TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES ALGORITHMES

CHAPITRE 1

INTRODUCTION

1.1 CONTEXTE DE LA RECHERCHE

De nos jours le vieillissement général de la population engendre de nombreuses problématiques, comme le manque de place dans les centres d'accueil, le coût des retraites, le maintien des personnes à domicile, et bien d'autres encore [1]. Notre société va devoir surmonter des défis dans les prochaines décennies afin de pouvoir aider le nombre grandissant des personnes âgées. L'un des défis actuels est d'apporter une meilleure assistance à domicile à ces personnes grâce aux nouvelles technologies.

En effet, l'intelligence artificielle (IA), dont les fondements en tant que domaine de recherche ont été érigés vers la fin des années 90, est de plus en plus présente de nos jours [2]. Elle est utilisée dans de nombreux domaines tels que : les jeux vidéo, la médecine, la logistique, etc. De plus, les avancées significatives obtenues en IA permettent aujourd'hui de l'appliquer dans l'assistance technologique des personnes. Cette assistance fait partie d'un sous domaine de l'IA qui est l'intelligence ambiante (IAm). L'IAm, définie par Capezio et Ramos [3, 4], fait référence à une approche multidisciplinaire qui consiste à enrichir un environnement standard (salle, voiture, immeuble, etc.) avec de la technologie (capteurs, effecteurs, étiquettes intelligentes RFID, etc.). L'IAm a pour but de construire un système pouvant prendre des décisions qui bénéficieront à l'utilisateur de cet

environnement. Elle constitue ainsi un nouveau paradigme de recherche qui tient aux capacités d'intégration des systèmes numériques dans le milieu physique. La principale infrastructure utilisant le concept d'IAm, communément appelée habitat intelligent [5], peut avoir divers objectifs. L'un d'eux est justement d'assister et de maintenir à domicile les personnes âgées, ou les personnes ayant des déficiences mentales. Le premier pas fondamental de cette assistance est de reconnaitre les activités de la vie quotidienne (AVQ) de l'habitant durant leurs exécutions, dans le but d'identifier les problèmes potentiels qui peuvent interférer dans leurs accomplissement [6]. Ainsi, la reconnaissance d'activités a pour but de prédire le comportement d'une personne en vue d'offrir, au moment opportun et sans intrusion, les services appropriés sans être rejetés par les habitants. Cependant, les techniques actuelles de reconnaissance d'activités sont encore incomplètes et ne peuvent pas assister complètement une personne âgée ou malade. C'est pourquoi de nombreuses recherches [7-9] sont effectuées afin d'améliorer les différents aspects de la reconnaissance d'activités, et ainsi combler les lacunes des systèmes existants. De plus, les nouvelles méthodes de forage de données permettent maintenant le traitement de grand ensemble de données, comme ceux générés par les capteurs des habitats intelligents par exemple. De ce fait, la combinaison des techniques de reconnaissance d'activités et du forage de données peut amener des solutions à notre problématique.

1.2 LA RECONNAISSANCE D'ACTIVITES

Schmidt et all. [10] définissent la reconnaissance d'activités par le fait de : « prendre en entrée une séquence d'actions exécutées par un acteur et d'inférer le but poursuivi par

l'acteur et d'organiser la séquence d'actions en terme d'une structure de plan ». L'acteur suit donc un plan d'action pour atteindre un but bien déterminé, alors que l'agent, qui possède normalement toutes les actions, essaie de trouver le plan adéquat en se basant sur les actions observées. Cette difficulté correspond en fait à une forme particulière d'une problématique bien connue en intelligence artificielle : la reconnaissance de plans [11]. Un plan correspond à une suite d'actions élémentaires représentant une activité donnée. Cette reconnaissance de plans a pour but d'interpréter le comportement d'une personne en vue d'offrir les services appropriés au bon moment. Ce procédé peut se définir comme étant la méthode d'extraction de signaux numériques des différents capteurs et leurs associations à des actions de bases qui peuvent constituer le plan d'une activité de la vie quotidienne (AVQ). Cette initiative est d'autant plus importante, car elle permet de déceler si le patient a commis des erreurs et de cibler le moment opportun pour effectuer une assistance efficace. Pour se faire, il faut définir plusieurs critères discriminatoires avec lesquels une présélection peut être faite. Afin de résoudre ces problèmes de reconnaissance de plans, les chercheurs ont développé des approches différentes.

Entre autres, Kautz et Allen [12] ont développé une approche basée sur les logiques traditionnelles, les activités sont déterminées par l'enchainement séquentiel des actions. Ce type d'approche vise à sélectionner, parmi les plans d'activités, par une série de déductions logiques, l'ensemble des activités possibles qui peuvent expliquer un ensemble d'actions observées. Le grand avantage de ces approches est leurs efficacités sur une très large base de connaissances. Elles permettent d'ignorer rapidement la majorité des activités et de n'en garder qu'un petit ensemble. Les traitements et calculs se font alors sur un ensemble réduit,

ce qui donne un meilleur temps de réponse. Par contre, elles peuvent facilement éliminer l'activité recherchée, car une action détectée qui n'appartient pas au plan d'une activité élimine automatiquement cette dernière des activités possibles, alors que le patient peut être en train d'effectuer la même activité et l'action détectée n'est due qu'à une erreur de type réalisation de sa part. Cette action peut aussi être expliquée par le commencement d'une deuxième activité avant l'achèvement de la première.

D'autres chercheurs [13] se sont servis de modèles probabilistes afin de déterminer les activités des agents observés, ces modèles consistent à augmenter les probabilités des plans dont les actions sont effectuées, puis d'inférer l'activité en court avec celle étant la plus probable. Ces approches probabilistes règlent le problème des approches logiques. Elles permettent de résoudre le problème d'irrationalité de l'entité observée. Une action, qui n'a aucun rapport avec l'objectif visé, n'éliminera plus l'activité la plus probable. Par contre, le grand inconvénient de ces approches est la lourdeur du calcul et du traitement. Pour chaque action détectée, c'est l'ensemble de toutes les activités de la base de connaissance qui est mis à jour.

Des approches temporelles ont également été utilisées par Jakkula et Cook [14], celles-ci permettent de déterminer les activités de l'agent observé en fonction du temps qui passe entre chacune de ses actions, il faut d'ailleurs comprendre qu'une activité peut être exécutée dans une séquence valide dans le temps, mais tout de même être erronée dû à des problèmes de nature spatiale. Par exemple, si l'aspect spatial de la reconnaissance d'activités n'est pas pris compte des problèmes particuliers peuvent subvenir. Entre autres, si un verre se situe en dessous d'un robinet, il est difficile de savoir si la personne le lave ou

si elle prend un verre d'eau. Par contre, en considérant l'aspect spatial relié à l'orientation de l'objet on peut déterminer laquelle des deux activités est effectuée. Si le verre reste droit alors l'action de prendre un verre d'eau est plus probable. C'est pourquoi certaines approches se basent sur des contraintes spatiales pour déduire les plans d'activités. Dans le travail d'Augusto [15], la position de l'agent observé est détectée précisément grâce a de nombreux capteurs, cela permet entre autres de suivre ses déplacements facilement. Toutefois, il est encore très difficile de nos jours d'identifier toutes les activités quotidiennes des utilisateurs en utilisant une seule de ces approches. Certains travaux cumulent plusieurs types d'approches dans ce but, mais ces nouvelles méthodes demandent beaucoup de temps de calcul dû au grand nombre de plans à gérer, c'est là qu'interviennent les techniques de forage de données.

1.3 LE FORAGE DE DONNEES

De nos jours, et dans tous les domaines, les bases de données chargées d'entreposer les informations sont gigantesques et de plus en plus inter-reliées. Nous ne parlons même plus de base de données mais d'entrepôt de données. Si les requêtes SQL sont parfaitement capables d'en retirer des informations, aussi compliquées soient-elles, elles sont totalement inutiles pour en extraire des connaissances. L'exemple d'un entrepôt de données d'un supermarché illustre très bien ces propos : les requêtes SQL peuvent facilement répondre à des questions sur la quantité vendue d'un produit ou sur la recette d'un mois, mais elles sont incapables de construire une base de connaissances sur les habitudes de consommation

des clients, comme trouver les produits que les clients ont l'habitude d'acheter ensemble par exemple [16].

Le forage de données a pour objectif d'extraire de la connaissance après l'analyse des données stockées dans l'entrepôt de données. Elle peut être définie plus précisément en étant un ensemble de méthodes et techniques automatiques ou semi-automatiques explorant les données en vue de détecter des règles, des associations, des tendances inconnues ou cachées, des structures particulières restituant l'essentielle de l'information utile pour la prise de décision [16]. Elle se divise en deux majeures catégories : la fouille de données descriptive qui ne fait que mettre en évidence des informations déjà existantes et qui considère toutes les données au même degré d'importance, et la fouille de données prédictive qui extrapole de nouvelles informations à partir de celles déjà présentes, en considèrant une ou plusieurs données plus importantes que les autres.

Les excellents résultats obtenus par le biais du forage de données lui ont permis de s'attaquer à des domaines diversifiés et nombreux : la sécurité, le marché boursier, le commerce électronique, la santé, et bien évidemment la reconnaissance d'activités. L'évolution des technologies de l'information et de l'électronique permet aujourd'hui d'envisager une solution permettant l'utilisation de nombreux capteurs dans les habitats intelligents. On peut ainsi identifier tous les objets présents dans l'habitat, la position de l'agent observé, et beaucoup d'autres informations encore. Toutefois, ce grand nombre de données doit être efficacement traité afin de ne pas ralentir l'algorithme de reconnaissance d'activité. Le nombre de plans étant pratiquement infini, les méthodes ne nécessitant pas

d'établir une bibliothèque de plan comme les méthodes d'apprentissage non supervisé peuvent se révéler un bon choix pour la prédiction d'activités dans les habitats intelligents.

1.3 CONTRIBUTION DE CE MEMOIRE

Tout d'abord, l'objectif général de cette recherche est d'améliorer la reconnaissance d'activités dans son ensemble afin d'obtenir une meilleure assistance pour les différents utilisateurs des habitats intelligents. L'hypothèse que nous avons posé est que le Flocking [17], comme une méthode de forage de données, peut améliorer la formation des clusters et aide à obtenir une meilleure reconnaissance des activités. En effet, le fait d'effectuer une activité déclenche obligatoirement un déplacement qui n'est pas pris en compte dans les méthodes existantes. Pourtant, on peut caractériser ces mouvements avec trois paramètres : la direction, la position, et la distance des objets entre eux. Or, ces paramètres sont formalisés et efficacement exploités dans l'algorithme du Flocking.

Ainsi, le but de notre travail est de détecter les activités effectués par l'utilisateur de l'habitat tout au long d'une journée, et quelques soit le nombre d'activités. Pour ce faire, les données des capteurs similaires génèrent des clusters grâce au Flocking, et ces clusters représentent les activités en cours. Puis, pour vérifier la qualité des clusters, des tests sont effectués avec les mêmes standards des méthodes classiques de forage de données.

1.4 METHODOLOGIE DE LA RECHERCHE

Le projet de recherche présenté dans ce mémoire s'est effectué en suivant une méthodologie de recherche se divisant en quatre principales étapes correspondant aux objectifs de recherche décrits dans la section précédente.

La première étape du projet a été de faire une étude approfondie sur les différentes techniques de reconnaissance d'activités intégrant des méthodes de forages de données, ainsi qu'une étude sur les différentes recherches qui utilise le Flocking comme méthode de segmentation. Cette première étape a permit de dégager des pistes de solution pour l'exploitation du Flocking dans le cadre des habitats intelligents. La revue de littérature a également permis au recueil des principaux formalismes du forage de données pour pouvoir établir un modèle pertinent.

La deuxième étape du projet a été de formaliser un nouveau modèle de reconnaissance d'activités intégrant les éléments tirés du flocking, afin de répondre à la problématique de la reconnaissance d'activités.

La troisième étape du projet a consisté à implémenter ce nouveau modèle afin d'entamer, dans une quatrième et dernière étape, une série de tests basés sur des scénarios de cas réels afin de valider la performance du nouvel algorithme, et de le comparer avec les autres méthodes de forage de données.

Ce projet à également été le sujet d'un article scientifique de huit pages [18] présenté à la « 4th International Conference on Ambient Systems, Networks and Technologies (ANT 2013) ». Cette distinction vient confirmer la pertinence de notre approche.

1.5 ORGANISATION DU MEMOIRE

Ce mémoire est organisé en quatre chapitres. Ce premier chapitre a pour but d'introduire le contexte de la recherche, de donner une première vue d'ensemble sur la

reconnaissance d'activité et le forage de données, puis de montrer la contribution réelle du mémoire et enfin de présenter la méthodologie qui a été utilisée tout au long du projet.

Le second chapitre est une revue de littérature sur le forage de données. Il est constitué d'un approfondissement sur les méthodes classiques de reconnaissance d'activités utilisant le forage de données. Puis, les objectifs et techniques de la segmentation sont présentés avec quelques-uns des algorithmes les plus connus. L'origine du Flocking est également revue, et deux recherches récentes qui utilisent le Flocking comme méthode de segmentation sont présentées avec leurs méthodologies et leurs résultats.

Le troisième chapitre constitue la contribution théorique et pratique de ce mémoire. Nous présentons dans ce chapitre notre extension du Flocking en commençant par les règles de bases du Flocking puis nos nouvelles règles qui permettent d'en faire une méthode de segmentation. Ensuite, le LIARA et ses spécificités sont détaillés pour comprendre nos choix d'implémentations. Enfin, la partie finale du chapitre présente les expérimentations effectuées et les résultats obtenus.

Le quatrième et dernier chapitre est la conclusion de ce projet de recherche. Il contient les objectifs réalisés et une revue du modèle développé. Puis, les limitations de la méthode et les futurs travaux sont discutés. Enfin, le chapitre termine sur un bilan personnel du travail de recherche.

CHAPITRE 2

LE FORAGE DE DONNEES

2.1 INTRODUCTION AU FORAGE DE DONNEES

Le forage ou la fouille de données (FD) est un ensemble de méthodes et d'algorithmes pour l'exploration et l'analyse de grandes bases de données. Ces méthodes ont pour objectif de détecter des règles, des associations, des tendances inconnues, des structures, désignées alors par le terme de connaissances relatives aux données analysées, et cela, afin de faciliter la prise de décisions [19, 20]. On peut ainsi reformuler le forage de données comme « l'analyse du passé pour prédire le futur ». Les motivations du développement du forage de données sont liées à la masse importante de données qui existe de nos jours, et qui double tous les 20 mois. La croissance en puissance des machines actuelles permet de supporter de gros volumes de données, et d'envisager l'exécution intensive dont requiert la fouille de donner. En effet, les données, souvent désignées par le terme « instance », sont multidimensionnelles, elles possèdent généralement des milliers d'attributs, et sont souvent inexploitables par les méthodes classiques. En autre, le traitement de ces données doit se faire en temps réel, car si les décisions prennent trop de temps à être prise elles seront probablement obsolètes.

La démarche méthodologique du forage de données est de comprendre le problème en sélectionnant un échantillon de donné avec une méthode d'échantillonnage adapté au problème, de supprimer le « bruit » de cet échantillon (les données superflues ou

manquantes) si nécessaire, et éventuellement de sélectionner une partie des attributs pour réduire la dimension du problème. Pour ensuite, appliquer une ou plusieurs techniques de fouille de données avec un algorithme spécifique choisi au préalable pour chaque technique. La méthodologie du forage de données peut être résumé comme un processus itératif à plusieurs passes, interactif dans le sens où l'utilisateur est dans la boucle, valide et compréhensible. La Figure 1 ci-dessous présente ce processus.

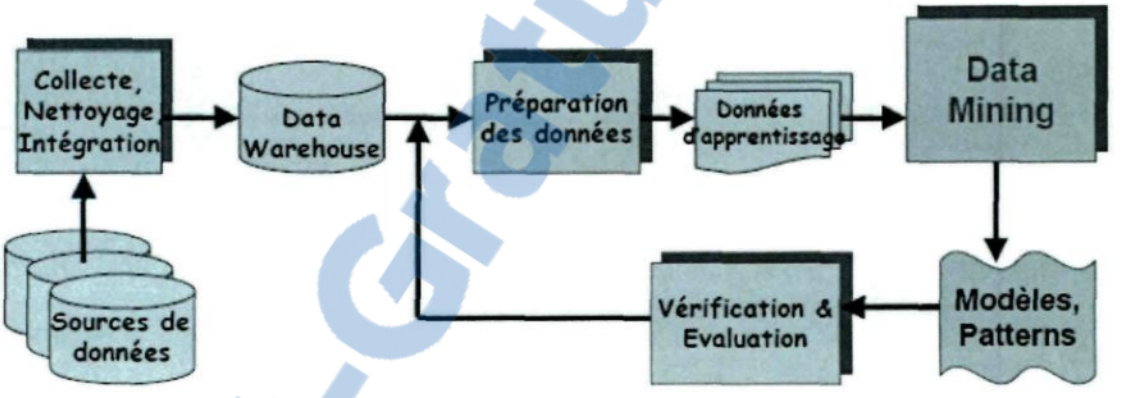


Figure 1 : Processus du forage de données

Il existe différentes méthodes de forage de données et chacune a des paramètres et des applications variés qu'on regroupe en trois catégories : la classification, la segmentation et l'association [21]. La classification correspond aux techniques permettant de prédire si une instance de données est membre d'un groupe ou d'une classe prédéfinie (arbre de décision, réseau de neurones, réseau bayésien, etc.) [19]. Sachant qu'une classe correspond à un groupe d'instances avec des profils particuliers. Ce type d'apprentissage est dit supervisé, car les classes sont connues à l'avance. La segmentation quant à elle n'a pas de classe à prédire, elle consiste à partitionner la base de données en clusters, c'est-à-dire à des

groupes d'instances ayant des caractéristiques similaires. Contrairement à cette dernière, les clusters sont inconnus au départ, la segmentation est un apprentissage non supervisé, en d'autres termes, nous n'avons aucune idée de ce que l'on va apprendre. Enfin, l'association, ou règle d'associations est également une sorte d'apprentissage non supervisé. Dans cette catégorie on cherche plutôt à établir des relations entre les attributs des données pour inférer les décisions.

Dans notre contexte d'IAm on doit se demander quelle est là catégorie de méthodes de forage de données qui s'accordent le mieux à ce paradigme. Nous allons donc voir les enjeux de la reconnaissance d'activités dans les habitats intelligents, puis les méthodes de forages de données utilisées dans ce même contexte. Ensuite, nous décrirons les techniques classiques de segmentation, suivi par la nouvelle approche du Flocking. Enfin, nous détaillerons deux recherches utilisant une version améliorée du Flocking comme méthode de segmentation.

2.1.1 LE FORAGE DE DONNEES DANS LES HABITATS INTELLIGENTS

Les méthodes classiques de reconnaissance d'activités se servent de différentes approches basées sur trois types de contraintes : les approches exploitant des contraintes logiques traditionnelles [6], les approches utilisant les contraintes temporelles [14] et celles exploitant les contraintes spatiales [15].

2.1.1.1 APPROCHES AVEC LOGIQUES TRADITIONNELLES

Kautz et Allen [12] ont établi l'une des plus importantes approches basées sur les logiques traditionnelles. Celle-ci consiste à utiliser la logique de premier ordre afin de

formaliser le processus d'inférence de l'activité en cours. Cependant, dans le cadre des habitats intelligents, son approche est rarement utilisée, car elle ne prend pas en compte les erreurs potentielles des acteurs. De ce fait, son modèle interprète les erreurs comme des changements d'activités. De plus, l'approche logique part du principe que toutes les activités possibles sont connues et répertoriées, ceci n'est pas réaliste dans le contexte où l'acteur peut être erratique et peut se tromper. Certains chercheurs, comme Py et Nerzic [22, 23] ont proposé des approches logiques avec gestion des erreurs. Mais ces méthodes n'ont pas encore été formalisées ni implémentées, car elles soulèvent de nouveaux problèmes, comme le fait de devoir définir toutes les erreurs possibles d'un plan. D'autres approches logiques se basent aussi sur des modèles probabilistes (modèles de Markov cachés et ses extensions, réseau bayésien, etc.) [13], faisant partie des méthodes de classification du forage de données. Ces modèles consistent à attribuer une probabilité à chaque plan de la bibliothèque selon les actions observées. Entre autres, lorsque l'acteur fait quelque chose, la probabilité des plans qui contiennent cette action augmente. Ainsi, le plan ayant le plus haut score est probablement l'activité réalisée par l'agent observé. Les modèlent probabilistes comportent néanmoins deux problèmes [6]. Tout d'abord, le temps de calcul peut devenir très long lorsque le nombre de plans est élevé, car ils sont tous évalués à chaque action. Ensuite, en ajoutant une nouvelle activité à la bibliothèque, il faut recalculer la distribution des probabilités afin qu'elles soient uniformes. Ces inconvénients limitent l'ajout de nouveaux comportements. Toutefois, d'autres approches combinent le processus d'inférence probabiliste avec une approche symbolique [24]. Ainsi, la bibliothèque de plans est filtrée, et seulement un sous-ensemble des plans est utilisé dans le

processus d'inférence, ce qui permet un temps de calcul plus raisonnable. Malgré tout, les approches avec logiques traditionnelles restent limitées, c'est pourquoi d'autres types de méthodes ont été développées, comme les approches temporelles.

2.1.1.2 APPROCHES TEMPORELLES

Les approches temporelles utilisent comme contraintes les durées, les délais, mais aussi des méthodes plus complexes comme les relations d'Allen [25]. Ces relations représentent 14 possibilités entre deux intervalles du temps, dont ce dernier est symbolisé par une droite infinie constituée d'un nombre infini de points. Par exemple, la partie temporelle du travail de Jakkula et Cook [14] est basé sur les relations d'Allen. Cela permet, dans les cas où une action est souvent suivie par une autre action spécifique, d'inférer une règle temporelle qui permet de savoir quelle est cette action suivante. Entre autres, si l'action *PrendreUnOeuf* est toujours suivie par l'allumage de la cuisinière, l'action *CuireUnOeuf* est inférée grâce aux relations d'Allen. De plus, ils utilisent les règles d'associations de l'exploration de données pour faire apprendre à leur système les activités clés. Cette technique est basée sur le la recherche des schémas d'associations [16] qui consistent à identifier les patrons séquentiels fréquents, et à les enregistrer. Cela leur permet d'apprendre à leur algorithme les activités fréquentes de l'agent grâce aux relations observées, et ainsi d'améliorer l'assistance procurée par leur système. Cependant, leur méthode dépend énormément de l'apprentissage de leur algorithme, car il doit connaître de nombreuses suites d'actions pour être efficace. Ceci demande beaucoup de temps d'apprentissage et doit être répété pour chaque nouvel agent observé. D'autres travaux [26,

27] utilisent l'aspect temporel, mais aucun d'eux ne peut reconnaitre les problèmes de type spatiaux qui peuvent subvenir dans les habitats intelligents. Il est donc nécessaire d'ajouter l'aspect spatial dans la reconnaissance d'activités pour pouvoir gérer le plus de cas possible.

2.1.1.3 APPROCHES SPATIALES

Les approches spatiales sont peu nombreuses à ce jour malgré qu'elles soient reconnues comme un aspect fondamental de la reconnaissance d'activités. Dans cette section, nous allons présenter deux approches récentes qui intègrent des notions spatiales. La première est celle de Augusto [15]. Tout d'abord, pour pouvoir suivre la position de l'agent observé, ils ont combiné des détecteurs infrarouges ainsi qu'une étiquette intelligente RFID portée par l'acteur. Puis, ils ont accroché des antennes RFID aux portes pour détecter l'acteur lorsqu'il franchit la porte, mais aussi des capteurs de mouvement qui indiquent dans quelle pièce il est entré. Ainsi, Augusto a pu suivre l'acteur assez précisément dans l'habitat intelligent. La seconde a été conçue par Riedel [28] et traite l'aspect spatial de la reconnaissance d'activités grâce à un modèle chimiotactique. Celui-ci est tiré du mouvement des bactéries nommé le chimiotactisme [29] qui leur permet de se diriger vers des gradients physiques. En outre, le modèle de Riedel exploite une abstraction de ce processus afin de représenter et reconnaitre certaines activités. Cependant, dans ces deux précédentes approches seulement la position de l'acteur est utilisée, alors qu'il faudrait utiliser l'ensemble des objets. Dans cette optique, on aurait beaucoup plus de données à traiter et on pourrait intégrer pleinement la reconnaissance spatiale de l'habitat

intelligent. C'est à ce niveau là que le forage de données intervient, puisque son objectif est d'extraire des informations à partir de grandes quantités de données.

En utilisant de nombreux capteurs on peut identifier tous les objets présents dans l'habitat intelligent, et ainsi combiner les approches temporelles et spatiales. Le fait d'effectuer une activité déclenche obligatoirement un déplacement des objets mobiles, un changement d'état des objets immobiles, et chacune de ces actions implique également un temps auquel elles ont été effectuées. Cependant, on ne peut pas directement reconnaitre les activités à partir de ces informations, on ne connait pas la « classe » des activités lorsqu'elles sont effectués. De ce fait, les méthodes de classifications ne sont pas appropriées pour ce problème, car il faut connaitre la classe à l'avance. De même, les règles d'associations ne sont pas très utiles, car on ne veut pas obtenir des corrélations entre les attributs de position ou temps, mais reconnaitre des activités directement à partir des données des capteurs. C'est pourquoi les méthodes de segmentation semblent les plus adéquates pour notre problématique, nous allons donc voir en détail en quoi consiste la segmentation.

2.1.2 LA SEGMENTATION

La segmentation, de l'anglais « clustering », consiste à séparer un large ensemble de données en plusieurs sous-ensembles nommés clusters. Les éléments contenus dans un cluster sont similaires entre eux et différents des éléments des autres clusters [30, 31]. L'objectif de la segmentation est de trouver des structures inhérentes aux données et de les présenter sous forme d'un ensemble de clusters [32]. Les applications typiques de la

segmentation sont comme outil d'exploration interne des distributions de données, ou comme étape de prétraitement pour les autres algorithmes. Recourir à la détection des clusters quand les regroupements naturels peuvent représenter des groupes de clients, de produits, ont beaucoup en commun. En effet, lors de la présence de plusieurs schémas en compétition au sein des données il est difficile d'extraire un schéma particulier, c'est pourquoi la création de clusters peut réduire la complexité en organisant les données en clusters. Les techniques de segmentation se séparent en deux catégories : le partitionnement et la répartition hiérarchique [30, 31]. Les méthodes hiérarchiques construisent une hiérarchie de clusters et une unique partition d'objets. Dans ces techniques, le nombre de clusters n'est pas requis et une matrice de distance est généralement utilisée comme critère de segmentation. De même, ces méthodes sont souvent caractérisées comme des approches de segmentation de meilleure qualité. Cependant, la complexité de la répartition hiérarchique est en général au moins $O(n^2)$, cela rend ces techniques trop lentes pour les grands ensembles de données. Au contraire, la complexité algorithmique des techniques de partitionnement est la plupart du temps linéaire. Celles-ci divisent les données en groupes qui ne se chevauchent pas pour maximiser le temps de traitement.

Afin de déterminer quelles sont les données avec le plus de similarité, une fonction de distance est généralement utilisée pour calculer la différence entre chaque instance [19]. Il existe plusieurs façons de définir cette fonction de distance, souvent désignée par « mesure de similarité ». La plupart des techniques de segmentation utilisent la distance euclidienne comme mesure de similarité. La distance entre deux instances a et b avec k

attributs est définie comme la somme de la différence au carré de chacun de leurs attributs, exprimé par la formule suivante :

$$d(a,b) = \sqrt{\sum_{i=1}^{k}(a_i - b_i)^2} \qquad (1)$$

Où a_i et b_i $(i = 1,n)$ sont respectivement des attributs des instances a et b. La racine carrée n'est généralement pas effectuée en pratique, car on peut comparer directement les distances au carré. Une alternative à la distance euclidienne est la distance de Manhattan, avec celle-ci la distance entre les attributs est simplement additionnée sans les mettre au carré, à la place on prend juste leurs valeurs absolues. Une autre mesure est la distance de Minkowski, au lieu de prendre le carré comme puissance on utilise un paramètre variable. Plus haute est la puissance choisie plus les grandes différences vont être amplifié par rapport aux petites différences. Généralement la distance euclidienne représente un bon compromis entre les deux autres. Un autre détail important dans l'utilisation de ces distances est de les normaliser entre 0 et 1. En effet, les différents attributs peuvent avoir plus d'influences que d'autres selon leur échelle. En conséquence, on normalise les attributs en retranchant à la valeur actuelle v_i d'un attribut la valeur minimale de toutes les instances, puis en divisant par la valeur maximale moins la valeur minimale. On peut résumer ainsi ce calcul :

$$a_i = \frac{v_i - \min v_i}{\max v_i - \min v_i} \qquad (2)$$

Cette formule suppose que les attributs sont de types numériques. Si un attribut est de type nominal, la différence entre deux valeurs identique sera 0. Si les valeurs ne sont pas égales alors la distance sera maximale, c'est-à-dire 1.

Tout ceci a pour objectif d'obtenir une bonne segmentation, la qualité des clusters sera d'autant plus importante avec une similarité intra-classe élevée et une similarité inter-classe faible. La mesure de similarité est l'un des principaux facteurs de la qualité des clusters, elle dépend de la distance utilisée et de son implémentation. Une méthode de segmentation est évaluée par son habileté à découvrir certains ou tous les schémas cachés dans les données, plus il y en aura et plus la méthode sera de bonne qualité.

2.2 LES METHODES CLASSIQUES DE SEGMENTATION

Maintenant que les principes de la segmentation ont été présentés nous examinerons le fonctionnement de trois techniques, la première est l'une des méthodes les plus connues et utilisées depuis plusieurs décennies, c'est l'algorithme K-means [19].

2.2.1 K-MEANS

L'objectif du K-means est de segmenter les données en k-groupes, il faut spécifier avant de lancer l'algorithme combien de clusters ont désire créer, c'est le paramètre k [33, 34]. Ensuite, k points sont choisi semi aléatoirement comme centre des clusters. Toutes les instances sont assignées au centre le plus proche d'eux, ceci étant calculé avec la distance euclidienne vue précédemment. Voici un exemple d'initialisation dans la Figure 2 ci-dessous avec trois clusters.

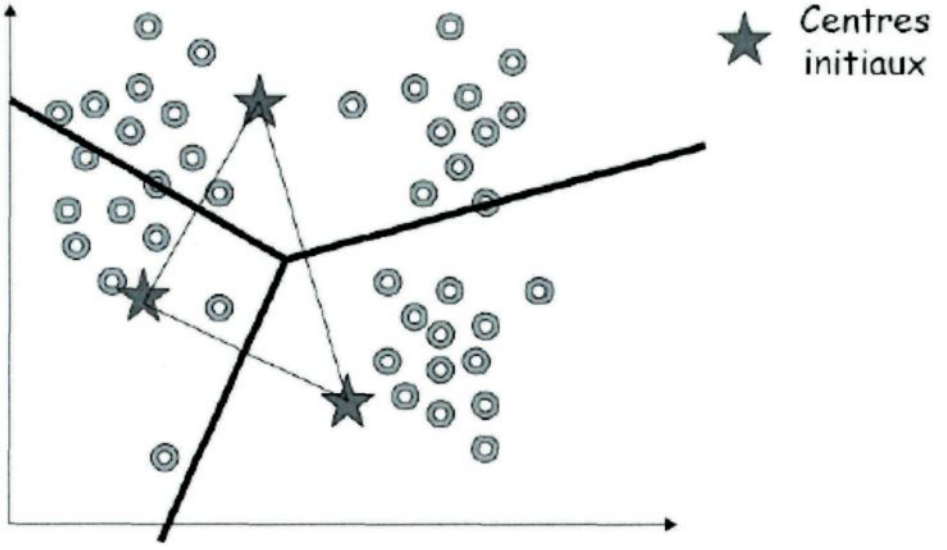


Figure 2 : Exemple d'initialisation du K-means avec trois clusters

Ensuite les centres de chaque cluster formés sont recalculés par rapport à la position des instances qu'ils contiennent. Puis, les instances sont réassignées à chacun des clusters en fonction de leur distance euclidienne par rapport aux nouveaux centres. Voir la Figure 3.

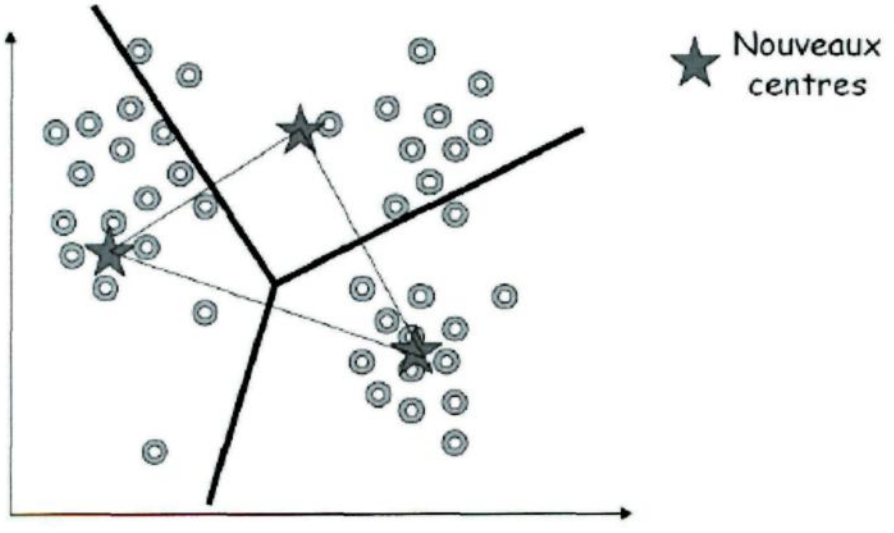


Figure 3 : Mise à jour des centres après une itération de K-means

Enfin, ce processus est répété jusqu'au moment où les centres des clusters varieront très peu d'une itération à l'autre, on appelle ce changement la stabilisation des centres de gravité. Ainsi, on obtient des clusters qui ne se recoupent pas, et englobe toutes les instances du jeu de données. Comme le montre la Figure 4.

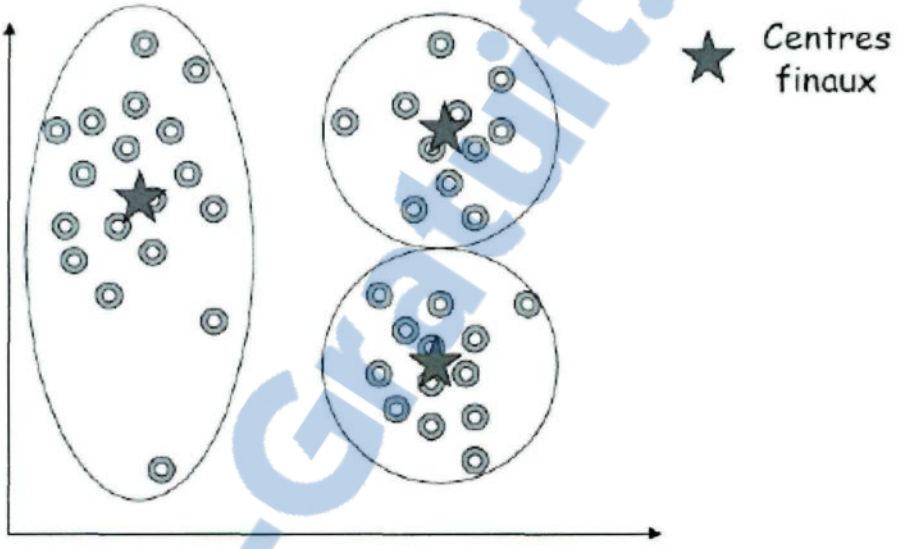


Figure 4 : Exemples de clusters finaux obtenus avec K-means

Nous venons de voir un exemple d'exécution du K-means avec un jeu de données précis, mais avant de décrire chaque instruction de l'algorithme il est important de définir les variables qui sont utilisées. Soit X un ensemble de données, chacune décrite par P attributs. On nomme « centre de gravité » g de X une donnée synthétique dont chaque attribut est égal à la moyenne de cet attribut dans X : $g = (m_1, m_2, \dots, m_p)$. Soit I_w l'inertie intraclasse où w_i est le poids du groupe G_i. Si toutes les données ont le même poids, le poids d'un groupe est $w_i = \frac{|G_i|}{N}$ avec $|G_i|$ est le cardinal du groupe G_i.

Algorithme K-means

Nécessite 2 paramètres : le jeu de données X, le nombre de groupe à constituer $k \in N$
$I_w \leftarrow \infty$
Prendre k centre arbitraries $c_j \in D$
Répéter
  Pour $j \in \{1, \dots k\}$ faire
    $G_j \leftarrow \emptyset$
  Fin pour
  Pour $i \in \{1, \dots N\}$ faire
    $j^* \leftarrow \arg\min_{j \in \{1,\dots k\}} d(x_i, c_j)$
    $G_{j^*} \leftarrow G_{j^*} \cup \{x_i\}$
  Fin pour
  Pour $j \in \{1, \dots k\}$ faire
    $c_j \leftarrow$ *centre de gravité de* G_j
  Fin pour
  ReCalculer I_w
Jusque $I_w < seuil$

Algorithme 1 : K-means

L'algorithme K-means converge rapidement vers une solution dû à sa complexité linéaire, c'est l'un des algorithmes de segmentation le plus performant en terme de temps de calcul, qui résulte du nombre d'itérations dont il a besoin pour former ses clusters [19]. Cependant, cet algorithme a plusieurs inconvénients. Le premier est que les clusters finaux sont très dépendants de leurs positions initiales qui a été établi semi-aléatoirement. Le second est que l'algorithme converge vers un minimum local, les centres des clusters se déplacent pour réduire la distance avec leurs données, mais il n'y a pas de garantie que ce soit la distance globale minimale. Une version amélioré de l'algorithme appelé K-means++ [19] consiste à choisir le centre du premier cluster aléatoirement parmi l'ensemble des données afin qu'il ait une probabilité de distribution uniforme, puis de déterminer le centre suivant de sorte que sa position soit proportionnelle à une certaine distance au carré par

rapport au premier, et ainsi de suite jusqu'au nombre de clusters désirés. Cela augmente ainsi la vitesse d'exécution et la précision de l'algorithme. Malgré tout, il faut quand même choisir un nombre de clusters au départ et donc avoir connaissance à l'avance du problème à résoudre pour que l'algorithme soit efficace. Néanmoins, dans notre contexte d'habitat intelligent nous ne savons pas à l'avance combien d'activités vont être réalisées, ainsi nous avons besoin d'un algorithme qui n'a pas besoin de préciser le nombre de clusters à l'avance. Parmi les méthodes de segmentations, l'algorithme Expectation Maximisation (EM) fait partie de ceux qu'on peut exécuter sans lui préciser le nombre de clusters, nous allons ainsi détailler son fonctionnement.

2.2.2 EXPECTATION MAXIMISATION (EM)

Cet algorithme, proposé par Dempster et al. en 1977 [35], permet de trouver le maximum de vraisemblance des paramètres de modèles probabilistes [19]. Le maximum de vraisemblance est une statistique utilisée pour inférer les paramètres de la distribution de probabilité d'un échantillon donné, autrement dit c'est cette statistique qui va permettre de trouver le nombre de clusters qu'il doit créer, si on ne lui indique pas ce nombre à l'avance. En effet, l'algorithme EM qui est plus connu avec son nom anglais que son nom français : Espérance-Maximisation, permet de choisir ou non le nombre de clusters. Dans le cas où on ne lui indique pas de nombre, il va le chercher lui-même.

La première étape E, l'évaluation de l'espérance, consiste à calculer l'espérance de la vraisemblance en tenant compte des dernières variables observées de l'ensemble de données. La seconde étape M, ou maximisation, consiste à estimer le maximum de

vraisemblance des paramètres en maximisant la vraisemblance trouvée à l'étape E. Puis, l'algorithme itère ces deux étapes en réutilisant les paramètres trouvés en M comme nouveau point de départ de la nouvelle phase E. Ces deux étapes permettent donc de créer les clusters à partir de l'ensemble de données, puis d'améliorer leurs centres et ou d'en créer de nouveau si nécessaire au fur et à mesure des itérations. À noter que ce ne sont pas vraiment les clusters qui sont créés, mais la probabilité des clusters, celle-ci étant calculée à partir des instances qui seraient contenues dans le cluster s'il était créé.

Algorithme EM

Nécessite 2 paramètres : le jeu de données X, une mixture de paramètres $\theta_{k,k\in\{1,\dots K\}}$
Initialiser, éventuellement aléatoirement, les paramètres de la mixture : les θ_k et les w_k
$I_{new} \leftarrow -\infty$
Répéter
  $I \leftarrow I_{new}$
  Pour $k \in \{1, \dots K\}$ faire
    Pour $i \in \{1, \dots N\}$ faire
      estimer $w_{i,k}$: la probabilité pour x_i d'appartenir à la distribution k : $w_{i,k} =$ Pr $[G_k|x_i]$ à l'aide de la formule décrivant la distribution du groupe G_k
    Fin pour
    $w_k \leftarrow \sum_{i=1}^{i=N} w_{i,k}$
  Fin pour
  Pour $k \in \{1, \dots K\}$ faire
    $calculer\ \theta_k$
  Fin pour
  $I_{new} \leftarrow \log - vraisemblance$
Jusque $|I - I_{new}| < seuil$

Algorithme 2 : Expectation Maximisation (EM)

Enfin, l'algorithme EM termine les itérations lorsque la moyenne des probabilités des clusters est inférieure à un certain seuil, 10^{-10} par exemple, sur une dixième d'itérations successives. Toutefois, l'algorithme converge vers un maximum local à cause

de la toute première itération, pour obtenir un maximum global il faut répéter l'algorithme plusieurs fois.

Comme avec K-means les données sont statiques, ce sont seulement les clusters qui sont améliorés au fil des itérations. Nous allons étudier maintenant un autre type d'algorithme de segmentation qui est basé sur le déplacement des données pour la formation des clusters : les colonies de fourmis.

2.2.3 COLONIES DE FOURMIS

Les algorithmes de colonies de fourmis sont des algorithmes inspirés du comportement des fourmis et qui constituent une famille de métaheuristiques d'optimisation. Mis en lumière par Marco Dorigo et al. en 1990 [36], le premier algorithme s'inspire du comportement des fourmis recherchant un chemin entre leur colonie et une source de nourriture, pour la recherche de chemins optimaux dans un graphe. L'idée originale s'est depuis diversifiée pour résoudre une classe plus large de problèmes et plusieurs algorithmes ont vu le jour, s'inspirant de divers aspects du comportement des fourmis. En anglais, le terme consacré à la principale classe d'algorithme est « Ant Colony Optimization » (abrégé ACO), ou « Ant Clustering ».

L'idée originale provient de l'observation de l'exploitation des ressources alimentaires chez les fourmis. En effet, celles-ci, bien qu'ayant individuellement des capacités cognitives limitées, sont capables collectivement de trouver le chemin le plus court entre une source de nourriture et leur nid. Des biologistes ont ainsi observé, dans une série d'expériences menées à partir de 1989 [37], qu'une colonie de fourmis ayant le choix

entre deux chemins d'inégale longueur menant à une source de nourriture avait tendance à utiliser le chemin le plus court. Un modèle [38] expliquant ce comportement est le suivant :

Une fourmi, appelée « éclaireuse », parcourt plus ou moins au hasard l'environnement autour de la colonie. Si celle-ci découvre une source de nourriture, elle rentre plus ou moins directement au nid, en laissant sur son chemin une piste de phéromones. Ces phéromones étant attractives, les fourmis passant à proximité vont avoir tendance à suivre, de façon plus ou moins directe, cette piste. En revenant au nid, ces mêmes fourmis vont renforcer la piste. Si deux pistes sont possibles pour atteindre la même source de nourriture, celle étant la plus courte sera, dans le même temps, parcourue par plus de fourmis que la longue piste. La piste courte sera donc de plus en plus renforcée, et donc de plus en plus attractive. La longue piste, elle, finira par disparaître, les phéromones étant volatiles. À terme, l'ensemble des fourmis a donc déterminé et « choisi » la piste la plus courte.

Algorithme Colonie de fourmis

```
Initialisation des agents dans le monde virtuel
Initialisation des phéromones
Répéter
   Pour chaque agent a_i faire
      Chercher les phéromones les plus proches de l'agent a_i
      Pour chaque phéromone proches faire
         Sauvegarder la phéromone la plus attractive
      Fin pour
      Calculer le nouveau vecteur position de a_i
   Fin pour
   Pour chaque phéromone faire
      Evaporation partielle de la phéromone
   Fin pour
Jusque temps écoulé t > temps maximal
```

Algorithme 3 : Colonie de fourmis

Les fourmis peuvent ainsi être vu comme des agents simple réflexe [39]. Ce type d'agent choisit ses actions uniquement sur le percept courant en ignorant les percepts précédents. Ainsi, la colonie de fourmis s'adaptera de façon relativement flexible aux changements. A partir de ces avantages, des recherches ont été effectué afin d'utiliser les colonies de fourmis comme algorithme de segmentation [40, 41]. Le principe est le suivant : Chaque agent se déplace d'une manière aléatoire au départ, puis en suivant les phéromones des autres agents. Durant son déplacement l'agent peut lacher les données qu'il transporte selon une certaine probabilité. S'il rencontre sur son chemin des données et qu'il n'en transporte pas, il peut ramasser ces nouvelles données. Après plusieurs itérations des clusters vont émerger à partir de l'activité collective. De plus, les colonies de fourmis n'ont pas besoin de connaitre le nombre de clusters au départ puisqu'ils vont être créés par les agents.

Cependant, les algorithmes des colonies de fourmis nécessitent un grand nombre d'itérations pour générer une bonne segmentation car les agents ne communiquent pas entre eux et peuvent se retrouver isolé durant une longue période. C'est pourquoi, d'autres recherches se sont inspirés de comportement plus évolués de la nature, comme celui du Flocking.

2.3 LE FLOCKING ET SES UTILISATIONS COMME ALGORITHME DE SEGMENTATION

Le Flocking, mis en lumière par Craig Reynolds [42], est à la base, un comportement attribué à certains animaux lorsqu'ils se déplacent en groupe, comme les oiseaux ou les poissons par exemple. Toutefois, le Flocking est plus qu'un simple modèle,

c'est un comportement émergent [43]. Ces comportements peuvent sembler complexes aux yeux des observateurs, mais en réalité ils reposent sur des règles très simples. Les agents virtuels, premièrement désignés avec le terme « boid » par Reynolds, suivent ces règles sans connaitre le but recherché, ils sont seulement conscients d'eux-mêmes et de quelques-uns de leurs voisins. Le modèle du Flocking [42] repose donc sur une combinaison de trois règles : l'alignement, la séparation et la cohésion. Ces règles, présentées à la figure 1, sont exécutées à chaque boucle du programme pour tous les agents, et permettent ainsi un déplacement cohérent.

L'alignement donne la même direction aux agents, elle est calculée en faisant la moyenne des directions de leurs voisins. La séparation éloigne les agents entre eux en fonction de la distance qui les sépare. La cohésion attire l'agent vers le centre de masse de ses voisins.

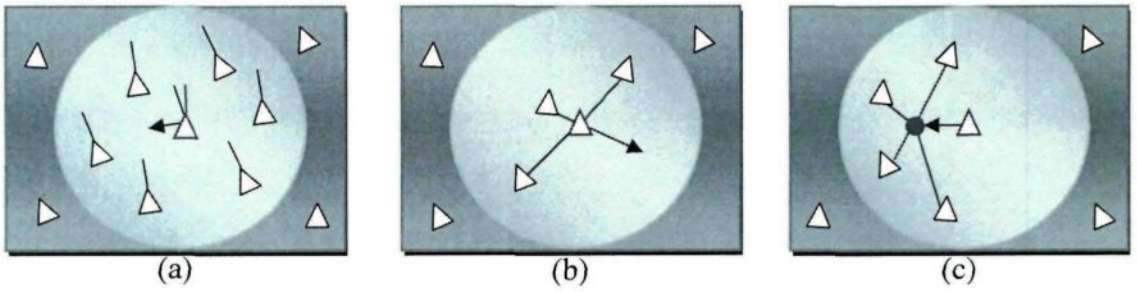


Figure 5 : Règles de base du Flocking
(a) Alignement (b) Séparation (c) Cohésion

A ce jour, le Flocking est principalement utilisé dans les jeux vidéo afin de déplacer des agents d'une façon plus naturelle, ou pour faire des simulations de la vie réelle. Mais, il

commence à être de plus en plus utilisé dans l'exploration de données. Cependant, les trois règles de bases du Flocking ne sont pas suffisantes pour faire de la segmentation, car elles forment un seul cluster dans la majorité des cas. Pour regrouper les données similaires et séparer les données différentes, il est nécessaire d'ajouter au moins une nouvelle règle qui va permettre la création de plusieurs clusters, on trouve souvent le terme « Similarité » pour désigner cette règle additionnelle. On peut observer ce genre d'implémentation dans le travail de Cui et al. [44]. Dans l'une de leurs recherches, ils utilisent le Flocking comme un algorithme de segmentation de documents, en d'autres mots ils se servent du Flocking pour trier les documents dans un espace virtuel en 2D. Ceux ayant le plus de similarité par rapport aux mots qu'ils contiennent se regroupent ensemble et forment des groupes facilement visibles à l'écran, ce qui permet ensuite de classer ces documents par catégorie et ainsi remplacer l'expert humain. Dans une autre recherche, Bellaachi et Bari [45] se sont servi du Flocking pour détecter les valeurs aberrantes dans les puces du cancer, ou « cancer microarrays », afin de les corriger. En effet, si un agent est éloigné de son groupe, il peut être ramené facilement en lui appliquant les règles du Flocking par rapport aux autres agents. Cela permet ainsi d'enlever les erreurs des puces.

Ainsi, le Flocking semble être un algorithme très performant dans l'exploration de données grâce aux segmentations qu'il forme rapidement. Voyons maintenant en détail l'utilisation de cet algorithme au travers des recherches citées précédemment.

2.3.1 LA SEGMENTATION DE DOCUMENT AVEC DU FLOCKING PAR CUI ET AL.

Dans cette section, nous détaillons le travail de Cui et al. [44]. Leurs recherches reposent sur l'utilisation d'un algorithme de segmentation non supervisé pour trier un ensemble de documents automatiquement, à la place d'un expert humain. Pour ce faire, ils ont utilisé les 3 règles de base du Flocking, ainsi que 2 nouvelles règles appelées respectivement similarité et dissimilarité.

2.3.1.1 REPRESENTATION DES DOCUMENTS

Comme dans de nombreux algorithmes de partitionnement, les données à segmenter sont représentées sous la forme d'un ensemble de vecteurs $X = [x_1, x_2, \dots, x_n]$, où x_i corresponds à un vecteur contenant les caractéristiques d'un document. Ces caractéristiques sont représentées avec le modèle « Vector Space Model » (VSM) [46]. Dans ce modèle, le contenu d'un document est formalisé comme un point dans un espace multidimensionnel, et représenté par un vecteur tel que $x = [w_1, w_2, \dots, w_n]$. Chaque w_i Corresponds à l'importance ou poids d'un terme dans un document. Afin de calculer le poids de toutes les expressions des documents, ils utilisent le système de pondération nommé « Term Frequency with Inverse Document Frequency » (TF-IDF) [46].

$$w_{ji} = TF_{ji} * IDF_{ji} = TF_{ji} * log_2(\frac{n}{DF_{ji}}) \qquad (3)$$

Où TF_{ji} est le nombre d'occurrence du terme i dans le document j, DF_{ji} indique la fréquence du terme dans la collection de documents, et n est le nombre total de document dans la collection. Ainsi, un mot avec une haute fréquence d'apparition dans un document,

et une basse fréquence dans la collection de documents obtiendra un poids élevé. Une fois les poids de chaque mot déterminé on peut calculer la similarité de deux documents.

2.3.1.2 MESURE DE SIMILARITE

La mesure de similarité choisie par les auteurs pour comparer les documents est la distance euclidienne, représentée par l'équation ci-dessous.

$$d(x_p, x_j) = \sqrt{\sum_{k=1}^{d_x} (w_{k,p} - w_{k,j})^2} \qquad (4)$$

Où $x_p \; et \; x_j$ correspondants aux vecteurs des deux documents à comparer, d_x est la dimension du vecteur document, $w_{k,p}$ et $w_{k,j}$ représentent respectivement les poids des termes des documents $x_p \; et \; x_j$ dans la dimension k.

Ainsi, ils obtiennent une distance qui symbolise la similitude d'une paire de documents. Plus cette distance est petite, plus les documents en question contiennent les mêmes termes. Une fois ces distances calculées le Flocking peut être appliqué. Comme présenté dans la section sur le Flocking : l'alignement, la séparation et la cohésion sont insuffisantes à elles seules pour faire des clusters cohérents. Ainsi, les auteurs ont ajouté non pas une, mais deux nouvelles règles : similarité et dissimilarité, afin de permettre une création efficace des clusters. Étudions donc les choix d'implémentation faits par Cui et al. pour ces deux règles.

2.3.1.3 DEUX NOUVELLES REGLES : SIMILARITE ET DISSIMILARITE

La règle de similarité consiste à garder les documents semblables proches les uns des autres dans le monde virtuel. Pour ce faire, la force d'attraction est proportionnelle à la distance entre les deux documents du monde virtuel multiplié par leur distance de similarité déterminé par l'équation 4. En voici la formule mathématique : $\overrightarrow{v_{ds}}$ est la vélocité de similarité, $S(B,X)$ est la similarité entre les termes des documents B et X, et $d(P_x, P_b)$ est la distance des documents dans le monde virtuel.

$$\overrightarrow{v_{ds}} = \sum_{x}^{n} (S(B,X) * d(P_x, P_b)) \qquad (5)$$

Au contraire, la règle de dissimilarité tend à éloigner les documents présentant peu de termes communs. La force de répulsion est égale à l'inverse de la force d'attraction de la similarité, comme le montre la formule suivante.

$$\overrightarrow{v_{dd}} = \sum_{x}^{n} \frac{1}{S(B,X) * d(P_x, P_b)} \qquad (6)$$

Lorsque les 5 vélocités sont calculées, v_{ar}, v_{sr}, v_{cr}, v_{ds} et v_{dd} respectivement alignement, séparation, cohésion, similarité et dissimilarité, elles sont multipliées par un poids spécifique prédéfini au départ pour chacune d'elles, puis elles sont additionnées entre elles. On obtient ainsi une force résultante pour chaque document dans le monde virtuel qui correspond à leur nouvelle vélocité.

2.3.1.4 EXPERIMENTATIONS ET RESULTATS

Pour effectuer les expérimentations de leur algorithme, les auteurs ont utilisé deux ensembles de données, un synthétique et un réel. Les données synthétiques contiennent 200 objets 2D (X,Y). Quant aux données réelles, elles représentent 100 articles collectés sur Internet et classés en 12 catégories par un expert humain, comme le montre le Tableau 1.

The document collection dataset

	Category/topic	Number of articles
1	Airline safety	10
2	Amphetamine	10
3	China and Spy Plane and Captives	4
4	Hoof and Mouth Disease	9
5	Hurricane Katrina	5
6	Iran Nuclear	8
7	Korea and Nuclear Capability	10
8	Mortgage Rates	10
9	Ocean and Pollution	6
10	Saddam Hussein and WMD	8
11	Storm Irene	10
12	Volcano	10

Tableau 1 : Données d'expérimentations de Cui et al. (100 articles)

Leur protocole expérimental a été de tester les deux ensembles de données avec trois algorithmes : K-means, colonies de fourmis, et leur version du Flocking. Chacun des algorithmes a utilisé la distance euclidienne comme mesure de similarité. Ensuite, ils ont évalué ces expérimentations en utilisant la mesure de F qui combine la précision et le rappel [19, 20]. Dans le cas d'une classification binaire à 2 classes, positif ou négatif, la précision mesure la proportion d'exemples vraiment positifs parmi ceux sont classés positifs, et même chose avec les négatifs, et le rappel mesure la proportion d'exemples

vraiment positifs parmi tous les exemples positifs, et même chose avec les négatifs. Dans le cas d'une segmentation la mesure de F détermine la pureté des clusters, si un cluster contient des données d'une seule classe il est complètement pur, et au contraire plus il y aura de classes différentes dans un cluster moins il sera pur. Une fois F calculé pour tous les clusters une moyenne est faite, plus la valeur est haute et plus la segmentation est de bonne qualité, sachant qu'elle est comprise entre 0 et 1. Chaque algorithme a été exécuté 20 fois chacun avec un arrêt systématique à la 300e itération. Le Tableau 2 présente les résultats obtenus. On s'aperçoit que leur implémentation du Flocking obtient une meilleure segmentation après 300 itérations, surtout pour les données réelles des 100 documents. L'algorithme des colonies de fourmis « Ant » ne peut pas effectuer une segmentation acceptable en si peu d'itérations pour les données réelles, et K-means obtient un résultat moyen.

The results of *K*-means, Ant clustering and Flocking clustering Algorithm on synthetic and real datasets after 300 iterations

Document Type	Algorithms	Average cluster number	Average *F*-measure value
Synthetic	Flocking	4	0.9997
Synthetic	*K*-means	(4)	0.9879
Synthetic	Ant	4	0.9823
Real	Flocking	10.083	0.8058
Real	*K*-means	(12)	0.6684
Real	Ant	1	0.1623

Tableau 2 : Résultats de segmentations de Cui et al. après 300 itérations

On peut observer la distribution des documents dans le monde virtuel 2D à travers la Figure 5. La partie (a) montre la distribution initiale des documents, et la partie (b)

montre le résultat obtenu après 300 itérations avec l'algorithme du Flocking. Une couleur à été associé à chaque catégorie des documents, on visualise mieux les clusters formés, et on peut ainsi constaté les documents d'une même couleur se sont regroupés et ont formés des clusters. De même, on peut voir que certains documents de deux catégories relativement similaires (Storm Irene en rose et Hurricane Katrina en orange) se sont proches les uns des autres malgré qu'ils n'appartiennent pas aux mêmes catégories. Cela est dû au fait que les termes contenus dans ces documents se recroisent et c'est tout à fait cohérent qu'ils fassent partie du même cluster, même s'ils ont été assignés en 2 catégories différentes par l'expert humain.

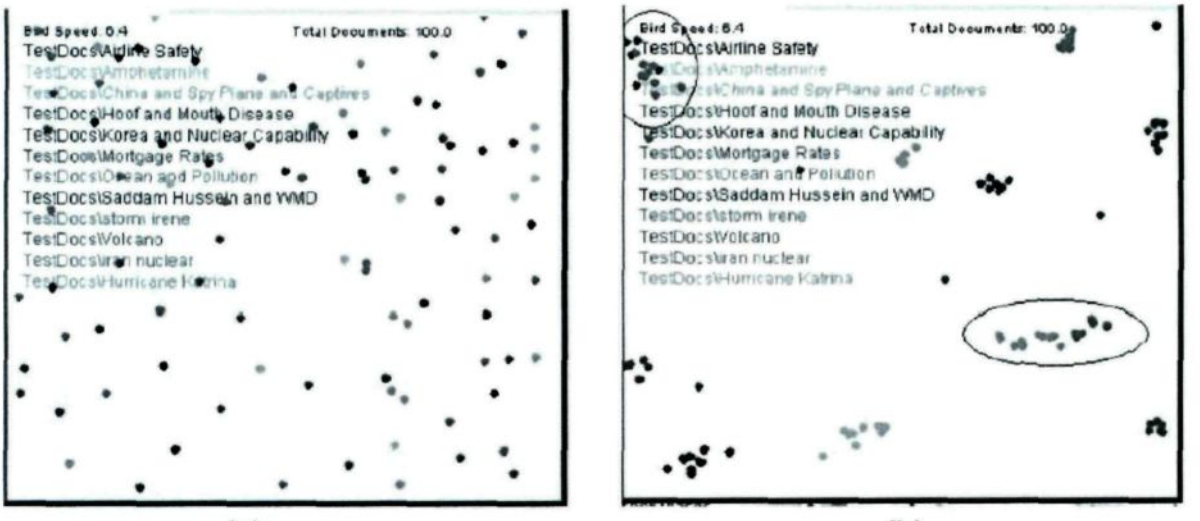


Figure 6 : Distribution de 100 articles avec le flocking de Cui et al.
(a) Positions initiales des données (b) Positions après 300 itérations

Les recherches de Cui et al. démontrent par leurs recherches l'efficacité du Flocking comme méthode de segmentation par rapport au technique classique tel que K-means et les colonies de fourmis. Malgré que l'algorithme K-means reste le plus performant en terme de

temps d'exécution, le paramètre k à fixer au départ est un véritable inconvénient sur la qualité de la segmentation. De plus, le Flocking permet d'ajouter de nouvelles instances facilement à n'importe quel moment de l'exécution de l'algorithme, car les agents continent à se déplacer même lorsque les clusters sont formés.

Étudions maintenant l'utilisation du Flocking dans le domaine de la biologie.

2.3.2 DETECTION DES « OUTLIERS » DE PUCE A ADN AVEC LE FLOCKING

Dans cette section, nous allons étudier le travail de Bellaachi et Bari [45]. Leurs recherches reposent sur l'utilisation du Flocking pour détecter les valeurs aberrantes, ou « outliers » en anglais, des puces à ADN du Cancer afin d'isoler et de supprimer ces valeurs qui sont en fait du bruit.

2.3.2.1 REPRESENTATION DES DONNEES

Comme pour les travaux de Cui et al., les données à segmenter sont représentées sous la forme d'un ensemble de vecteurs $S = [s_1, s_2, \dots, s_x]$, où s_i corresponds à un vecteur contenant un échantillon de la puce à ADN. Chaque vecteur $s_i = [d_1, d_2, \dots, d_m]$ contiennent m principaux composants de l'échantillon, déterminé au départ avec la méthode PCA « Principle Component Analysis » afin de garder seulement les gènes importants et ainsi réduire le nombre de données à traiter [47]. Enfin, la mesure de similarité choisie par les auteurs pour comparer les échantillons des puces à ADN est la distance euclidienne.

2.3.2.2 ALGORITHME OFLOSCAN

L'algorithme OFLOSCAN a pour but d'utiliser le Flocking afin de séparer les échantillons incohérents des échantillons normaux des puces à ADN. Pour cela trois étapes sont nécessaires et résumées dans le pseudo-code suivant.

Algorithme OFLOSCAN

Nécessite 1 paramètre : le jeu de données X
Initialisation des agents dans le monde virtuel
Calculer la similarité de toutes les instances du jeu de données
Répéter
  Pour chaque Agent a_i de l'instance s_i faire
    Chercher les N voisins de l'agent a_i avec la distance maximale d_{max}
    Pour chaque voisin a_j de l'agent a_i faire
      Calculer le vecteur aligement et attraction de a_i par rapport à a_j
      Calculer le nouveau vecteur direction de a_i
    Fin pour
    Calculer le nouveau vecteur position de a_i
  Fin pour
Jusque sous-ensemble optimal trouvé

Algorithme 4 : OFLOSCAN

2.3.2.3 EXPERIMENTATIONS ET RESULTATS

Pour effectuer les expérimentations, les auteurs ont utilisé comme ensemble de données une puce à ADN du cancer du côlon contenant 62 instances dont 40 sont des échantillons issus de tissus de tumeur et 22 de tissus normaux. Ils ont également choisi de comparer les résultats de leur algorithme avec cinq autres méthodes de forage de données spécialisé à ce problème, par exemple ils utilisent les SVM, ou « Séparateurs à Vaste Marge », qui ramène un problème de classification à un problème d'optimisation en se basant sur un sous ensemble de données appelées supports et l'astuce du noyau, cette

technique fait partie des méthodes d'apprentissage supervisé du forage de données [48, 49]. Les résultats obtenus sont présentés dans la figure 4.

	Tumor Outliers					Normal Outliers				Extra Samples Detected	
	T2	T30	T33	T36	T37	N8	N12	N34	N36	Tumor	Normal
SVM Classifier & LOOCV [19]		✔	✔	✔		✔		✔	✔		
ROBPCA[20]						✔				T5,T6,T12,T19, T25,T29,T39, T40	N4,N8, N29
CL-stability (SVM) [17]	✔	✔	✔	✔		✔					N2,N28
CL-stability (3-NN) [17]		✔		✔		✔		✔	✔		N2, N7, N27, N39
Enhanced ROBPCA[3]	✔	✔	✔	✔		✔	✔	✔	✔		
OFLOSCAN	✔	✔	✔	✔		✔	✔	✔	✔	T3,T30,T32,T39	N3,N6,N9,N15

Tableau 3 : Valeurs détectées par OFLOSCAN et d'autres méthodes

[45] page 6

On observe ainsi que l'algorithme OFLOSCAN arrive à détecter toutes les outliers connues, mais aussi quelques échantillons qui peuvent être des valeurs aberrantes sans être connus à l'avance. Le Flocking permet donc une meilleure efficacité de détection par rapport aux autres algorithmes comparés.

2.4 CONCLUSION SUR LA REVUE DE LITTERATURE

Ce chapitre avait pour but de présenter un état de l'art sur les méthodes de forages de données. Nous avons décrit les caractéristiques et les limites des trois catégories d'approches utilisées dans les habitats intelligents : logiques, temporelles et spatiales. Ensuite, nous avons mis en lumière quelques méthodes de segmentations comme K-means, EM et les colonies de fourmis, avec leurs avantages et leurs inconvénients respectifs. Puis, nous avons montré deux utilisations du Flocking comme algorithme de fouille de données pour la segmentation de document et de cellules cancéreuses des puces à ADN, avec dans les deux recherches de bons résultats obtenus. C'est pourquoi nous avons décidé de nous inspirer des travaux de [44] pour développer une extension du Flocking dans le cadre des habitats intelligents afin de dépasser les limites des méthodes classiques de reconnaissance d'activités.

Le chapitre suivant présente donc notre extension du Flocking pour la segmentation des actions d'une personne dans un habitat intelligent.

CHAPITRE 3

EXTENSION DU FLOCKING ET IMPLEMENTATION

3.1 INTRODUCTION

Dans le chapitre précédent, nous avons présenté quelques approches qui ont montré l'intérêt de l'utilisation du Flocking dans la segmentation de données dans deux domaines différents : la segmentation de documents et la biologie. Les résultats obtenus sont très intéressants comparés aux méthodes classiques de forage de données comme K-means et les colonies de fourmis pour les recherches de Cui et al. [44], et SVM pour les travaux de Bellaachi [45]. Cependant, le modèle initial du Flocking proposé par Craig Reynolds a quelques limites en tant que méthode de forage de données. En effet, dans un monde fini en deux dimensions, les agents virtuels vont former très rapidement un seul et unique cluster, car il n'y a aucune discrimination entre les agents. De même, si un agent se retrouve seul il ne pourra plus bouger si on applique les règles de bases du Flocking tel quel. Afin de dépasser ces limites, nous avons modifié légèrement la règle de l'alignement et nous avons ajouté deux nouvelles règles : la similarité et la dissimilarité. De plus, pour accélérer le temps de traitement de chaque itération nous avons découpé le monde en zone de même taille. En effet, avec cette segmentation si le nombre d'agents dans le monde est doublé alors le temps de calcul sera seulement doublé au lieu d'être au carré comme pour les algorithmes de complexité $O(n^2)$ [43]. Lorsque le nombre d'agents est petit, ce découpage du monde est peu significatif, mais dans notre contexte d'habitat intelligent le nombre de

données liées aux capteurs est très nombreux, ce qui en fait un atout considérable pour la rapidité d'exécution de notre algorithme.

Nous allons donc voir au travers de ce chapitre, notre implémentation du Flocking dans le cadre d'un habitat intelligent, nous allons détailler chacune des règles qui constituent l'algorithme et la méthode utilisée pour segmenter le monde virtuel. Ensuite, nous présenterons les spécifications liées à notre habitat intelligent, le Laboratoire d'Intelligence Ambiante pour la Reconnaissance d'Activité (LIARA), dans lequel nous avons testé notre approche. Enfin, nous partagerons les résultats obtenus et les comparerons avec les autres algorithmes de segmentations tels que K-means et EM. Commençons tout d'abord par présenter les forces d'alignement, de séparation et de cohésion qui constitue la base du Flocking.

3.1.1 FORCES D'ALIGNEMENT, DE SEPARATION ET DE COHESION

3.1.1.1 ALIGNEMENT

La force d'alignement a pour objectif de garder les agents alignés avec leurs voisins et dans la même direction. La force est calculée en faisant la somme des vecteurs directions des voisins de l'agent courant, en y ajoutant son propre vecteur direction, puis en divisant le tout par le nombre de voisins plus un. L'ajout de sa propre direction permet qu'il continue d'avancer s'il se retrouve seul. On considère k le nombre de voisins locaux de l'agent courant, $\overrightarrow{Ha}$ est le vecteur direction de l'agent, $\overrightarrow{Hn}$ est le vecteur direction des voisins, et l'équation de la force $\overrightarrow{F_A}$ donné par la règle d'alignement est :

$$\overrightarrow{F_A} = \frac{1}{k+1}\left(\overrightarrow{Ha} + \sum_{n}^{k} \overrightarrow{Hn}\right) \qquad (7)$$

On peut également représenter la force $\overrightarrow{F_A}$ par l'algorithme suivant :

```
FONCTION Alignement() RETOURNE Vecteur2D
    Vecteur2D moyenneDirection
    POUR CHAQUE voisin DANS agentsVoisins FAIRE
        moyenneDirection += voisin.direction
    FIN POUR
    moyenneDirection ← moyenneDirection / (nombreDeVoisins+1)
    RETOURNE moyenneDirection
FIN FONCTION
```

Algorithme 5 : Force d'alignement

3.1.1.2 SEPARATION

La force de séparation créer une force qui éloigne les agents proches. Quand elle est appliquée à plusieurs agents, ceux-ci vont se disperser en essayant de maximiser leur distance avec les agents les plus proches. On considère k le nombre de voisins locaux de l'agent courant, $\overrightarrow{Pa}$ est le vecteur position de l'agent, $\overrightarrow{Pn}$ est le vecteur position des voisins, et $\overrightarrow{F_P}$ est la force donné par la règle de séparation avec pour la formule suivante :

$$\overrightarrow{F_P} = \sum_{n}^{k} \frac{\overrightarrow{Pa} - \overrightarrow{Pn}}{\left\|\overrightarrow{Pa} - \overrightarrow{Pn}\right\|^2} \qquad (8)$$

On peut également représenter la force $\overrightarrow{F_P}$ par l'algorithme suivant :

```
FONCTION Séparation() RETOURNE Vecteur2D
    Vecteur2D directionVersAgent
    Vecteur2D forceSeparation
    POUR CHAQUE voisin DANS agentsVoisins FAIRE
        directionVersAgent ← agent.position – voisin.position
        forceSeparation += directionVersAgent.Normalise() / directionVersAgent.norme
    FIN POUR
    RETOURNE forceSeparation
FIN FONCTION
```

Algorithme 6 : Force de séparation

3.1.1.3 COHESION

La cohésion produit une force qui dirige l'agent vers le centre de masse de ses voisins. Cette règle est utilisée afin de garder les agents ensemble tel un groupe. Elle est calculée par la moyenne des vecteurs positions des voisins de l'agent courant, ceci résultant en centre de masse, et l'agent cherche à atteindre cette position. La force $\overrightarrow{F_C}$ dont l'équation est présentée ci-dessous a pour composants le vecteur position de l'agent courant $\overrightarrow{Pa}$, la vitesse maximum Ms de l'agent (constante prédéfinie), la vélocité de l'agent $\overrightarrow{Va}$, et le centre de masse du groupe $\overrightarrow{CoM}$. Ce dernier élément est déterminé par k le nombre de voisins et leur vecteur position $\overrightarrow{Pn}$.

$$\overrightarrow{F_C} = \left(\frac{\overrightarrow{CoM} - \overrightarrow{Pa}}{\|\overrightarrow{CoM}\|} * Ms\right) - \overrightarrow{Va}, \quad \overrightarrow{CoM} = \frac{1}{k}\left(\sum_{n}^{k} \overrightarrow{Pn}\right) \qquad (9)$$

On peut également représenter la force $\overrightarrow{F_P}$ par l'algorithme suivant :

On peut également représenter la force $\overrightarrow{F_P}$ par l'algorithme suivant :

```
FONCTION Cohésion() RETOURNE Vecteur2D
    Vecteur2D centreDeMasse
    Vecteur2D forceCohésion
    POUR CHAQUE voisin DANS agentsVoisins FAIRE
        centreDeMasse += positionVoisin
    FIN POUR
    centreDeMasse ← centreDeMasse / nombreDeVoisins
    forceCohésion ← DirectionVers(centreDeMasse)
    forceCohésion.Normalise()
    RETOURNE forceCohésion
FIN FONCTION
```

Algorithme 7 : Force de cohésion

À partir de ces trois forces, nous pouvons déjà former un cluster d'agents qui va se déplacer comme un seul et unique groupe. Cependant, pour faire de la segmentation sur des données nous avons besoin de créer plusieurs clusters. C'est pourquoi nous avons ajouté deux nouvelles règles : la similarité et la dissimilarité.

3.1.2 Extension du Flocking pour la segmentation

3.1.2.1 Dissimilarite

La force de dissimilarité quant à elle éloigne les agents entre eux comme la règle de séparation, mais uniquement entre les agents dissimilaire. L'équation de cette force étend celle de séparation en modulant la direction de l'agent voisin vers l'agent courant par la

fonction $d^*(a,b)$ qui est la distance euclidienne normalisée. Elle compare les données des agents dans le but de calculé une certaine distance entre a et b. Cette distance représente notre mesure de similarité, c'est-à-dire la différence entre les valeurs des attributs des agents, et la variable n est le nombre d'attributs de l'ensemble des données. L'équation de la fonction $d^*(a,b)$ est la suivante :

$$d^*(a,b) = \sqrt{\left(x_1^a - x_1^b\right)^2 + \cdots + \left(x_n^a - x_n^b\right)^2} \qquad (11)$$

Ainsi en multipliant $\overrightarrow{Pa} - \overrightarrow{Pn}$ de la formule de la force de séparation, on obtient la formule ci-dessous qui corresponds à la force de dissimilarité.

$$\overrightarrow{F_D} = \sum_{n}^{k} \frac{\left(\overrightarrow{Pa} - \overrightarrow{Pn}\right) * d^*(a,b)}{\left\| \overrightarrow{Pa} - \overrightarrow{Pn} \right\|^2} \qquad (12)$$

On peut également représenter la force $\overrightarrow{F_D}$ par l'algorithme suivant :

```
FONCTION Dissimilarité() RETOURNE Vecteur2D
    Vecteur2D forceVersAgent
    Vecteur2D forceDissimilarité
    double différence
    POUR CHAQUE voisin DANS agentsVoisins FAIRE
        différence ← 0
        CalculerLaDifférenceAvecLeVoisin(voisin, différence)
        forceVersAgent ← agent.position – voisin.position
        forceVersAgent ← forceVersAgent.Normalise() / forceVersAgent.norme
        forceDissimilarité += forceVersAgent * différence
    FIN POUR
    RETOURNE forceDissimilarité
FIN FONCTION
```

Algorithme 8 : Force de dissimilarité

Nous détaillerons la méthode qui calcule la différence avec un agent et son voisin dans la prochaine section, en même temps que nos choix sur la définition sur la similarité de deux agents.

3.1.2.2 SIMILARITE

La force de similarité produit une force semblable à la cohésion, mais seulement entre les agents similaires. Cependant, un nouveau centre de masse ($\overrightarrow{CoM_S}$) est calculé en prenant en compte le degré de similitude des agents. Les voisins peu semblables à l'agent courant auront moins d'influence sur le centre de masse par rapport aux agents ayant des attributs proche. On considère pour l'équation de la similarité ci-dessous : k le nombre de voisins locaux de l'agent courant, $\overrightarrow{Pa}$ est le vecteur position de l'agent, $\overrightarrow{Pn}$ est le vecteur position des voisins, et $S(a,b)$ est la similarité entre l'agent a et b.

$$\overrightarrow{F_S} = \left(\frac{\overrightarrow{CoM_S} - \overrightarrow{Pa}}{\|\overrightarrow{CoM_S}\|} * Ms\right) - \overrightarrow{Va}, \quad \overrightarrow{CoM_S} = \frac{1}{k}\left(\sum_{n}^{k}\left(\left(\overrightarrow{Pa} - \overrightarrow{Pn}\right) * S(a,b) * \overrightarrow{Pn}\right)\right) \quad (13)$$

La similarité $S(a,b)$ est déterminée à partir de fonction $d^*(a,b)$ comme ceci :

$$S(a,b) = 1 - d^*(a,b) \quad (14)$$

On peut également représenter la force $\overrightarrow{F_S}$ par l'algorithme suivant :

```
FONCTION Similarité() RETOURNE Vecteur2D
    Vecteur2D centreDeMasse
    Vecteur2D forceSimilarité
    double différence
    POUR CHAQUE voisin DANS agentsVoisins FAIRE
        différence ← 0
        CalculerLaDifférenceAvecLeVoisin (voisin, différence)
        forceSimilarité ← agent.position – voisin.position
        centreDeMasse += forceSimilarité * (1 – différence) * voisin.position
    FIN POUR
    centreDeMasse ← centreDeMasse / nombreDeVoisins
    forceSimilarité ← DirectionVers(centreDeMasse)
    forceSimilarité.Normalise()
    RETOURNE forceSimilarité
FIN FONCTION
```

Algorithme 9 : Force de similarité

3.1.2.3 FORCE RESULTANTE

Afin de compléter le comportement du Flocking, les résultats des règles précédentes sont multipliés par un certain poids spécifique à chacune d'entre elles. Ces poids ont été déterminés après de nombreux essais sur des données synthétiques afin de rendre le comportement des agents cohérent et efficace, les valeurs des poids sont comprises entre 0.5 et 3.0 dans la dernière version de l'algorithme et sont présenté dans le Tableau 4 ci-dessous.

Force	Similarité	Dissimilarité	Cohésion	Séparation	Alignement
Poids	1.5	3.0	0.8	1.2	0.5

Tableau 4 : Valeurs des poids associés aux forces

On considère donc w_X les valeurs prédéfinies des poids et $\vec{F}$ notre force de Flocking qui peut être vu comme la force résultante de la combinaison linéaire des autres forces.

$$\vec{F} = w_S\overrightarrow{F_S} + w_D\overrightarrow{F_D} + w_C\overrightarrow{F_C} + w_P\overrightarrow{F_P} + w_A\overrightarrow{F_A} \qquad (13)$$

Il est important de noter que les forces sont ajoutées dans cet ordre là précis, si l'une des forces dépasse la force maximale qu'un agent peut avoir, alors les forces suivantes ne sont pas ajoutées et la magnitude de la force résultante est maximale. On peut représenter le calcul de la force $\vec{F}$ par le pseudo-code ci-dessous :

```
FONCTION Flocking() RETOURNE Vecteur2D
    Vecteur2D flocking
    agent.DéterminerLesVoisins(agentsVoisins)
    SI nombreVoisin = 0 ALORS
        flocking ← agent.direction
    SINON
        SI ImpossibleAjouterForce(flocking, Similarité() * poidsSimilaritée) ALORS
            RETOURNE flocking
        FIN SI
        SI ImpossibleAjouterForce(flocking, Dissimilarité() * poidsDissimilarité) ALORS
            RETOURNE flocking
        FIN SI
        SI ImpossibleAjouterForce(flocking, Cohésion() * poidsCohésion) ALORS
            RETOURNE flocking
        FIN SI
        SI ImpossibleAjouterForce(flocking, Séparation() * poidsSéparation) ALORS
            RETOURNE flocking
        FIN SI
        SI ImpossibleAjouterForce(flocking, Alignement() * poidsAlignement) ALORS
            RETOURNE flocking
        FIN SI
    FIN SI
    RETOURNE flocking
FIN FONCTION
```

Algorithme 10 : Force résultante (Flocking)

Afin de mieux comprendre le précédent algorithme, il est nécessaire de détailler la fonction qui essaie d'ajouter chaque force à celle du Flocking, la fonction

ImpossibleAjouterForce. Celle-ci prend en paramètre deux vecteurs à deux dimensions, le premier étant la force cible et le second la force à ajouter. Si l'ajout de la force ne dépasse pas la force maximale alors elle sera ajoutée et la fonction retournera faux, sinon elle ne sera pas ajouté et vrai sera retourné. Voici le pseudo-code de la fonction.

```
FONCTION ImpossibleAjouterForce (Vecteur2D forceCible, Vecteur2D forceAjouter)
RETOURNE booléen
    double magnitude ← forceCible.norme
    double magnitudeRestante ← ForceMaximale - magnitude
    SI magnitudeRestante <= 0 ALORS
        RETOURNE  vrai
    FIN SI
    magnitude ← forceAjouter.norme
    SI magnitude < magnitudeRestante ALORS
        forceCible += forceAjouter
    SINON
        forceCible += forceAjouter.Normaliser() * magnitudeRestante
    FIN SI
    RETOURNE faux
FIN FONCTION
```

Algorithme 11 : Fonction ImpossibleAjouterForce

Maintenant que nous avons présenté notre extension du Flocking, nous allons voir l'implémentation de celui-ci, ainsi que les spécifications liées à l'habitat intelligent.

3.2 IMPLEMENTATION DU FLOCKING

Notre projet de recherche vise à explorer de nouvelles directions pour procurer une assistance cognitive aux personnes souffrant de la maladie d'Alzheimer. Comme nous l'avons présenté au préalable, la principale problématique dans ce projet est de reconnaitre et de prédire les activités de tous les jours (ADLs) des habitants. Afin de comprendre les différents choix techniques de notre approche, il est nécessaire de présenter le type

d'environnement où elle a été appliquée : le Laboratoire d'Intelligence Ambiante pour la Reconnaissance d'Activité (LIARA).

3.2.1 L'INFRASTRUCTURE DU LIARA

Ce laboratoire est un habitat intelligent construit sur cent mètres carrés à l'intérieur de l'Université du Québec à Chicoutimi (UQAC), il comprend les pièces standard d'un appartement : cuisine, salle à manger, chambre et salle de bain. Il est également équipé de nombreux capteurs, tags intelligents (RFID), tapis de pressions, systèmes de localisation et d'identification pour les objets et les résidents, appareils audio et vidéo, etc. La Figure 7 montre des images de l'habitat intelligent sous différents angles.

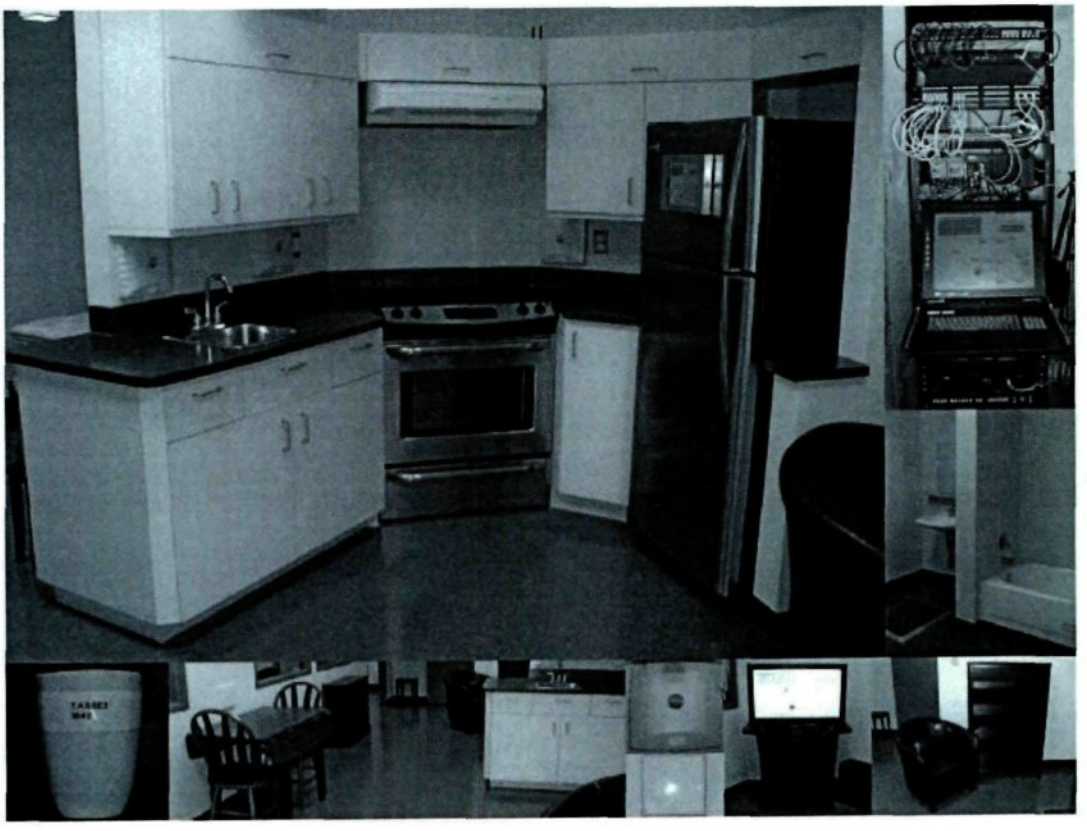


Figure 7 : Photos du LIARA

L'image centrale de la figure montre la cuisine où les activités les plus complexes sont effectuées. Ce sont les activités de préparation de repas qui se composent de plusieurs actions et utilisent différents objets comme la tasse, affichée en bas à gauche, sur laquelle nous avons installé une étiquette RFID. Encore sur l'image centrale nous pouvons constater l'iPAD installé en haut du réfrigérateur qui permet de contrôler les expériences faites dans l'habitat, tester les équipements, ou jouer des vidéos dans le but d'assister l'utilisateur dans ces activités complexes. La télévision, en bas à droite de la Figure 7, peut elle aussi être contrôlée à distance pour fournir le même type d'assistance. En haut à gauche, nous pouvons voir le serveur Dell qui se charge du traitement des informations, il récupère les données de tous les capteurs ou antennes RFID qui sont connectés à des ilots à chaque deux cent cinquante millisecondes. Les autres images montrent la salle à manger, les toilettes et la bibliothèque. Tous ces capteurs ont été choisis surtout pour leur facilité d'installation, car notre but est de transformer l'habitat du patient en un habitat intelligent et non pas de l'amener à vivre dans un laboratoire où les installations sont très compliquées. Ce type d'habitat nous permet ainsi de faire une reconnaissance non intrusive, à l'insu de la personne, une reconnaissance d'activités basée sur les objets. Dans ce contexte nous allons voir les spécifications liées au Flocking et à la base de données du LIARA, mais aussi nos choix dans la définition de la similarité entre les agents.

3.2.2 SPECIFICATIONS LIEES A LA BASE DE DONNEES

Les structures de données utilisées sont des tables de la base de données MS-SQL du LIARA. On extrait les données des tables qui correspondent aux capteurs et aux tags

RFID de l'habitat intelligent. Ces tables sont mises à jours toutes les deux cent cinquante millisecondes et se présentent sous deux formes différentes. La première table, celle des capteurs, contient deux lignes et environ cent colonnes. À chaque colonne sont associés le nom du capteur et sa valeur, qui peut être un entier (zéro ou un) ou un réel. On peut voir un échantillon de ces capteurs dans le Tableau 5 ci-dessous.

MV6	MT1	MT2	MT3	LTC	TC1	TC2
0	1	1	1	1	20.1	20.4

Tableau 5 : Échantillon de la table Capteurs_IA du LIARA

De plus, chaque capteur à un emplacement fixe dans l'habitat intelligent, on récupère cette information par un fichier dont le contenu est présenté dans le Tableau 6 suivant.

Nom	*Zone*	*X*	*Y*	*Description*
CA1	Cuisine	400	285	Capteur magnétique de panneau
CA2	Cuisine	400	265	Capteur magnétique de panneau
DB1	Cuisine	400	290	Capteur de débit
DB2	Cuisine	400	285	Capteur de débit
CB1	Cuisine	390	300	Capteur magnétique de panneau
CB2	Cuisine	390	200	Capteur magnétique de panneau
CB3	Cuisine	390	280	Capteur magnétique de panneau
CB4	Cuisine	390	260	Capteur magnétique de panneau
CA3	Cuisine	400	190	Capteur magnétique de panneau
CA4	Cuisine	400	170	Capteur magnétique de panneau
CA5	Cuisine	300	100	Capteur magnétique de panneau
CA6	Cuisine	250	100	Capteur magnétique de panneau
CB6	Cuisine	270	100	Capteur magnétique de panneau
MT1	Cuisine	365	130	Capteur de marche moteur vitesse 1
MT2	Cuisine	365	130	Capteur de marche moteur vitesse 2
MT3	Cuisine	365	130	Capteur de marche moteur vitesse 3
LTC	Cuisine	365	130	Capteur de marche et effecteur lumière hotte
LT1	Cuisine	365	130	Capteur de marche lumière hotte
CB7	Cuisine	250	250	Capteur de porte-cuisinière

CB8	Cuisine	250	250	Capteur de tiroir
MCR_i	Cuisine	250	250	Signal de marche du contacteur de cuisinière
LTF	Cuisine	250	250	Lumière intérieure du four
TC1	Cuisine	250	250	Capteur de température plaque gauche avant
TC2	Cuisine	250	250	Capteur de température plaque gauche arrière
TC3	Cuisine	250	250	Capteur de température plaque centrale
TC4	Cuisine	250	250	Capteur de température plaque droite avant
TC5	Cuisine	250	250	Capteur de température plaque droite arrière
TC6	Cuisine	250	250	Capteur de température four
RL13	Cuisine	250	250	Bouton de réarmement d'arrêt d'urgence
MV2	Chambre	450	450	Capteur de mouvement chambre
MV3	Chambre	450	450	Capteur de mouvement salle de bain
MV4	Chambre	450	450	Capteur de mouvement hall d'entrée
CD1	Chambre	450	450	Capteur infrarouge bibliothèque 1
CD2	Chambre	450	450	Capteur infrarouge bibliothèque 2
CD3	Chambre	450	450	Capteur infrarouge bibliothèque 03
CD4	Chambre	450	450	Capteur infrarouge bibliothèque 04
TP1	Chambre	450	450	Capteur tapis tactile chambre
CC1	Chambre	450	450	Capteur panneau du haut à gauche
CC3	Chambre	450	450	Capteur panneau du bas à gauche
CC2	Chambre	450	450	Capteur panneau du haut à droite
CC4	Chambre	450	450	Capteur panneau du bas à droite
TP2	Salle de bain	50	50	Capteur tapis tactile sale de bain
CC5	Salle de bain	50	50	Capteur magnétique de porte salle de bain
DB3	Salle de bain	50	50	Eau chaude bain adapté
DB4	Salle de bain	50	50	Eau froide bain adapté
DB5	Salle de bain	50	50	Eau chaude évier
DB6	Salle de bain	50	50	Eau froide évier
DB7	Salle de bain	50	50	Remplissage toilette
CE1	Salon	150	150	Capteur magnétique de porte salle d'entrée
MV5	Salon	150	150	Capteur de mouvement zone salon
CE2	Salon	150	150	Capteur magnétique de porte salle de conférence
CA10	Cuisine	250	250	Capteur réfrigérateur congélateur du haut
CA11	Cuisine	250	250	Capteur magnétique de porte du réfrigérateur
LD1	Salon	150	150	Capteur de luminosité entrée
LD2	Salon	150	150	Capteur de luminosité salle à manger
LD3	Cuisine	250	250	Capteur de luminosité cuisine
LD4	Chambre	450	450	Capteur de luminosité chambre
LD5	Salle de bain	50	50	Capteur de luminosité salle de bain
TC8	Cuisine	250	250	Température robinet évier cuisine
TC9	Salon	150	150	Température salle à manger
TC10	Chambre	450	450	Température chambre
TC11	Salle de bain	50	50	Température salle de bain

TC12	Salle de bain	50	50	Température robinet évier salle de bain
TC13	Salle de bain	50	50	Température robinet bain

Tableau 6 : Positions des capteurs dans le LIARA

La table des tags RFID quant à elle contient autant de lignes qu'il y a de tags dans l'habitat intelligent, et six colonnes qui sont visibles dans le Tableau 7 ci-dessous. Ces colonnes contiennent respectivement : le numéro d'identification du tag, son alias, sa position spatiale, les valeurs en X et Y de sa position, et sa visibilité. Ce dernier attribut sera égal à un si le tag est détecté par une antenne RFID ou zéro dans le cas contraire.

Num_tag	Alias	Position	PositionX	PositionY	Visible
00000000001B2A46...	Bol2	Dans le réfrigérateur	160	130	1
00000000001B2A46...	Lait	Dans le réfrigérateur	160	130	1
00000000001B2A46...	Bol4	Hors Champs	0	0	0
00000000001B2A46...	Bol3	Hors Champs	0	0	0
00000000001B2A46...	Bol3	Hors Champs	0	0	0
00000000001B2A46...	Sel	Hors Champs	0	0	0
00000000001B2A46...	Sucre	Cuisine Compoir du fond près de la cuisi	330	87	1
00000000001B2A46...	Assiette2	Hors Champs	0	0	0
00000000001B2A46...	Chaudron	Hors Champs	0	0	0
00000000001B2A46...	Tasse1	Hors Champs	0	0	0
00000000001B2A46...	Tasse4	Hors Champs	0	0	0
00000000001B2A46...	Verre1	Hors Champs	0	0	0
00000000001B2A46...	Beurre	Dans le réfrigérateur	160	130	1
00000000001B2A46...	Bouilloir	Hors Champs	0	0	0
00000000001B2A46...	Bol1	Salle commune sur la petite table	220	395	1
00000000001B2A46...	Assiette1	Cuisine Antenne2 droite de l'évier	406	180	1
00000000001B2A46...	Bol2	Dans le réfrigérateur	160	130	1
00000000001B2A46...	Lait	Dans le réfrigérateur	160	130	1
00000000001B2A46...	Poelon	Cuisine Sur la cuisinière	365	140	1
00000000001B2A46...	Bol4	Hors Champs	0	0	0
00000000001B2A46...	Bol1	Salle commune sur la petite table	220	430	1
00000000001B2A46...	Sucre	Cuisine Compoir du fond près du réfrigér	255	91	1
00000000001B2A46...	Tasse2	Chambre autour de la bibliothèque	591	300	1
00000000001B2A46...	Tasse3	Hors Champs	0	0	0
00000000001B2A46...	Assiette1	Cuisine entre Antenne 1 et 2	396	227	1
00000000001B2A46...	Poivre	Hors Champs	0	0	0
00000000001B2A46...	sAssiette1	Hors Champs	0	0	0

Tableau 7 : Échantillon de la table Def_RFID du LIARA

Cependant, les valeurs des attributs de la base de données changent fréquemment à cause des activités du patient, et ces changements de valeurs représentent indirectement les actions de l'utilisateur. Nous qualifierons d'ailleurs ces modifications de valeurs comme des évènements. Ensuite, pour utiliser notre algorithme étendu du Flocking nous devons représenter les données comme un ensemble mobile d'agents. C'est pourquoi chaque évènement dans la base de données est représenté par un agent dans le monde virtuel à deux dimensions. Lorsqu'un évènement est détecté, un nouvel agent est créé avec les nouvelles valeurs du capteur ou de l'étiquette RFID correspondante. Après sa création, l'agent applique immédiatement les règles du Flocking décrites plus tôt. Un agent est essentiellement défini par trois principaux attributs : région spatiale, position, et date de génération de l'évènement. Ces caractéristiques ont été choisies parmi les attributs de la base de données qui sont importants dans la comparaison de la similarité des agents afin d'obtenir une bonne segmentation. De nombreux paramètres peuvent être pris en compte pour définir les ressemblances entre des données, par exemple : durée de l'évènement, temps entre deux évènements, type (objet mobile ou immobile), relation spatiale, région spatiale (salle de bain, cuisine, chambre, etc.), position, heure de génération de l'évènement. Parmi l'ensemble de ces variables, nous avons choisi d'investiguer trois d'entre elles : la relation temporelle entre deux évènements, la région spatiale et la position cartésienne. La comparaison entre des données de deux agents contrôle si leurs valeurs correspondent les unes les autres : petit écart des temps de génération, positions proches et même région spatiale. De tels agents sont considérés similaires et sont attirés entre eux à cause de la règle de similarité. Au contraire, si leurs régions spatiales sont différentes, ou

s'ils sont éloignés, ou encore si la différence de temps de création est grande, alors la règle de dissimilarité les conduira à s'éloigner.

Les spécificités du LIARA et de sa base de données ont ainsi déterminé une grande partie de nos choix d'implémentations, mais pas tous. Ceux que nous allons voir maintenant sont plus liés directement à l'algorithme du Flocking et des méthodes de segmentation.

3.2.3 CHOIX D'IMPLEMENTATIONS DE L'ALGORITHME

Tout d'abord, nous avons choisi d'implémenter notre algorithme avec le langage Java avec pour principale raison la portabilité de ce langage de programmation. Chaque donnée est représentée comme un agent, et tous les agents suivent les cinq règles mentionnées au début du chapitre. Un agent peut seulement percevoir les autres agents présents dans sa distance de vue, plus cette distance est grande et plus rapide s'effectue la segmentation. En contrepartie, plus la distance de vue est grande et plus le temps de traitement de chaque agent augmente, surtout avec un nombre important d'agents. Ensuite, le Flocking étant un comportement basé sur le déplacement de certains animaux nous avons décidé d'ajouter une interface graphique permettant d'observer facilement les résultats de l'algorithme à l'œil nu, comme pour les autres recherches sur le Flocking [44, 45]. Les calculs de la physique et des graphismes sont bien entendu dans deux processus différents afin que le calcul des différentes forces ne soit pas pénalisé par l'affichage des agents, sachant que le monde virtuel 2D pour représenter les agents est de 500 x 500 pixels ce qui ne demande pas d'énormes ressources.

Nous allons maintenant détailler les différentes étapes d'une itération de l'algorithme pour le calcul de la physique de chaque agent.

3.2.3.1 LES ETAPES D'UNE ITERATION

Pour commencer, chaque itération est traitée en fonction du temps afin de garder le mouvement des agents fluides. La première étape est donc de calculer le temps écoulé depuis la précédente itération et de sauvegarder le temps actuel pour l'itération suivante.

La seconde étape est de mettre à jour la vélocité et la position de chaque agent en fonction du Flocking et du temps écoulé. Cette étape est celle qui demande le plus de traitement pour chaque itération, elle comporte de nombreux calculs qui sont résumés dans le pseudo-code suivant.

```
PROCEDURE MiseAJourAgent(double tempsEcoulé)
    anciennePosition ← position
    forceFlocking ← Flocking() / masse * tempsEcoulé
    velocité += forceFlocking
    velocité ← Tronquer(velocité, VitesseMax)
    position +=  velocité * tempsEcoulé
    SI velocité.normeAuCarré > 0 ALORS
        direction ← velocité.Normaliser()
        directionDroite ← Perpendiculaire(direction)
    FIN SI
    monde.cellules.MiseAJourEntité(this)
FIN PROCEDURE
```

Algorithme 12 : Mise à jour de la vélocité et de la position d'un agent

Enfin, la troisième et dernière étape n'est effectuée qu'une fois toutes les quarante millisecondes. Celle-ci a pour but de mettre à jour les clusters formés par le déplacement des agents. Comme les autres techniques de forage de données, nous avons séparé

l'exécution de l'algorithme en deux phases : la phase d'apprentissage et la phase de test, dont nous verrons les détails plus tard dans le mémoire. Ainsi, le calcul des clusters s'effectue de deux façons différentes selon si le programme se situe en phase d'apprentissage ou en phase de test. Durant la phase d'apprentissage, les clusters sont mis à jour récursivement à partir de la liste d'agent. Si la distance au carré entre deux agents est inférieure à leurs distances de vue, alors ils sont considérés dans le même cluster. Lorsque tous les agents ont été assignés à un cluster, le centre et le rayon du cluster sont calculés relativement aux agents à l'intérieur de celui-ci. Durant la phase de test, les clusters sont calculés à partir des agents représentatifs de chaque cluster. Ces agents sont créés au début de la phase de test, ils représentent une moyenne des valeurs des attributs de tous les autres agents contenus dans les clusters à la fin de la phase d'apprentissage.

Ainsi, chaque itération peut se résumer à trois étapes : calculer le temps écoulé par rapport à la précédente itération, mettre à jour la vélocité et la position des agents grâce au Flocking, et recalculer les clusters. Toutefois, lorsque le nombre d'agents est important le temps d'exécution de chaque itération augmente énormément, car le temps nécessaire au calcul du Flocking et des clusters dépend des agents voisins, ce qui donne une complexité de calcul quadratique $O(n^2)$ car on devrait comparer tous les agents entre eux pour obtenir ceux qui sont proches les uns des autres. Pour réduire cet effet, il existe différentes techniques de partitionnement du monde virtuel comme ceux utilisant des arbres tels que BSP trees, quad-trees, etc. Dans ce projet nous avons choisi une méthode simple qui est exécutée une seule fois à l'initialisation du programme : la technique du partitionnement du monde en zone [43].

3.2.3.2 PARTITIONNEMENT DU MONDE VIRTUEL EN ZONES

Avec cette méthode le monde à deux dimensions est divisé en plusieurs zones de taille identique, également appelé cellule. Chaque zone a une liste des entités qu'elle contient, et cette liste est mise à jour à chaque fois qu'un agent change de position. Si l'agent se déplace dans une nouvelle zone, il est supprimé de la liste de son ancien emplacement et ajouté à la liste de la zone actuelle. Ainsi, à la place de contrôler la distance entre chaque agent, nous avons seulement à déterminer les agents présents dans les zones voisines de l'agent actuel et ainsi la complexité de calcul devient linéaire $O(n)$.

Nous pouvons d' ailleurs résumer la procédure effectuée en trois étapes. Premièrement, la distance de vue de l'agent est approximée avec un carré. Deuxièmement, les zones qui s'entrecoupent avec le précédent carré sont testées pour voir si elles contiennent d'autres agents. Troisièmement, la distance entre chaque agent contenu dans ces zones et l'agent courant est calculée, si la distance est inférieure à sa distance de vue, alors l'autre agent est ajouté dans la liste des voisins de l'agent courant. La Figure 8 montre chacun des éléments nécessaires, et les agents entourés d'un cercle blanc sont ceux considérés comme voisin de l'agent en blanc.

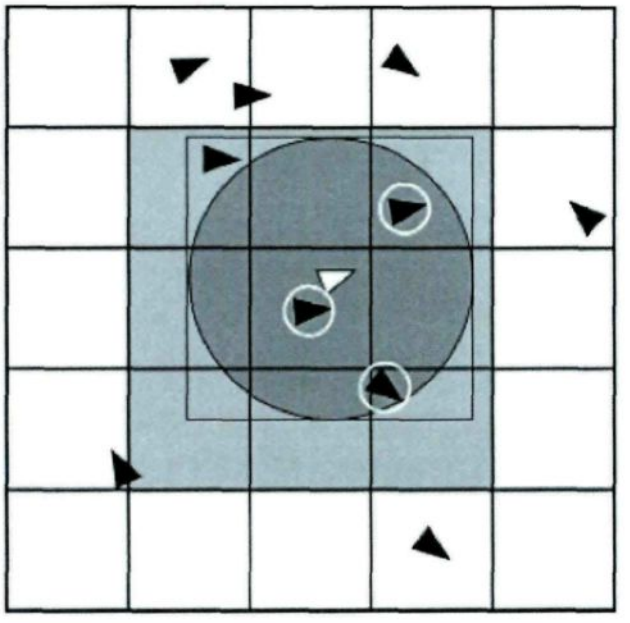


Figure 8 : Partitionnement du monde virtuel en zones.

Les choix d'implémentations de notre algorithme du Flocking étant présentés, nous allons voir maintenant les expérimentations que nous avons effectuées, ainsi que les résultats obtenus.

3.3 EXPERIMENTATIONS ET RESULTATS

Afin de tester l'efficacité et la précision de notre nouveau modèle, nous avons mené de nombreuses expérimentations dans notre habitat intelligent. Entre autres, nous avons réalisé cinq tâches différentes sans indications particulières, et nous avons enregistré les évènements de ces cinq scénarios pour pouvoir les refaire à l'identique autant de fois que nécessaires.

3.3.1 SCENARIOS DE CAS REELS

Ces scénarios contiennent seulement les informations pertinentes retournées par les capteurs et antennes RFID. De ce fait, les activités simples comme lire un livre contiennent moins d'évènements qu'une tâche demandant plus d'opérations complexe tel que cuisiner. En effet, peu de capteurs changent d'état lorsqu'un résident lit un livre. Le nom des scénarios et le nombre d'évènements qui sont associés sont donnés dans le Tableau 4 ci-dessous.

Nom du scénario	*Nombre d'évènements*
Lire un livre	12
Aller aux toilettes	14
Dormir dans la chambre	16
Cuisiner pour le souper	98
Cuisiner pour le déjeuner	101

Tableau 8 : Liste des scénarios de test

Tous les scénarios sont représentés par un fichier texte contenant six données par ligne, et chaque ligne symbolise un évènement. Dans l'ordre, le premier attribut correspond au type de l'évènement, soit déclenché par un capteur soit par un tag RFID. Le second attribut est le nom du capteur ou tag RFID, chacun d'eux à un nom particulier qui permet de l'identifier plus facilement dans le LIARA. Le troisième attribut correspond à la zone spatiale où se situe le capteur ou tag RFID. Les quatrième et cinquième attributs sont les positions en X et Y de l'objet, ces valeurs sont comprises entre 0 et 600. Les tags RFID sont posés sur des objets déplaçables comme des tasses ou des assiettes par exemple, de ce

fait leur position peut varier contrairement aux capteurs qui sont immobiles. Enfin, le sixième et dernier attribut est l'heure où est survenu l'évènement, celle-ci est exprimée en minutes à partir de minuit (0h00), par exemple 710 min est égale à 11h50 du matin.

Nous allons maintenant partager la liste complète des évènements de chaque scénario dans le même ordre que le Tableau 4 précédent.

3.3.1.1 SCENARIO LIRE UN LIVRE

Type	*Nom*	*Zone*	*X*	*Y*	*Temps*
Capteur	CD2	Chambre	450	450	694
Capteur	CD1	Chambre	450	450	694
RFID	Livre1	Chambre	580	300	694
Capteur	CD2	Chambre	450	450	694
Capteur	CD1	Chambre	450	450	694
Capteur	LD2	Salon	250	250	694
Capteur	LD2	Salon	250	250	694
RFID	Livre1	Chambre	580	300	695
Capteur	CD2	Chambre	450	450	695
Capteur	CD1	Chambre	450	450	695
Capteur	CD2	Chambre	450	450	695
Capteur	CD1	Chambre	450	450	695

Tableau 9 : Évènements du scénario Lire un livre.

3.3.1.2 SCENARIO ALLER AUX TOILETTES

Type	*Nom*	*Zone*	*X*	*Y*	*Temps*
Capteur	DB6	Salle	50	50	475
Capteur	DB6	Salle	50	50	475
Capteur	LD5	Salle	50	50	475
Capteur	DB7	Salle	50	50	475
Capteur	LD5	Salle	50	50	475
Capteur	TP2	Salle	50	50	475
Capteur	DB6	Salle	50	50	475
Capteur	DB6	Salle	50	50	475

Capteur	DB5	Salle	50	50	475
Capteur	DB5	Salle	50	50	476
Capteur	TP2	Salle	50	50	476
Capteur	LD5	Salle	50	50	476
Capteur	DB7	Salle	50	50	476
Capteur	TC12	Salle	50	50	476

Tableau 10 : Évènements du scénario Aller aux toilettes.

3.3.1.3 Scenario Dormir dans la chambre

Type	*Nom*	*Zone*	*X*	*Y*	*Temps*
Capteur	TP1	Chambre	450	450	1390
Capteur	LD6	Chambre	450	450	1390
Capteur	TP1	Chambre	450	450	1390
Capteur	LD6	Chambre	450	450	1390
Capteur	LD6	Chambre	450	450	1390
Capteur	LD6	Chambre	450	450	1390
Capteur	LD6	Chambre	450	450	1391
Capteur	CD2	Chambre	450	450	1391
Capteur	LD6	Chambre	450	450	1391
Capteur	TP1	Chambre	450	450	1391
Capteur	LD6	Chambre	450	450	1391
Capteur	LD6	Chambre	450	450	1391
Capteur	CD2	Chambre	450	450	1391
Capteur	TP1	Chambre	450	450	1391
Capteur	LD4	Chambre	450	450	1391
Capteur	LD4	Chambre	450	450	1391

Tableau 11 : Évènements du scénario Dormir dans la chambre.

3.3.1.4 Scenario Cuisiner pour le souper

Type	*Nom*	*Zone*	*X*	*Y*	*Temps*
Capteur	CB3	Cuisine	390	280	1009
Capteur	CB4	Cuisine	390	260	1009
Capteur	CB3	Cuisine	390	280	1009
Capteur	CA1	Cuisine	400	285	1009
Capteur	CA2	Cuisine	400	265	1009

Capteur	CA3	Cuisine	400	190	1009
Capteur	CA4	Cuisine	400	170	1009
RFID	Assiette2	Cuisine	364	140	1009
RFID	Assiette2	Cuisine	398	180	1009
RFID	Assiette2	Cuisine	362	140	1009
RFID	Assiette2	Cuisine	399	180	1009
Capteur	LD3	Cuisine	250	250	1009
RFID	Casserole	Cuisine	395	180	1009
RFID	Casserole	Cuisine	396	180	1009
Capteur	LD3	Cuisine	250	250	1009
Capteur	MT2	Cuisine	365	130	1009
Capteur	MT1	Cuisine	365	130	1009
Capteur	LTC	Cuisine	365	130	1009
Capteur	MT3	Cuisine	365	130	1009
Capteur	CB4	Cuisine	390	260	1009
RFID	Casserole	Cuisine	330	98	1009
RFID	Casserole	Cuisine	330	100	1009
RFID	Casserole	Cuisine	362	139	1009
RFID	Casserole	Cuisine	290	95	1009
Capteur	CA4	Cuisine	400	170	1009
Capteur	CA3	Cuisine	400	190	1009
Capteur	CA2	Cuisine	400	265	1009
Capteur	CA1	Cuisine	400	285	1009
Capteur	MT2	Cuisine	365	130	1009
Capteur	MT1	Cuisine	365	130	1009
Capteur	MT2	Cuisine	365	130	1009
Capteur	MT1	Cuisine	365	130	1009
RFID	Casserole	Cuisine	330	95	1009
RFID	Assiette2	Cuisine	330	100	1009
Capteur	TC2	Cuisine	250	250	1010
Capteur	TC2	Cuisine	250	250	1010
Capteur	TC2	Cuisine	250	250	1010
Capteur	TC2	Cuisine	250	250	1010
Capteur	TC1	Cuisine	250	250	1010
Capteur	TC2	Cuisine	250	250	1010
Capteur	TC1	Cuisine	250	250	1010
Capteur	TC1	Cuisine	250	250	1010
Capteur	TC2	Cuisine	250	250	1010
Capteur	TC1	Cuisine	250	250	1010
Capteur	TC1	Cuisine	250	250	1010

Capteur	TC2	Cuisine	250	250	1010
Capteur	CA5	Cuisine	300	100	1010
RFID	Casserole	Cuisine	362	139	1010
RFID	Pâtes	Cuisine	322	140	1010
Capteur	TC2	Cuisine	250	250	1010
Capteur	TC1	Cuisine	250	250	1010
RFID	Pâtes	Cuisine	395	180	1010
Capteur	MT2	Cuisine	365	130	1010
Capteur	TC1	Cuisine	250	250	1010
Capteur	TC1	Cuisine	250	250	1010
Capteur	TC1	Cuisine	250	250	1010
Capteur	CA5	Cuisine	300	100	1010
RFID	Pâtes	Cuisine	396	180	1010
RFID	Pâtes	Cuisine	330	96	1010
RFID	Pâtes	Cuisine	363	138	1010
RFID	Pâtes	Cuisine	330	100	1010
RFID	Pâtes	Cuisine	362	140	1010
Capteur	TC2	Cuisine	250	250	1010
Capteur	TC1	Cuisine	250	250	1010
Capteur	TC1	Cuisine	250	250	1010
Capteur	TC2	Cuisine	250	250	1010
Capteur	TC2	Cuisine	250	250	1010
Capteur	TC1	Cuisine	250	250	1010
Capteur	TC2	Cuisine	250	250	1010
Capteur	TC1	Cuisine	250	250	1010
Capteur	TC2	Cuisine	250	250	1010
Capteur	TC1	Cuisine	250	250	1010
Capteur	TC2	Cuisine	250	250	1010
Capteur	TC1	Cuisine	250	250	1010
RFID	Pâtes	Cuisine	290	96	1010
RFID	Pâtes	Cuisine	250	100	1010
Capteur	MT1	Cuisine	365	130	1010
Capteur	LTC	Cuisine	365	130	1010
Capteur	MT3	Cuisine	365	130	1010
Capteur	TC1	Cuisine	250	250	1010
Capteur	TC2	Cuisine	250	250	1010
Capteur	TC1	Cuisine	250	250	1010
Capteur	TC1	Cuisine	250	250	1011
Capteur	LD3	Cuisine	250	250	1011
Capteur	TC2	Cuisine	250	250	1011

Capteur	TC1	Cuisine	250	250	1011
Capteur	TC1	Cuisine	250	250	1011
Capteur	TC1	Cuisine	250	250	1011
Capteur	TC2	Cuisine	250	250	1011
Capteur	TC1	Cuisine	250	250	1011
Capteur	TC1	Cuisine	250	250	1011
Capteur	TC2	Cuisine	250	250	1011
RFID	Casserole	Cuisine	366	139	1010
RFID	Casserole	Cuisine	363	140	1010
Capteur	TC1	Cuisine	250	250	1011
Capteur	TC2	Cuisine	250	250	1011
Capteur	TC1	Cuisine	250	250	1011
Capteur	TC1	Cuisine	250	250	1011

Tableau 12 : Évènements du scénario Cuisiner pour le souper.

3.3.1.5 SCENARIO CUISINER POUR LE DEJEUNER

Type	*Nom*	*Zone*	*X*	*Y*	*Temps*
Capteur	LD3	Cuisine	250	250	709
Capteur	MT2	Cuisine	365	130	709
Capteur	MT1	Cuisine	365	130	709
Capteur	LTC	Cuisine	365	130	709
Capteur	MT3	Cuisine	365	130	709
Capteur	CB4	Cuisine	390	260	709
Capteur	CB3	Cuisine	390	280	709
Capteur	CB4	Cuisine	390	260	709
Capteur	CB3	Cuisine	390	280	709
Capteur	CA1	Cuisine	400	285	709
Capteur	CA2	Cuisine	400	265	709
Capteur	CA3	Cuisine	400	190	709
Capteur	CA4	Cuisine	400	170	709
RFID	Assiette2	Cuisine	364	140	709
RFID	Assiette2	Cuisine	398	180	709
RFID	Assiette2	Cuisine	362	140	709
RFID	Assiette2	Cuisine	399	180	709
Capteur	LD3	Cuisine	250	250	709
RFID	Casserole	Cuisine	395	180	709
RFID	Casserole	Cuisine	396	180	709

RFID	Casserole	Cuisine	330	98	709
RFID	Casserole	Cuisine	330	100	709
RFID	Casserole	Cuisine	362	139	709
RFID	Casserole	Cuisine	290	95	709
RFID	Casserole	Cuisine	330	95	709
Capteur	CA4	Cuisine	400	170	709
Capteur	CA3	Cuisine	400	190	709
Capteur	CA2	Cuisine	400	265	709
Capteur	CA1	Cuisine	400	285	709
Capteur	MT2	Cuisine	365	130	709
Capteur	MT1	Cuisine	365	130	709
Capteur	MT2	Cuisine	365	130	709
Capteur	MT1	Cuisine	365	130	709
RFID	Assiette2	Cuisine	330	100	709
Capteur	TC2	Cuisine	250	250	710
Capteur	TC2	Cuisine	250	250	710
Capteur	TC2	Cuisine	250	250	710
Capteur	TC2	Cuisine	250	250	710
Capteur	TC1	Cuisine	250	250	710
Capteur	TC2	Cuisine	250	250	710
Capteur	TC1	Cuisine	250	250	710
Capteur	TC1	Cuisine	250	250	710
Capteur	TC2	Cuisine	250	250	710
Capteur	TC2	Cuisine	250	250	710
Capteur	TC1	Cuisine	250	250	710
Capteur	TC2	Cuisine	250	250	710
Capteur	TC1	Cuisine	250	250	710
Capteur	TC2	Cuisine	250	250	710
Capteur	TC1	Cuisine	250	250	710
Capteur	TC2	Cuisine	250	250	710
Capteur	TC1	Cuisine	250	250	710
Capteur	TC2	Cuisine	250	250	710
Capteur	TC1	Cuisine	250	250	710
Capteur	TC1	Cuisine	250	250	710
Capteur	TC2	Cuisine	250	250	710
Capteur	TC1	Cuisine	250	250	710
Capteur	TC1	Cuisine	250	250	710
Capteur	TC1	Cuisine	250	250	710
Capteur	TC1	Cuisine	250	250	710
Capteur	TC1	Cuisine	250	250	710

Capteur	TC1	Cuisine	250	250	710
Capteur	TC1	Cuisine	250	250	710
Capteur	CA5	Cuisine	300	100	710
RFID	Pâtes	Cuisine	396	180	710
RFID	Pâtes	Cuisine	330	96	710
RFID	Pâtes	Cuisine	363	138	710
RFID	Pâtes	Cuisine	330	100	710
RFID	Pâtes	Cuisine	362	140	710
RFID	Pâtes	Cuisine	290	96	710
RFID	Pâtes	Cuisine	250	100	710
Capteur	TC2	Cuisine	250	250	710
Capteur	CA5	Cuisine	300	100	710
RFID	Casserole	Cuisine	362	139	710
RFID	Pâtes	Cuisine	322	140	710
Capteur	TC2	Cuisine	250	250	710
Capteur	TC1	Cuisine	250	250	710
RFID	Pâtes	Cuisine	395	180	710
Capteur	MT2	Cuisine	365	130	710
Capteur	MT1	Cuisine	365	130	710
Capteur	LTC	Cuisine	365	130	710
Capteur	MT3	Cuisine	365	130	710
Capteur	TC1	Cuisine	250	250	710
Capteur	TC2	Cuisine	250	250	710
Capteur	TC1	Cuisine	250	250	710
RFID	Casserole	Cuisine	366	139	710
RFID	Casserole	Cuisine	363	140	710
Capteur	TC1	Cuisine	250	250	711
Capteur	TC2	Cuisine	250	250	711
Capteur	TC1	Cuisine	250	250	711
Capteur	TC1	Cuisine	250	250	711
Capteur	LD3	Cuisine	250	250	711
Capteur	TC2	Cuisine	250	250	711
Capteur	TC1	Cuisine	250	250	711
Capteur	TC1	Cuisine	250	250	711
Capteur	TC1	Cuisine	250	250	711
RFID	Casserole	Cuisine	366	139	711
RFID	Casserole	Cuisine	363	140	711
Capteur	TC2	Cuisine	250	250	711
Capteur	TC1	Cuisine	250	250	711
Capteur	TC2	Cuisine	250	250	711

Capteur	TC1	Cuisine	250	250	711

Tableau 13 : Évènements du scénario Cuisiner pour le déjeuner.

Dans le but de tester ces scénarios rigoureusement, nous avons établi une interface graphique et un protocole expérimental pour toujours appliquer notre algorithme avec la même procédure et les mêmes paramètres entre chaque test.

3.3.2 INTERFACE GRAPHIQUE ET PROTOCOLE EXPERIMENTAL

L'interface graphique de notre application Java mesure 500x500 pixels et est présentée dans la Figure 10 avec le mode de débogage activé. On retrouve en haut à gauche les poids associés à chacune des forces, respectivement : similarité, dissimilarité, cohésion, séparation, alignement. Le dernier terme de la liste « Vis » correspond au rayon de la distance de vision des agents en pixels, on peut voir le nombre 140 sur l'exemple, ce qui signifie que le cercle représentant le champ de vision des agents aura un rayon de 140 pixels. Pour un souci de visibilité, ce champ de vision est représenté dans l'interface par un carré au lieu d'un cercle. On peut voir ce carré pour l'agent en cours de sélection sur l'image, cet agent se situe au centre du carré et est entouré par un cercle rouge. Les autres agents entourés par un cercle jaune correspondent aux voisins de l'agent au centre du carré. On peut distinguer que les agents placés dans les coins du carré ne sont pas considérés comme des voisins, car ils sont plus loin que 280 pixels par rapport à l'agent sélectionné. De même, les attributs de cet agent sont affichés entre parenthèses en bas à gauche, ils correspondent respectivement au : type de l'agent, nom, zone spatiale, position en X, position en Y, heure de création de l'agent (exprimé sous la forme « heure : minute »),

classe. Dans cet exemple la classe de l'agent est « Dejeuner » ce qui signifie que l'agent sélectionné appartient au scénario « Cuisiner pour le Dejeuner » vu précédemment. Si la classe de l'agent était inconnue, il y aurait indiqué le mot clé « Unknow ».

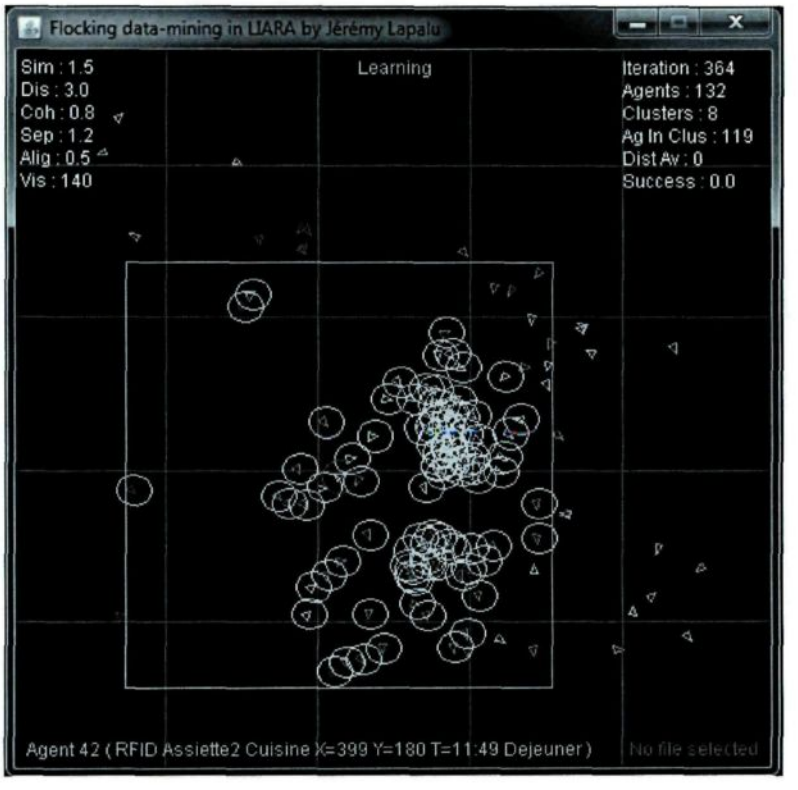


Figure 9 : Interface graphique avec débogage 1

Les classes des agents sont connues seulement lorsqu'ils sont créés à partir des fichiers de test comme les scénarios de cas réels présentés plus tôt. Si un fichier de ce type est sélectionné son nom apparait en bas à droite de l'interface, on peut ainsi voir quel scénario il contient et sa date d'enregistrement, comme le montre la Figure 10. De plus, d'autres informations sont visibles en haut à droite de l'interface de débogage : nombre d'itérations, d'agents, de clusters et le nombre d'agents dans les clusters. Les deux

dernières données sont utilisées seulement durant la phase de test de l'algorithme, ils correspondent à la distance moyenne des agents intra-clusters et au taux de succès moyens des clusters. On peut également distinguer sur l'image les huit clusters qui sont affichés par des cercles blancs. À noter que les agents appartiennent à un seul cluster à la fois, même si on pourrait croire le contraire dans l'exemple. Les clusters s'entrecroisent sur l'interface seulement, car on les représente par des cercles.

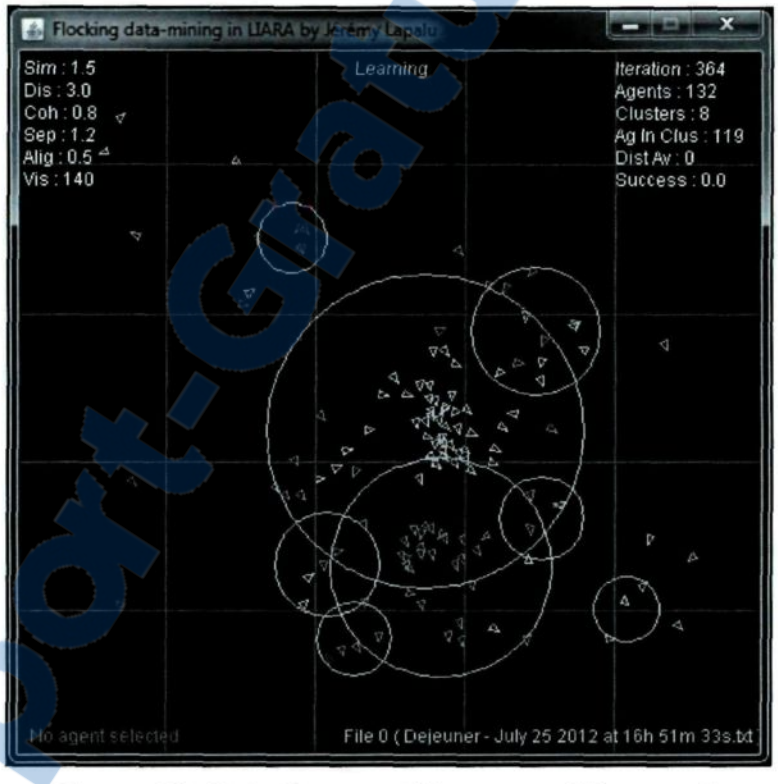


Figure 10 : Interface graphique avec débogage 2

Enfin, la phase en cours est affichée en haut au centre de l'interface avec « Learning » pour la phase d'apprentissage, et « Testing » pour la phase de test. Nous

pouvons maintenant présenter le protocole expérimental. Celui-ci se distingue par les deux phases mentionnées.

La première, la phase d'apprentissage ont pour but de former les clusters à partir des agents présents. Pour cela, les scénarios de test sont chargés manuellement un par un au lancement du programme, trois quart des évènements contenus dans les fichiers sont créés sous forme d'agents, et ils sont placés semi aléatoirement vers le centre du monde virtuel. Une fois tous les scénarios chargés l'algorithme est lancé. Après un certain nombre d'itérations et lorsque tous les agents font parties d'un cluster, la phase de test commence.

La seconde phase a pour objectif de vérifier la qualité des clusters formés précédemment. Au début de la phase de test, un agent représentatif de chaque cluster est créé, ces agents représentent une moyenne des valeurs des attributs de tous les autres agents contenus dans le cluster. Puis, tous les autres agents sont détruits et le tiers non utilisé des données remplacent ces agents en apparaissant tous semi aléatoirement vers le centre du monde virtuel comme pour la phase d'apprentissage. Dans cette nouvelle phase, les clusters sont formés différemment. En effet, ils sont créés à partir des agents représentatifs établis au départ, s'ils sont au nombre de cinq agents cela signifie que le nombre de clusters sera cinq et ce nombre ne variera pas contrairement à l'apprentissage. Ceci nous permet de confirmer si les clusters créés précédemment sont de bonne qualité ou non. Pour vérifier cela, nous calculons à chaque itération le taux de succès de tous les clusters, et la moyenne est inférée. À noter que le taux de succès d'un cluster correspond au nombre d'agents de la même classe divisé par le nombre total d'agents dans le cluster. Après vingt-cinq mesures, une moyenne de ces taux de succès et le nombre d'itérations en cours sont sauvegardés

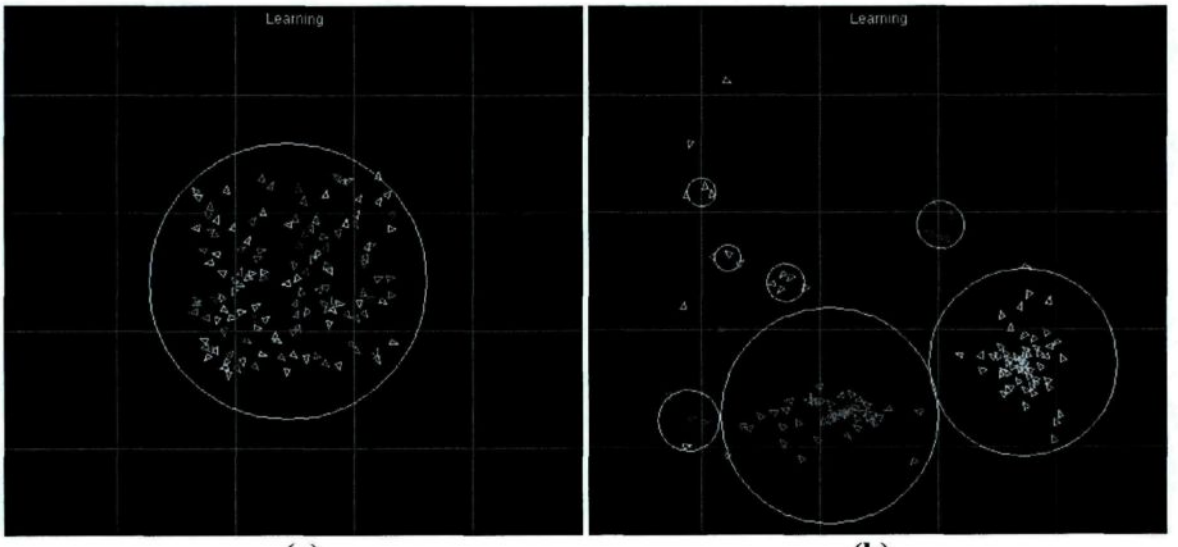
dans un fichier, ainsi nous gardons facilement les résultats obtenus, et lorsque le taux de succès ne varie plus de manière significative l'algorithme s'arrête. Ceci termine le protocole concernant le Flocking.

Ensuite, pour pouvoir comparer les résultats obtenus nous avons choisi deux autres algorithmes de segmentations. Nos choix se sont portés en premier sur le K-means pour sa popularité, sa réputation, et aussi qu'il a déjà été comparé au Flocking dans les travaux de Cui et al. [44]. En second nous avons choisi EM pour ses options concernant le nombre de clusters : soit l'algorithme se charge d'estimer par lui-même ce nombre, soit nous pouvons le fixer au départ. Les versions de EM et K-means utilisé sont celles contenues dans l'outil Weka 3.6. De plus, pour chacun des trois algorithmes nous avons utilisé la distance Euclidienne comme mesure de similarité, et le même ensemble de données : les cinq scénarios présentés précédemment.

Ainsi, nous allons présenter maintenant un exemple d'exécution du Flocking du début jusqu'à la fin avec les scénarios de test, puis les résultats obtenus.

3.3.3 EXEMPLE D'EXECUTION ET RESULTATS

Commençons tout d'abord par l'exemple d'exécution, la distribution initiale des données au début de la phase d'apprentissage est présentée dans la Figure 11 (a). Comme vous pouvez le constater, un seul cluster contient tous les agents au commencement de la phase puisque les agents ont été placés de façon semi-aléatoire vers le centre du monde virtuel. La Figure 11 (b) montre que plusieurs clusters se sont formés après cinq cents itérations, mais certains agents n'ont pas encore trouvé leurs agents similaires.



(a) (b)

Figure 11 : Exemple d'une phase d'apprentissage avec le Flocking
(a) Distribution des données au début de la phase (b) Après 500 itérations

Après mille itérations par contre, tous les agents font parties d'un cluster, comme le montre la Figure 12. Chacun des clusters correspond en réalité à un scénario précis dont nous avons parlé plus tôt. Notre algorithme arrive donc à trouver le nombre exact d'activités durant la phase d'apprentissage, et cela, sans supervision.

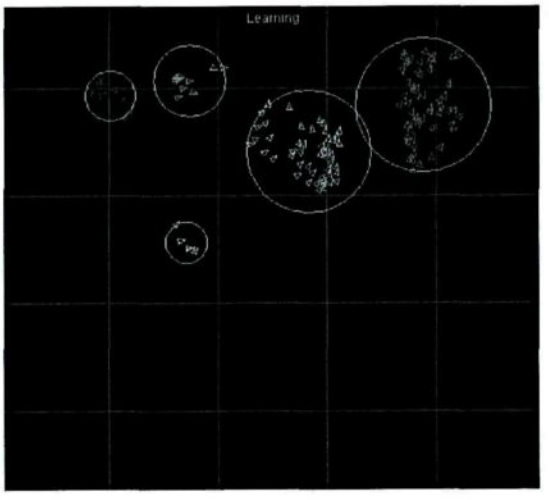
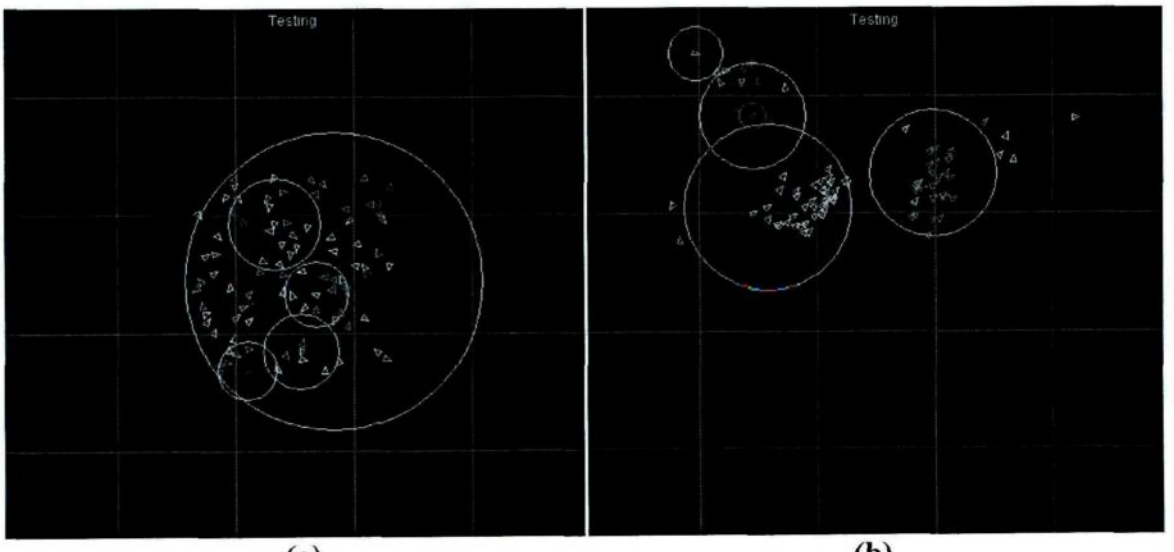


Figure 12 : Exemple de la fin d'une phase d'apprentissage (1000 itérations)

Ensuite, si le nombre de clusters ne varie plus après quelques itérations cette phase se termine et la phase de test commence. Comme nous l'avons déjà expliqué dans la section précédente, les agents sont remplacés par les données restantes des scénarios, dont on peut voir le placement dans la Figure 13 (a) et le résultat après trois mille itérations dans la partie (b).



(a) **(b)**

Figure 13 : Exemple d'une phase de test avec le Flocking
(a) Début de la phase (b) Après 3000 itérations

Dans cet exemple le taux de succès atteint le seuil de quatre-vingt-douze pour cent après seulement huit mille itérations. On peut voir la distribution des agents dans la Figure 14 suivante.

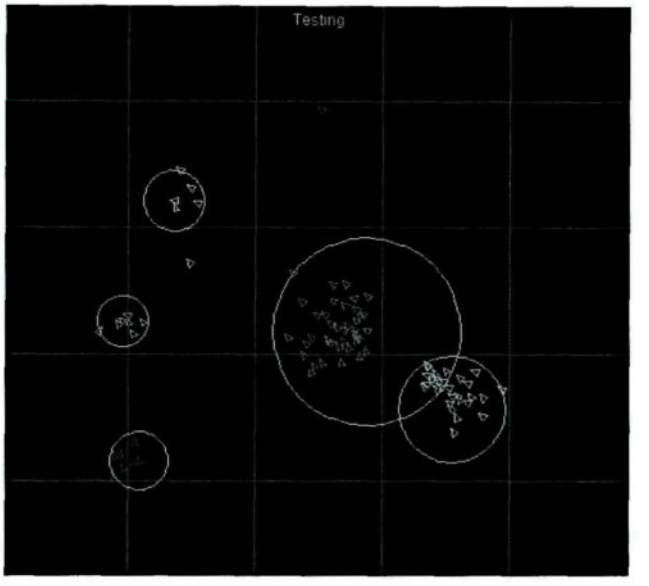


Figure 14 : Exemple de la fin d'une phase de test (8000 itérations)

Puis, la phase de test se termine avec quelques itérations supplémentaires et conclues ainsi notre exemple. Afin d'obtenir des résultats concrets, nous avons exécuté notre algorithme dix fois avec les cinq scénarios, et nous avons calculé la moyenne des résultats obtenus qui sont présentés dans le Tableau 14. Le taux de succès est compris entre 0 et 1, plus la valeur est proche de 1 et plus le cluster est pur.

Itérations	3371	4591	5753	6897	8064	8882	10227	13404
Succès	0.6359	0.7184	0.8038	0.8716	0.8863	0.8902	0.9000	0.9249

Tableau 14 : Moyenne des résultats du Flocking sur des données réelles.

Comme nous pouvons le constater, au départ de la phase de test le taux de succès augmente rapidement au fil des itérations vers environ quatre-vingt-cinq pour cent, puis il va progressivement atteindre le seuil de quatre-vingt-douze pour cent autour des treize mille itérations. Ceci signifie que les clusters des cinq activités qui ont été détectées par

l'algorithme durant la phase d'apprentissage ont une pureté moyenne d'environ quatre-vingt-douze pour cent. Le taux de succès n'est pas égal à cent pour cent à cause du bruit de l'habitat intelligent (interférence des fréquences radio pour les tags RFID, échec de détection d'un capteur, etc.), mais aussi parfois à cause d'évènements aléatoires générés par l'imprécision de certains capteurs. Pour comparaison, le Tableau 15 présente les résultats obtenus avec les algorithmes K-means et EM.

Algorithme	*Itérations*	*Succès*	*Nombre de clusters*
K-Means	5	0.6033	5 (fixé au départ)
EM (k fixé)	5	0.7603	5 (fixé au départ)
EM (k non fixé)	5	0.6240	7

Tableau 15 : Résultats avec K-means et EM sur des données réelles.

Ces deux algorithmes sont très efficaces en terme de nombre d'itérations. Cependant, ils ne peuvent atteindre un haut taux de succès en partie à cause du petit nombre de données contenu dans notre ensemble de test, et K-means requiert obligatoirement de fixé le nombre de clusters au départ. Quant à EM, il n'obtient pas le nombre exact d'activités si on ne lui indique pas le nombre de clusters, et dans ce cas son taux de succès est bas. Il est également important de noter la différence entre les façons de compter le nombre d'itérations des algorithmes. Dans les deux méthodes de segmentation classique, une itération correspond à la réattribution complète de tous les éléments dans les clusters, alors que dans notre cas c'est seulement le déplacement des agents avec le Flocking et la mise à jour c'est pourquoi, la différence de performance entre notre algorithme et les méthodes classiques n'est pas aussi importante que cela en a l'air.

3.4 CONCLUSION DU CHAPITRE

Ce chapitre avait pour but de présenter une nouvelle approche pour la segmentation de données d'un habitat intelligent. Dans la première partie, nous avons décrit le modèle du Flocking avec l'addition de deux nouvelles règles, similarité et dissimilarité, qui nous permettent d'utiliser l'émergence du Flocking comme méthode de forage de données.

La deuxième partie de ce chapitre a permis de présenter l'implémentation du modèle dans l'habitat intelligent du LIARA. Nous avons détaillé l'infrastructure du laboratoire avec ses différents capteurs, sa base de données, et nos choix pour la mesure de similarité des nombreux paramètres possibles.

Dans la troisième section, nous avons détaillé le protocole expérimental que nous avons utilisé pour évaluer et valider notre approche. Puis, nous avons partagé les résultats obtenus sur des scénarios de cas réels avec notre algorithme du Flocking, ainsi que K-means et EM.

Le chapitre suivant vient conclure le projet en décrivant les objectifs réalisés, la contribution du mémoire, les limitations de cette méthode et les futurs travaux en liens pour l'améliorer, et enfin, un bilan personnel sur ce travail de recherche.

CHAPITRE 4

CONCLUSION GENERALE

L'assistance technologique des personnes est un domaine encore jeune, mais très prometteur pour l'avenir. Les nouvelles technologies permettant d'obtenir plus de données dans les habitats intelligents de nombreuses recherches sont réalisées pour améliorer les algorithmes de reconnaissance d'activités afin de donner une meilleure assistance aux utilisateurs de l'habitat d'une manière non intrusive. Parmi toutes les techniques de reconnaissance d'activités et de forage de données, le Flocking est une méthode pertinente pour la segmentation des données grâce à son comportement émergent. Il permet aussi bien de traiter de petit ou de grand ensemble de données, sans baisser la qualité des clusters formés, car les agents peuvent changer facilement et rapidement de cluster. En effet, les agents suivent seulement quelques règles simples par rapport à leurs voisins et ont ainsi peu de probabilité de se retrouver dans le mauvais cluster, contrairement à beaucoup d'algorithmes comme le K-means où les données sont statiques et peuvent donc engendrer plus d'erreurs. En terme de temps de calcul, le Flocking nécessite effectivement plus d'itérations que les algorithmes classiques, mais au prix d'une meilleure segmentation. De plus, la durée de traitement reste raisonnable en temps humain, nous n'avons pas besoin que la segmentation se fasse en moins de quelques millisecondes, car la personne assistée n'effectuera pas plus d'une action par seconde de toute façon. Enfin, le Flocking a été mis en lumière depuis 1987 [42] mais il est utilisé comme méthode de segmentation seulement

depuis quelques années, il y a donc de nombreuses possibilités d'améliorations qui pourront rendre la méthode encore plus performante.

4.1 OBJECTIFS REALISES

Le premier objectif du projet de recherche qui consistait à établir une revue de littérature sur les différentes techniques de reconnaissance d'activités intégrant des méthodes de forages de données, afin de dégager des pistes de solution pour l'exploitation du Flocking dans le cadre des habitats intelligents, a été plus difficile à ce qu'on pourrait croire dans un premier temps. En effet, de nombreuses recherches ont été effectuées dans le cadre de la reconnaissance d'activités [6, 7, 50], ainsi que pour les méthodes de forages de données [16, 19, 20]. Toutefois, l'utilisation du Flocking dans le cadre des habitats intelligents a été une première. Cette piste de solution a émergé grâce aux utilisations du Flocking dans d'autres domaines que l'intelligence ambiante avec entre autres les travaux de Cui et al. [44] et ceux de Bellachi et al. [45].

Le second objectif qui consistait à formaliser un nouveau modèle de reconnaissance d'activités intégrant les éléments tirés du Flocking a été plus simple à accomplir que le premier objectif, grâce aux recherches de Cui et al. cité précédemment. Nous nous sommes fortement basés sur leurs nouvelles règles afin d'apporter notre propre extension au Flocking tout en modifiant quelques éléments afin de rendre la segmentation plus efficace avec nos données, mais en obtenant un modèle théorique rigoureux et longuement détaillé dans le chapitre trois de ce mémoire.

Le troisième objectif du projet qui consistait à implémenter ce nouveau modèle a été réalisé assez rapidement grâce au langage Java. Une interface complète de test a pu être développée dans la même lignée afin de pouvoir réaliser une série de tests basés sur des scénarios de cas réels qui constituait l'objectif final du projet.

Finalement, déterminer la performance du nouvel algorithme a été effectuée avec notre interface. Nous avons obtenu dans un temps raisonnable des clusters de très bonnes qualités avec le Flocking à chacun des tests, alors que les autres méthodes de forage de données comme K-means et EM même si elles sont extrêmement rapides n'ont pas pu obtenir de bon pourcentage de réussite.

4.2 REVUE DU MODELE DEVELOPPE

Notre extension du Flocking repose sur des bases solides, car nous avons suivi les mêmes méthodes d'implémentation conseillées par ceux ayant déjà utilisé l'algorithme [43, 44]. Notre version reprend les règles essentielles qui font l'essence même du Flocking, c'est-à-dire la séparation, la cohésion, et l'alignement, pour ainsi former un seul et unique groupe d'agents. Puis, nos deux nouvelles règles de similarité et dissimilarité permettent la formation de plusieurs clusters grâce à la comparaison des attributs des agents, qui sont eux-mêmes créés par les évènements de l'habitat intelligent, ou autrement dit de la modification des valeurs de la base de données.

De plus, nous avons découpé le monde virtuel en zone afin de réduire la complexité de calcul de l'algorithme qui est normalement quadratique $O(n^2)$ en une complexité linéaire $O(n)$, et ainsi réduire le temps de calcul de chaque itération. Le modèle suit

également les principes des méthodes de forages de données qui divisent les algorithmes en deux phases distinctes : phase d'apprentissage et phase de test. Avec tous ses éléments, le Flocking peut ainsi créer des clusters de manière efficace et obtenir exactement le bon nombre d'activités qui se déroulent dans l'habitat intelligent. Cependant, de nombreuses améliorations sont possibles, nous allons justement voir quelques limitations du modèle qui pourraient être résolues dans de futurs travaux.

4.3 LIMITATIONS ET TRAVAUX FUTURS

À l'heure actuelle, le plus grand défaut de notre algorithme est le nombre d'itérations nécessaire pour aboutir à une bonne segmentation. Comme nous l'avons constaté dans le chapitre trois notre extension du Flocking à besoin de milliers d'itérations pour créés les clusters, même si le temps de calcul reste raisonnable en temps humain il est beaucoup trop lent comparé aux méthodes classiques. Nous pensons qu'il est possible de diviser ce nombre d'itérations au moins par deux. Pour cela il faudra faire beaucoup plus de tests pour trouver les valeurs des poids de chaque force qui accélère à coup sûr le traitement, mais aussi ajuster dynamiquement durant l'exécution la vitesse maximale des agents et leur distance de vue en fonction du nombre d'agents présent dans le monde virtuel. S'il y a peu d'agents, on peut augmenter ces deux paramètres afin d'accélérer la formation des clusters, et si le nombre d'agents est important il vaut mieux réduire les paramètres afin de diminuer le temps de calcul de chaque itération.

Le second problème de notre modèle est que l'on ignore quelles sont exactement les activités en train d'être effectués. On peut déterminer le nombre d'activités dont une

personne a effectué durant une journée avec leurs emplacements, la tranche d'heures, les objets utilisés, les capteurs déclenchés, mais on ignore le type exact de l'activité. Notre modèle est un pré-traitement des actions de l'utilisateur de l'habitat intelligent. Ainsi, pour pouvoir prédire les activités et assister la personne au moment opportun, il faudrait combiner le Flocking avec un second algorithme de reconnaissance d'activité comme celui de Bouchard [50] par exemple.

4.4 BILAN PERSONNEL DU TRAVAIL DE RECHERCHE

En conclusion, j'ai énormément appris durant ce projet de recherche. J'ai pu découvrir de nouveaux domaines comme la reconnaissance d'activités et le forage de données dont j'ignorais complètement l'existence avant ma maîtrise. L'équipe du LIARA m'a permis d'obtenir une excellente expérience de travail en équipe, avec en plus une très bonne ambiance générale. J'ai pu également écrire mon premier article scientifique et comprendre les enjeux et les difficultés de rédiger ce genre d'article. Ce projet m'a ouvert un nouvel horizon qu'est le monde de la recherche scientifique.

BIBLIOGRAPHIE

[1] U. Nations, "World population ageing 2009: United Nations," *Dept. of Economic and Social Affairs, Population Division,* 2010.

[2] Russell S. and Norvig P., *Artificial Intelligence: A Modern Approach. Prentice-Hall*: Englewood Cliffs, NJ, 3nd edition, 2009.

[3] Capezio F., Giuni A., Mastrogiovanni F., Sgorbissa A., Vernazza P., Vernazza T., and Z. R., "Sweet Home! Perspectives of Ambient Intelligence. ," *In Journal of the Italian AEIT Association,* pp. 42 - 49, 2007.

[4] Ramos C., Augusto J.C., and S. D., "Ambient Intelligence: the Next Step for Artificial Intelligence," *IEEE Intelligent Systems,* vol. 23, pp. 5-18, 2008.

[5] Augusto J. C. and Nugent C. D., "Designing Smart Homes: The Role of Artificial Intelligence," *Lecture Notes in Artificial Intelligence (LNAI), Springer,* pp. 1-183, 2006.

[6] Roy P.C., Bouchard B., Bouzouane A., and G. S., "Challenging issues of ambient activity recognition for cognitive assistance," in *Handbook of research on Ambient Intelligence and Smart Environments: Trends and Perspectives*, IGI global ed: IGI global, F. Mastrogiovanni and N. Chong Editors, Information Science Publishing, 2010.

[7] Bouchard B., Bouzouane A., and G. S., "A Keyhole Plan Recognition Model for Alzheimer's Patients: First Results," *Journal of Applied Artificial Intelligence (AAI),* vol. Vol. 22, pp. 623-658, July 2007.

[8] Singla G., Cook D.J., and S.-E. M., "Incorporating Temporal Reasoning into Activity recognition for Smart Home Residents," *AAAI Workshop on Spatial and Temporal Reasoning,* pp. 53-61, 2008.

[9] Patterson D.J., Fox D., Kautz H., and P. M., "Fine–grained activity recognition by aggregating abstract object usage," *In Proceedings of IEEE 9th International Symposium on Wearable Computers (ISWC),* 2005.

[10] Schmidt C., Sridharan N., and G. J., "The plan recognition problem: an intersection of psychology and artificial intelligence," *Artificial Intelligence,* pp. 45-83, 1978.

[11] T. Asfour, Geib C., Goldman R., Kautz H., "Plan recognition," *Report from Dagstuhl Seminar 11141, Germany,* 2011.

[12] Kautz H. and Allen J., "Generalized plan recognition," presented at the National Conference on Artificial Intelligence (AAAI), 1986.

[13] Rabiner L. R., "A tutorial on hidden Markov models and selected applications in speech recognition," in *Readings in speech recognition*, ed: Morgan Kaufmann Publishers Inc., 1990, pp. 267-296.

[14] Jakkula V.R. and Cook D.J., "Enhancing Smart Home Algorithms Using Temporal Relations," *Technology and Aging,* vol. Vol. 21, pp. 3-10, 2008.

[15] Augusto J. C., "Management of uncertainty and spatio-temporal aspects for monitoring and diagnosis in a Smart Home," *International Journal of Computational Intelligence Systems,* vol. Vol. 1, pp. 361-378, 2008.
[16] Witten I. H. and al., "Data Mining: Practical Machine Learning Tools and Techniques," *Elsevier Science & Technology,* 2011.
[17] Reynolds C., "Steering behaviors for autonomous characters," presented at the Proceedings of Game Developers Conference, 1999.
[18] J. Lapalu, K. Bouchard, A. Bouzouane, B. Bouchard, and S. Giroux, "Unsupervised Mining of Activities for Smart Home Prediction," *The 4th International Conference on Ambient Systems, Networks and Technologies (ANT 2013),* 2013.
[19] Ian H. Witten and E. Franck, "Data mining: practical machine learning tools and techniques," *Elsevier editor,* 2010.
[20] Daniel T. Larose, "Data mining Methods and Models," *Wiley pub.,* 2009.
[21] Han J. and Kamber M., "Data Mining: Concepts and Techniques," *2nd ed. Academic Press,* 2006.
[22] Nerzic P., "Two Methods for Recognizing Erroneous Plans in Human-Machine Dialogue," *In AAAI'96 Workshop : Detecting, Repairing and Preventing Human-Machine Miscommunication,* 1996.
[23] Py D., "Reconnaissance de plan pour l'aide à la démonstration dans un tuteur intelligent de la géométrie," Thèse de Doctorat, Université de Rennes 1, 1992.
[24] Geib C. and Goldman R., "Partial Observability and Probabilistic Plan/Goal Recognition," presented at the Int. Joint Conference on Artificial Intelligence, 2005.
[25] Allen J. F., "Maintaining knowledge about temporal intervals," *Artificial Intelligence and Language Processing,* vol. Vol. 26, pp. 832-843, 1983.
[26] Song F. and Cohen R., "Temporal reasoning during plan recognition," in *Ninth National conference on Artificial intelligence*, Anaheim, California, 1991.
[27] Weida R., "Terminological constraint network reasoning and its application to plan recognition: Dept. of Computer Science," *Columbia University,* 1993.
[28] Riedel D. E. and al., "Spatial Activity Recognition in a Smart Home Environment using a Chemotactic Model," in *International Conference on Intelligent Sensors Networks and Information Processing*, 2005.
[29] Adler J., "Chemotaxis in Bacteria," *Annual Review of Biochemistry,* vol. Vol. 44, pp. pp. 341-356, 1975.
[30] P. Berkhin, "Survey of clustering data mining techniques," *Accrue Software Research Paper,* 2002.
[31] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM Computing Survey 31 (3),* pp. 264–323, 1999.
[32] M. R. Anderberg, "Cluster Analysis for Applications," *Academic Press Inc. New York,* 1973.
[33] J. A. Hartigan, "Clustering Algorithms," *John Wiley and Sons Inc. New York,* 1975.
[34] S. Z. Selim and M. A. Ismail, "K-means type algorithms: a generalized convergence theorem and characterization of local optimality," *IEEE Transactions on Pattern Analysis and Machine Intelligence 6,* pp. 81–87, 1984.

[35] A. P. Dempster, N. M. Laird, and D. Rubin, "Maximum Likelihood from Incomplete Data via the EM Algorithm," *Journal of the Royal Statistical Society. Series B (Methodological),* vol. vol. 39, pp. 1–38, 1977.

[36] A. Colorni, M. Dorigo, and V. Maniezzo, "Distributed Optimization by Ant Colonies, actes de la première conférence européenne sur la vie artificielle," *Paris, France, Elsevier Publishing,* pp. 134-142, 1991.

[37] S. Goss, S. Aron, J.-L. Deneubourg, and J.-M. Pasteels, "The self-organized exploratory pattern of the Argentine ant," *Naturwissenschaften,* vol. volume 76, pp. pages 579-581, 1989.

[38] C. Solnon, *Optimisation par colonies de fourmis*, 2008.

[39] M. Wooldridge, *An Introduction to MultiAgent Systems - Second Edition*: John Wiley & Sons, 2009.

[40] M. Dorigo, E. Bonabeau, and G. Theraulaz, "Ant algorithms and stigmery," *Future Generation Computer Systems 16 (8),* pp. p. 851–871, 2000.

[41] G. Folino and G. Spezzano, "SPARROW: A Spatial Clustering Algorithm using Swarm Intelligence," *Applied Informatics,* vol. Innsbruck, Austria, pp. p. 50–5, 2003

[42] Reynolds C., "Flocks, herds, and schools: a distributed behavioral model," *Computer Graphics,* vol. Vol. 21, pp. 25-34, 1987.

[43] Buckland M., "How to Create Autonomously Moving Game Agents," in *Programming Game AI by Example*, ed: Wordware Publishing, 2005, pp. 113-119.

[44] Xiaohui Cui, Jinzhu Gao, and Thomas E. Potok, "A flocking based algorithm for document clustering analysis," *Journal of Systems Architecture,* vol. Vol. 52, pp. 505-515, 2006.

[45] Bellaachi A. and Bari A., "OFLOSCAN : A Flocking-Based Data Mining Algorithm for Detecting Outliers in Cancer Microarrays," 2010.

[46] B. Everitt, "Cluster Analysis second ed.," *Halsted Press New York,* 1980.

[47] Shieh Albert D. and Hung Yeung Sam, "Detecting Outlier Samples in Microarray Data," *Statistical Applications in Genetics and Molecular Biology,* vol. 8, issue 1, number 13, 2009.

[48] Campbell Colin and Ying Yiming, "Learning with Support Vector Machines," *Morgan and Claypool,* 2011.

[49] Cristianini Nello and Shawe-Taylor John, "An Introduction to Support Vector Machines and other kernel-based learning methods," *Cambridge University Press,* 2000.

[50] Bouchard K., "Qualitive spatial reasoning for activity recognition using tools of ambient intelligence," Master, Science, UQAC, Chicoutimi, 2011.