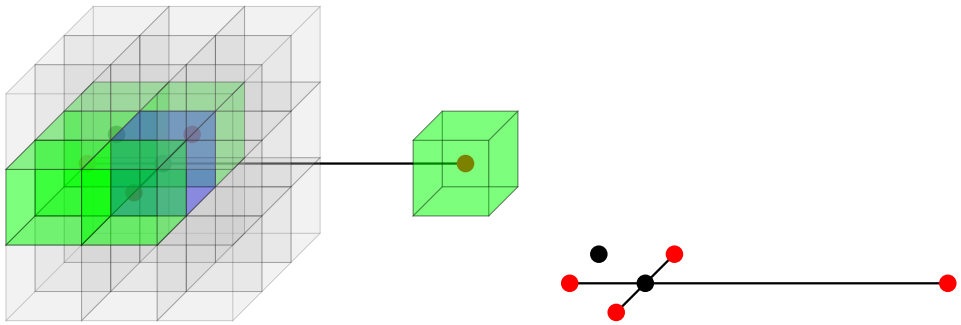
La figure 3.1.5 montre le graphe du voisinage tissé qui évoluera dans le temps. on peut remarquer qu'un voxel peut appartenir aux 2 voisinages 26-connexes et tissés (cas du noeud rouge).

Désormais nous pouvons considérer le voisinage total comme un graphe dynamique suivant la définition de Harary et Gopta (Harary and Gupta, 1997) :

Définition 16 *Un réseau (ou graphe entièrement pondéré) est un 4-uplet (S, A, F_1, F_2) où S est un ensemble de sommets, A est un ensemble d' arêtes entre S_i et S_j où F_1 est une fonction $F_1 : S \rightarrow N_1$, où N_1 est une certaine valeur numérique, par exemple, une quantité et/ou un poids, et F_2 est également une fonction, $F_2 : A \rightarrow N_2$, où N_2 est une autre valeur numérique, représenté aussi par une quantité et/ou un poids.*

Selon cette définition, on distingue 4 types de graphes dynamiques suivant l'élément du 4-uplets qui évolue dans le temps :

1. un graphe nœud-dynamique : l'ensemble S varie avec le temps. Ainsi, certains nœuds peuvent être ajoutés ou supprimés : ajout d'un nœud au voisinage tissé. Lorsque les nœuds sont supprimés, les arêtes incidentes avec eux sont également supprimées ;
2. un graphe arête-dynamique : l'ensemble A varie avec le temps. Ainsi, les arêtes peuvent être ajoutées ou supprimées à partir du graphe : ajout d'un fil ;
3. un graphe nœud-pondéré-dynamique : la fonction F_1 varie avec le temps. Ainsi, le poids et/ou la quantité sur les nœuds aussi change : le nombre de fils pour un voxel donné varie ;
4. un graphe arête-pondéré-dynamique : la fonction F_2 varie avec le temps. Ainsi, le poids et/ou la quantité sur les arêtes aussi change : le nombre de fils pour un couple de voxels varie.



(a) (b)

FIGURE 3.5: Voisinage d'un voxel : (a) représentation de l'environnement, (b) représentation des fils, les voxels tissés représentent les nœuds en rouge.

Ainsi dans notre modèle, le graphe dynamique construit satisfait les quatre propriétés citées ci-dessus puisque nous avons une évaluation dans le temps du tissage des fils. Le poids des nœuds sera utilisé dans le processus de décision pour déterminer si un nœud est saturé ou non. À chaque pas de temps, on testera si ce poids est inférieur ou non au paramètre de **Saturation** décrit dans le tableau 3.1. Les arêtes/fils portent l'identité de la colonie ayant tissée le fil. Pour chaque arête, on aura un compteur par colonie. La décision pour un agent-araignée de recruter ou non un voxel dans son voisinage tissé utilisera cette information.

3.1.6 Processus de segmentation

Comme nous l'avons vu dans la section 3.1.3, un agent-araignée possède un état interne constitué de sa position courante et de la dernière position où elle a tissé. Les agents-araignées appartenant à une même colonie partagent le même ensemble de valeurs des paramètres du tableau 3.1. Ainsi, tous les agents-araignées de la même colonie ont pour tâche de détecter la même région (Moussa et al., 2009b). Les fils tissés sont alors leur média de communication. Un cycle de vie d'agent, comme le montre la figure 3.2, consiste à exécuter chaque comportement selon une probabilité qui dépend des

valeurs des paramètres et des caractéristiques environnementales liées à la position de l'agent-araignée. Nous allons maintenant décrire précisément le cycle de vie des agents-araignées tel que nous l'avons défini pour la segmentation d'images IRM 3D.

Perception

À cette étape du cycle de vie, un agent-araignée perçoit son voisinage total, soit une cible de voxels candidats potentiels à la sélection, comme nouvelle destination. Le choix du meilleur candidat se fait suivant la procédure suivante :

1. exploration du voisinage 26-connexe. Tous les voxels non-tissés de ce voisinage sont alors candidats équiprobable au choix de la meilleure destination ;
2. si tous les voxels du voisinage 26-connexe sont tissés, on considère alors le voisinage tissé (quelle que soit la colonie qui a tissée). On exclut uniquement la dernière position tissée et les voxels saturés de cette liste. On affecte à chaque candidat une probabilité d'être choisi correspondant aux nombres de fils déjà présents.

Cas particulier : Après les deux étapes, la liste de candidats est toujours vide. Ceci se produit si le voisinage total est saturé. Dans ce cas, les voxels de ce voisinage sont considérés comme candidats potentiels avec une équiprobabilité et le déplacement de l'agent-araignée se fera sans tissage.

Cet algorithme permet de considérer des voxels à différentes distances de la position actuelle de l'agent-araignée et grâce à l'étape 1 favorise l'exploration des voxels non encore tissés.

L'algorithme 6 présente la sélection d'un ensemble de voxels candidats potentiels.

La complexité de cet algorithme est de l'ordre de $O(Voisinage_{total})$. Dans cet algorithme, nous communiquons à la procédure de décision non seulement la liste des candidats mais aussi leur type : Non tissé (NT), tissé (T) ou saturé (S).

Décision

Après la construction de la liste des candidats potentiels, chaque agent-araignée procède à une évaluation de ces candidats. Si le type de l'ensemble appartient à la famille **"S"**, aucun processus décisionnel n'est nécessaire (tous les voxels ont la même probabilité d'être choisi), une sélection aléatoire d'un voxel de la liste est effectuée. Dans le cas où la liste est composée de voxels **NT**, une première sélection sera faite en suivant l'information colorimétrique de la colonie à laquelle appartient l'agent-araignée.

La décision de choisir un voxel plutôt qu'un autre dans la liste des candidats est conditionnée par l'histogramme de l'image (figure 3.3). Pour chaque colonie k, nous connaissons la moyenne m_k de la couleur recherchée et l'écart-type σ_k de cette moyenne. Nous avons choisi de prendre un intervalle de confiance de 99% dans ce processus de sélection, soit 3 écart-type. Pour rester candidat à la future destination, un voxel devra donc suivre la définition suivante :

Définition 17 *Un voxel V_j est considéré comme un voxel de tissage si :*

$$m_k - 3 \times \sigma_k \leq Intensité(V_j) \leq m_k + 3 \times \sigma_k \tag{3.4}$$

Algorithme 6 La sélection des candidats potentiels pour une araignée

Entrées: Image : Matrice $\in \mathbb{N}^3$.
Sorties: V : liste de voisins $\in \mathbb{N}$, TypeListe $\in$ [S, T, NT].

$V_{Tissés} \leftarrow$ ListeVide().
$V_{Saturés} \leftarrow$ ListeVide().
$V_{NonTissés} \leftarrow$ ListeVide().
Type $\leftarrow$ NULL.
V $\leftarrow$ ListeVoisinsTotal(Position(A), Image).
Pour Chaque voisin V_i de V **Faire**
  Si (V_i != Position(A)) et (V_i != Position(DernierFil(A))) **Alors**
    Si Tissé((V_i) == Faux **Alors**
      $V_{Tissés} \leftarrow$ Ajouter(V_i).
    FinSi
    Si EstSaturé(V_i) == Faux **Alors**
      $V_{Saturés} \leftarrow$ Ajouter(V_i).
    Sinon
      $V_{NonTissés} \leftarrow$ Ajouter(V_i).
    FinSi
  FinSi
Fin Pour
Si Taille($V_{Tissés}$) !=0 **Alors**
  V $\leftarrow V_{tempo1}$.
  TypeListe $\leftarrow$ NT.
Sinon Si Taille($V_{Saturés}$) !=0 **Alors**
  V $\leftarrow V_{Saturés}$.
  TypeListe $\leftarrow$ T.
Sinon
  V $\leftarrow V_{NonTissés}$.
  TypeListe $\leftarrow$ S.
FinSi

Dans le dernier cas où le voisinage est constitué de fils tissés et non saturés, une fonction de poids est alors définie pour chaque voxel voisin afin d'évaluer la probabilité d'être sélectionné. Ainsi, pour chaque voxel V_j dans le voisinage de V_i, la probabilité de se déplacer vers V_j est :

$$P(Déplacement(V_j)) = \frac{Poids(V_j)}{\sum_{V_k \in Voisinage_{total}(V_i)} Poids(V_k)} \tag{3.5}$$

Pour calculer Poids(V_j), nous distinguons les fils tissés par la colonie de l'araignée de ceux tissés par les autres colonies. Deux paramètres de simulation entrent en jeu, $Attraction_{MaColonie}$ et $Attraction_{AutresColonies}$ pour calculer ce poids. Ces paramètres sont respectivement la probabilité de tisser un fil par sa colonie et la probabilité de tisser un fil des autres colonies. Ces deux probabilités sont choisies empiriquement et doivent satisfaire la condition suivante:

$$Attraction_{MaColonie} + Attraction_{AutresColonies} = 1. \tag{3.6}$$

Le calcul du poids de chaque voxel devient alors :

$$\begin{aligned} Poids(MaColonie) &= Attraction_{MaColonie} \times F_0(MaColonie) \\ Poids(AutresColonies) &= Attraction_{AutresColonies} \times F_0(AutresColonies) \\ Poids(V_j) &= Poids(MaColonie) + Poids(AutresColonies) \end{aligned} \tag{3.7}$$

Où $F_0(.)$ représente un critère dépendant du nombre de fils présents sur le voxel en cours d'analyse. Il peut être simplement le nombre de fils, c'est la procédure que nous avons retenue ; mais on pourrait choisir de prendre en compte une variation locale des intensités entre la position courante de l'agent-araignée et le voxel candidat. Il serait aussi possible de combiner ces deux informations pour pondérer l'attraction entre les voxels.

L'algorithme 7 décrit le processus de sélection de la nouvelle position pour un agent-araignée, il présente les cas explorés ci-dessus.

Algorithme 7 La décision du tissage d'un voisin pour une araignée

Entrées: Image : Matrice $\in \mathbb{N}^3$, V : liste de voisins $\in \mathbb{N}$, TypeListe $\in$ [S, T, NT].
Sorties: NouvellePosition $\in \mathbb{N}^3$, Tissage : booléen $\in$ [Vrai, Faux].

Tissage ← Faux.
V_{tempo} ← ListeVide().
$PoidsMax \leftarrow 0$.
$Index_{PoidsMAx} \leftarrow 0$.
Si TypeListe == S **Alors**
  NouvellePosition ← ChoixAléatoire(V).
Sinon Si TypeListe == NT **Alors**
  Pour Chaque voisin V_i de V **Faire**
    Si V_i satistfait l'information couleur **Alors**
      $V_{tempo} \leftarrow V_i$.
    FinSi
  Fin Pour
  Si Taille(V_{tempo}) != 0 **Alors**
    NouvellePosition ← ChoixAléatoire(V_{tempo}).
    Tissage ← Vrai.
  Sinon
    NouvellePosition ← ChoixAléatoire(V).
  FinSi
Sinon
  Pour Chaque voisin V_i de V **Faire**
    Si W(V_i) > PoidsMax **Alors**
      PoidsMax ← W(V_i).
      $Index_{PoidsMax}$ ← i ;
    FinSi
  Fin Pour
  NouvellePosition ← $V_{Index_{PoidsMax}}$
  Si NouvellePosition satisfait l'information couleur **Alors**
    Tissage ← Vrai.
  FinSi
FinSi

La complexité de cet algorithme est de l'ordre de $O(\times Voisinage_{total})$. À la fin de cet algorithme, l'agent-araignée connait sa prochaine destination et si un fil doit être tissé ou non.

Action

Une fois que le voxel de destination est choisi, l'agent-araignée va mettre à jour l'environnement, en déposant l'information fil, s'il y en a, entre sa position actuelle et sa nouvelle position. Deux cas se présentent : soit le voxel de destination fera l'objet d'un tissage avec la position courante, soit il recevra l'agent-araignée sans création d'un nouveau fil.

Lorsqu'un tissage doit être effectué, soit il suffit d'incrémenter le compteur de la colonie de l'agent, soit il est nécessaire de créer le compteur et d'ajouter la position courante et la nouvelle position pour définir le fil.

La procédure action se termine par le déplacement de l'agent-araignée et la mise à jour de sa position courante. L'algorithme 8 présente l'action de l'agent-araignée sur l'environnement.

Algorithme 8 Le tissage d'un fil par une araignée

Entrées: Image : Matrice $\in \mathbb{N}^3$, NouvellePosition $\in \mathbb{N}^3$ et Tissage : booléen $\in$ [Vrai, Faux].

1: **Si** Tissage == Vrai **Alors**
2: Tisser(Position(A), NouvellePosition).
3: MettreAJour(Tissé(Image), Saturé(Image))
4: Position(DernierFil(A)) ← Position(A).
5: **FinSi**
6: Position(A) ← NouvellePosition.

La complexité de cet algorithme est de l'ordre de O(RechercherFils(Position(A), NouvellePosition)), où RechercherFils(Position(A), NouvellePosition) recherche si il existe déjà un fil entre Position(A) et NouvellePosition.

3.1.7 Construction de l'image segmentée

Pour extraire le résultat de la segmentation, nous devons analyser la toile tissée par les araignées. Après exécution de tous les pas de temps, chaque voxel est porteur de 0 à N fils. Le degré d'un voxel définit le nombre de fils entrant et sortant de ce voxel. Comme nous avons autant de nombre de colonies que de régions à détecter, ce degré est fourni en fonction des colonies.

Nous pouvons donc déterminer pour chaque voxel la colonie dominante, c'est à dire la colonie C_d tel que $deg(V_i, C_d) = max_{k=1}^{k=NbColonies} deg(V_i, C_k)$. Une première image segmentée peut être créée de la façon suivante :

1. pour chaque voxel, l'affectation de la couleur de référence de sa colonie dominante ;
2. affectation d'une couleur pour les voxels de degré NULL.

La figure 3.6(a) montre une image de taille 5×5×5 voxels pour laquelle il y a un objectif de segmentation en deux régions. Dans cette image, on peut voir l'étape initiale avec la valuation de tous

les voxels en niveau de gris. Deux régions sont définies. Les agents-araignées de chacune d'entre elles vont tisser des fils sur ces voxels.

À l'issue du processus, on calcule les degrés de chaque voxel. Dans l'image 3.6(b), on voit des sommets rouges pour la colonie 1 et des sommets bleus pour la colonie 2. Les sommets verts représentent les voxels tissés par les deux colonies. La figure 3.6(c) représente le graphe des régions de l'image et la figure 3.6(d) montre l'image segmentée pour laquelle un choix a été fait d'affecter à l'une ou à l'autre les voxels pour lesquels il y avait une compétition (le maximum de degré du compteur).

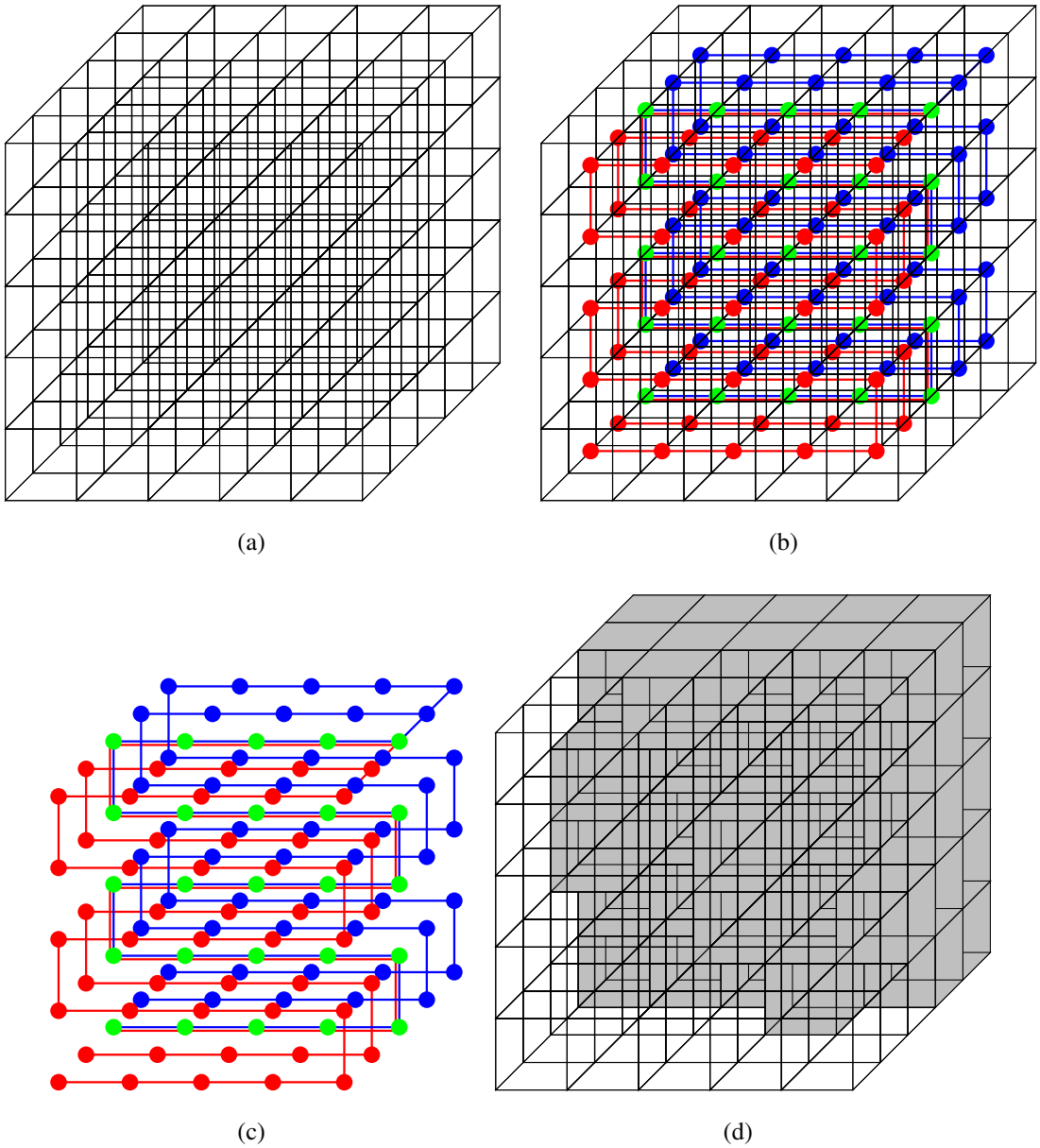
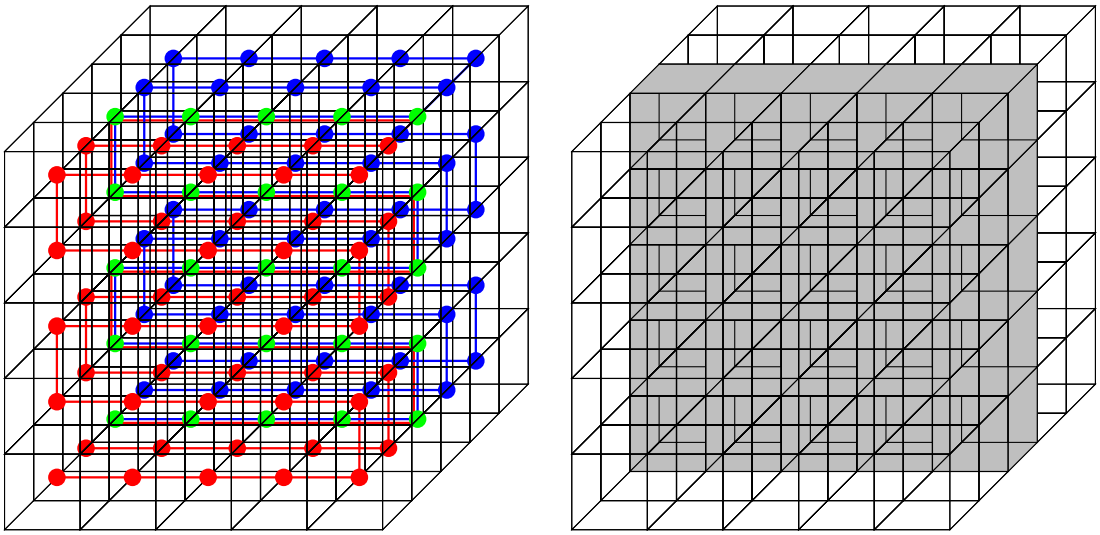


FIGURE 3.6: Les étapes de la segmentation en régions : (a) image initiale, (b) image à jour par les informations des fils, (c) les informations des fils et (d) l'image segmentée.

Les zones de compétition entre les colonies sont intéressantes à analyser. En effet, comme le montre la figure 3.7 ; à partir de cette information, on peut aussi extraire une information topologique sur l'image.



(a) (b)

FIGURE 3.7: L'extraction des voxels en compétition : (a) image segmentée et (b) image de compétition.

3.2 Segmentation contours : les fourmis

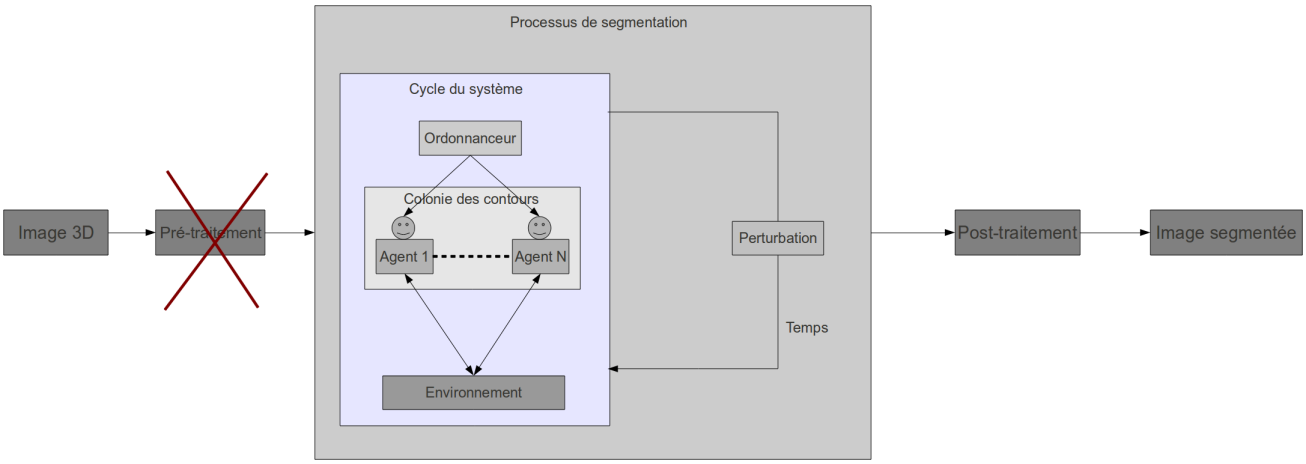


FIGURE 3.8: Chaine de segmentation en contours.

Comme pour le modèle des araignées décrit ci-dessus, dans le modèle des agents-fourmis (figure 3.8), nous n'avons pas besoin d'effectuer un pré-traitement tel que l'application de filtres sur l'image puisque notre objectif principal est de tester la robustesse du résultat de la segmentation sur des données bruitées.

3.2.1 Création de l'environnement

Dans une segmentation classique, il est possible de considérer deux types de partitionnement : un partitionnement en voxels ou en inter-voxels. Le modèle d'agents-fourmis pourra utiliser l'un ou l'autre de ces partitionnements.

Suivant le partitionnement choisi, le codage de l'image dans l'environnement sera différent. Les agents-fourmis seront toujours positionnés sur un point de l'image. Mais, dans un cas ce point correspondra à un voxel (comme pour le modèle des araignées) et dans l'autre cas il correspondra à un pointel (la notion de pointel sera expliquée dans la section 3.2.4).

De part le fait que les agents-fourmis sont des détecteurs de contours et pas de régions, ce modèle n'utilise qu'une seule colonie.

3.2.2 Description du cycle de vie

Le cycle de vie des agents-fourmis est similaire à celui des agents-araignées : **Perception**, **Décision** et **Action**. À chaque pas de temps, un agent-fourmi se déplace dans l'environnement afin de rechercher des "voxels intéressants" et de les phéromoner.

Perception

Pour favoriser une exploration maximum de l'environnement, les points de l'environnement mémorisent l'information suivant qu'ils aient été ou non déjà visité. Une probabilité conditionnelle est affectée à chaque point et sera prise en compte lors de l'étape de décision de déplacement. Le détail de ces probabilités est donné dans la section suivante. À cette étape de perception, l'agent-fourmi construit sa liste de destination potentielle suivant le principe suivant :

1. si au moins un point du voisinage n'est pas visité, il est retenu ;
2. si plusieurs points du voisinage ne sont pas visités, leur liste est construite ;
3. si tous les points du voisinage ont déjà été visités au moins une fois, tous les points appartiennent à la liste des destinations potentielles.

Décision

Le rôle de cette étape est de choisir le nouveau point à phéromoner. Pour cela, on prendra en compte d'une part le statut du point (quantité de phéromones déjà déposée) et d'autre part l'état interne de l'agent-fourmi : direction privilégiée et colorimétrie.

Action

Le déplacement de l'agent-fourmi consiste non seulement en la mise à jour de sa position mais aussi le marquage du point d'arrivée en ajoutant des phéromones dans le compteur du point. Afin de pouvoir faire marche arrière, chaque agent-fourmi mémorise aussi le dernier point quitté.

Avant l'exécution d'un pas de temps pour l'ensemble des agents-fourmis, la liste des agents-fourmis est réorganisée aléatoirement. Après l'exécution d'un pas de temps, une évaporation prédéfinie est appliquée à tous les points de l'environnement. La valeur 0 est la valeur minimum pour un point.

La complexité de l'algorithme des fourmis diffère de celui des araignées par l'ajout d'une étape d'évaporation des phéromones dans l'environnement. Ceci est réalisé à chaque pas de temps.

3.2.3 Identification des paramètres

Dans cette simulation, la stigmergie est aussi présente : chaque agent-fourmi partage des phéromones au travers l'environnement avec les autres agents-fourmis et cette information sera utilisée dans leur processus décisionnel pour le prochain pas de temps. Ces contours contenus dans l'image sont donc les chemins tracés par les phéromones des fourmis qui auront résisté à l'évaporation.

3.2.3.1 Des problèmes

Bien que la méthode des fourmis ne requiert que peu de paramètres, leur définition reste un point crucial, générateur éventuel de biais. Plusieurs problèmes se posent :

1. comment sélectionner le nombre d'agents nécessaires pour la segmentation sans aucune reconnaissance du nombre de points qui composent le contour à détecter ?
2. comment adapter et évaluer la fonction d'évaporation ?
3. comment calculer le nombre de pas nécessaires pour arrêter la simulation ?

3.2.3.2 Des solutions

Afin de répondre aux problèmes soulevés dans la section précédente, nous proposons les solutions suivantes :

1. pour calculer le nombre d'agents nécessaires pour la détection des contours de l'image, nous pouvons nous référer au gradient de l'image qui permettra de déterminer le pourcentage de voxels appartenant aux contours. Ce nombre pourra être utilisé pour déterminer le nombre d'agents ;
2. pour définir le rythme de l'évaporation des phéromones, plusieurs éléments peuvent être pris en compte :
   a) considérer une information complètement évaporée s'il n'y a pas eu de dépôt de phéromones pendant une période $\triangle t$;
   b) seuiller la distribution finale des phéromones déposées sur l'environnement.

   Nous utiliserons la solution b) dans notre application.
3. Pour arrêter la simulation, la solution la plus courante est d'observer à chaque pas de temps combien de nouveaux points sont phéromonés et d'arrêter la simulation en dessous d'un valeur minimale.

La détection des contours par les agent-fourmis dépend des phéromones déposées ayant une concentration supérieure à 0 dans l'environnement. Si le nombre de nouveaux points phéromonés tend vers 0, nous pouvons considérer que le système converge vers une solution et les agents-fourmis ne font alors que renforcer les contours détectés.

Pour le modèle des fourmis, il est intéressant de mémoriser le nombre de passage d'un agent sur un point. En effet, avoir un compteur de phéromones à 0 peut résulter de 2 situations différentes : le point n'a jamais été visité ou bien les phéromones qui y étaient déposées se sont évaporées (voxels

non visités depuis un certain temps). Ainsi, un point sera considéré comme non-encore tagué non pas quand son compteur était précédemment à zero mais quand le nombre de passages mémorisé est égal à zéro.

Comme pour le modèle des araignées, nous pouvons maintenant définir une condition d'arrêt de la simulation qui vérifie ces contraintes.

Définition 18 *Soit α le nombre de points phéromonés en un pas de temps donné et dont les compteurs de passages étaient précédemment à zéro.*

On définit 2 seuils : $Seuil_{Min}$ et $Seuil_{Max}$.

- $Seuil_{Min}$ représente le nombre minimal de points nouvellement phéromonés attendu pour considérer le pas de simulation comme significatif ;
- $Seuil_{Max}$ représente le nombre de pas de simulation consécutifs acceptés où $\alpha < Seuil_{Min}$. Lorsque $Seuil_{Max}$ est atteint, la simulation est arrêtée automatiquement car on considère alors que le système se stabilise et qu'il converge.

3.2.4 Instanciation de l'environnement

Comme nous l'avons vu au chapitre 1, les fourmis ont besoin d'un environnement pour exécuter leur cycle de vie. Cet environnement est défini selon le partitionnement à effectuer : partitionnement voxels ou partitionnement inter-voxels. La détection de contours suppose qu'une colonie de fourmis va à la recherche des voxels ou des lignels (cas inter-voxels) pouvant avoir une certaine discontinuité. Nous allons donc présenter une représentation de l'environnement pour chaque cas. À chacun de ces cas correspond un partitionnement :

1. un maillage cubique classique de l'image : dans ce cas les voxels seront les points sur lesquels se déplacent les fourmis. Ce partitionnement est identique à celui utilisé comme solution par le modèle des araignées. Nous retiendrons aussi le 26-voisinage ;
2. un nouveau type de partitionnement issus des méthodes "inter-voxels".

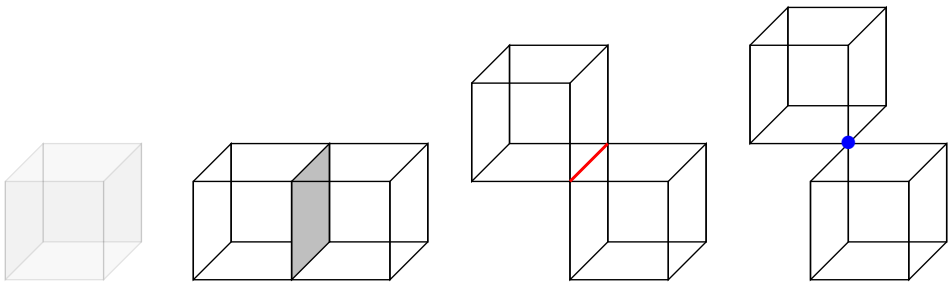
Le deuxième point correspond à un partitionnement inter-voxels d'une extension en 3D du partitionnement classique inter-pixels (Braquelaire and Brun, 1998), *i.e.* qu'une image n'est pas seulement considérée comme une matrice de voxels mais comme une subdivision d'un espace 3D en un ensemble d'éléments de dimension 3, 2, 1 et 0. Ces éléments, représentés dans la figure 3.9, sont respectivement le voxel, le surfel, le lignel et le pointel (Kovalevsky, 2003). Leurs définitions sont données ci-dessous:

Définition 19 *Un voxel (figure 3.9(a)) est représenté par un cube.*

Définition 20 *Un surfel (figure 3.9(b)) est représenté par l'intersection de deux voxels tel que ces deux derniers sont 6-connexes.*

Définition 21 *Un lignel (figure 3.9(c)) est représenté par l'intersection de deux voxels tel que ces deux derniers sont 18-connexes et pas 6-connexes.*

Définition 22 *Un pointel (figure 3.9(d)) est représenté par l'intersection de deux voxels tel que ces deux derniers sont 26-connexes et pas 18-connexes.*



(a) (b) (c) (d)

FIGURE 3.9: Les éléments inter-voxels : (a) voxel, (b) surfel, (c) lignel et (d) pointel.

Le principe de base du partitionnement inter-voxels repose sur la considération des faces des voxels séparant chaque région de ses voisines en tant que contours séparant les régions de l'image. Dans notre cas, nous considérons les lignels séparant les voxels adjacents comme contours des régions. À l'étape finale, les voxels encodant ces lignels représentent les contours obtenus. La modélisation de l'environnement permet ensuite de définir le voisinage de l'agent. Contrairement au modèle araignées pour lequel le voisinage pouvait dynamiquement être modifié par la présence d'un fil, le modèle fourmis affecte à chaque agent un voisinage strictement constitué des points adjacents au sens des partitionnements que nous venons d'expliquer. Ce voisinage restera identique au fil des pas de simulation.

Pour coder le partitionnement inter-voxels d'une image 3D de taille $X \times Y \times Z$, nous utilisons une matrice de taille $(X + 1) \times (Y + 1) \times (Z + 1)$. Cette matrice contient soit les caractéristiques de colorimétrie de chaque voxel (niveaux de gris) soit pour chacun des axes x, y et z les valeurs de colorimétrie associées au lignel correspondant, *i.e.* la variation des niveaux de gris entre les 2 voxels adjacents (figure 3.10).

3.2.5 Processus de segmentation

L'état interne d'une fourmi est identique à celui d'une araignée au détail près qu'elle possède également la notion de direction pour ses déplacements. Afin de généraliser notre plateforme de segmentation, nous conservons la même définition du cycle de vie pour les araignées et les fourmis : **Perception**, **Décision** et **Action**.

À l'initialisation de la simulation, Les fourmis vont se disperser dans l'environnement dans le but de détecter les contours de l'image (Moussa et al., 2011b). Puis, à chaque pas de temps, tous les agents-fourmis exécuteront leur cycle de vie tel que décrit ci-dessous.

Perception

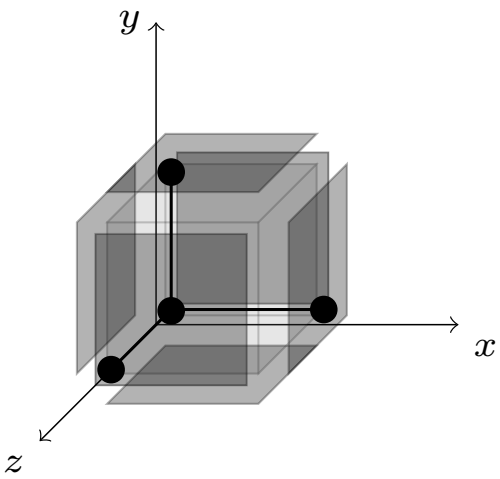


FIGURE 3.10: Le codage inter-voxels

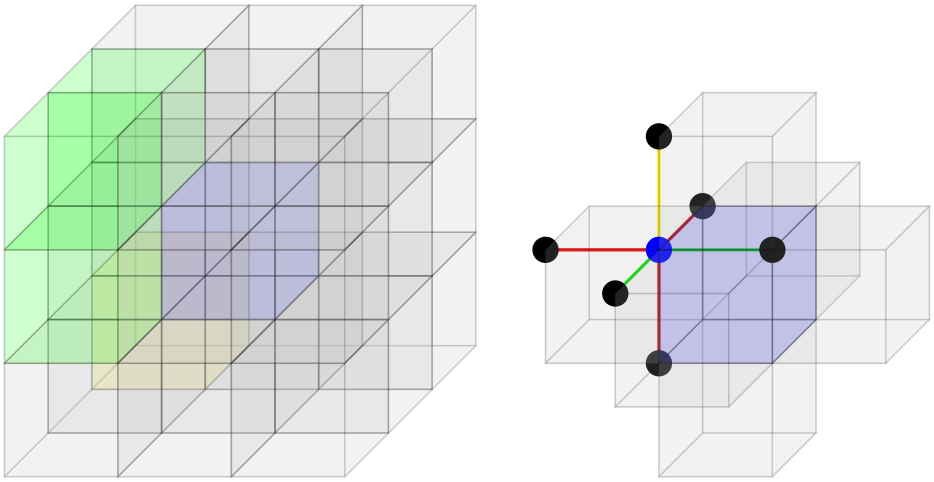
Comme expliqué précédemment, la fourmi n'a qu'une vision purement locale de sa prochaine destination. Cette visibilité permet la construction d'une liste de points candidats où la fourmi souhaite se déplacer et déposer une phéromone. Comme pour le modèle des araignées, chaque fourmi favorise des points par rapport aux autres. Elle commence à regarder les points qu'aucun membre de sa colonie n'a encore visité, représentés par les voxels en gris dans la figure 3.11(a) et par les lignels en jaune dans la figure 3.11(b). Si il existe des candidats, c'est cette liste qui sera évaluée dans la partie décisionnelle pour le choix de la nouvelle position. Sinon, c'est la liste composée des points déjà phéromonés qui va représenter les candidats potentiels en excluant toutefois la dernière position où il y a eu un dépôt de phéromone (*cf.* algorithme 9).

Décision

Pour procéder à la sélection du meilleur chemin à traverser entre les points, chaque fourmi analyse la liste de choix qu'elle possède. Si cette liste correspond à de nouveaux points à explorer, une sélection aléatoire est réalisée afin d'en tirer la nouvelle destination. Dans le cas où tous les points sont déjà phéromonés, une fonction de transition est calculée reposant sur des caractéristiques entre chaque couple de points (p_1, p_2) tel que p_1 correspond à la position actuelle de la fourmi et p_2 la prochaine destination. Ces caractéristiques sont d'une part la variation de couleur entre les deux points, le gradient de p_2, la concentration de phéromones présente en p_2 et d'autre part une orientation de la fourmi signalant la direction de son déplacement. Dans le cas où la décision est faite par rapport à un partitionnement en voxels de l'environnement, la sélection du meilleur voxel v_j destinataire est calculé selon une probabilité P_{DV} de passer au pas de temps t d'un voxel v_i où se situe la fourmi au voxel v_j :

$$P_{DV_{v_i \to v_j}}(t) = \frac{Poids_{DV_{v_i \to v_j}}(t)}{\sum_{v_{\hat{n}} \in Voisins(v_i)} Poids_{DV_{v_i \to v_{\hat{n}}}}(t)} \tag{3.8}$$

Où le poids de décision $Poids_{DV_{v_i \to v_j}}(t)$ dépend de l'orientation (O) de la fourmi, du gradient (G) du



(a) (b)

FIGURE 3.11: Voisinage d'un point : (a) cas d'un partitionnement en voxels : les voxels en gris sont les voxels non visités, les voxels verts sont les voxels phéromonés et le voxel en jaune est la dernière position d'où vient la fourmi, (b) cas du partitionnement inter-voxels : les lignels en rouges sont ceux non visités, les lignels en vert sont ceux déjà phéromonés et le lignel jaune représente la dernière voie traversé par la fourmi.

Algorithme 9 La sélection d'un voisinage locale par une fourmi

ENTRÉES: Image : Matrice $\in \mathbb{N}^3$, A : Agent
SORTIES: V : liste de voisins $\in \mathbb{N}$

```
1:  V ← ListeVide().
2:  V_tempo1 ← ListeVide().
3:  V_tempo2 ← ListeVide().
4:  V ← ListeVoisins(Position(A), Image).
5:  Pour Chaque voisin V_i de V Faire
6:      Si (V_i != Position(A)) Alors
7:          Si Visité((V_i) == Faux Alors
8:              V_tempo1 ← Ajouter(V_i).
9:          FinSi
10:         Si V_i != Position(DernièrePhéromone(A)) Alors
11:             V_tempo2 ← Ajouter(V_i).
12:         FinSi
13:     FinSi
14: Fin Pour
15: Si Taille(V_tempo1) !=0 Alors
16:     V ← V_tempo1.
17: Sinon
18:     V ← V_tempo2.
19: FinSi
```

voxel destination, de la variation locale (V) entre le voxel source et le voxel destination et du montant de la concentration de phéromones (φ) déposé au cours des étapes précédentes.

$$Poids_{DV_{v_i \to v_j}}(t) = Poids_{O_{v_i \to v_j}}(t) \times Poids_{G(v_j)}(t) \times Poids_{V_{v_i \to v_j}}(t) \times Poids_{\varphi(v_j)}(t) \tag{3.9}$$

$$Poids_{O_{v_{i \to j}}}(t) = \begin{cases} 1 & \text{Si} \quad -\frac{\pi}{4} \leq \theta_{v_{ij}} - \theta_0 \leq \frac{\pi}{4} \\ \frac{1}{2} & \text{Si} \quad |\phi_{v_{ij}} - \phi_0| = \frac{\pi}{2} \; et \; \theta_{v_{ij}} = \theta_0 \\ \frac{1}{4} & \text{Si} \quad |\theta_{v_{ij}} - \theta_0| = \pi \\ \frac{1}{2^{\frac{|\theta_{v_{ij}} - \theta_0|}{\frac{\pi}{4}}}} & \text{Sinon} \end{cases} \tag{3.10}$$

Où $Poids_{O_{v_{i \to j}}}(t)$ dépend de la direction de la fourmi. Cette direction est donnée par différence des coordonnées de la position de l'agent-fourmi au temps $(t-1)$ et au temps (t) sur les 3 axes. La figure 3.12 montre à partir d'une position (cas bleu) les deux angles θ et ϕ correspondant au déplacement en 3 dimensions.

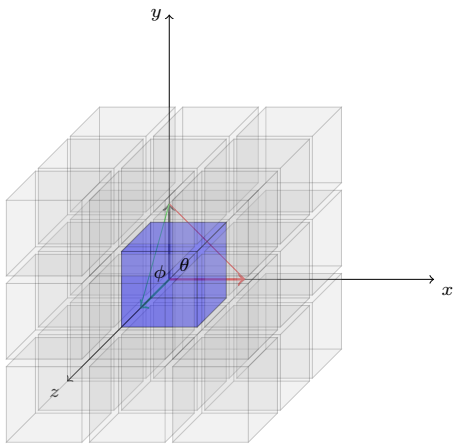


FIGURE 3.12: Calcul du poids de l'orientation.

Pour le calcul de $Poids_{O_{v_{i \to j}}}(t)$, on distingue les quatre cas de l'équation 3.10. La figure 3.13 montre ces différents cas de déplacement. Par exemple le cas 2 exprime un déplacement sur l'axe des z.

$$Poids_{G(v_i)}(t) = \frac{max(\frac{\sum_{v_{\tilde{n}} \in Voisins_{v(i)}} F(v_{\tilde{n}}) \times S(index(v_{\tilde{n}}))}{\sum_{k \in S} |k|})}{Intensitémaximale} \tag{3.11}$$

Le gradient (G) de chaque voxel voisin est calculé à travers tous les configurations 3D grâce à l'opérateur de Sobel (S) et le gradient maximal est sélectionné. Le poids $Poids_{G(v_i)}(t)$ est ensuite normalisé.

$$Poids_{V_{v_i \to v_j}}(t) = \begin{cases} 1 & \text{Si} \quad |F(v_i)| = |F(v_j)| \\ \frac{1}{|F(v_i) - F(v_j)|} & \text{Sinon} \end{cases} \tag{3.12}$$

La variation d'intensité locale entre 2 points est calculée à partir de l'intensité de chaque point selon l'équation 3.12 où F(.) représente l'intensité du point.

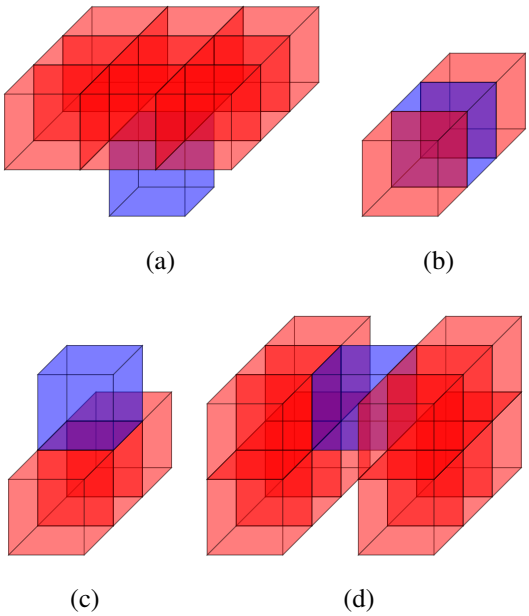


FIGURE 3.13: Les différents cas d'orientation : (a) cas 1, (b) cas 2, (c) cas 3, (d) cas 4.

Le dernier terme de l'équation 3.9 représente la prise en compte de la quantité de phéromones déjà déposée sur le point j.

Dans le cas où le partitionnement est de type inter-voxels, la fourmi doit pouvoir se déplacer entre les pointels. Dans ce modèle, les surfels sont implicites pour connaitre la variation entre les 2 voxels en raison du codage choisi (*cf.* section 3.2.4). La figure 3.14 montre que le calcul de la variation du lignel entre le pointel du voxel bleu et le pointel du voxel noir est donné par le surfel gris foncé. Afin de calculer la probabilité de chaque destination possible $P_{DIV_{p(i)\to p(j)}}(t)$, le poids de chaque lignel (l_i) est normalisée par le poids total de tous les voisins :

$$P_{DIV_{p(i)\to p(j)}}(t) = \frac{Poids_{DIV_{p(i)\to p(j)}}(t)}{\sum_{p_{\hat{n}}\in Voisins(p_i)} Poids_{DIV_{p(i)\to p_{\hat{n}}}}(t)} \tag{3.13}$$

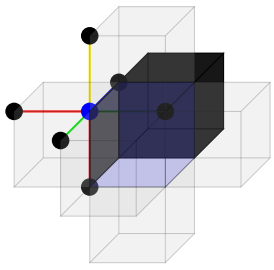


FIGURE 3.14: Représentation du calcul de la variation d'intensité sur un surfel.

La fonction de décision utilise le même calcul de probabilité de mouvement décrit par l'équation 3.10.

On obtient désormais l'équation 3.14 pour calculer $Poids_{DIV_{p(i)\to p(j)}}(t)$ de la décision inter-voxels :

$$Poids_{DIV_{p(i)\to p(j)}}(t) = Poids_{O_{p_i\to p_j}}(t) \times Poids_{Vl(l_{i\to j})}(t) \times Poids_{V_{p_{p_i\to p_j}}}(t) \times Poids_{\varphi(l_{i\to j})}(t) \tag{3.14}$$

Avec désormais $Poids_{Vl(l_{i\to j})}(t)$ la variation de l'intensité du lignel $l_{i\to j}$.

$$Poids_{Vl_{l_{i\to j}}}(t) = \frac{|F_1(i) - F_1(j)|}{Intensitémaximale} \tag{3.15}$$

$Poids_{\varphi(l_{i\to j})}(t)$ représente de la même façon la quantité de phéromones sur le lignel $l_{i\to j}$.

L'algorithme 10 décrit le processus décisionnel pour un agent-fourmi à partir de la liste de voisins fournie par l'étape **Perception** et suivant le type de ces voisins (phéromonés ou pas). Le choix de la prochaine destination sera aléatoire dans un cas et dépendant du calcul du poids de chaque candidat, calculé à partir des équations 3.9 et 3.14 en fonction du type de partitionnement.

Algorithme 10 La décision sur le prochain voisin à phéromoner

Entrées: Image : Matrice $\in \mathbb{N}^3$, V : liste de voisins $\in \mathbb{N}$, VoisinPhéromoné : booléen in [Vrai, Faux]
Sorties: NouvellePosition $\in \mathbb{N}^3$, Concentration $\in \mathbb{R}$.

```
Concentration ← 0.
PhéromoneMax ← 0.
Si VoisinPhéromoné == Faux Alors
    NouvellePosition ← ChoixAléatoire(V).
    Concentration Poids(φ(.)) selon les équations 3.16 et 3.17.
Sinon
    Pour Chaque voisin V_i de V Faire
        Si Poids_Dec(position(A), V_i) > PhéromoneMax Alors
            PhéromoneMax ← Poids_Dec(.) selon les équations 3.8 et 3.14.
            NouvellePosition ← V_i.
            Concentration ← Poids(φ(.)) selon les équations 3.16 et 3.17.
        FinSi
    Fin Pour
FinSi
```

La complexité de cet algorithme est de l'ordre de $O(Voisinage_{total})$, où $Voisinage_{total}$ est le nombre de candidats potentiels pour une fourmi.

Lorsqu'un agent-fourmi aura choisi son déplacement, on passe à l'étape **Action**.

Action

Soit j la nouvelle position de la fourmi, on mettra à jour La quantité de phéromones de j $Poids_{\varphi(v_j)}(t)$ à partir de sa précédente valeur à laquelle on ajoute une quantité Q prédéfinie par le poids de la variation de l'intensité entre i et j et le gradient du voxel j (équation 3.16). Pour un partitionnement inter-voxels, on utilisera l'équation 3.17.

$$Poids_{\varphi(v_j)}(t) = Poids_{\varphi(v_j)}(t-1) + Q \times Poids_{V_{v_i\to v_j}}(t) \times Poids_{G(v_j)}(t) \tag{3.16}$$

$$Poids_{\varphi(l_{i\to j})}(t) = Poids_{\varphi(l_{i\to j})}(t-1) + Q \times Poids_{Vp_{p_i\to p_j}}(t) \times Poids_{Vl(l_{i\to j})}(t) \tag{3.17}$$

La mise à jour de la quantité de phéromones à chaque pas de temps s'effectue sur le voxel destination dans le cas du partitionnement voxels et sur le lignel joignant les 2 pointels dans le cas du partitionnement inter-voxels.

Un des problèmes bien connus de la méthode fourmi est le problème des "aller-retour", *i.e.* l'enfermement d'une fourmi dans un périmètre restreint qu'elle parcourt perpétuellement (Chevrier, 2002).

Notre algorithme, en pondérant le choix du nouveau point par sa direction permet en général de résoudre le problème et nos simulations ont montré que notre modèle ne présente pas ce type de biais.

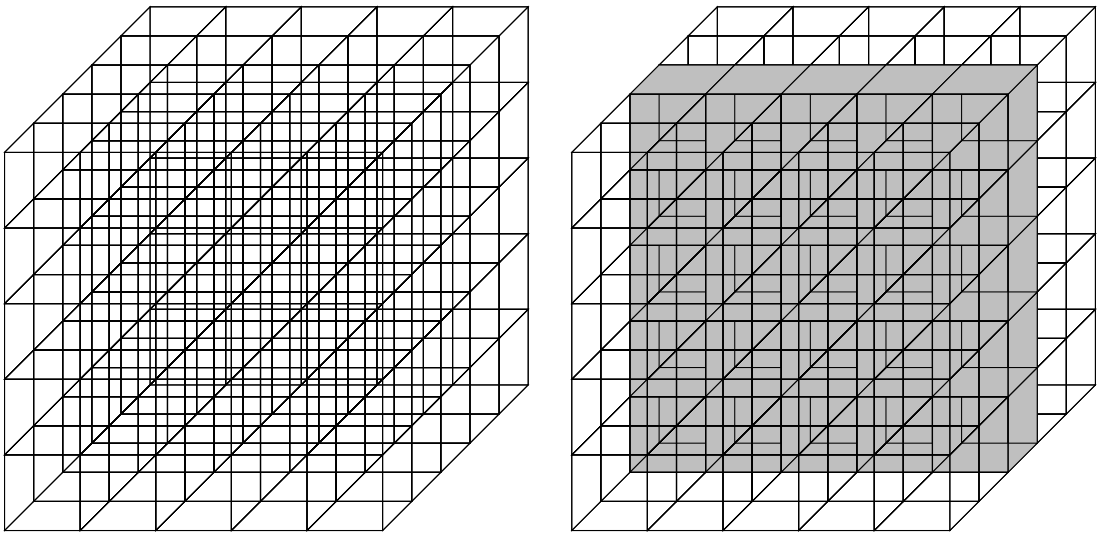
3.2.6 Construction de l'image segmentée

Pour construire les contours recherchés dans l'image, on va appliquer à chaque voxel ou lignel l'équation 3.18 où IntensitéMaximale est la couleur maximum de l'image (255 ou 4095 selon le codage).

$$Couleur(V_i) = \frac{Concentration(V_i)}{ConcentrationMax} \times Intensitémaximale \tag{3.18}$$

À partir de cette image, on peut construire une image binaire qui déterminera pour un seuil donné si un voxel (lignel) appartient au contour. Le seuil peut être choisi à partir de l'histogramme des quantités de phéromones déposées sur l'image.

L'exemple de la figure 3.16 est le même que celui que nous avons utilisé pour le modèle des araignées, l'image à segmenter est de taille 5×5×5 voxels. Nous avons simulé une segmentation avec les agents-fourmis. Dans un cas, des phéromones sont déposées sur les voxels . Après avoir analysé les phéromones de l'image, nous obtenons l'image segmentée présentée par la figure 3.15(b).



(a) (b)

FIGURE 3.15: L'extraction des voxels contours : (a) image origine et (b) image segmentée en contours.

Dans le cas où nous simulons la segmentation inter-voxels, les phéromones sont déposées au fil du temps sur les lignels de l'images (3.16(b)). Ce sont les voxels représentants ces lignels (figure 3.16(c)) qui vont construire les contours (figure 3.16(d)).

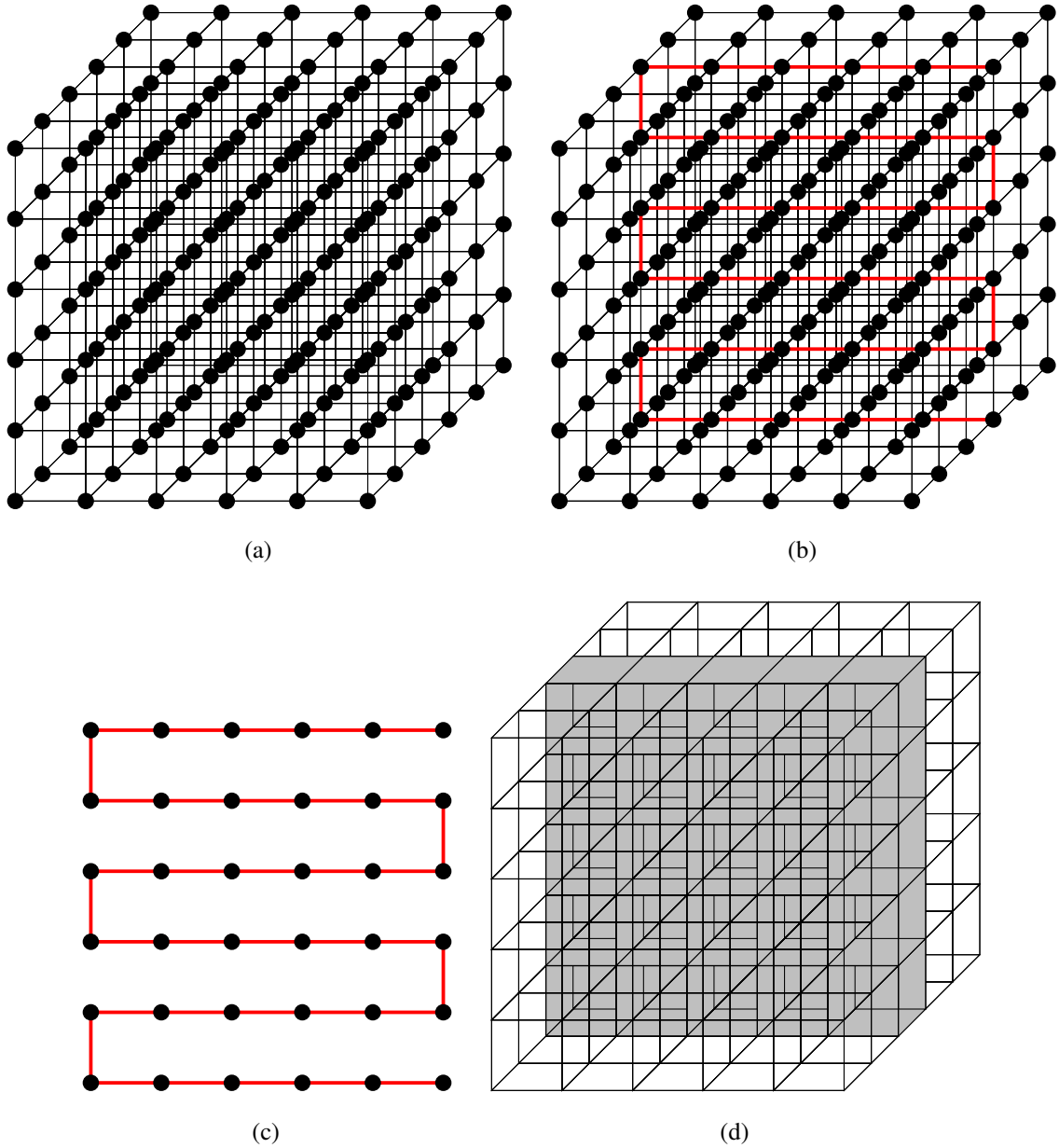


FIGURE 3.16: L'extraction des voxels contours : (a) image origine, (b) les lignels phéromonés, (c) le graphe des lignels et (d) l'image segmentée en contours par les voxels représentants des lignels.

3.2.7 Amélioration de la recherche des contours

Classiquement, les modèles fourmis prévoient une évaporation des phéromones dans le temps. Cette évaporation a pour objectif d'effacer le bruit induit par la dispersion des fourmis dans l'environnement et d'annuler l'effet de passage peu fréquent sur un point.

Toutefois, ce traitement est apparu comme insuffisant pour obtenir un contour de bonne qualité. Nous avons donc utilisé l'information concernant le nombre de passage sur un point comme élément de pondération dans l'évaluation de la quantité de phéromones déposées.

Pour cela, nous écrivons les équations 3.19 et 3.20 :

$$Poids_{\varphi(v_i)}(t) = (1-\rho) \times Poids_{\varphi(v_i)}(t-1) + NbPassage * F(v_i) \tag{3.19}$$

$$Poids_{\varphi(l_i)}(t) = (1-\rho) \times Poids_{\varphi(l_i)}(t-1) + NbPassage * F(l_i) \tag{3.20}$$

Dans cette équation, ρ est le facteur d'évaporation ($0 \leq \rho \leq 1$), $Poids_{\varphi(.)}(t)$ est la valeur de phéromones avant évaporation. F(.) représente le poids du gradient pour un partitionnement en voxels et le poids de la variation locale présente sur le surfel pour le cas inter-voxels.

La complexité de cette opération est de O(Taille(Image)).

3.3 Ordonnanceur

Les simulations SMA peuvent en général être exécutées en mode séquentiel ou en mode parallèle. Les ordinateurs étant des machines séquentielles lorsqu'on dispose d'un seul coeur, la simulation est à priori en mode séquentiel. Dans le cas d'un processeur multi-coeurs (CPU ou GPU), il devient possible de paralléliser l'exécution d'un pas de temps.

3.3.1 Exécution séquentielle

Dans ce cas, la liste des agents sera réordonnée à chaque début d'exécution d'un pas de temps et tous les agents exécuterons leur cycle de vie l'un après l'autre. Il a été prouvé qu'à partir d'un nombre d'agents assez grand ; ce type de simulation ne présentait pas d'artéfact de synchronisation (Ballet, 2000). Nos simulations impliquent toujours plusieurs milliers d'agents, nous sommes donc dans ce cas précis. Toutefois, le temps d'exécution d'une segmentation est massivement dépendant du nombre de pas de temps de la simulation. Lorsqu'on a une exécution séquentielle, le temps de la simulation devient $TempsCalcul(cycledevie) \times NbAgents \times NbPasDeTemps$. Ceci est un des reproches majeurs fait aux simulations SMA, car il conduit à des simulations extrêmement longues.

3.3.2 Exécution parallèle

À l'heure actuelle, il existe plusieurs solutions pour exécuter des simulations en parallèle dont certaines sont peu coûteuse à la fois en coût matériel, mais aussi en temps pour paralléliser des algorithmes.

En ce qui concerne l'aspect matériel, on dispose aisément des machines multi-coeurs CPU-8coeurs. Pour les GPU, une carte graphique un peu spécialisée telle que NVIDIA QUADRO 4000 fournira 256 coeurs. Dans le cas d'un SMA à base d'agents sociaux dédié au traitement d'images, nous pouvons déposer l'image dans l'environnement commun à tous les coeurs et répartir les agents sur les coeurs.

La question de l'interaction entre les agents se pose. En effet, les agents appartenant à différents coeurs ne peuvent pas communiquer directement. Le modèle d'agents sociaux spécifie que les agents communiquent au travers de l'environnement, dans la mesure où pour le GPU la mémoire, et donc l'environnement, sont partagés, ils deviennent naturellement le média de communication des agents. La synchronisation des cycles de vie des agents n'est donc pas cruciale. Dans l'implémentation à mémoire partagée, l'algorithme garantit que tous les agents exécutent leur cycle de vie à chaque pas de temps. Il y a donc une synchronisation au niveau de chaque pas de temps et les différents agents sont activés en mode asynchrone. Bien sûr, il reste la possibilité qu'à un pas de temps donné, 2 agents modifient concurremment le même point. Nous avons décidé de ne pas tenir en compte de cette erreur après avoir mesuré lors de tests l'incidence de cette perte d'information. En fait, étant donné le très grand nombre de points d'une image 3D (en général, une image IRM est de taille 256x256x128), même avec plusieurs milliers d'agents, la probabilité que 2 agents soient sur le même point est très faible. Tous les détails de l'implémentation sont données en annexe (annexe A). Pour cette implémentation, à la fois les versions de parallélisation CUDA et OpenCL.

On notera que la parallélisation en mémoire partagée est particulièrement favorable aux modèles SMA. En effet, chaque agent "embarque" son algorithme de cycle de vie, identique pour tous, pour l'exécuter sur le cœur auquel il est dédié. Aucune modification de cet algorithme n'est induite par la parallélisation.

Désormais, le temps de la simulation est $Temps(CycleDevie) \times \frac{Nbagents}{NbMaxThread} \times NbPas$ où $NbMaxThread$ est le nombre de traitement maximum qu'une machine multi-cœurs exécutera en un pas de temps. Pour une carte graphique NVIDIA QUADRO 4000 (qui est celle que nous avons utilisée pour l'application), ce nombre est $(65535)^3 \times 512$. Comme nous ne dépassons pas ce nombre d'agents, nos simulations sont de l'ordre de $TempsCalcul(cycledevie) \times NbPasDeTemps$. Dans la pratique, comme nous le montrerons au chapitre suivant, nous avons vu nos temps de segmentation passer de plusieurs heures à quelques minutes, rendant ainsi les SMA concurrentiels par rapport aux méthodes classiques : Otsu multi-niveaux, Croissance de régions ou Sobel.

Il reste à traiter la question de la synchronisation de la mise à jour de l'état interne de chaque agent. Il est possible soit de maintenir un état actuel et un état futur, soit d'effectuer une mise à jour immédiate. La première solution est plus coûteuse en taille mémoire que la seconde. Si nous examinons le comportement réel des agents sociaux dans la nature, leur comportement n'est pas synchronisé et les conséquences d'une action sont immédiatement prises en compte.

Nous choisissons donc de réaliser des mises à jour asynchrones des états internes des agents (figure 3.17). Trois possibilités pour cela :

1. tous les agents sont mis à jour à chaque pas de temps du système. Les agents sont réordonnancés selon un tirage aléatoire sur les agents et nous pouvons garantir une équiprobabilité pour chaque agent de déposer une information ;

2. chaque agent est mis à jour après l'exécution de son cycle de vie. L'ordonnancement des agents se traduit ici par le tirage d'une probabilité à être sélectionné en premier. Cette liste de probabilité est ensuite triée par ordre décroissant et les agents sont activés suivant leur rang.
3. la sélection d'un seul agent parmi l'ensemble pour qu'il dépose son information sur l'environnement. Cette sélection est faite d'une manière aléatoire sur l'ensemble des agents présents dans l'environnement. Dans ce cas, le nombre de pas de temps de la simulation jouera le rôle le plus important dans la sélection des agents à activer.

La solution 2 est celle qui a été adopté pour ces simulations sur GPU présentées en annexe.

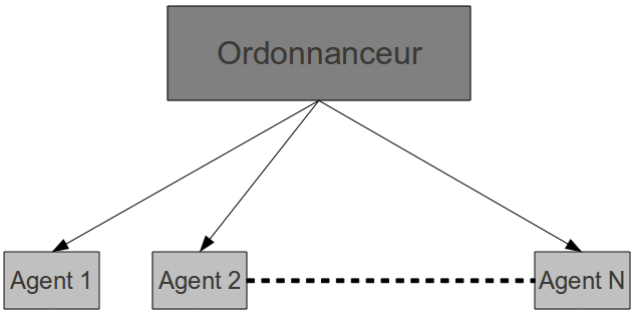


FIGURE 3.17: Ordonnancement des agents.

La complexité de cette opération est de $\mathcal{O}$(Taille(Agents) + Taille(Agents) $\times$ log (Taille(Agents))), où Taille(Agents) représente le nombre d'agents dans l'environnement.

3.4 Vers une modélisation Orienté Objet

La plateforme agent de IPSiMAS a été réalisée suivant le paradigme Objet qui consiste à définir et à assembler des objets. Chaque objet représente un concept, une idée ou une entité physique. Dans un modèle Orienté Objet, les entités sont définies à partir de classes et de relations entre ces classes. Chaque classe est caractérisée par des attributs et des comportements. Nous désignons une relation d'héritage entre les classes lorsqu'une classe est spécialisée en sous-classes et une relation de composition ou d'agrégation lorsque une classe contient des instances d'autres classes. L'intérêt de cette modélisation est de favoriser la conception de modules génériques faciles à étendre pour des besoins spécifiques.

Dans la suite, nous allons présenter l'architecture d'IPSiMAS (figure 3.18), et plus particulièrement, les classes essentielles pour la décomposition de l'environnement et la construction de colonies d'agents sans entrer dans une dissertation des méthodes utilisées par les différentes classes. Dans les sections suivantes, nous allons décrire les éléments essentiels des 3 principales classes du modèle : Agent, Environnement et Application.

3.4.1 Agent

Dans un modèle SMA, de façon assez triviale, nous devons définir une classe générique `Agent` qui sera la super-classe pour différents types d'agents que nous utilisons.

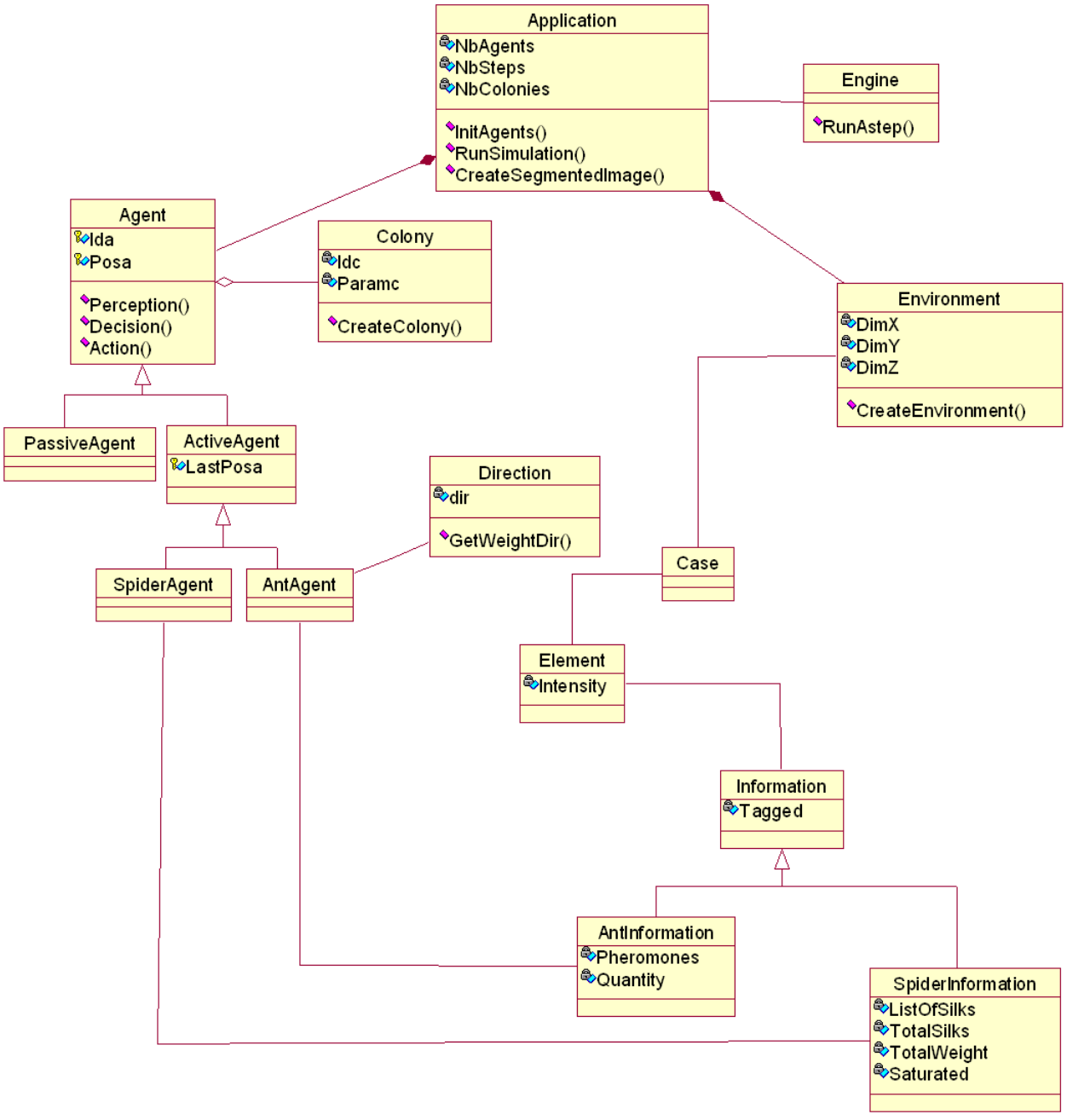


FIGURE 3.18: Architecture d'IPSiMAS.

Notre équipe développe des SMA dans différents contextes : la segmentation d'images (concernée par ce travail), la simulation de réactions métaboliques, la représentation de molécules physiques en interaction ou encore la simulation de mouvements de macro molécules. Nous avons décidé de tous utiliser le même modèle de classe `Agent` en développant plus ou moins certaines parties de l'arborescence suivant les besoins. Pour la segmentation d'images, nous n'utilisons que des agents de la classe `ActiveAgent`. Nous avons spécialisé la classe `ActiveAgent` en deux sous-classes : `AntAgent` et `SpiderAgent`. Les comportements génériques tels que les méthodes `Perception/Decision/Action` sont définis à partir de la classe `Agent` car elle caractérise tout comportement d'agent. Chaque sous-classe aura ensuite la responsabilité de spécialiser chaque comportement.

La classe `ActiveAgent` hérite d'un identifiant `Ida`, une position actuelle `Posa` de la classe `Agent`. La classe `ActiveAgent` définit un paramètre, la dernière position `LastPosa`. La `direction` est un attribut spécifique de la classe `AntAgent`. Lors de la création des colonies d'agents, un identifiant unique est généré pour chaque agent, la position est traitée automatiquement par une sélection aléatoire d'un placement dans l'environnement, la dernière position est vide et c'est lors de la simulation que cet attribut sera mis à jour. Concernant la direction de chaque `AntAgent`, une direction initiale est donnée pour chaque agent afin de lui définir un sens pour son mouvement. La méthode `GetPoidsDir` permet d'attribuer un poids selon son orientation actuelle par rapport au voisinage qu'il souhaite analyser. Ce poids est calculé selon l'équation 3.10.

La classe `Colony` permet de mémoriser la couleur de référence de chaque colonie. Elle contient aussi pour une colonie de `SpiderAgent` un intervalle de couleur pour représenter la plage d'attraction des voxels de l'environnement, $Attraction_{MaColonie}$ et $Attraction_{AutresColonies}$ qui sont utilisées pour paramétrer la coopération et la compétitivité et la `Saturation` pour définir le nombre de fils maximal que peut contenir un voxel dans l'environnement.

3.4.2 Environnement

La classe `Environment` sert à discrétiser l'image 3D pour la simulation. Dans notre cas, l'environnement est vu comme une grille régulière de cubes non hiérarchiques. Elle contient sa dimension qui sont `DimX`, `DimY` et `DimZ`, et la méthode `CreateEnvironment` pour spécifier l'espace de recherche pour les agents (par exemple 6-connexes ou 26-connexes). De plus, il est souvent utile de pouvoir mettre en place différentes résolutions de grilles pour optimiser certaines tâches et pratiquer une forme de multi-résolution. C'est pour cette raison que nous avons la classe `Case` permettant de regrouper un ensemble d'éléments et ainsi avoir une vision multi-échelles de cet environnement. Ceci est utile, par exemple, si nous voulons permettre la prise en compte de plusieurs positions dans une case. La classe `Element` représente la plus petite partition acceptée par le modèle.

À chaque voxel est associé des informations relatives à la catégorie de segmentation que nous souhaitons effectuer. Par exemple, pour une détection des contours de l'image, l'attribut `Tagged` recevra l'information si le voxel a été phéromoné. La classe `AntInformation` contient aussi des informations sur la concentration actuelle de phéromones présentes (`Pheromones`), le nombre de

fois qu'il y a eu un dépôt (`Quantity`). Tandis que pour extraire les régions de l'image, ce sont les attributs relatifs à la classe `SpiderInformation` qui vont contenir les positions des voxels tissés (`ListOfSilks`), le nombre total de fils (`TotalSilks`), le poids total des fils (`TotalWeight`) et un booléen `Saturated` pour spécifier s'il reste de la place pour tisser de nouveaux fils.

3.4.3 Simulation

Pour simuler la segmentation d'images, nous avons la classe `Application` composée d'un environnement, d'un groupe d'agents répartis et d'une ou plusieurs colonies. Cette classe contient les attributs `NbAgents` pour définir le nombre total d'agents présents dans l'environnement, le nombre de pas de temps système (`NbSteps`) et le nombre de colonies que nous voulons créer (`NbColonies`). Elle contient des méthodes pour initialiser la simulation suivant le mode d'exécution choisi (CPU ou GPU, synchrone ou asynchrone), pour instancier la méthode définissant un pas de temps (`RunAStep` de la classe `Engine`) et pour créer l'image résultat (`CreateSegmentedImage`). La première version du code parallèle est disponible via l'adresse suivant : *http://sourceforge.net/p/ipsimas/home/Home/*.

3.5 Conclusion

Dans ce chapitre, nous avons présenté deux approches de segmentation basées sur le comportement des araignées et des fourmis. La première approche, celle des araignées, consiste à segmenter une image en plusieurs régions. Le principe est de construire un environnement sous forme d'un grille et d'y placer l'information couleur ou intensité. Cette information est ensuite utilisée par les araignées pour qu'elles déposent leurs fils entre chaque couple de voxels "intéressants". Un graphe dynamique est ainsi construit représentant l'image segmentée. Ce processus de segmentation est terminé par une condition d'arrêt relative à l'information portée par les fils (`Saturation`). C'est ainsi qu'est créée l'image représentant la segmentation finale. La deuxième approche, celle des fourmis, a deux visions de l'environnement pour détecter les contours. La première vision est de percevoir l'environnement, comme pour le cas des araignées, comme un partitionnement de voxels. Les fourmis vont alors se déplacer sur les voxels pour créer des chemins. Ces chemins, au fil du temps, vont être validés par un grand nombre de fourmis présentes dans l'environnement malgré une perturbation induite par l'évaporation des phéromones. Ce processus de segmentation finit en renforçant les chemins existants. Ce sont ces chemins qui vont construire l'image segmentée. La deuxième vision consiste à utiliser une partition basée sur les inter-voxels. Pour encoder les voxels, ce type de partitionnement induit la définition de pointels et de lignels. Dans ce cas, les fourmis vont se déplacer entre les pointels suivant les lignels. Ils vont renforcer les lignels présentant une forte chance d'être un contour de l'image tout en résistant toujours à une perturbation du système (évaporation). Ce sont ces lignels qui fourniront les voxels appartenant aux contours de l'image. Finalement, la modélisation Orientée Objet du SMA est décrite.

Le chapitre suivant présente l'ensemble des résultats obtenus pour chaque type de SMA et par leur combinaison.

Chapitre 4

Expérimentations

Dans ce chapitre, nous allons montrer les résultats tirés d'expérimentations produites à l'aide de nos méthodes de segmentation basées sur les SMA (Moussa et al., 2011a). Pour pouvoir estimer la pertinence de ces résultats, nous les avons comparés avec des algorithmes classiques en utilisant des critères tels que la corrélation, la spécificité et la sensibilité de la segmentation produite par rapport à la vérité terrain. Pour ce faire, nous avons choisi d'utiliser des images obtenues à l'aide d'un simulateur d'acquisition d'images IRM (BrainWeb) car il est difficile d'obtenir des résultats objectifs par comparaison avec une segmentation manuelle. En effet, la précision du placement des contours est dépendante de l'expérience de l'utilisateur. De plus, le fait d'utiliser un simulateur permet de donner des premiers résultats en s'affranchissant de l'erreur inter- et intra-observateur.

Dans cette partie, nous commencerons donc par présenter BrainWeb et les images sur lesquelles nous allons faire nos comparaisons, puis les critères que nous utiliserons et les méthodes auxquelles nous nous comparerons. Enfin, nous présenterons nos résultats et discuterons de la pertinence de nos algorithmes.

4.1 Les images de test

BrainWeb contient une base de données d'images cérébrales représentant un ensemble de volumes de données IRM réalistes produites par un simulateur (Collins et al., 1998). Ces données peuvent être utilisées par la communauté de neuro-imagerie pour évaluer la performance des différentes méthodes d'analyse d'images dans un contexte où la vérité est connue. Les volumes de données ont été simulés pour diverses séquences d'acquisition (T1, T2, *etc*) et une variété d'épaisseurs de coupe, de niveau de bruit, et d'inhomogénéité. Le bruit ajouté par BrainWeb correspond au modèle mathématique de l'acquisition IRM *i.e.* il suit la loi de Rayleigh pour le fond et la loi de Rice dans les régions de signal. Le pourcentage de bruit représente le ratio de l'écart-type du bruit blanc gaussien par rapport au signal d'un tissu de référence. Dans le cadre de nos expérimentations, nous avons choisi des volumes T1

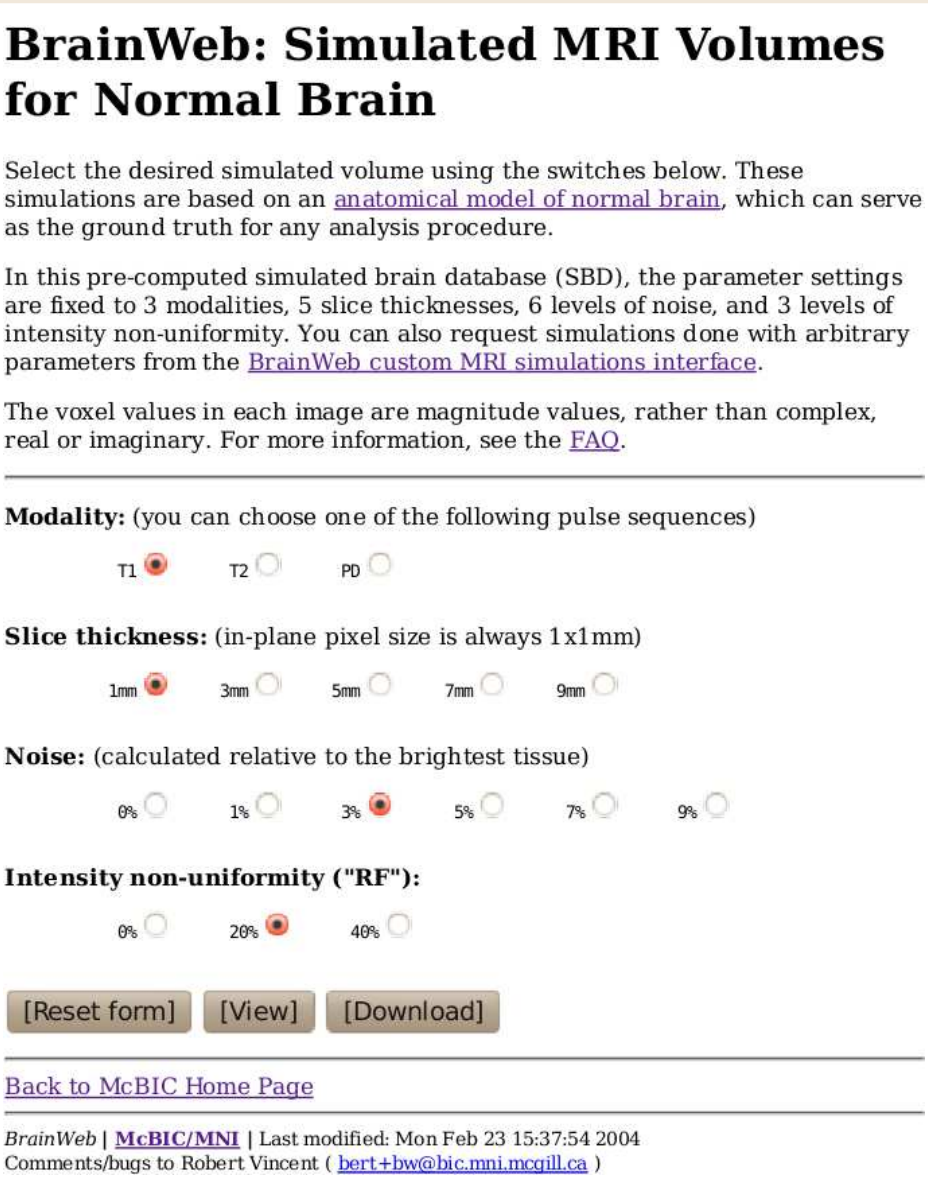
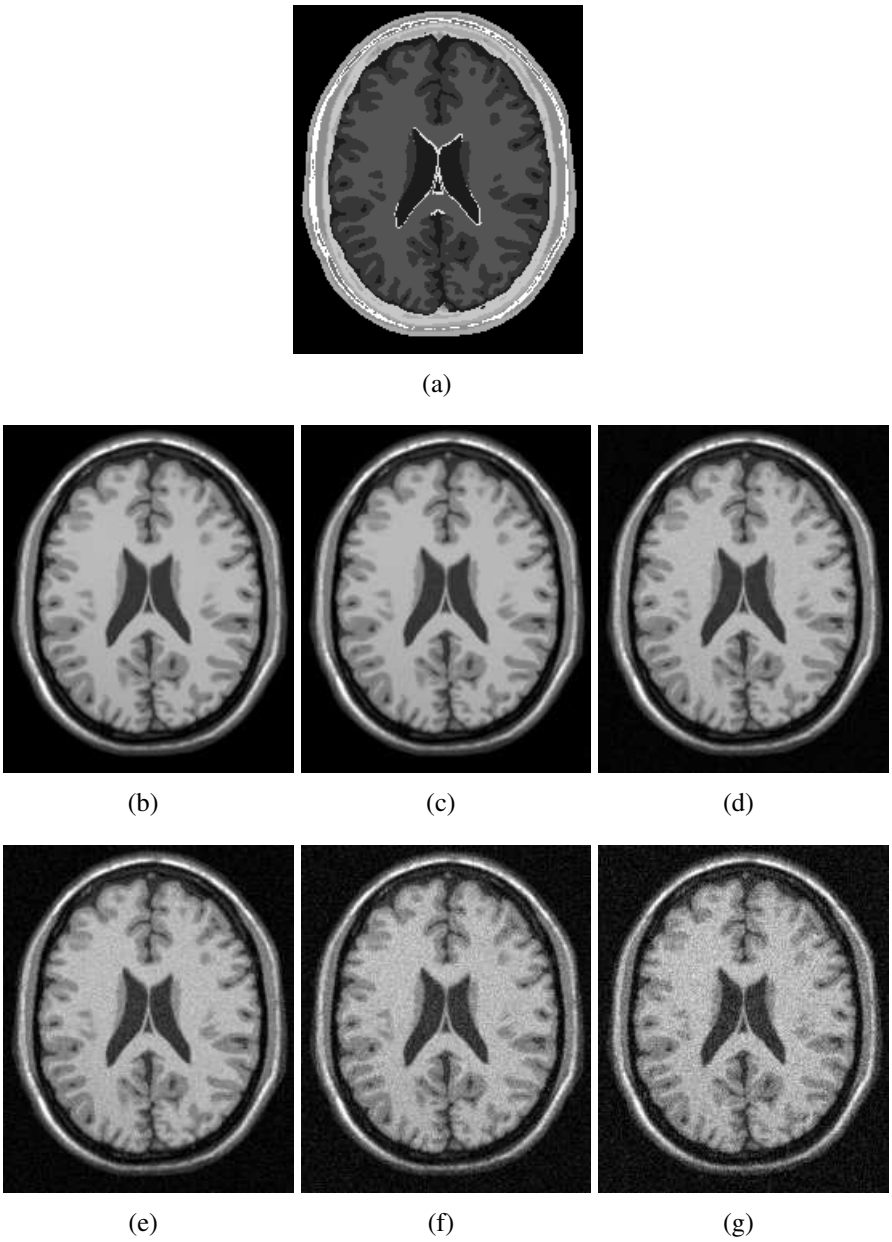


FIGURE 4.1: Interface du simulateur BrainWeb : choix des paramètres.

(séquence anatomique classique) pour lesquels l'épaisseur de coupe est de 1mm (taille utilisée en routine clinique). Le choix du pourcentage de bruit proposé par BrainWeb est compris entre 0% et 9%. L'intégralité de ces pourcentages de bruit a été utilisé dans nos expérimentations pour étudier la résistance de nos algorithmes au bruit. Pour l'inhomogénéité, nous avons laissé la valeur par défaut de 20% ce qui correspond à une acquisition réaliste en application clinique. En résumé, les cerveaux simulés ont les caractéristiques suivantes :

- Séquence : T1.
- Épaisseur de coupe : 1mm.
- Bruit : X% ($0 \leq X \leq 9$).
- Inhomogénéité : 20%.
- Dimensions : 181 x 217 x 181 voxels (données par le simulateur).
- Codage : 2 octets dont 12 bits significatifs (*i.e.* 4096 niveaux de gris).

La figure 4.2 représente ce qui sera considéré comme la "vérité terrain" à savoir le résulat d'une image segmenté de cerveau en régions et les différents fantômes [1] avec le bruit variant entre 0% et 9 %. Ces informations sont ensuite être utilisées pour valider le résultat de nos segmentations.



(a)

(b) (c) (d)

(e) (f) (g)

FIGURE 4.2: La coupe 94, représentant les différents "fantômes" utilisés : (a) La vérité terrain, (b) 0% de bruit, (c) 1% de bruit, (d) 3% de bruit, (e) 5% de bruit, (f) 7% de bruit, (g) 9% de bruit.

BrainWeb propose un cerveau segmenté en 10 régions où les niveaux de gris représentent des étiquettes. Nous illustrons la segmentation de ces 10 régions par la figure 4.2(a) et le tableau 4.1.

L'ordinateur de test ayant servi pour nos expérimentations est équipé de 2 processeurs Intel Xeon (4 coeurs à 2.53GHz), 16 Go de RAM et 4 GPUs avec 2 Go de mémoire chacun. Le système d'exploitation de cet ordinateur est un noyau Linux x86_64 2.6.21. En raison de la taille importante de l'environnement ainsi que du nombre d'agents traités et du nombre de pas de temps, nous avons

1. Cerveaux simulés dont les différents volumes sont connus précisément.

Eléments	Pourcentage	Forme
Fond	42.2%	
LCR	5.2%	
MG	12.7%	
MB	9.5%	
Graisse	2.1%	

Eléments	Pourcentage	Forme
Peau / muscle	8.7%	
Peau	10.2%	
Crâne	5.1%	
Matière gliale	0.1%	
Conjonction	4.2%	

TABLE 4.1: "Fantôme" BrainWeb : Les différents éléments de la partition.

décidé de mettre en œuvre les approches sur GPU afin d'avoir un temps de calcul réduit. Le temps de calcul ne pourra pas donc pas être pris en compte pour la comparaison des méthodes de segmentation car les méthodes basées sur les SMA sont codées sur GPU (voir annexe A) et les autres sur CPU. Dans la suite de ce travail, la coupe présentée dans les captures d'écrans représente la $94^{ème}$ coupe de l'image 3D. Ce choix de représentation a été fait car cette coupe contient les 10 régions de la partition fournie par BrainWeb.

4.2 Les méthodes de comparaison

Dans ce qui suit, nous allons présenter les méthodes de segmentation classiques avec lesquelles nous allons nous comparer. Pour l'approche des colonies d'araignées, les comparaisons vont nous permettre de déterminer si elle conduit à une bonne segmentation par rapport à deux méthodes :

- une méthode de classification par seuillage (voir section 2.2.2.1) : la méthode d'Otsu multi-niveaux ;
- une méthode basée régions (voir section 2.2.2.3) : la méthode de croissance de régions.

Pour les approches des fourmis, nous comparons les résultats de segmentation avec un détecteur de contours classique : l'opérateur de Sobel.

4.2.1 Otsu multi-niveaux

La méthode que nous avons utilisée est une méthode de seuillage multi-niveaux basée sur la méthode de Otsu (Otsu, 1979). Elle consiste à déterminer, pour un nombre de régions donné, les valeurs optimales des différents seuils en se basant sur la variance des subdivisions ainsi créées. La méthode de base consiste à séparer le premier plan du fond. On cherche dans ce cas le seuil optimal pour partager les voxels en deux régions. Pour un seuil t, il est possible de calculer la variance

inter-classes $\sigma^2(t)$. Cette mesure est obtenue à partir des intensités moyennes μ_1, μ_2 et μ des classes [0 ; t], [t + 1 ; L] et [0 ; L] où L représente l'intensité maximale. L'équation 4.1 montre le calcul de $\sigma^2(t)$ représentant la proportion de voxels dans la classe [0 ; t] et [t+1 ; L] par rapport au nombre total de voxels.

$$\sigma^2(t) = w_1(t) \times (\mu_1(t) - \mu)^2 + w_1(t) \times (\mu_2(t) - \mu)^2 \tag{4.1}$$

Otsu montre que le seuil optimal t^* est obtenu pour une variance inter-classe maximale. La méthode consiste donc à calculer cette variance pour tous les seuils possibles (t ∈ 1, . . ., L - 1) et d'en déterminer la valeur maximale.

Cette méthode s'étend facilement au calcul de M classes avec M-1 seuils $\{ t_1, t_2, \ldots, t_{M-1} \}$ avec $(t_1 < t_2 < \ldots < t_{M-1})$ (Liao et al., 2001). La variance inter-classes est alors définie de la façon suivante :

$$\sigma^2(t_1, \ldots, t_{M-1}) = \sum_{k=1}^{M} w_k \times (\mu_k - \mu)^2 \tag{4.2}$$

où w_k représente la proportion de voxels dans la classe $[t_{k-1} ; t_k]$ tel que $t_0 = 0$ et $t_M = L$, μ_k l'intensité moyenne de cette même classe et μ l'intensité moyenne de la classe [0 ; L].

On calcule pour chaque M-1-uplet de seuils la variance inter-classes correspondante. Les seuils optimaux, $(t_1^*, \ldots, t_{M-1}^*)$, correspondent à la valeur maximale de la variance inter-classes.

L'algorithme 11 présente l'implémentation de Liao et al. pour le calcul de plusieurs seuils à partir de l'histogramme de l'image.

La complexité de cet algorithme est O(K× (L-K-1) × (K-1)) où :

1. L est le nombre de niveaux de gris dans l'image ;
2. K est le nombre de seuils à trouver.

4.2.2 Croissance de régions

Cette technique est probablement la technique la plus simple. Elle consiste à extraire une zone d'intérêt d'une image en faisant croître une région à partir d'un ou de plusieurs germes constituant un sous-ensemble de la zone recherchée. À partir de cet ensemble de germes initiaux, cette méthode agrège les voxels suivant un double critère : l'homogénéité et l'adjacence. Cette agrégation de voxels est contrôlée par un prédicat (généralement, une fonction booléenne). La croissance est régie par un critère de propagation pouvant être basé sur des propriétés colorimétriques, mais également géométriques ou topologiques. Le prédicat peut avoir plusieurs formes tel que :

- la variance σ_R^2 des niveaux de gris de l'image associés aux points de la région R est inférieure à un seuil préfixé ;
- la proposition α_R de points dont le niveau de gris se situe hors de l'intervalle [m_R - α_R, m_R + α_R] (où m_R est la valeur moyenne des niveaux de gris), ne dépasse pas un seuil préfixé ;

Algorithme

Pour chaque région, le processus de croissance comprend une phase d'initialisation et une phase itérative. On suppose que l'ensemble des objets de l'image a été partiellement partitionné en N régions.

Algorithme 11 Seuillage multi-niveaux (Liao et al., 2001)

Entrées: Histo: Histogramme de l'image, K $\in \mathbb{N}$, L $\in \mathbb{N}$
Sorties: l: liste de seuils $\in \mathbb{N}$

l ← ListeVide().
P ← matrice triangulaire de dimension L
S ← matrice triangulaire de dimension L
H ← matrice triangulaire de dimension L
P[0][0] ← Histo[0]
S[0][0] ← Histo[0]
H[0][0] ← Histo[0]
Pour i ∈ 1, . . ., L **Faire**
    P[0][i] = P [0][i-1] + Histo [i]
    S[0][i] = S [0][i-1] + i × Histo[i]
    H[0][i] = $\frac{(S[0][i])^2}{P[0][i]}$
Fin Pour
Pour i ∈ 1, . . ., L **Faire**
    Pour j ∈ 1, . . ., L **Faire**
        P[i][j] = P[0][j] - P[0][i-1]
        S[i][j] = S[0][j] - S[0][i-1]
        H[i][j] = $\frac{(S[i][j])^2}{P[i][j]}$
    Fin Pour
Fin Pour
TantQue Taille(l) != K-1 **Faire**
    max ← 0
    Pour chaque d = (t_1, . . ., t_{k-1}) tel que 0 < t_1 < . . . < t_{k-1} < L **Faire**
        somme ← h [0][t_1]
        Pour i ∈ 1, . . ., k-1 **Faire**
            somme ← somme + h [t_i + 1][t_{i+1}]
        Fin Pour
        somme ← somme + h [t_k][L]
        Si somme > max **Alors**
            max ← somme
            l ← Ajouter(d)
        FinSi
    Fin Pour
Fin TantQue

La phase itérative modifie la région R_i ($1 \leq i \leq N$) représentant l'objet segmenté. Le processus de croissance de région étant itératif, le contenu de la région R_i à l'itération n est notée $R_i^{(n)}$.

Initialisation des germes

L'initialisation de la croissance de région nécessite le positionnement de germes désignant la ou les premières régions. Bien entendu, le point ou la zone germe doit faire partie de la région recherchée, sinon on risque d'obtenir une segmentation partiellement, voire totalement erronée. Ces germes constituent donc la région initiale $R_i^{(0)}$. Le choix d'un germe $R_i^{(0)}$ peut se faire par seuillage sur un attribut (par exemple le niveau de gris), ou de toute autre manière (extraction de formes géométriques, *etc.*). Un germe peut éventuellement être réduit à un seul point.

Construction des régions

Suite à la phase d'initialisation, un processus itératif de déformation ajoute progressivement des points situés à la périphérie de la région en train de croître, s'ils respectent le prédicat. Ces nouveaux points sont regroupés dans l'ensemble que l'on appelle couronne $S^{(k)}$. Considérons une croissance de région ne faisant croître qu'une seule région. Étant donné une région initiale $R^{(0)}$, la région suivante $R^{(k+1)}$ est obtenue par union de la région courante $R^{(k)}$ et de la couronne $S^{(k)}$. Dans la croissance de région, la condition d'agglomération implique la définition d'un terme de similarité entre un point candidat et la région segmentée. Ce terme de similarité, appelé critère, est utilisé par le prédicat pour décider de l'ajout ou non d'un voxel.

Condition d'arrêt Comme l'itération porte sur une région croissante et bornée, la convergence est assurée au bout d'un nombre fini d'itérations. Le processus de croissance peut s'arrêter selon deux conditions:

- toutes les régions satisfont le prédicat et la segmentation comprend N régions ;
- il existe N-1 régions qui vérifient le prédicat, la $N^{ème}$ région comprend les points ne vérifiant pas le prédicat.

Algorithme 12 Croissance de régions

Entrées: Voxels: matrice de points $\in \mathbb{N}^3$, $Height \in \mathbb{N}$, $Width \in \mathbb{N}$, $Depth \in \mathbb{N}$ et Seuil $\in \mathbb{N}$
Sorties: Régions: liste de régions

```
1:  Régions ← ListeVide().
2:  Pour k allant de 0 à Depth Faire
3:     Pour j allant de 0 à Height  Faire
4:        Pour i allant de 0 à Width Faire
5:           Si Voxels[i][j][k] n'est dans aucune région Alors
6:              AccroîtreRégion(Voxels, Voxels[i][j][k], Régions, Seuil).
7:           FinSi
8:        Fin Pour
9:     Fin Pour
10: Fin Pour
```

La complexité de cet algorithme est de $O(N^2 \times V)$ où :

Algorithme 13 Création d'une région

Entrées: Voxels: matrice de points $\in \mathbb{N}^3$, Germe $\in \mathbb{N}$, Régions : Liste de régions et $Seuil \in \mathbb{N}$

```
1: candidats ← ListeVide().
2: région ← ListeVide() ;
3: candidats.ajouter(Germe)
4: TantQue candidats != ∅ Faire
5:     c ← retirer un voxel de candidats
6:     région.ajouter(c)
7:     Pour chaque voisin v de c Faire
8:         Si  NON(v ∈ région) ET | Intensité(v) - Intensité(Germe)| < Seuil Alors
9:             candidats.ajouter(v)
10:        FinSi
11:    Fin Pour
12: Fin TantQue
13: Régions.ajouter(région)
```

1. N est la taille de l'image.
2. V est la taille du voisinage considérée.

4.2.3 Opérateur de Sobel

Cet opérateur vise à détecter des contours dans l'image pour effectuer la segmentation. Les contours sont formés par les frontières des régions. Ils sont l'un des principaux indices de distinction visuelle de deux régions. L'algorithme correspondant se déroule en deux étapes:

1. les contours locaux sont détectés en utilisant une certaine forme de différenciation ;
2. ces contours locaux sont regroupés ensuite pour former des frontières séparant les voxels de la région désirée des autres voxels de l'image.

Cet opérateur part de l'hypothèse qu'un voxel appartient à une frontière entre deux objets s'il existe une forte variation d'intensité dans son voisinage. L'approximation de la variation locale peut se faire par exemple à l'aide d'un opérateur de convolution défini sur le voisinage du voxel. L'opérateur de Sobel a été défini en 2D et s'adapte facilement en 3D. Sa définition en 2D utilise deux noyaux de convolution de taille 3×3, un pour évaluer les variations selon la direction x :
et l'autre pour évaluer les variations selon la direction y :

$$\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

Cette formulation en 2D de l'opérateur est facilement extensible en 3D par l'utilisation de 6 noyaux de convolution de taille $3 \times 3 \times 3$ pour évaluer la variation suivant la direction x, la direction y, la direction z, la direction xy, la direction xz et la direction yz.

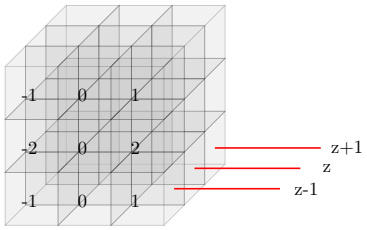


FIGURE 4.3: La représentation du premier masque de l'opérateur de Sobel en 3D.

$$\underset{\text{z-1}}{\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}} \quad \underset{\text{z}}{\begin{pmatrix} -2 & 0 & 2 \\ -4 & 0 & 4 \\ -2 & 0 & 2 \end{pmatrix}} \quad \underset{\text{z+1}}{\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}}$$

$$\underset{\text{z-1}}{\begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}} \quad \underset{\text{z}}{\begin{pmatrix} 2 & 4 & 2 \\ 0 & 0 & 0 \\ -2 & -4 & -2 \end{pmatrix}} \quad \underset{\text{z+1}}{\begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}}$$

$$\underset{\text{z-1}}{\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}} \quad \underset{\text{z}}{\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}} \quad \underset{\text{z+1}}{\begin{pmatrix} -1 & -2 & -1 \\ -2 & -4 & -2 \\ -1 & -2 & -1 \end{pmatrix}}$$

$$\underset{\text{z-1}}{\begin{pmatrix} -2 & -1 & -2 \\ -1 & -4 & -1 \\ -2 & -1 & -2 \end{pmatrix}} \quad \underset{\text{z}}{\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}} \quad \underset{\text{z+1}}{\begin{pmatrix} 2 & 1 & 2 \\ 1 & 4 & 1 \\ 2 & 1 & 2 \end{pmatrix}}$$

$$\underset{\text{z-1}}{\begin{pmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{pmatrix}} \quad \underset{\text{z}}{\begin{pmatrix} -4 & -2 & 0 \\ -2 & 0 & -2 \\ 0 & 2 & 4 \end{pmatrix}} \quad \underset{\text{z+1}}{\begin{pmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{pmatrix}}$$

L'algorithme utilisant cet opérateur s'exécute en trois phases. Tout d'abord, le gradient est calculé pour tous les voxels de l'image. Ensuite, une opération de seuillage est utilisée pour déterminer quels sont les voxels correspondant à des contours. Enfin, une attribution d'une coloration aux contours et une autre au fond.

$$\underset{\text{z-1}}{\begin{pmatrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{pmatrix}} \quad \underset{\text{z}}{\begin{pmatrix} 0 & -2 & -4 \\ 2 & 0 & -2 \\ 4 & 2 & 0 \end{pmatrix}} \quad \underset{\text{z+1}}{\begin{pmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{pmatrix}}$$

4.3 Critères de comparaisons

Dans cette section, nous allons décrire les critères de comparaison que nous avons choisi pour valider les segmentations que nous avons effectuées.

4.3.1 Pour la segmentation en régions

Pour comparer les méthodes de segmentation en régions, nous avons besoin d'établir des critères que nous utiliserons sur toutes les images de test. Nous comparons les résultats sur plusieurs points :

1. le nombre de régions ;
2. la corrélation de Spearman (R_s) avant et après le post-traitement entre la vérité terrain et le résultat de la segmentation.

Le nombre de régions est un critère permettant de déterminer la fiabilité de la segmentation. Nous ajoutons au nombre total des régions le nombre de régions ayant une taille non significative. Pour les méthodes classiques de segmentation, nous considérons une région comme non significative si sa taille est inférieure à 10 voxels.

Pour comparer les segmentations en régions, nous avons opté pour un test de corrélation. Ce test basé sur le calcul du coefficient de corrélation de Spearman (R_s) qui permet de préciser l'existence d'une relation entre deux variables quantitatives et son intensité (Dalgaard, 2002). Cette relation est non-linéaire et monotone.

Deux hypothèses sont définies par ce test :

1. Hypothèse nulle : "$H_0 : R_s = 0$"
2. Hypothèse alternative : "$H_1 : R_s$ est différent de 0"

Comme dans tout test non-paramétrique, et à la différence des tests paramétriques, le calcul ne porte pas sur les valeurs numériques des mesures issues des échantillons représentatifs des populations, mais sur leurs rangs attribués suite au classement des valeurs par ordre croissant. Ainsi, nous nous affranchissons des conditions de normalité des distributions et d'homogénéité des variances indispensables à la fiabilité des tests paramétriques.

Nous considérons une image comme la réalisation d'une variable aléatoire X pour l'image "vérité terrain" et d'une variable Y pour l'image segmentée. Pour calculer ce coefficient, nous commençons par classer séparément les valeurs de la variable aléatoire X et celles de la variable Y. Supposons qu'il existe une liaison linéaire positive entre X et Y, le calcul des rangs permettrait de se rendre compte que les sujets qui auraient les plus petites valeurs de X auraient également les plus petites valeurs de Y, et inversement les sujets qui auraient les plus grandes valeurs de X auraient également les plus grandes valeurs de Y. En revanche, s'il n'existe aucune liaison linéaire entre X et Y, les sujets qui auraient

Image segmentée \ Vérité terrain	Contour	Fond
Contour	VP	FP
Fond	FN	VN

TABLE 4.2: VP (Vrais Positifs) sont les voxels de contours qui ont été bien segmentés, FN (Faux Négatifs) sont les faux contours, VN (Vrais Négatifs) sont les voxels du fond qui ont été bien détectés et FP (Faux Positifs) sont les voxels qui ont été des voxels de contours et sont devenus des voxels de fond.

les plus petites valeurs de X auraient des valeurs de Y éparses dans leur classement. Ce coefficient s'obtient de la manière suivante :

$$R_s = \frac{d_i^2}{N \times (N^2 - 1)} \tag{4.3}$$

Où d_i représente la différence des rangs au niveau de l'observation i et N la taille de l'échantillon. Ce coefficient est proche de 1 s'il y a une forte corrélation entre les deux échantillons testés et inversement, il est proche de 0 si la corrélation est faible.

4.3.2 Pour l'extraction de contours

Pour pouvoir évaluer les approches de détection de contours, nous jugeons la segmentation obtenue par rapport à des estimateurs classiques (Huguier and Flahault, 2003). Ces critères reposent sur la mesure de la sensibilité et la spécificité. Ces mesures sont définies comme suit (voir table 4.2 pour la définition des termes Vrais Positifs (VP), Faux Négatifs (FN), Vrais Négatifs (VN) et Faux Positifs (FP) dans notre cas) :

Définition 23 *La **sensibilité** (SE) est la proportion des Vrais Positifs par rapport à l'ensemble des contours (Vrais Positifs et Faux Négatifs) qui devraient être segmentés :*

$$SE = \frac{VP}{VP + FN} \tag{4.4}$$

Ce critère permet d'évaluer la capacité à détecter tous les contours.

Définition 24 *La **spécificité** (SP) est la proportion des Vrais Négatifs par rapport à l'ensemble des contours qui ne devraient pas être segmentés (Vrais Négatifs et Faux Positifs) :*

$$SP = \frac{VN}{VN + FP} \tag{4.5}$$

Ce critère permet d'évaluer dans quelle mesure des voxels n'appartenant pas aux contours y auraient été intégrés.

Ces deux critères s'interprète de la même façon que le test de corrélation, la valeur est proche de 1 pour montrer la pertinence de la segmentation et proche de 0 pour montrer sa non pertinence.

4.4 Résultats sur la segmentation en régions : méthode des araignées

La méthode des araignées a été testée avec les paramètres suivants (*cf.* section 3.1.4.2):

1. NbAgent = 1 000 000 (valeur empirique dépendante de la taille de l'image (13%)) ;
2. Couleur et Sélectivité (calculés automatiquement à partir de l'histogramme de l'image comme vu au chapitre précédant dans la section 3.1.4.2) ;
3. Saturation = 100 (valeur empirique 1% de la taille de chaque colonie).

Les paramètres d'attraction ($Attraction_{MaColonie}$, $Attraction_{AutresColonies}$) et le nombre de pas de temps (condition d'arrêt) sont évalués empiriquement à partir des résultats des différentes simulations ($0.5 \leq Attraction_{MaColonie} \leq 1$) (*cf* figure 4.4).

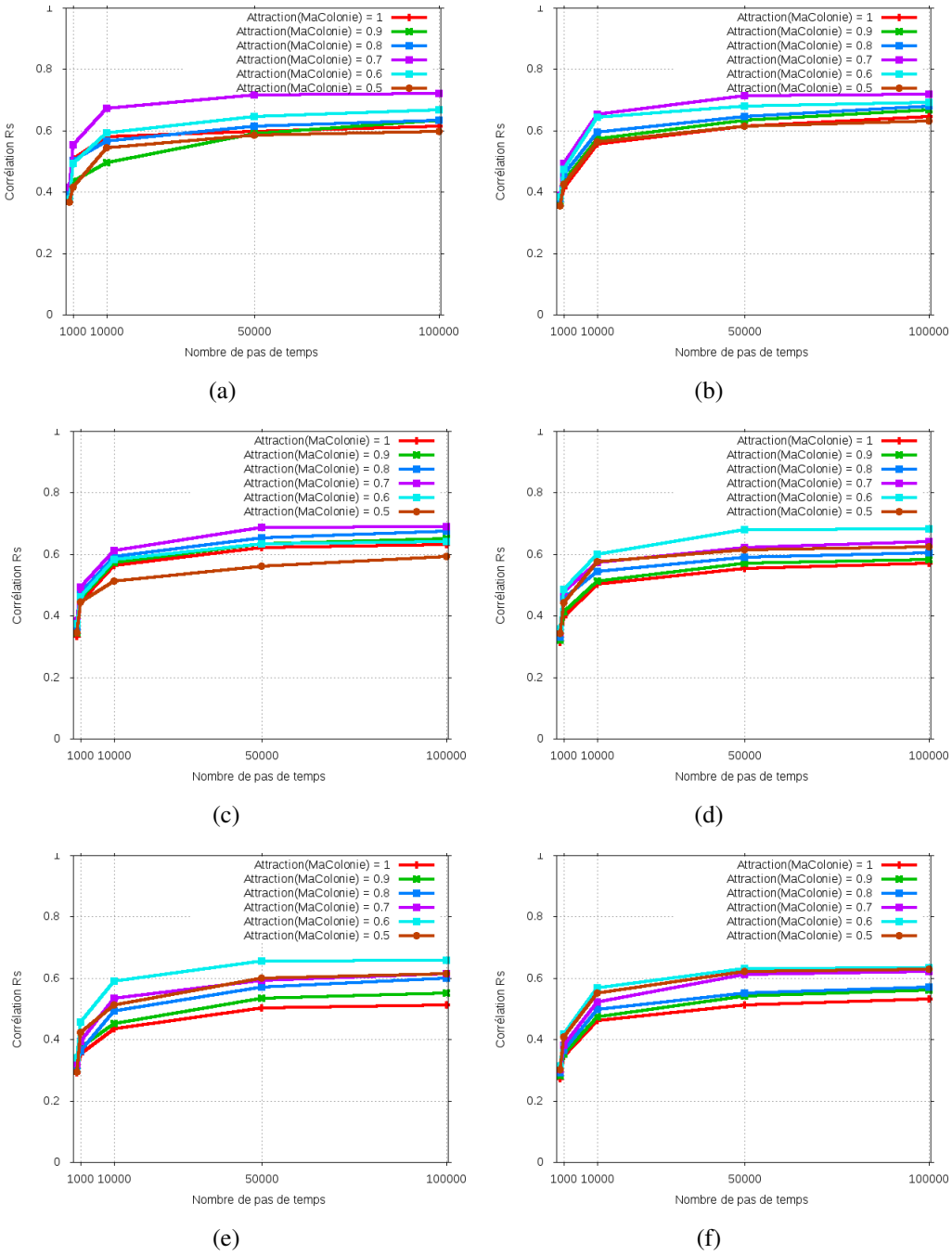


FIGURE 4.4: La combinaison la plus optimale pour la sélection de paramètres pour différents niveaux de bruit : (a) 0%, (b) 1%, (c) 3%, (d) 5%, (e) 7%, (f) 9%.

La figure 4.4 montre que le système a commencé à converger à partir de l'itération 50 000. C'est cette valeur qui sera retenue pour la comparaison des résultats. La valeur 100 000 n'a pas été retenue, malgré sa corrélation un peu plus élevé, puisqu'elle engendre un temps de calcul beaucoup plus important pour un gain de précision négligeable par rapport à 50 000.

Maintenant que nous avons défini le pas de temps choisi, nous allons regarder le paramétrage de $Attraction_{MaColonie}$ et $Attraction_{AutresColonies}$. Nous voyons que ce paramétrage est dépendant du pourcentage de bruit présent dans l'image, les paramètres optimaux pour les niveaux de bruit entre 0% et 3%, $Attraction_{MaColonie} = 0.7$ et $Attraction_{AutresColonies} = 0.3$, ont la meilleure corrélation. Cette probabilité ($Attraction_{AutresColonies} = 0.3$) d'être attiré par les fils des autres colonies provient du fait qu'il y a un petit nombre de voxels en compétition. Lorsque le bruit est entre 5% et 9%, les paramètres optimaux deviennent $Attraction_{MaColonie} = 0.6$ et $Attraction_{AutresColonies} = 0.4$. Ceci provoque une forte compétition entre les agents de colonies différentes pour taguer les voxels. Notons que pour 9% de bruit, une valeur de $Attraction_{MaColonie} = 0.5$ amène à une corrélation très proche de la solution optimale retenue. Lorsque le niveau de bruit augmente, la probabilité $Attraction_{AutresColonies}$ croît et la probabilité $Attraction_{MaColonie}$ décroît jusqu'à une probabilité de 0.5 (MaColoniet 50% *ie.* tirage complètement aléatoire). C'est la cause des artéfacts dont le bruit.

Le tableau 4.3 présente les résultats des différentes techniques de segmentation appliquées sur une image du cerveau avec différents niveaux de bruit. L'évaluation quantitative de ces résultats est donnée dans le tableau 4.4 et sur la figure 4.5.

Après avoir sélectionné le nombre de pas et les valeurs du comportement interne de chaque agent-araignée ($Attract_{MaColonie}$ et $Attract_{AutresColonies}$). Nous allons analyser qualitativement les résultats obtenus avec les différentes méthodes de segmentation. La méthode des araignées présente les meilleurs résultats lorsque le bruit augmente (tableau 4.3). En particulier, l'interface MG/MB voit sa forme préservée.

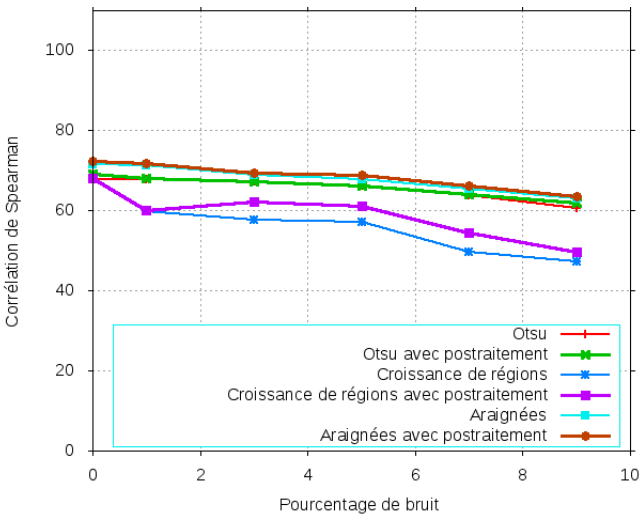


FIGURE 4.5: Segmentation en régions : corrélation de Spearman.

Le nombre de régions que fournit la méthode des araignées est celui qu'a donné l'utilisateur pour détecter les composantes du cerveau. Le nombre de composant étant à 10, nous obtenons 11 régions

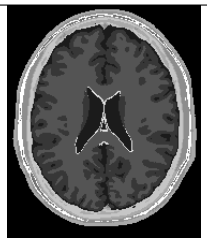
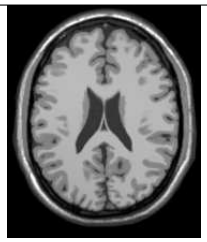
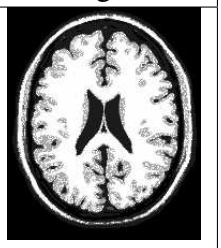
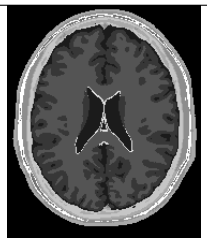
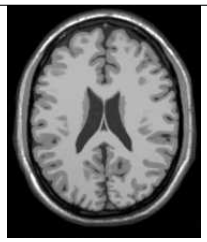
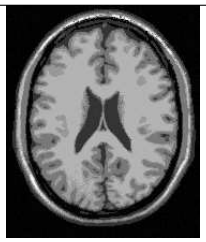
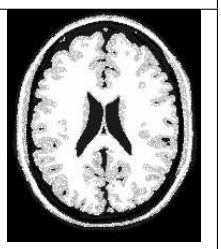
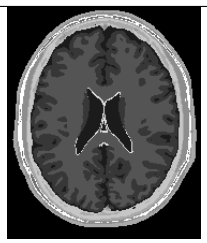
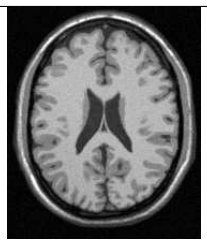
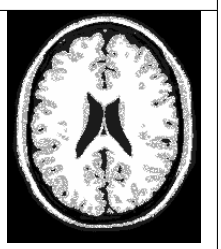
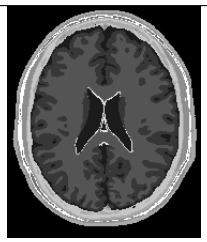
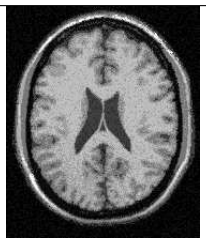
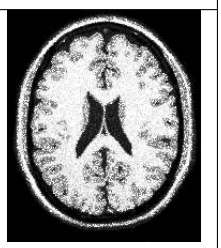
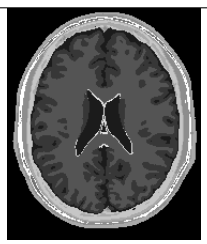
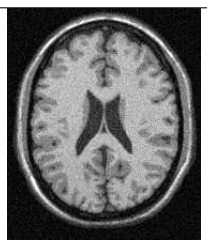
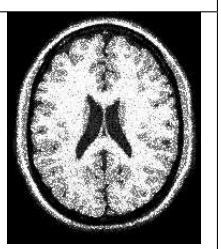
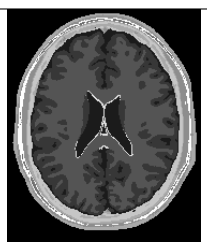
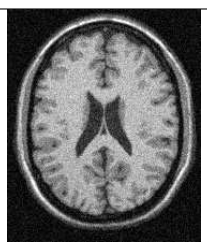
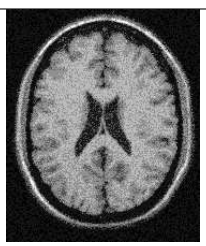
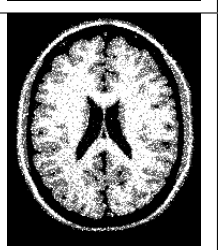
Bruit	Vérité terrain	Image originale	Otsu	Croissance de régions	Araignées
0 %					
1 %					
3 %					
5 %					
7 %					
9 %					

TABLE 4.3: Résultats qualitatifs : segmentation d'images avec différentes valeurs de bruit.

avec une région supplémentaire représentant un petit nombre de voxels non tagué. Ce nombre de régions est le résultat d'une légère sur-segmentation.

Les méthodes classiques ont toujours sur-segmenté l'image, quel que soit le niveau de bruit, en particulier avec de petites régions (régions < 10 voxels). Cette sur-segmentation est le résultat de leur sensibilité au bruit. Malgré le fait que le post-traitement des méthodes classiques a mené à minimiser le nombre de régions dans l'image, leurs résultats montrent toujours une forte sur-segmentation de

% de bruit		Otsu	Croissance de régions	Colonies d'araignées
0 %	**Régions**	917	70231	11
	Régions > 10 voxels	203	1006	11
	R_s	67.91 %	68.02 %	71.63 %
	R_s^p	68.19 %	68.20 %	72.3 %
1 %	**Régions**	1150	135093	11
	Régions > 10 voxels	205	1609	11
	R_s	67.90 %	59.90 %	71.35 %
	R_s^p	68.01 %	60.13 %	71.7 %
3 %	**Régions**	2503	224653	11
	Régions > 10 voxels	198	4890	11
	R_s	67.17 %	57.63 %	68.76 %
	R_s^p	67.25 %	62.16 %	69.32 %
5 %	**Régions**	9669	241482	11
	Régions > 10 voxels	209	3483	11
	R_s	66.11 %	57.23 %	67.93 %
	R_s^p	66.32 %	61.18 %	68.82 %
7 %	**Régions**	23138	249995	11
	Régions > 10 voxels	317	4673	11
	R_s	63.83 %	49.73 %	65.46 %
	R_s^p	64.21 %	54.54 %	66.33 %
9 %	**Régions**	42392	186376	11
	Régions > 10 voxels	330	2935	11
	R_s	60.69 %	47.13 %	63.11 %
	R_s^p	61.95 %	49.73%	63.65 %

TABLE 4.4: Résultats quantitatifs : segmentation d'images avec différentes valeurs de bruit.

l'image.

Le test de corrélation avant et après post-traitement, R_s et R_s^p, a donné les meilleurs résultats dans tous les cas pour la méthode des araignées. Ces résultats montre la résistance de notre méthode aux différents artéfacts de l'image et plus particulièrement au bruit car on peut voir dans le tableau 4.4 que plus le bruit augmente plus l'écart entre nos résultats et ceux des méthodes classiques est grand.

4.5 Résultats sur la segmentation contours : modèles des fourmis

La méthode des fourmis possède les paramètres suivants (*cf.* section 3.2.5) :

1. NbAgent = 1 000 000 (valeur empirique dépendante de la taille de l'image (13%)) ;
2. le seuillage de Sobel ainsi que celui des fourmis se fait de la même façon (seuillage à 10% de l'intensité maximale) ;
3. Q = 1 (valeur empirique désignant le nombre de fois qu'une concentration de phéromones doit être déposé par chaque agent-fourmi, la valeur par défaut est 1 ce qui notre cas).

Le coefficient de simulation ρ et le nombre de pas de temps sont obtenus par la configuration optimale observée sur la figure 4.6. Comme pour la méthode des araignées, nous avons procédé par expérimentation pour la sélection des paramètres optimaux. La figure 4.6 montre que le système basé sur un partitionnement en voxels a commencé à converger à partir de l'itération 5 000. C'est cette

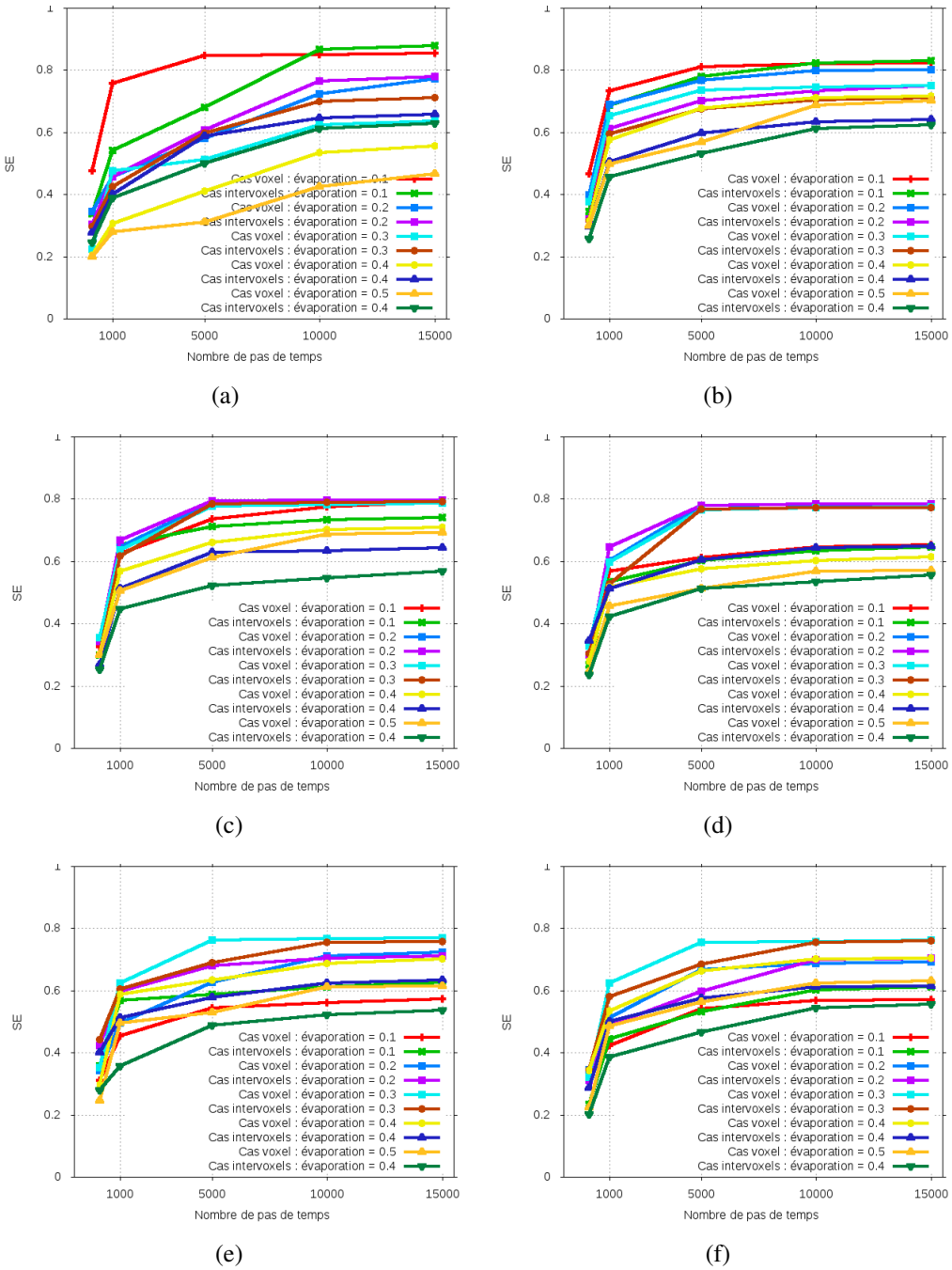


FIGURE 4.6: La combinaison la plus optimale pour la sélection de paramètres pour différents niveaux de bruit : (a) 0%, (b) 1%, (c) 3%, (d) 5%, (e) 7%, (f) 9%.

valeur qui sera retenue pour la comparaison des résultats. Les valeurs 10 000 et 15 000 n'ont pas été retenues, malgré leur valeur de sensibilité un peu plus élevé, puisqu'elle engendre un temps de calcul beaucoup plus important pour un gain de précision négligeable par rapport à un nombre de pas 5 000. Pour le modèle basé sur un partitionnement voxels, le système a commencé à se stabiliser à partir de l'itération 10 000. Ceci s'explique par le nombre important de lignels à traverser pour créer les contours de l'image. De même, nous avons obtenu une meilleure SE pour un nombre de pas de 15 000 mais cet avantage reste minime en comparaison du coût en temps de calcul engendré.

Maintenant que nous avons défini le pas de temps choisi, nous allons regarder le paramétrage de l'évaporation ρ. Nous voyons que ce paramètre est lié au pourcentage de bruit. Ainsi, le paramètre optimal pour les niveaux de bruit de 0% et 1%, $\rho = 0.1$, a obtenu les meilleures sensibilité/spécificité. Cette valeur faible d'évaporation est due à une faible valeur de bruit présent dans l'image. Lorsque le bruit est de 3% ou 5%, la valeur de l'évaporation devient un peu plus importante ($\rho = 0.2$) et permet de réduire les nuisances liées au bruit. Le facteur d'évaporation continue sa croissance ($\rho = 0.3$) lorsque les images sont plus bruitées (7% et 9%).

Le tableau 4.5 présente les résultats des différentes techniques de segmentation appliquées sur une image du cerveau avec différents niveaux de bruit. L'évaluation quantitative de ces résultats est donnée dans le tableau 4.6 et la figure 4.7.

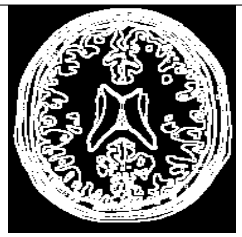
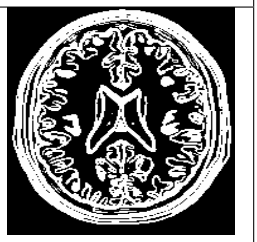
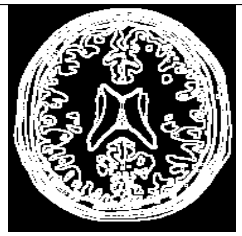
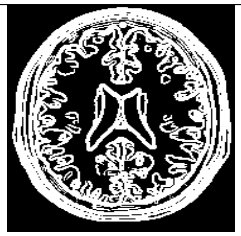
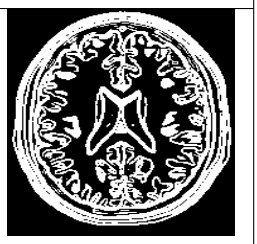
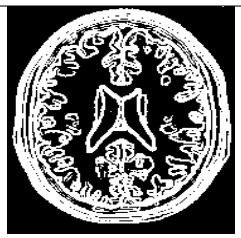
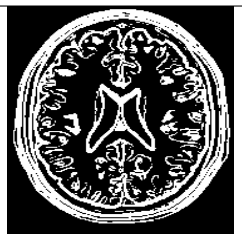
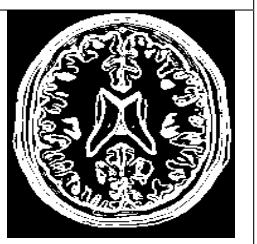
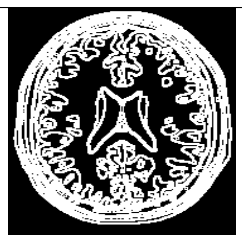
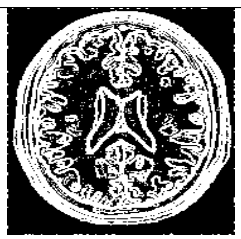
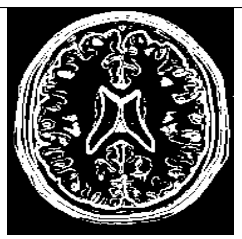
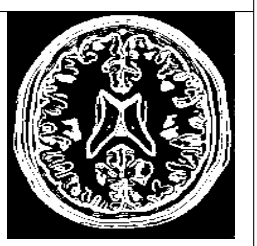
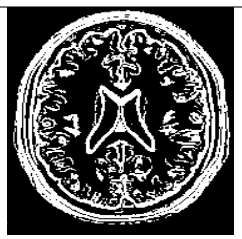
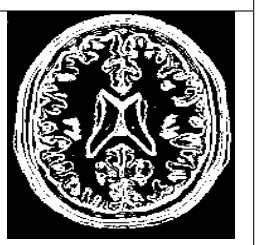
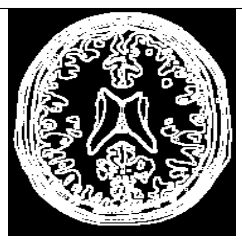
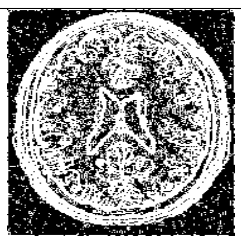
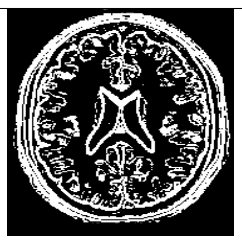
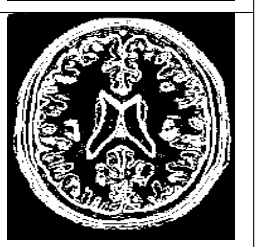
Bruit	Vérité terrain	Sobel	Fourmi-voxel	Fourmi-intervoxel
0%				
1%				
3%				
5%				
7%				
9%				

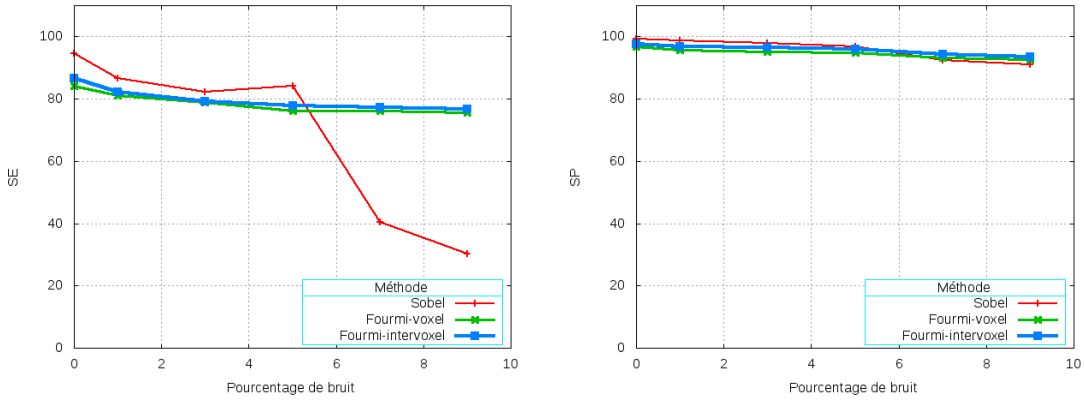
TABLE 4.5: Résultats qualitatifs : détection de contours avec différents valeurs de bruit.

Les résultats présentés dans le tableau 4.5 montre que l'opérateur de Sobel devient de moins en moins pertinent lorsque le bruit augmente. La méthode des fourmis basée sur le partitionnement en voxels montre une meilleure résistance au bruit avec une détection d'un peu moins de contours, tandis que la méthode des fourmis basée sur le partitionnement inter-voxels est aussi résistante au bruit mais avec la particularité d'une meilleure détection de contours.

Comme le montre le tableau 4.6 et la figure 4.7, l'opérateur de Sobel devient peu fiable à partir de 7% de bruit (SE < 50%). Ces méthodes des fourmis ont donné les meilleurs SE avec en plus une

% de bruit	Critère	Sobel	Fourmi-voxel	Fourmi-intervoxel
0 %	**SE**	94.5%	84.2%	86.7%
	SP	99.2%	96.8%	97.6%
1 %	**SE**	86.6%	81.1%	82.2%
	SP	98.8%	95.8%	96.7%
3 %	**SE**	82.2%	78.9%	79.3%
	SP	97.8%	95.2%	96.5%
5 %	**SE**	84.1%	76.3%	77.9%
	SP	96.8%	94.9%	96.1%
7 %	**SE**	40.5%	76.1%	77.2%
	SP	92.4%	93.2%	94.3%
9 %	**SE**	30.2%	75.5%	76.8%
	SP	91.1%	92.8%	93.6%

TABLE 4.6: Résultats quantitatifs : détection de contours avec différentes valeurs de bruit.



(a) (b)

FIGURE 4.7: Détection de contours : (a) sensibilité (SE), (b) spécificité (SP).

meilleure détection des contours et la meilleure résistance au bruit pour les agents-fourmis utilisant un partitionnement inter-voxels.

La spécificité nous permet de voir la fiabilité de ces méthodes pour détecter le fond de l'image. L'opérateur de Sobel devient peu fiable à partir de 7% de bruit. Comme pour la SE, le choix du partitionnement inter-voxels a mené à une SP meilleure que celle d'un partitionnement classique (voxels).

Remarquons que pour aucune de ces méthodes un post-traitement n'a été effectué (fermeture des contours notamment). Tous ces éléments nous permettent d'affirmer que les méthodes des fourmis ont une meilleure détection que Sobel pour les valeurs de bruit les plus importantes (7% et 9%).

4.6 Conclusion

Dans ce chapitre, nous avons présenté la procédure d'évaluation de nos méthodes. Nous avons commencé par introduire le choix des images utilisées, puis, nous nous sommes focalisés sur les

critères de validation que ce soit pour la détection de régions ou la détection de contours. Nous avons choisi une sélection de méthodes classiques avec lesquelles nous nous sommes comparées. Les résultats des expérimentations nous ont permis de valider nos méthodes que ce soit qualitativement ou quantitativement tout en passant par le choix des paramètres ayant donné les résultats optimaux. Cette validation nous a montré la résistance de nos méthodes au bruit, résistance meilleure que celle des méthodes classiques.

Nous tenons à souligner, par ailleurs, que bien qu'un certain nombre de paramètres reste à définir : nombre de colonies, nombre d'agents, évaporation ou saturation, nous avons à chaque fois fourni une méthode qui permet de déterminer des plages de valeurs acceptables et ce à partir d'une simple analyse de l'image : taille, histogramme, niveau de bruit. Nous n'avons pas voulu dans ce chapitre donner de temps de calcul car il n'est pas correct intellectuellement de comparer "basiquement" des temps de calcul séquentiel (sur le CPU) avec des temps de calcul parallèle (sur le GPU). Le temps de calcul est un des reproches majeurs adressés aux SMA. En utilisant des solutions GPU, nous avons à chaque fois pu ramener les segmentations à des temps de calcul de l'ordre de quelques minutes sans pour autant rencontrer de difficultés majeures. La mise en œuvre de cette parallélisation est présentée en annexe.

Ce travail réalisé au sein de l'équipe Image et Son du LaBRI [2] connue pour son savoir-faire en traitement d'images médicales par des méthodes s'appuyant sur des modèles classiques, a ouvert de nouvelles perspectives en abordant la segmentation d'images avec avec des SMA. En ce qui concerne les SMA, nous avons montré qu'une "bonne" segmentation émerge de la combinaison des deux types de segmentation : segmentation régions réalisée par des agent-araignées et segmentation contours par des agent-fourmis. Pour l'instant, ces opérations sont appliquées indépendamment. Il pourrait être intéressant d'écrire une application qui réaliserais les deux traitements en parallèle, à condition d'imaginer un mode de communication entre les araignées et les fourmis. Enfin, les développements technologiques avec les cartes GPU et la mémoire partagée, créent un climat favorable aux développements des simulations SMA en permettant le passage à l'échelle de traitement de problèmes de taille réelle comme le montre nos applications sur les images IRM 3D.

2. Laboratoire Bordelais de Recherche en Informatique.

Conclusion

Dans ce mémoire, nous avons avons abordé la problématique de la segmentation d'images opérée par les Systèmes Multi-Agents. Notre objectif était de concevoir des approches de segmentation pour des images IRM 3D de cerveau. Ces approches de segmentation doivent être résistantes aux différents artéfacts présents dans l'image et plus particulièrement au bruit induit par des phénomènes physiques et physiologiques lors de l'acquisition de l'image. Nos approches ont été conçues pour détecter les régions et les contours de l'image. Pour remplir ces objectifs, nous avons conçu une méthode inspirée des comportements des araignées et des fourmis. Tout d'abord, les araignées, regroupées sous forme de plusieurs colonies, ont pour objectif de détecter les différents éléments du cerveau. Cette tâche est réalisée par le placement de fils au cours du temps sur des voxels qualifiés *"d'intéressants"*. Dans la mesure où il y a toujours plusieurs régions à détecter, nous avons utilisé le concept de colonies d'araignées, chacune ayant la tâche de détecter une région spécifique. Ce sont ensuite les informations portées par les fils qui vont construire les régions de l'image segmentée. Pour leur part, les fourmis ont pour objectif de détecter les contours de l'image. Pour réaliser cette tâche, l'environnement dans lequel les fourmis se déplacent est partitionné de deux façons : un partitionnement voxel et un partitionnement intervoxel. Avec un partitionnement voxel, les fourmis vont se déplacer de voxel en voxel et dans l'autre cas, les fourmis vont se déplacer sur des lignels entre les pointels. Elles vont déposer des phéromones sur les positions considérées comme des éléments des contours de l'image. Ce dépôt de phéromone est pondéré par une *"évaporation"* à chaque pas de temps. Les phéromones qui ont persisté jusqu'à la fin de la simulation marqueront les contours de l'image.

Nous avons évalué nos approches de segmentation en les comparant à des approches classiques. Pour les araignées, nous avons comparées les résultats aux approches de croissance de régions et de seuillage multi-niveaux d'Otsu. Ces comparaisons ont utilisé le test de corrélation de Spearman avec lequel nous avons évalué si l'étiquetage des régions réalisé par les araignées correspondait à la *"vérité terrain"* (fournie par l'application BrainWeb sur nos images de test). Pour les fourmis, nous avons effectué une mesure de sensibilité et de spécificité et nous avons comparé ces mesures à celles obtenues avec un opérateur classique de détection de contours : l'opérateur de Sobel en 3D.

L'analyse des résultats de l'approche des araignées a montré que le comportement interne des agents de chaque colonie joue un rôle très important pour l'obtention d'une segmentation satisfaisante en terme de résistance au bruit. Nous avons pu voir que la probabilité d'attraction d'une araignée pour les fils de sa propre colonie est dépendante de la quantité de bruit présent dans l'image, les meilleurs résultats étant obtenus avec une probabilité inversement proportionnelle au pourcentage

du bruit. Pour toutes les segmentations effectuées, l'approche des araignées a toujours obtenu une meilleure corrélation que les approches classiques quelque soit le niveau de bruit considéré et on a pu constater que l'algorithme de croissance de régions est très peu résistant au bruit. Quant à l'algorithme de seuillage multi-niveaux d'Otsu, bien que compétitif, il donne toujours de moins bon résultats que la méthode des araignées. Il est à noter que dans notre implémentation de la méthode des araignées, nous avons de plus proposé une méthode d'automatisation de la recherche des valeurs des paramètres liés à l'image. Dans le modèle fourmis, nous avons identifié que le niveau de bruit pouvait être pris en compte par la valeur *"évaporation"* qui s'applique à chaque itération. Plus le bruit augmente, plus *"l'évaporation"* doit être importante. La mesure de sensibilité montre que les fourmis offrent une meilleure détection de contours que l'opérateur de Sobel à partir de 7 % de bruit. Cette information indique qu'il est recommandé d'utiliser nos approches lorsque le niveau de bruit est important, ce qui est le cas des images IRM de cerveau. Grâce au partitionnement intervoxel, nous avons obtenu une mesure de spécificité meilleure que la méthode de Sobel à partir de 7 % de bruit. En effet, grâce à ce partitionnement, nous détectons moins de faux contours (Faux Positifs) que Sobel.

Nous n'avons pas inclus le temps de calcul comme élément de comparaison entre nos méthodes et les méthodes classiques car les programmes dont nous disposions pour ces méthodes étaient essentiellement séquentielles et implémentées sur CPU et donc pas réellement comparables aux programmes parallèles. Nos temps de calculs sur GPU restent toujours de l'ordre de quelques minutes, ils sont donc parfaitement utilisables sur des images de taille réelle. Toutefois, il est toujours intéressant d'essayer d'optimiser les temps de simulation. Nous avons remarqué que la répartition aléatoire des agents sur l'environnement au moment de l'initialisation du système est un des facteurs pénalisant du temps de simulation car ces agents vont avoir un temps d'inactivité important lors de la recherche du premier voxel à taguer dans le cas des araignées. Quant aux fourmis, elles vont taguer des points non intéressants en premier et engendrer une activité inutile pour le comportement global du système. Pour résoudre ces problèmes, une solution consiste à placer des nids lors de la phase d'initialisation. Pour les araignées, ces nids vont représenter un des voxels à détecter pour chaque colonie présente dans l'image. Pour les fourmis, les nids vont représenter des points de contours à détecter en priorité. La prochaine version du programme devrait intégrer cette possibilité.

Une des perspectives plus large de ce travail est une intégration dans un schéma général d'analyse d'images développé par notre équipe. Ce schéma repose sur un processus de division/fusion et nous pourrions y intégrer de la manière suivante :

1. En effectuant une segmentation avec les araignées comme une première opération de division de l'image en régions. On obtiendrait ainsi une première partition de l'image avec des contours imprécis. Ces contours pourraient être ensuite raffinés par des fusions successives de régions qui seraient guider par les informations délivrées par les araignées (fils provenant de différentes colonies).
2. En effectuant une segmentation avec les fourmis qui pourrait apporter des informations sur la qualité des contours détectés. En effet, lorsque le poids d'un contour est faible, on pourrait alors favoriser la fusion des régions séparées par ces contours.

En conclusion, nous pouvons espérer que cette approche de segmentation par les Systèmes Multi-

Agents deviendra un des éléments supplémentaire de la boite à outils de l'analyse d'images tant pour l'imagerie médicale pour toute analyse d'images complexes.

Bibliographie

S. M. Ali and R. M. Zimmer. The Question Concerning Emergence. In *Computing Anticipatory Systems: CASYS - First International Conference, D.M. AIP Conference Proceedings 437*, pages 138–156, 1998.

W. Ashby. Principles of the self-organizing dynamic system. *The Journal of General Psychology*, 37 (2):125–128, 1947.

H. Atlan. *Entre le cristal et la fumée: essai sur l'organisation du vivant*. Collection Points: Série Sciences. Éditions du Seuil, 1986. ISBN 9782020093620.

P. Ballet. *Intérêts mutuels des systèmes multi-agents et de l'immunologie. Application à l'immunologie, l'hématologie et au traitement d'images*. PhD thesis, Université de Bretagne Occidentale, Jan. 2000.

F. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent Systems with JADE*. Wiley, 2007.

N. Benamrane and S. Nassane. *Medical image segmentation by a multi-agent system approach*, volume 4687. LNAI, 2007.

J. Besag. On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society*, B-48: 259–302, 1986.

J. Bezdek, M. Pal, J. Keller, and R. Krisnapuram. *Fuzzy Models and Algorithms for Pattern Recognition and Image Processing*. Kluwer Academic Publishers, Norwell, MA, USA, 1999. ISBN 0792385217.

B. Bhanu, S. Lee, and J. Ming. Adaptive image segmentation using a genetic algorithm. *IEEE Transactions On Systems Man And Cybernetics*, 25(12):1543–1567, 1995.

O. Boissier, S. Gitton, and P. Glize. caractéristiques des systèmes et des applications. In y. Demazeau, editor, *Systèmpes multi-agents*, volume 29 of *ARAGO*, pages 25–54. Editions TEC DOC, février 2004. OFTA.

E. Bonabeau and G. Theraulaz. *Auto-organisation et comportements collectifs : la modélisation des sociétés d'insectes*, pages 91–140. Editions Hermes, 1997.

C. Bourjot, V. Chevrier, and V. Thomas. A new swarm mechanism based on social spiders colonies : from web weaving to region detection. *Web Intelligence and Agent Systems*, 1:47–64, 2003.

J.-P. Braquelaire and L. Brun. Image segmentation with topological maps and inter-pixel representation. *Journal of Visual Communication and Image representation*, 9(1):62–79, 1998.

L. Breiman, W. Meisel, and E. Purcell. Variable Kernel Estimates of Multivariate Densities. *Technometrics*, 19(2):135–144, 1977. ISSN 00401706.

H. Cao, P. Huang, and S. Luo. *A novel image segmentation algorithm based on artificial ant colonies*, volume 4987. LNCS, 2008.

V. Chevrier. Contributions au domaine des systemes multi-agents. Technical report, Université Henry-Poincarré - Nancy 1, 2002. URL `http://www.loria.fr/~chevrier/Publi/HDR.zip`.

V. Chevrier. Tisser des toiles d'araignées sociales. *Pour la Science*, pages 62–64, 2006.

S. Cho, D. Jones, W. E. Reddick, R. J. Ogg, and R. G. Steen. Establishing norms for age-related changes in proton t1 of human brain tissue in vivo. *Magn Reson Imaging*, 15(10):1133–43, 1997. ISSN 0730-725X.

M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *Evolutionary Computation, IEEE Transactions on*, 6(1):58–73, 2002.

M. L. College. Global optimization and meta-heuristics, 2002.

D. L. Collins, A. P. Zijdenbos, V. Kollokian, J. G. Sled, N. J. Kabani, C. J. Holmes, and A. C. Evans. Design and construction of a realistic digital brain phantom. *IEEE Trans Med Imaging*, 17(3): 463–468, June 1998. ISSN 0278-0062.

F. D. Croce, R. Tadei, and G. Volta. A genetic algorithm for the job shop problem. *Computers and Operations Research*, 22(1):15 – 24, 1995. ISSN 0305-0548.

P. Dalgaard. *Introductory statistics with R*. Statistics and computing. Springer, 2002. ISBN 9780387954752.

T. De Wolf. *Analysing and engineering self-organising emergent applications*. PhD thesis, Informatics Section, Department of Computer Science, Faculty of Engineering, May 2007. Berbers, Yolande and Holvoet, Tom (supervisors).

T. De Wolf and T. Holvoet. Emergence and self-organisation: a statement of similarities and differences. In *Eds), Lecture Notes in Artificial Intelligence*, pages 96–110. SpringerVerlag, 2004.

M. Dempster, U. of Waterloo. School of Urban, and R. Planning. *A self-organizing systems perspective on planning for sustainability*. University of Waterloo, 1998. URL `http://books.google.com/books?id=Uh4fcgAACAAJ`.

B. Denis De Senneville, P. Desbarats, B. Quesson, and C. Moonen. Real-time artefact corrections for quantitative mr temperature mapping. In *Journal of WSCG*, volume 11, No.1, pages 87–94. WSCG, 2003.

T. Denœux. A k-nearest neighbor classification rule based on dempster-shafer theory. *IEEE Trans. on Systems, Man and Cybernetics*, 25(05):804–813, 1995.

R. Descartes. *Discours de la méthode: pour bien conduire sa raison et chercher la vérité dans les sciences*. Hachette, 1856.

M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Trans. on Systems, Man and Cybernetics-Part B*, 26(1):29–41, 1996.

A. Drogoul. *De la simulation multi-agents à la résolution collective de problèmes*. PhD thesis, Université Paris VI, 1994.

E. Duchesnay, J. . Montois, Y. Jacquelet, and A. Kinie. An agent-based implementation of irregular pyramid for distributed image segmentation. In *Emerging Technologies and Factory Automation, 2001. Proceedings. 2001 8th IEEE International Conference on*, volume 1, pages 409–415, 2001.

E. Duchesnay, J.-. Montois, and Y. Jacquelet. Cooperative agents society organized as an irregular pyramid: A mammography segmentation application. *Pattern Recognition Letters*, 24(14):2435–2445, 2003.

J. C. Dunn. A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters. *Journal of Cybernetics*, 3(3):32–57, 1973.

J. Ferber. *Les systèmes multi-agents : vers une intelligence collective*. Informatique, Intelligence Artificielle. InterÉditions, 1995.

M. Fisher. An application oriented guide to lagrangian relaxation. *Interfaces*, 1985.

M. Flynn. Some computer organizations and their effectiveness. *Computers, IEEE Transactions on*, C-21(9):948 –960, sept. 1972.

L. Germond, M. Dojat, C. Taylor, and C. Garbay. A cooperative framework for segmentation of mri brain scans. *Artificial Intelligence in Medicine*, 20(1):77–93, 2000.

P. P. Grassé. La reconstruction du nid et les coordinations interindividuelles chez bellicositermes natalensis et cubitermes sp. La theorie de la stigmergie: essai d'interpretation du comportement des termites constructeurs. *Insectes Sociaux*, 6:41–81, 1959.

R. M. Haralick and L. Shapiro. Image segmentation techniques. *Computer Vision, Graphics, and Image Processing*, 29(1):100 – 132, 1985. ISSN 0734-189X.

F. Harary and G. Gupta. Dynamic graph models. *Mathematical and Computer Modelling*, 25(7): 79–88, 1997.

R. Haroun, F. Boumghar, S. Hassas, and L. Hamami. *A massive multi-agent system for brain MRI segmentation*, volume 3446. LNAI, 2005.

A. L. G. Hayzelden and J. Bigham. Agent technology in communications systems: an overview. *Knowl. Eng. Rev.*, 14:341–375, December 1999. ISSN 0269-8889.

F. Heylighen. The science of self-organization and adaptivity. In *in: Knowledge Management, Organizational Intelligence and Learning, and Complexity, in: The Encyclopedia of Life Support Systems, EOLSS*, pages 253–280. Publishers Co. Ltd, 1999.

J. H. Holland. *Adaptation in natural and artificial systems*. MIT Press, Cambridge, MA, USA, 1992. ISBN 0-262-58111-6.

S. Horowitz and T. Pavlidis. Picture Segmentation by a Directed Split-and-Merge Procedure. In *Proc. Second Intern. Joint Conf. on Pattern Recognition*, pages 424–433, Aug. 1974.

P. Huang, H. Cao, and S. Luo. An artificial ant colonies approach to medical image segmentation. *Computer Methods and Programs in Biomedicine*, 92(3):267–273, 2008.

M. Huguier and A. Flahault. *Biostatistiques au quotidien*. Elsevier, 2003. ISBN 9782842994013.

Y. Jacquelet, J. . Montois, E. Duchesnay, and A. Kinie. Combinatorial pyramid transposed to behavioural space for object recognition process. In *Systems, Man and Cybernetics, 2002 IEEE International Conference on*, volume 1, pages 212–217, 2002.

J. James Odell. Agents and complex systems. *Journal of Object Technology*, 1(2):35–45, 2002.

N. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1:7–38, 1998. ISSN 1387-2532.

J. Juntu, J. Sijbers, D. Van Dyck, and J. Gielen. Bias field correction for mri images. *Computer Recognition Systems*, 30:1–8, 2005.

M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International Journal Of Computer Vision*, 1(4):321–331, 1988.

J. Kennedy and R. Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948, Aug. 2002.

V. Kovalevsky. Multidimensional cell lists for investigating 3-manifolds. *Discrete Applied Mathematics*, 125:25–43, 2003.

P. J. M. Laarhoven and E. H. L. Aarts, editors. *Simulated annealing: theory and applications*. Kluwer Academic Publishers, Norwell, MA, USA, 1987.

A. Lefohn, J. Cates, and R. Whitaker. Interactive, GPU-Based Level Sets for 3D Segmentation. In *MICCAI*, pages 564–572. Springer, 2003.

Y. Lespérance, H. Levesque, F. Lin, D. Marcu, R. Reiter, and R. Scherl. A logical approach to high-level robot programming — a progress report. In *Control Of The Physiscal World By Intelligent Systems: Papers From The 1994 AAAI Fall Symposium*, pages 79–85, 1994.

G. H. Lewes. *Problems of Life and Mind Vol 2*. Kegan Paul, Trench, Turbner, & Co., London, 1875.

P.-S. Liao, T.-S. Chen, and P.-C. Chung. A Fast Algorithm for Multilevel Thresholding. *Journal of Information Science and Engineering*, 17:713–727, 2001.

H. Liu. Two- and three-dimensional boundary detection. *CGIP*, 6(2):123–134, April 1977.

J. B. Macqueen. Some methods of classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

P. Maes. Artificial life meets entertainment: lifelike autonomous agents. *Commun. ACM*, 38:108–114, November 1995. ISSN 0001-0782.

I. N. Manousakas, P. E. Undrill, G. G. Cameron, and T. W. Redpath. Split-and-merge segmentation of magnetic resonance medical images: performance evaluation and extension to three dimensions. *Comput. Biomed. Res.*, 31:393–412, December 1998. ISSN 0010-4809.

T. McInerney and D. Terzopoulos. Deformable models in medical image analysis: a survey. *Medical Image Analysis*, 1(2):91 – 108, 1996. ISSN 1361-8415.

S. Mostefaoui, O. Rana, N. Foukia, S. Hassas, G. Di Marzo-Serugendo, C. Van Aart, and A. Karageorgos. Self-Organising Applications: A Survey, 2003.

R. Moussa, M. Beurton-Aimar, M., G. Savin, and P. Desbarats. Image segmentation using socials agents. 21 pages, Dec 2008.

R. Moussa, M. Beurton-Aimar, and P. Desbarats. On the use of social agents for image segmentation. In *International Conference on complex systems and applications (iccsa 2009)*, Le Havre, France, Jun 2009a. 5 pages.

R. Moussa, M. Beurton-Aimar, and P. Desbarats. Multi-agent segmentation for 3d medical images. In *Information Technology and Applications in Biomedicine, 2009. ITAB 2009. 9th International Conference on*, pages 1–5, nov. 2009b.

R. Moussa, M. Beurton-Aimar, and P. Desbarats. Image processing on mr brain images using social spiders. In F. Alkhateeb, E. Al Maghayreh, and I. Abu Doush, editors, *Modeling, Control, Programming, Simulations and Applications*, pages 45–68. Intech, 2011a. ISBN 978-953-307-174-9.

R. Moussa, M. Beurton-Aimar, and P. Desbarats. Towards noise robust edge detection using ants colonies in 3d medical mr images. *System*, pages 3–10, 2011b.

J. Müller and M. Pischel. An architecture for dynamically interacting agents. In *CoopIS*, pages 114–121, 1994.

G. Nicolis and I. Prigogine. *Self-Organization in Non-Equilibrium Systems*. Wiley, New York, 1977.

M. Omran, A. Salman, and A. Engelbrecht. Dynamic clustering using particle swarm optimization with application in image segmentation. *Pattern Analysis and Applications*, 8:332–344, 2006. ISSN 1433-7541.

S. Osher and J. A. Sethian. *Fronts propagating with curvature dependent speed [microform] : algorithms based on Hamilton-Jacobi formulations*. National Aeronautics and Space Administration, Langley Research Center, Hampton, Va. :, 1987.

N. Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man and Cybernetics*, 9(1):62–66, Jan. 1979.

H. V. Parunak. "Go to the ant": Engineering principles from natural multi-agent systems. *Annals of Operations Research*, 75(1):69–101, 1997. ISSN 0254-5330.

C. A. Petri. Introduction to general net theory. In W. Brauer, editor, *Advanced Course: Net Theory and Applications*, volume 84 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 1975. ISBN 3-540-10001-6.

D. Pham. Robust fuzzy segmentation of magnetic resonance images. In *Proceedings of the Fourteenth IEEE Symposium on Computer-Based Medical Systems*, CBMS '01, pages 127–, Washington, DC, USA, 2001. IEEE Computer Society.

D. L. Pham, C. Xu, and J. L. Prince. A Survey of Current Methods in Medical Image Segmentation. *Annual Review of Biomedical Engineering*, pages 315–338, 2000.

J. Rajapakse, J. Giedd, and J. Rapoport. Statistical approach to segmentation of single-channel cerebral mr images. *IEEE Trans. Med. Imaging*, 16(2):176–186, 1997.

A. Rao and M. Georgeff. An abstract architecture for rational agents. In C. Rich, W. Swartout, and B. Nebel, editors, *Proceedings of Knowledge Representation and Reasoning (KRgR-92)*, pages 439–442, 1992.

N. Richard, M. Dojat, and C. Garbay. Automated segmentation of human brain mr images using a multi-agent approach. *Artificial Intelligence in Medicine*, 30(2):153–175, 2004.

N. Richard, M. Dojat, and C. Garbay. Distributed markovian segmentation: Application to mr brain scans. *Pattern Recognition*, 40(12):3467–3480, 2007.

T. Ridler and S. Calvard. Picture thresholding using an iterative slection method. *Ieee Transactions On Systems Man And Cybernetics*, 8(Aug):630–632, 1978.

S. Ruan, C. Jaggi, J.-H. Xue, M.-J. Fadili, and D. Bloyet. Brain tissue classification of magnetic resonance images using partial volume modeling. *IEEE Trans. Med. Imaging*, pages 1179–1187, 2000.

S. Russell, P. Norvig, J. Candy, J. Malik, and D. Edwards. *Artificial intelligence: a modern approach*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996. ISBN 0-13-103805-2.

T. Sandholm. Distributed rational decision making, 1999.

B. Scherrer, M. Dojat, F. Forbes, and C. Garbay. Mrf agent based segmentation: Application to mri brain scans. In *Proceedings of the 11th conference on Artificial Intelligence in Medicine*, AIME '07, pages 13–23, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-73598-4.

J. Serra. *Image Analysis and Mathematical Morphology*. Academic Press, Inc., Orlando, FL, USA, 1983. ISBN 0126372403.

J. Sijbers, A. Den Dekker, A.-J. Den Dekker, P. Scheunders, and D. Van Dyck. Maximum-likelihood estimation of rician distribution parameters. *IEEE Transactions on Medical Imaging*, 17:357–361, 1998.

L. Steels. Towards a theory of emergent functionality. In *Proceedings of the first international conference on simulation of adaptive behavior on From animals to animats*, pages 451–461, Cambridge, MA, USA, 1990. MIT Press. ISBN 0-262-63138-5.

D. Terzopoulos, A. Witkin, and M. Kass. Constraints on deformable models:Recovering 3D shape and nonrigid motion. *Artificial Intelligence*, 36(1):91–123, Aug. 1988.

D. W. Thompson. *On Growth and Form*. Cambridge University Press, 1992.

P. Thompson and A. Arthur Toga. Detection, visualization and animation of abnormal anatomic structure with a deformable probabilistic brain atlas based on random vector field transformations. *Medical Image Analysis*, 1(4):271–294, 1997.

O.-D. Trier and A.-K. Jain. Goal-directed evaluation of binarization methods, 1995.

J. Wansapura, S. Holland, R. Dunn, and W. Ball. Nmr relaxation times in the human brain at 3.0 tesla. *Journal of Magnetic Resonance Imaging*, 9(4):531–538, 1999. ISSN 1522-2586.

D. Weyns and T. Holvoet. Architecture-centric software development of situated multiagent systems. In *Proceedings of the 7th international conference on Engineering societies in the agents world VII*, ESAW'06, pages 62–85, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 3-540-75522-5, 978-3-540-75522-7.

M. Wooldridge and N. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10:115–152, 1995.

J.-H. Xue, S. Ruan, B. Moretti, M. Revenu, and D. Bloyet. Knowledge-based segmentation and labeling of brain structures from mri images. *Pattern Recogn. Lett.*, 22:395–405, March 2001. ISSN 0167-8655.

X. Zenglin, H. Kaizhu, Z. Jianke, K. Irwin, and M. R.L. Kernel maximum a posteriori classification with error bound analysis. In M. Ishikawa, K. Doya, H. Miyamoto, and T. Yamakawa, editors, *Neural Information Processing, 14th International Conference, ICONIP 2007, Kitakyushu, Japan, November 13-16, 2007, Revised Selected Papers, Part I*, volume 4984 of *Lecture Notes in Computer Science*, pages 841–850. Springer, 2007. ISBN 978-3-540-69154-9.

S. Zucker and R. Hummel. A three-dimensional edge operator. *PAMI*, 3(3):324–331, May 1981.

Table des figures

1.1 Relation unissant l'environnement et l'ensemble des agents. 7
1.2 Principe de la réaction d'un agent réactif sur une environnement à partir de son cycle de vie. 9
1.3 L'émergence. 12
1.4 L'auto-organisation. 13
1.5 La stigmergie (De Wolf, 2007). 14

2.1 Le principe de la caractérisation des tissus de l'image. 21
2.2 Coupe du cerveau. 22
2.3 Exemples de seuillage simple et multiple sur deux histogrammes. 27
2.4 Segmentation extraite de (Haroun et al., 2005) : (a) image originale et (b) image segmentée en MB, MG et LCR. 34
2.5 Segmentation extraite de (Scherrer et al., 2007) : (a) image originale, (b) image segmentée décomposée en blocs et (c) l'histogramme de la représentation des tissus dans le bloc rouge. 34
2.6 Segmentation extraite de (Germond et al., 2000) : (a) image originale et (b) image segmentée en MB, MG et LCR et les contours les englobant. 35
2.7 Segmentation extraite de (Bourjot et al., 2003) : (a) image originale et (b) image segmentée en 4 régions. 35
2.8 Segmentation extraite de (Huang et al., 2008) : (a) image originale et (b) image segmentée en MB, MG et LCR. 36
2.9 Segmentation extraite de (Benamrane and Nassane, 2007) : (a) image originale et (b) extraction de contours par croissance de régions. 37
2.10 Segmentation extraite de (Richard et al., 2004) : (a) image originale décomposée en blocs, (b) image segmentée décomposée en blocs, (c) l'histogramme de l'image, (d) l'histogramme du bloc noir de l'image originale et (e) l'histogramme du bloc noir de l'image segmentée. 37

3.1 Chaine de segmentation en régions. 41
3.2 Cycle de vie des agents-araignées. 43
3.3 Sélection des informations colorimétries pour les colonies d'araignées : (a) Histogramme de l'image, (b) Histogramme lissé et sélection des maxima et minima locaux, (c) intervalles de recherche pour les différentes colonies. 48
3.4 Le voisinage d'un voxel : (a) 6-voisins, (b) 18-voisins et (c) 26-voisins. 50

3.5 Voisinage d'un voxel : (a) représentation de l'environnement, (b) représentation des fils, les voxels tissés représentent les nœuds en rouge. 51
3.6 Les étapes de la segmentation en régions : (a) image initiale, (b) image à jour par les informations des fils, (c) les informations des fils et (d) l'image segmentée. 56
3.7 L'extraction des voxels en compétition : (a) image segmentée et (b) image de compétition. 57
3.8 Chaine de segmentation en contours. 57
3.9 Les éléments inter-voxels : (a) voxel, (b) surfel, (c) lignel et (d) pointel. 61
3.10 Le codage inter-voxels 62
3.11 Voisinage d'un point : (a) cas d'un partitionnement en voxels : les voxels en gris sont les voxels non visités, les voxels verts sont les voxels phéromonés et le voxel en jaune est la dernière position d'où vient la fourmi, (b) cas du partitionnement inter-voxels : les lignels en rouges sont ceux non visités, les lignels en vert sont ceux déjà phéromonés et le lignel jaune représente la dernière voie traversé par la fourmi. 63
3.12 Calcul du poids de l'orientation. 64
3.13 Les différents cas d'orientation : (a) cas 1, (b) cas 2, (c) cas 3, (d) cas 4. 65
3.14 Représentation du calcul de la variation d'intensité sur un surfel. 65
3.15 L'extraction des voxels contours : (a) image origine et (b) image segmentée en contours. 67
3.16 L'extraction des voxels contours : (a) image origine, (b) les lignels phéromonés, (c) le graphe des lignels et (d) l'image segmentée en contours par les voxels représentants des lignels. 68
3.17 Ordonnancement des agents. 71
3.18 Architecture d'IPSiMAS. 72

4.1 Interface du simulateur BrainWeb : choix des paramètres. 77
4.2 La coupe 94, représentant les différents "fantômes" utilisés : (a) La vérité terrain, (b) 0% de bruit, (c) 1% de bruit, (d) 3% de bruit, (e) 5% de bruit, (f) 7% de bruit, (g) 9% de bruit. 78
4.3 La représentation du premier masque de l'opérateur de Sobel en 3D. 84
4.4 La combinaison la plus optimale pour la sélection de paramètres pour différents niveaux de bruit : (a) 0%, (b) 1%, (c) 3%, (d) 5%, (e) 7%, (f) 9%. 88
4.5 Segmentation en régions : corrélation de Spearman. 89
4.6 La combinaison la plus optimale pour la sélection de paramètres pour différents niveaux de bruit : (a) 0%, (b) 1%, (c) 3%, (d) 5%, (e) 7%, (f) 9%. 92
4.7 Détection de contours : (a) sensibilité (SE), (b) spécificité (SP). 95

A.1 Architecture d'un GPU. 115
A.2 Gestion des Block Thread. 115
A.3 Les performances : (a) araignées, (b) fourmis. 122

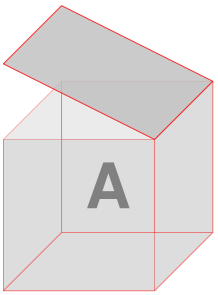
Liste des tableaux

3.1 Les paramètres de chaque colonie. 45

4.1 "Fantôme" BrainWeb : Les différents éléments de la partition. 79
4.2 VP (Vrais Positifs) sont les voxels de contours qui ont été bien segmentés, FN (Faux Négatifs) sont les faux contours, VN (Vrais Négatifs) sont les voxels du fond qui ont été bien détectés et FP (Faux Positifs) sont les voxels qui ont été des voxels de contours et sont devenus des voxels de fond. 86
4.3 Résultats qualitatifs : segmentation d'images avec différentes valeurs de bruit. 90
4.4 Résultats quantitatifs : segmentation d'images avec différentes valeurs de bruit. 91
4.5 Résultats qualitatifs : détection de contours avec différents valeurs de bruit. 94
4.6 Résultats quantitatifs : détection de contours avec différentes valeurs de bruit. 95

A.1 Organisation du kernel lancé sur le GPU en blocs de threads et grilles de blocks. Chaque thread d'un bloc a un identifiant (par exemple, coordonnée 1D) unique au bloc. De même chaque bloc de la grille a un identifiant unique. On peut accéder à un thread particulier connaissant la taille des blocs de la grille. Dans nos modèles, l'agent est le thread. 115
A.2 Les informations sur les mémoires du GPU. 116

Annexe



Implémentation

Les méthodes développées dans le chapitre précédent sont gourmandes en temps de calculs. Elles sont donc peu exploitées pour des applications de segmentation d'images volumineuses. Une façon de diminuer la durée des traitements consiste à paralléliser les simulations des méthodes.

La parallélisation consiste à exécuter des calculs de façon simultanée, en divisant un problème en sous problèmes, exécutés en parallèle sur des architectures matérielles compatibles. Elle s'oppose à la programmation séquentielle, qui consiste à exécuter des instructions les unes à la suite des autres sur un processeur. Lors de la parallélisation d'une application, il faut aussi bien considérer la méthode de parallélisation que la machine employée. La méthode de parallélisation dépend de l'application elle même, tandis que le choix du matériel doit être pris en compte selon son disponibilité et son prix.

A.1 Catégories de Parallélisation

Afin qu'une application bénéficie de la parallélisation, il est d'abord nécessaire d'identifier les portions de celle-ci susceptibles d'être parallélisées. L'application pourra alors être subdivisée en portions parallèles et séquentielles. Le traitement impliqué dans les portions parallèles doit ensuite subir une décomposition qui peut être effectuée de deux façons :

1. **Parallélisation de tâches :** Le traitement est lui-même décomposé en sous-tâches. Ces dernières seront exécutées sous formes de threads ou de processus. Elles seront réparties sur différentes unités de calcul par le système d'exploitation. Le temps d'exécution d'une portion parallèle est alors réduite au temps d'exécution de la tâche la plus longue.
2. **Parallélisme de données :** Dans le cas ou un même traitement est répété plusieurs fois sur différentes données, on peut alors distribuer ces données sur les unités de calcul et traiter chaque donnée en parallèle. On parle alors de parallélisation de données. Chaque processeur réalise la même tâche sur différents éléments d'un ensemble de données. La parallélisation de

données nécessite que les données soit distribuées, et n'est donc compatible qu'avec certaines applications. Un bon exemple d'application adaptée à ce type de parallélisme est l'addition de 2 matrices. Idéalement, chaque processeur se voit attribuer une case de la matrice et réalise l'opération d'addition. L'accélération du traitement ne dépend pas, dans ce cas, du traitement lui-même mais seulement de la taille des données et du nombre d'unités de calcul à disposition.

Selon les applications, on pourra paralléliser un traitement selon l'un de ces paradigmes, selon les deux, ou bien pas du tout si celui-ci est strictement séquentiel et que les données ne sont pas de nature distribuée.

Dans le cadre des SMA, les deux types de parallélisation peuvent être envisagés. Les systèmes comportant peu d'agents de types différents, comme ceux impliqués dans les architectures logicielles, peuvent bénéficier de la parallélisation de tâche. Si le système contient un nombre important d'agents au comportement similaire, comme dans notre cas, la parallélisation de données est plus adaptée. L'image constitue l'environnement des agents, c'est à dire les données à traiter, et les agents qui exécutent tous le même code sont répartis sur les processeurs.

A.2 Modèles de parallélisation

Pour choisir le modèle de parallélisation qui convient le mieux aux SMA, nous avons étudié la taxonomie de Flyn, une classification des architectures ordinateur selon le flux d'instructions et de données (Flynn, 1972). 4 architectures ont été proposées :

1. **SISD (Single Instruction, Single Data) :** son rôle est d'exécuter une instruction sur une donnée. Elle correspond aux ordinateurs mono-coeur ;
2. **SIMD (Single Instruction, Multiple Data) :** son rôle est d'exécuter l'instruction parallèlement sur plusieurs données. Elle correspond à un ordinateur qui utilise la parallélisation au niveau des données, comme celles équipés de GPU ;
3. **MISD (Multiple Instruction, Simple Data) :** son rôle est d'exécuter des instructions différentes sur la même donnée. Ce type d'architecture est rarement utilisé ;
4. **MIMD (Multiple Instruction, Multiple Data) :** son rôle est d'exécuter plusieurs instructions sur plusieurs données. Les processeurs sont alors autonomes et fonctionnent de manière asynchrone. Elle correspond à un ordinateur équipé d'un ou de plusieurs CPU multi-coeurs.

À partir des besoins des SMA, nous avons considéré que les deux architectures MIMD et SIMD sont les plus adaptés. Généralement, on distingue deux types de mémoires :

1. **Mémoire distribuée :** chaque processeur à son propre espace mémoire qui ne peut pas être accédé par les autres processeurs. Toute communication entre les processeurs doit alors passer par un réseau d'interconnexions. Par exemple, la biblithèque MPI (Message Parssing Interface) [1] est consacrée à des architectures qui utilisent ce type de mémoire;

1. http://www.mcs.anl.gov/research/projects/mpi/

2. **Mémoire partagée :** tous les processeurs partagent la même espace mémoire. Ils peuvent accéder à cette mémoire via un espace d'adressage global. L'usage de ces architectures nécessite de gérer les conflits d'accès à une même zone. Une mémoire partagée peut être encore émulée de façon logicielle sur une architecture à mémoire distribuée (par exemple, celle contenant un GPU). On parle ainsi de mémoire partagée virtuelle.

Le choix d'une mémoire partagée est lui essentiel pour nos modèles. En effet, les agents partagent le même environnement, et devront avoir un accès direct aux modifications faites par les autres agents sur l'environnement.

Nous avons choisi d'utiliser les GPU, architectures spécialisées dans le traitement de données graphiques et présents dans la plupart des ordinateurs de bureaux. Les GPU, qui disposent d'une mémoire partagée, sont de plus en plus utilisés dans le cadre de la parallélisation en raison de leur prix comparable à un CPU multi-coeurs et de leurs performances qui peuvent être jusqu'à 100 fois supérieures à ces mêmes CPU, selon l'application concernée.

A.3 Programmation GPGPU

La programmation GPGPU (General Purpose computing on Graphic Processing Units) consiste à utiliser les GPU, présents sur les cartes graphiques et initialement prévus pour effectuer des calculs liés au rendu graphique 3D. Ils sont utilisés dans des domaines comme le traitement audio/vidéo, les calculs scientifiques et les simulations. Ils peuvent tirer avantage du type de parallélisation massive lié au fonctionnement du GPU.

L'architecture des GPU est conçue pour exécuter des calculs massivement parallèles, selon le mode de fonctionnement SIMD présenté dans la section A.2. Ils sont composés de centaines d'unités de calcul capables d'exécuter un même traitement en parallèle. Ils montrent un intérêt pour les applications parallélisables par une partage de donnée. Leur utilisation présente de nombreux avantages :

1. De fortes possibilités d'accélérations de performances, variables selon les propriétés de l'application.
2. Organisation des processeurs du GPU autour d'une mémoire partagée.
3. Coût de plus en plus réduit par rapport à la plupart des autres architectures parallèles pouvant être envisagées.
4. Disponibilité sur la plupart des ordinateurs de bureau.

Le premier inconvénient d'une programmation GPU reste à nos jours la nécessité d'une réorganisation ou de la réécriture du code de l'application pour l'adapter au modèle de programmation GPU. Le deuxième inconvénient est la limitation de la taille de la mémoire présente sur le GPU (quelques GO). De plus les échanges de données entre les mémoires du GPU et du CPU doivent être réduits au minimum en raison du faible débit du bus PCI-Express qui les relient. Toutefois les GPU sont de plus en plus utilisés pour l'accélération d'applications grâce au développement de plateformes dédiées

OpenCL (Open Computing Language) [2] et CUDA (Computed Unified Device Architecture) [3], qui facilitent la programmation GPGPU.

CUDA

Elle est développée depuis 2007 par Nvidia. Elle est compatible avec tous les systèmes d'exploitation. Elle fournit un SDK, une API bas niveau (CUDA driver) et une API de haut niveau (CUDA Runtime). Des interfaces vers l'API Runtime sont disponibles dans de nombreux langages dont C++, Java, *etc*. Sa principal limitation est de permettre seulement la programmation des GPU de marque Nvidia. Elle permet de programmer la carte graphique via un langage dérivé du langage C avec des extensions/restrictions (C for CUDA). Certaines fonctionnalités du langage C++ ont été intégrées dans les dernières versions, permettant une programmation de haut niveau (classes, templates, *etc*) tout en restant linéaire en terme de stockage de données. Elle bénéficie de mises à jour régulières, et d'un grand nombre de documentation/tutoriels.

OpenCL

Initiée en 2009 par le Khronos group. Elle fournit une API et un langage de programmation des GPU dérivé de la norme C99. Chaque spécification OpenCL constitue un standard libre, que les différents constructeurs doivent suivre dans leur implémentation d'OpenCL. Son principal avantage est son aspect multi-architecture. En effet, elle permet de programmer aussi bien sur des GPU Nvidia qu'AMD. L'API OpenCL est similaire à l'API CUDA Runtime, et les mêmes interfaces peuvent être utilisées. Le langage de programmation du GPU ne possède pas de fonctionnalité de haut-niveau comme celles de CUDA pour l'instant, de plus des restrictions par rapport au C99 ont été ajoutées. Ainsi des fonctionnalités comme les pointeurs de fonctions, la récursivité et les tableaux / structures à taille variable sont interdits. En terme de performances, OpenCL est légèrement inférieur à CUDA sur des GPU Nvidia.

A.4 Mise en œuvre

Les principes de bases de la programmation GPU, avec CUDA ou OpenCL, sont quasiment les mêmes. On distingue dans une application GPU une partie hôte qui correspond aux instructions exécutées sur le CPU, et la partie machine qui sera exécutée sur l'architecture cible (figure A.1). Le code hôte gère le transfert des données vers et depuis la mémoire du GPU, et joue le rôle d'ordonnanceur pour l'exécution du code machine. Tous les aspects de chargement des paramètres et d'initialisation d'une simulation devront être gérés sur la partie hôte, de même pour le traitement des résultats. La partie machine concernera uniquement les agents et leur comportement durant chaque pas de temps. Le code machine dans le domaine du GPGPU est appelé noyau [4]. Un noyau sera exécuté sur chaque unité de calcul du GPU.

2. www.khronos.org/OpenCL/
3. http://www.nvidia.com/object/cuda_home_new.html
4. kernel en anglais

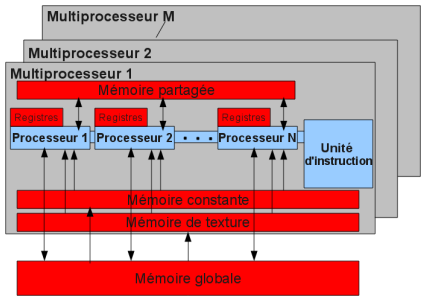


FIGURE A.1: Architecture d'un GPU.

CUDA	OpenCL
Grid	NDRange
Thread Block	Work Group
Thread	Work Item
Thread ID	Global Id
Block Index	Block Id
Thread Index	Local ID

TABLE A.1: Organisation du kernel lancé sur le GPU en blocs de threads et grilles de blocks. Chaque thread d'un bloc a un identifiant (par exemple, coordonnée 1D) unique au bloc. De même chaque bloc de la grille a un identifiant unique. On peut accéder à un thread particulier connaissant la taille des blocs de la grille. Dans nos modèles, l'agent est le thread.

Le modèle d'exécution d'un noyau (figure A.2) est basé sur une grille de calcul à un niveau de granularité initialisé et pouvant profiter des informations décrites dans la table A.1. CUDA ou OpenCL dispose de trois niveaux de granularité que ce soit pour les blocks ou les threads. Premièrement cette grille est divisée en plusieurs *Thread Block*. Ceux-ci sont exécutés indépendamment des autres *Thread Block*, et dans un ordre qui ne peut pas être contrôlé par le programmeur. Un *Thread Block* est à son tour subdivisé en plusieurs *Thread*.

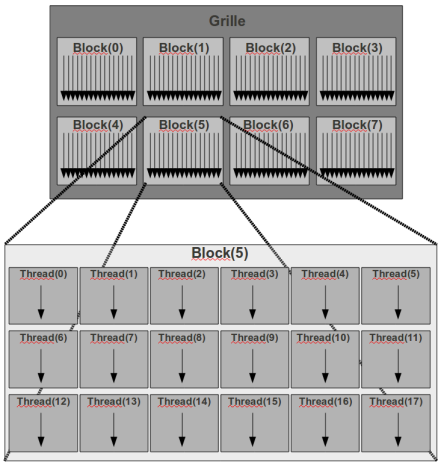


FIGURE A.2: Gestion des Block Thread.

Pour adapter notre modèle agent à ce système, le noyau doit correspondre au cycle de vie d'un agent. Chaque *Thread* sera donc une instance de l'agent.

CUDA	OpenCL
Global Memory	Global Memory
Local Memory	Global Memory
Constant Memory	Constant Memory
Texture Memory	Global Memory
Shared Memory	Local Memory
Registers	Private Memory

TABLE A.2: Les informations sur les mémoires du GPU.

A.4.1 Différents types de mémoire

Les *Thread* exécutant un noyau ont accès à plusieurs espaces mémoires (table (A.2)), qui diffèrent par leur latence, leur visibilité depuis les différents *Thread* et *Thread Block*, et leurs tailles. Une utilisation adaptée des différents types de mémoires pourra influencer considérablement les performances de l'application. Les GPU de NVIDIA disposent de 6 types de mémoires :

1. **Mémoire globale (Global Memory) :** elle permet un accès en lecture/écriture à tous les *Thread* et *Thread Block*. C'est la mémoire la plus importante en terme d'espace avec une taille de quelques GO. Cependant, elle est la plus lente en terme d'accès aux données. Elle a la durée de vie de la simulation ;

2. **Mémoire Texture (Global Memory) :** elle est une dérivée de la mémoire globale. Elle possède un accès en lecture seule à tous les *Thread* et *Thread Block*. Elle est plus rapide que la mémoire globale en terme d'accès aux données. Comme pour la mémoire globale, elle a la durée de vie de la simulation ;

3. **Mémoire Locale (Local Memory) :** elle réside dans la mémoire globale. Elle est utilisée en recours des registres.

4. **Mémoire Constante (Constant Memory) :** elle correspond à une mémoire en lecture seule, accessible à tous les *Thread*. Sa taille est faible de quelques KO. Elle est cachée et d'une durée de vie de la simulation. Elle possède une meilleure latence que celle de la mémoire globale. Elle doit être allouée et initialisée du coté hôte ;

5. **Mémoire partagée (Shared Memory) :** elle correspond à mémoire cachée accessible en lecture/écriture par tous les *Thread Block*. Sa taille est aussi faible de quelques KO. elle possède la durée de vie d'un *Thread Block*.

6. **Registres (Private Memory) :** ils correspondent aux variables allouées pour chaque *Thread*. Sa taille est également faible de quelques KO. S'il y aura un dépassement de taille, c'est la mémoire locale qui s'en charge pour allouer les nouvelles variables. Les variables allouées ont la durée de vie d'un *Thread*.

Donc, Les données de la simulation devront, dans un premier temps, être chargées sur les mémoires globales et textures. Il est donc nécessaire de profiter des autres mémoires présentes dans les GPU pour faire les optimisations.

A.4.2 Le noyau

Le noyau implémente le cycle de vie des agents. C'est lui qui va s'exécuter d'une façon parallèle. Nous détaillerons içi les modalités de l'utilisation des différentes mémoires des GPU pour profiter au maximum de leurs caractéristiques. Les données utiles pour le processus de segmentation sont divisées en 2 catégories :

1. Celles en lecture seule envoyées à la mémoire de texture.
2. Celles en lecture/écriture envoyées à la mémoire globale.

Pour initialiser le processus de segmentation par les araignées, nous envoyons les informations de niveaux de gris de l'image et celles des paramétrages des colonies à la mémoire de texture puisqu'ils sont non modifiables. Les informations relatives aux fils et celles aux agents seront envoyées à la mémoire globale. Ce sont eux qui vont être modifié au cours de la simulation. Les agents vont se propager pendant un nombre de pas de temps et modifier leurs propres informations et celles des fils tissés au cours de la simulation. De la même manière, pour la segmentation par les fourmis, nous envoyons les informations de niveaux de gris et du gradient de l'image pré-calculé à l'avance par un autre noyau sur la mémoire de texture. Les informations relatives aux phéromones et aux agents seront envoyées à la mémoire globale. Au cours de la simulation, les phéromones vont être modifiées et les agents vont mettre à jour leurs informations personnelles. Quelque soit le processus de segmentation déployé, nous avons utilisé la mémoire partagée pour calculer le voisinage de chaque agent. Cette mémoire nous a permis d'optimiser le calcul du voisinage puisque les agents vont se déplacer le plus souvent dans le voisinage local.

Algorithme du processus de segmentation sur un GPU

La gestion de cet algorithme ne passe pas par l'ordonnanceur puisque c'est le GPU qui gère les exécutions de façon asynchrone. Désormais, le nombre d'agents n'influence plus important le temps de simulation puisqu'ils sont tous traités simultanément. L'algorithme se déroule en 5 étapes :

1. Création des agents ;
2. Envoi des données du CPU vers GPU ;
3. Pour un nombre de pas de temps, exécution du noyau composé du cycle de vie de l'agent ;
4. Récupération des données du GPU vers CPU ;
5. Analyse des données et construction l'image.

Au niveau de complexité, l'algorithme des araignées passe de

$$O(C_{lissage} + C_{InitAgent} + C_{Cond} \times C_{Ordonnancer} \times NbAgent \times C_{Cycle} + C_{ConstruireImage})$$

à

$$O(C_{lissage} + C_{InitAgent} + C_{Cond} \times C_{Cycle} + C_{ConstruireImage})$$

Tandis que l'algorithme des fourmis devient passe de

$$O(C_{Cond} \times (C_{Ordonnancer} + NbAgent \times C_{Cycle} + C_{Evaporer}) + C_{ConstruireImage})$$

à

$$O(C_{Cond} \times (C_{Cycle} + C_{Evaporer}) + C_{ConstruireImage})$$

Code du cycle de vie de l'agent

Ce code présente le moyau d 'un cycle de vie d'un agent-araignée en langage CUDA.

```
__global__ void SEG_RunKernelSpider(AGE_Spider *Gpucolonies,
    ENV_SilksSpiders *Gpusilksin, ENV_SilksSpidersOut *Gpusilksout, float
    * Gpucolors, float *Cudaparam , ushort *Gpuintensities, uint *
    Gpuvisited, uint *Gpustatglobal, uint *Gpustatcol , uint NbAgent, uint
     NbColonies, uint DimX, uint DimY, uint DimZ, Rand48 RandomVar)
{
// Thread index
uint idx = blockIdx.x * blockDim.x + threadIdx.x;
if (idx >= NbAgent)
return;
int tid = threadIdx.x;
uint idc , ida, numagentcol;
numagentcol = NbAgent / NbColonies;
idc = idx/numagentcol;
ida = idx%numagentcol;
__shared__ struct Voisinage Voisins[MAXTHREAD];
bool nothing = false, newsilk = false;
float weight = 0.;
uint newpos = 0;
//Perception
Gpucolonies[idc*numagentcol+ida].AGE_DiscretePerception(Gpusilksout, &
    Voisins[tid] , Gpuvisited, idc, ida, &nothing , DimX, DimY, DimZ);
//Decision
Gpucolonies[idc*numagentcol+ida].AGE_DiscreteDecision(Gpusilksin,
    Gpusilksout, Voisins[tid], Cudaparam, Gpucolors, Gpuintensities, idc,
    Gpuvisited, Gpustatglobal,nothing, &newpos, &newsilk, &weight , DimX,
    DimY, DimZ, RandomVar);
//Action
Gpucolonies[idc*numagentcol+ida].AGE_DiscreteAction(Gpusilksin,
    Gpusilksout, CudaParam, newpos, weight, newsilk, Gpuvisited,
    Gpustatglobal, Gpustatcol, idc, DimX, DimY, DimZ);
}
```

La mise en oeuvre de verion parallèle avec CUDA ou OpenCL nécessite toujours l'écriture des morceaux de code totalement spécifique qui seront ensuite compilés avec le compilateur associé (nvcc ou g++ avec un lien dynamique avec les libraires associées).

Le traitement de la répartition des données sur les mémoires et des threads sur les blocks (étape 2) s'effectue de la manière suivante :

```
1  // Send Read/Write data to the global memory.
2
3  // Colonies.
4  cutilSafeCall(cudaMalloc((void**)&cudacolonies, Taille(cpucolonies) ));
5  cutilSafeCall(cudaMemcpy(cudacolonies, cpucolonies, Taille(cpucolonies),
       cudaMemcpyHostToDevice));
6  cudaCheckError("Copy from host to device error-Colonies");
7
8  // Silks.
9  cutilSafeCall(cudaMalloc((void**)&cudasilksin, Taille(cpusilksin) ));
10 cutilSafeCall(cudaMemcpy(cudasilksin, cpusilksin, Taille(cpusilksin),
       cudaMemcpyHostToDevice));
11 cudaCheckError("Copy from host to device error-silkin");
12
13 cutilSafeCall(cudaMalloc((void**)&cudasilksout, Taille(cpusilksout) ));
14 cutilSafeCall(cudaMemcpy(cudasilksout, cpusilksout, Taille(cpusilksout),
       cudaMemcpyHostToDevice));
15 cudaCheckError("Copy from host to device error-silkout");
16
17 // Statistics.
18 cutilSafeCall(cudaMalloc((void**)&cudavisited, Taille(cpuvisited) ));
19 cutilSafeCall(cudaMemcpy(cudavisited, cpuvisited, Taille(cpuvisited) ,
       cudaMemcpyHostToDevice));
20 cudaCheckError("Copy from host to device error-visited");
21
22 cutilSafeCall(cudaMalloc((void**)&cudastatglobal, Taille(cpustatglobal) )
       );
23 cutilSafeCall(cudaMemcpy(cudastatglobal, cpustatglobal, Taille(
       cpustatglobal), cudaMemcpyHostToDevice));
24 cudaCheckError("Copy from host to device error-Statglobal");
25
26 cutilSafeCall(cudaMalloc((void**)&cudastatcol, Taille(cpustatcol)));
27 cutilSafeCall(cudaMemcpy(cudastatcol, cpustatcol, Taille(cpustatcol),
       cudaMemcpyHostToDevice));
28 cudaCheckError("Copy from host to device error-Stat"col);
29
30
31 // Send readonly memory to the texture memory / till now they are
       attached to the global memory.
32
```

```
// Color information for each colony.
cutilSafeCall(cudaMalloc((void**)&cudacolors, Taille(cpucolors)));
cutilSafeCall(cudaMemcpy(cudacolors, cpucolors, Taille(cpucolors),
    cudaMemcpyHostToDevice));
cudaCheckError("Copy from host to device error-colors");


//Sils attractions and saturation information.
cutilSafeCall(cudaMalloc((void**)&cudaparam, Taille(cpuparam)));
cutilSafeCall(cudaMemcpy(cudaparam, cpuparam, Taille(cpuparam),
    cudaMemcpyHostToDevice));
cudaCheckError("Copy from host to device error-Param");

// Gray level of the voxels.
cutilSafeCall(cudaMalloc((void**)&cudaintensities, Taille(cpuintensities)
    ));
cutilSafeCall(cudaMemcpy(cudaintensities, cpuintensities, Taille(
    cpuintensities), cudaMemcpyHostToDevice));
cudaCheckError("Copy from host to device error-Intensities");

// Create blocks and threads for the system.
uint numThread = MAXTHREAD;
uint numBlock = NbAgent / numThread + (NbAgent % numThread > 0 ? 1 : 0);

dim3 dimBlock(numThread);
dim3 dimGrid(numBlock);
```

Il est à noter que la déclaration des mémoires de textures doit être faite de manière globale pour que tout les fichiers du programmes puissent accéder à ces informations. Nous allons données les points importants pour créer la mémoire texture. Tout d'abord, il faut créer des variables globales de texture visibles par tout le programme. Après cette étape, les données sont initialisées dans la mémoire globale puis affecter à la mémoire de texture. Par exemple, pour affecté les intensités de l'image à la mémoire de texture, nous avons avons besoin de suivre la procédure suivante :

```
// Creation of the global variable of the texture memory, in our case it
    is in 3D.
texture<ushort, 3, cudaReadModeElementType> texcudaintensities;
// Create the memory on the GPU side.
cutilSafeCall(cudaMalloc((void**)&cudaintensities, Taille(cpuintensities)
    ));
cutilSafeCall(cudaMemcpy(cudaintensities, cpuintensities, Taille(
    cpuintensities), cudaMemcpyHostToDevice));
cudaCheckError("Copy from host to device error-Intensities");
```

```
7
8  //Linking the global memory to the texture memory in order to be cached.
9   cudaBindTexture(0, texcudaintensities, cudaintensities, Taille(
       cpuintensities));
10
11 // Setting the options.
12 texcudaintensities.normalized = false;
13 texcudaintensities.filterMode = cudaFilterModePoint;
```

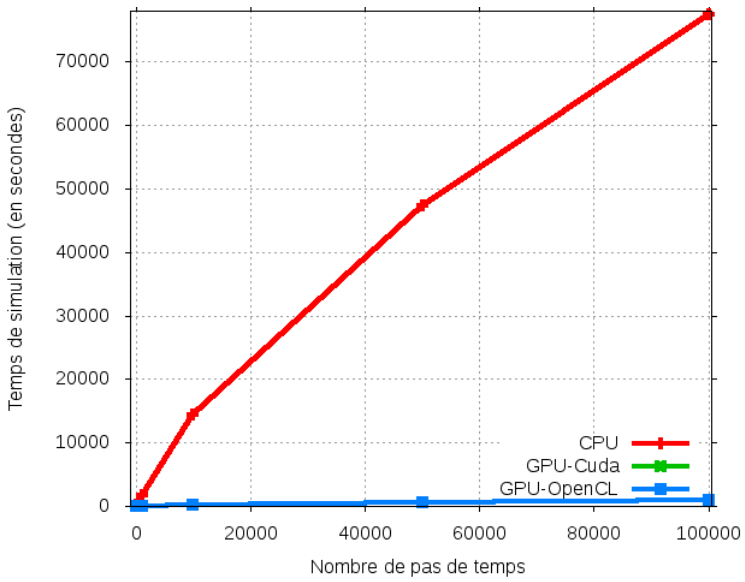
L'affection de ces données à la mémoire de texture induit la suppression des paramètres qui les représentent du noyau et de tout les fonctions exécutées sur la partie GPU.

L'agent-fourmi a exactement le même format mais à la différence du travail sur les phéromones et non pas les fils. Le programme IPSiMAS dont découle ce cycle de vie est disponible sur **sourceforge** (*http://sourceforge.net/p/ipsimas/home/Home/*).

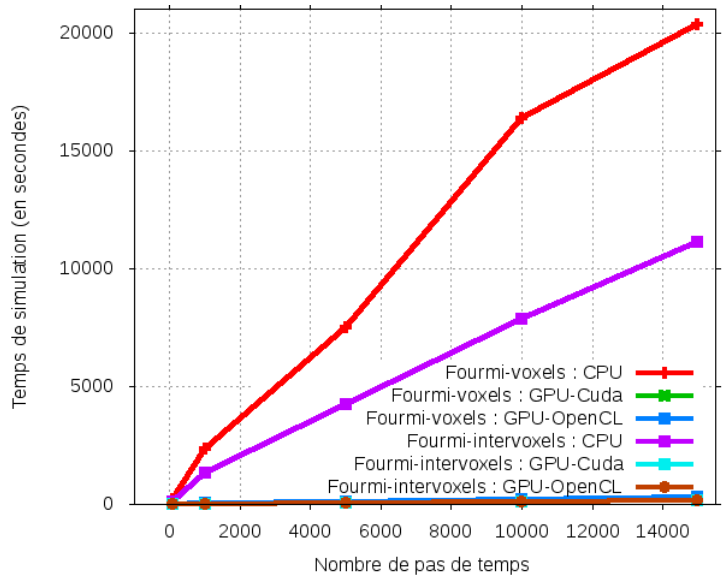
A.4.3 Performance

Nous avons voulu évaluer le gain de performances obtenu par l'utilisation du GPU. Pour cela nous avons implémenté une version de l'application écrite uniquement en langage C++. Dans celle-ci, l'exécution du cycle des agents est séquentielle et n'utilise qu'un seul coeur du CPU.

Les comparaisons des performances des différents Systèmes Multi-Agents est effectué sur la base de la fixation du nombre d'agent à 1 million et la variation du nombre de pas de temps. La figure A.3 représente le temps de simulation moyen obtenu avec les paramètres optimaux utilisés. L'analyse des temps de simulations nous a montré que le gain de performance par les GPU peut atteindre un facteur 135 fois pour le modèle des fourmis utilisant un partitionnement en voxels, un facteur 102 lors du partitionnement intervoxels et un facteur 86 pour le modèle des araignées. Nous avons remarqué que la plateforme OpenCL est inférieure à CUDA sur des GPU Nvidia. Ceci est dû au fait que l'API CUDA est plus mature que l'API OpenCL et que CUDA est spécifiquement conçu et optimisé pour programmer des GPU NVIDIA.



(a)



(b)

FIGURE A.3: Les performances : (a) araignées, (b) fourmis.