UNIVERSITE LIBRE DE BRUXELLES
Faculté des Sciences Appliquées
Département CoDE - The Computer & Decision Engineering
Service IRIDIA - Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle

Contrôle de la taille de groupe en robotique en essaim : conception et étude d'une méthode scalable et robuste.

Directeur de mémoire : M. Marco Dorigo
Co-Promoteurs : M. Mauro Birattari,
M. Alexandre Campo,
M. Shervin Nouyan

Mémoire présenté en vue de l'obtention du grade d'Ingénieur Civil en Informatique

Antoine Dubois
Année académique 2006–2007

Résumé

Ce mémoire de fin d'étude traite de la conception d'un alogorithme de contrôle de la taille de groupe en robotique en essaim et de l'implémentation de celui-ci. Le protocole de communication entre agents mobiles requit pour la réalisation de cet algorithme a été développé durant l'élaboration de ce mémoire. En créant une structure virtuelle en arbre entre agents groupés, la taille du groupe peut être calculée et controlée. Le protocole de communication permet aux agents de créer et de maintenir un arbre virtuel les reliant, de calculer la taille du groupe et de contrôler cette taille en acceptant ou en refusant des agents s'approchant du groupe. L'algorithme a été testé sur le simulateur TWODEEPUCK. Une série d'expérimentations confirmant son comportement et démontrant sa réponse dynamique, sa robustesse et sa scalabilité a été réalisée avec fruit.

Table des matières

Remerciements

Je remercie tout d'abord Mopnsieur le Professeur Marco Dorigo, mon directeur de mémoire, d'avoir accepté de diriger ce travail. Grâce à la liberté qu'il m'a accordé dans la réalisation de ce travail, j'ai eu l'opportunité d'aborder cette étude selon une approche personnelle.

Mes remerciments vont aussi à mes co-promotteurs, Monsieur le Docteur Mauro Birattari, Monsieur Alexandre Campo et Monsieur Shervin Nouyan pour leur disponibilité et leurs conseils avisés. Je les remercie plus particulièrement pour leur soutien dans les moments plus difficiles où les idées manquaient.

Je tiens également à remercier les membres du projet E-PUCK pour notre entente, notre sérieux et pour les bons moments de détente que nous avons pu partager.

Mes remerciements s'adressent aussi aux membres du service IRIDIA pour leur soutien et leur conceils.

Je remercie mes parents, Jean-Jacques Dubois et Bernadette Cornet, de m'avoir donner une seconde chance. Ce travail est l'aboutissement de la confiance qu'ils ont placé en moi. Je les remercie également pour leur soutien moral inestimable. J'ai eu la chance de bénéficier de leurs conseils avisés et je leur suis reconnaisant des nombreuses heures qu'ils ont consacrées à la relecture ainsi que la patience dont ils ont fait preuve à mon égard.

Je remercie tout particulièrement mes grands-parents, Jacques Dubois et Jeanine Vandamme, pour leur soutien, leur confiance, leur disponibilité, leur patience et leur gentillesse permanente.

Je remercie également ma soeur, Emilie Dubois, mon frère et sa compagne, Vincent Dubois et Virginie Lago, pour leurs encouragements et leur support moral inconditionnel.

Mes remerciement s'adressent également à Anne-Laurence Delor, Benoit Minon et Colin Salmon pour leur soutien et leurs encouragements dans les moments difficiles.

Pour finir je tiens à te remercier, ma chère et tendre Claire. Sans toi, les moments difficiles m'auraient été insurmontables. Je te suis reconnaissant de tous tes conseils et de ton soutien.

Chapitre 1

Introduction

1.1 Choix du sujet

Aborder l'étude comportementale en robotique en essaim constitue pour moi une réelle opportunité d'aborder de manière plus approfondie le secteur de l'intelligence artificielle, plus particulièrement sur le secteur de l'intelligence distribuée. En effet, la robotique en essaim me fascine par le minimalisme comportemental et l'émergence de comportements complexes. La reproduction sur robots de comportements inspirés par la nature est depuis quelques années sujète à de nombreuses études. En particulier, la réalisation de tâches complexes par un ensemble de robots est un aboutissement dans ce domaine.
Ce sujet allie à la fois mon attirance pour la robotique avec mon interêt pour l'informatique en général. La rédaction d'un travail sur la conception d'un comportement auto-organisé m'offre la possibilité de mettre en pratique les connaissances acquises en informatique, tout en les applicant à un domaine fascinant.

1.2 Objectif du travail

L'objectif de ce mémoire est double. Le premier est de concevoir une méthode de contrôle auto-organisé de la taille de groupe en robotique en essaim sans leader. Le deuxième est d'appliquer cette méthode à un simulateur et d'étudier ses propriétés, en particulier sa robustesse et sa scalabilité.

1.3 Délimitation du sujet de mémoire

Le domaine de la robotique en essaim est vaste et très hétérogène. En effet, les systèmes multi-agents peuvent souvent être divisés en deux catégories : ceux utilisant des techniques basées sur le comportement et les communications implicites et ceux utilisant une représentation symbolique et des communications explicites. Les systèmes de la première catégorie ont tendance à explorer les stratégies comportementales inspirées des systèmes biologiques

comme les insectes sociaux alors que les systèmes de la deuxième catégorie sont plutôt inspirés des technologies informatiques de communication déjà développées. Ce mémoire traite de la conception d'un système multi-agents de la deuxième catégorie appliquée à des robots par l'intermédiaire du simulateur TWODEEPUCK. Il faut noter qu'aucune expérience physique n'a été réalisée. C'est-à-dire qu'aucune expérience ne s'est effectuée par l'observation simple des robots.
Le support de communication entre les robots n'est pas étudié dans le cadre de ce mémoire, non plus le modèle d'agrégation des robots.

1.4 Structure du travail et méthodologie

1.4.1 Plan

Préalablement à la description de la méthode, les définitions concernant la robotique en essaim, la communication entre agents et la théorie des arbres sont exposées afin d'autoriser une compréhension précise et claire des termes utilisés.

La description de la méthode commence par un résumé des concepts dont elle s'inspire. Suit la définition du protocole de communication ainsi que la structure des messages transitant dans l'arène expérimentale.
Pour conclure, une description détaillée de chaque pas d'éxécution du contrôleur des robots est présentée ainsi que le schéma de l'algorithme.

La deuxième partie de ce travail concerne l'étude de la méthode. Une description de l'environnement expérimental est faite avant d'aborder les propriétés intrinsèques du système. Une étude de la robustesse et de la scalabilité de la méthode est obtenue à partir d'une série d'expériences ciblées.

1.4.2 Méthodologie

La conception de la méthode est basée sur l'étude des robots E-PUCKS menée au service I.R.I.D.I.A de l'Université Libre de Bruxelles. Une série de tests ont été effectués sur les robots, entre autre l'étude des moteurs et des senseurs de proximité. Cette étude a permis de développer le simulateur TWODEEPUCK qui modélise le comportement et l'environnement des robots E-PUCKS. Pour plus de détails sur le simulateur, consultez L. Bury [3]. Pour plus de détails sur les robots E-PUCKS eux-mêmes, consultez O. Dedriche [4].

Le contrôle auto-organisé de la taille de groupe a dèjà été abordé par C. Melhuish [6] qui propose de contrôler la taille de groupes autour d'un site particulier par chorusing. Cette méthode, inspirée du comportement d'insectes sociaux comme le cricket, se base sur l'erreur de synchronisation des agents émettant des signaux à fréquence variable.

Le contrôle de la taille de groupe peut être utilisé dans une multitude d'applications, entre autre dans les système multi-tâches. En effet, une tâche complexe divisée en tâches plus simples requérant un certain nombre d'agents actifs peut être automatisée à un niveau supérieur si les agents effectuent un comptage auto-organisé.

Chapitre 2

Algorithme

Ce chapitre décrit la méthode conçue. Dans un premier temps, les termes spécifiques et leurs définitions sont listés afin d'en faciliter la compréhension. Nous allons ensuite aborder de manière concise les étapes de conceptions de l'algorithme avant d'approfondir les concepts plus sophistiqués, comme la table de voisinage et la communication entre agents. En conclusion, les deux parties du protocole conçu sont détaillées et accompagnées d'exemples concrets montrant le fonctionnement de l'algorithme étape par étape.

2.1 Définitions

2.1.1 Robotique en essaim

Agent Unité implémentant un algorithme de comportement. Cette unité est en général mobile et peut éventuellement communiquer. Un robot correspond parfaitement au rôle d'agent. Par la suite, des noms génériques de la forme 'A#' seront donnés aux agents.

Population Ensemble d'agents participant à une expérience.

Arène Surface dédiée à une population d'agents.

Groupe Sous-ensemble d'une population d'agents agrégés.

Voisin Un voisin d'un agent A1 est un agent que l'agent A1 "voit", en l'occurence, un agent avec lequel l'agent A1 peut communiquer.

Voisinnage Le voisinage d'un agent A1 est le sous-ensemble d'une population dont tous les agents sont voisins de A1.

2.1.2 Communication

Message Trame d'informations transmissibles d'un agent à un autre.

Source La source d'un message est l'agent qui envoye ou a envoyé ce message.

Destinataire Le destinataire d'un message est l'agent auquel le message est destiné.

Zone d'émission Surface autour d'un agent A1 dans laquelle un agent peut recevoir un message de l'agent A1. Cette surface est généralement circulaire et le rayon en est défini.

Rayon d'émission Rayon de la zone d'émission

Broadcast Un message émis par un agent A1 dont le destinataire est défini à broadcast peut être reçu par tous les agents dans la zone d'émission de l'agent A1.

2.1.3 Théorie des arbres

Elément Unité d'un ensemble. En l'occurence, un élément est un agent et l'ensemble est un groupe.

Relation Lien, éventuellement ordonné, entre deux éléments dans un ensemble.

Chemin Ensemble de relations reliant deux éléments.

Arbre Un ensemble d'éléments et de relations. Les relations d'un arbre sont ordonnées et sont de type parent-enfant. Dans un arbre sans cycle, il n'existe qu'un seul chemin entre deux éléments.

Parent Le parent d'un agent A1 dans un arbre est le seul agent A2 avec lequel A1 est relié par une relation

Enfant Un enfant d'un agent A1 est un agent dont A1 est le parent.

Racine La racine d'un arbre sans cycle est le seul élément qui n'a pas de parent.

Extrémité Une extrémité d'un arbre est un agent qui n'a pas d'enfant.

Ascendant de A Un ascendant d'un agent A1 est un agent qui est sur le chemin reliant l'agent A1 et la racine.

Descendant de A Un descendant d'un agent A1 est un agent A2 dont le chemin jusuqe la racine contient l'agent A1.

Branche Une branche d'un agent A1 est l'ensemble des descendants d'un agent A2, enfant de A1. Un agent possède donc autant de branches que celui-ci a d'enfants.

Génération La génération d'un agent est la distance mesurée en relation de l'agent à la racine. La racine est de génération zéro. Un enfant de la racine est de première génération.

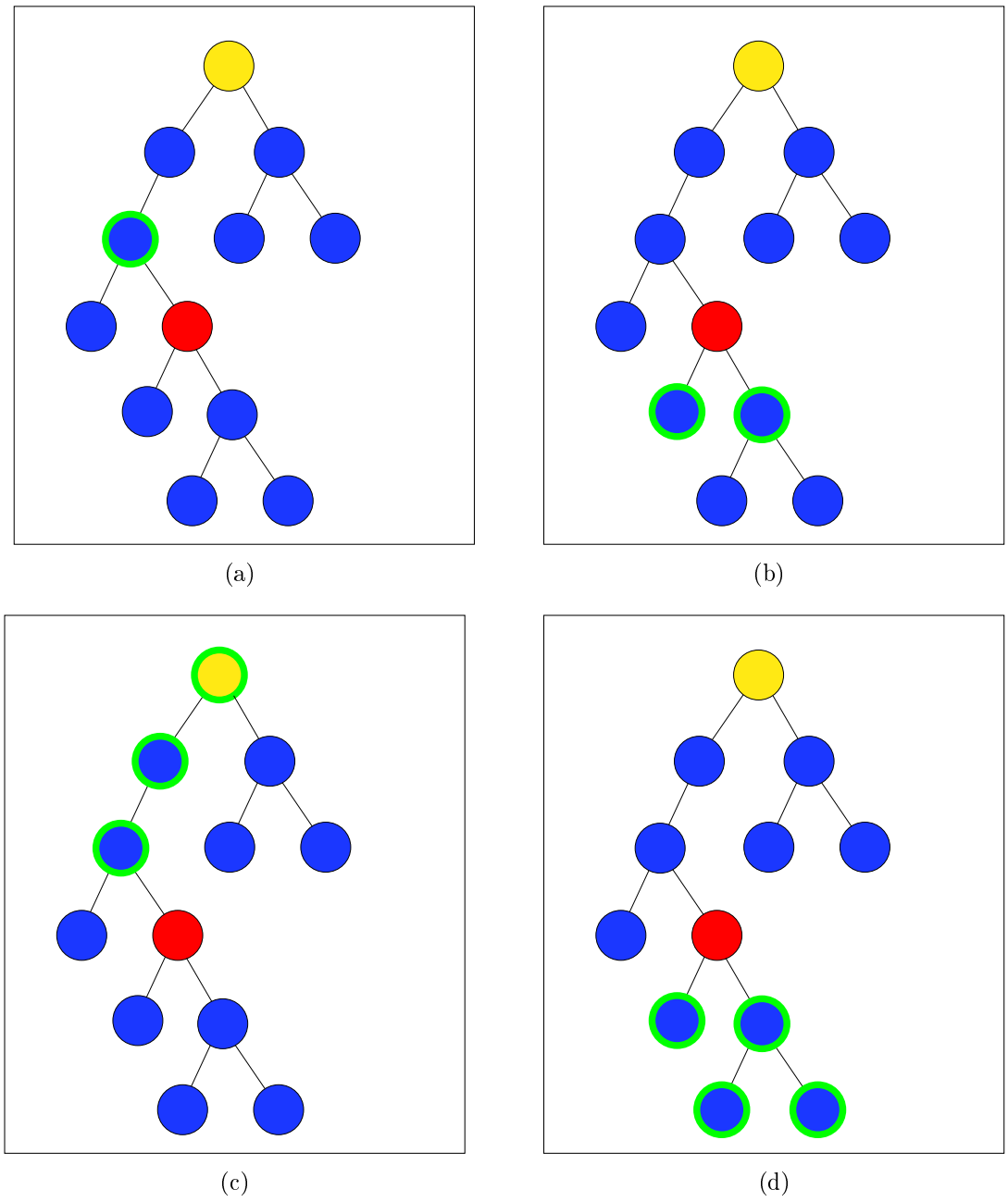


FIG. 2.1 – Definitions des ensembles relationnels relatifs à l'élément rouge - (a) Le parent - (b) Les enfants - (c) Les ascendants - (d) Les descendants

2.2 Conditions d'application

L'algorithme décrit est applicable dans certaines conditions. Il a été conçu pour être appliqué à des agents pouvant se déplacer et communiquer par des messages structurés.
Une condition nécessaire est l'identification unique des agents. Chaque agent doit posséder

un identifiant unique au sein de la population. Beaucoup de scientifiques rejettent cette pratique car pour une grande population il est difficile d'établir cet identifiant unique et rend donc le système non-scalable. Une solution à ce problème est de générer aléatoirement des nombres dont la taille peut etre adaptée au nombre d'agents. Cet identifiant ne peut être de trop grande taille. En effet, dans la suite de ce mémoire, il est montré que la majeure partie de l'information véhiculant par message est une série d'identifiants d'agents. Si l'identifiant des agents est trop grand, la taille des messages est affectée et donc la réception et le traitement aussi. Si l'identifiant est trop petit, il se peut que deux agents aient le même identifiant et donc que le système ne fonctionne pas correctement. Il est donc nécessaire de prévoir une taille adaptée au nombre d'agents et à la dynamique du système.
Tous les agents de la population doivent implémenter l'algorithme de façon identique.

2.3 Description

Cette section aborde les concepts généraux de la méthode. Le problème posé est de trouver une méthode qui permet le contrôle de la taille d'un groupe en robotique en essaim. La solution proposée s'inspire de la théorie des arbres et de la communication entre ordinateurs.

2.3.1 Comptage du nombre d'éléments d'un groupe

En créant une structure d'arbre sans cycle dans un groupe d'agents, on peut tirer avantage des propriétés de cette structure pour effectuer un comptage des éléments. L'existence d'un chemin unique entre deux éléments d'un arbre sans cycle est une propriété intéressante qui permet d'en déduire qu'il n'existe qu'un chemin entre la racine de l'arbre et tous les autres éléments de celui-ci. La racine possède autant de branches que d'enfants. On peut donc diviser la tâche en contrôlant le nombre d'éléments dans chaque branche.

Le problème est de former des groupes dont la taille serait fixée. Il a été décidé de scinder ce problème en considérant des limites inférieure et supérieure de tailles valides. En égalisant ces deux limites, l'intervalle devient vide et une seule taille est valide. Cet intervalle de tailles valides permet d'aborder le problème par ces deux limites fixées.

En ce qui concerne la limite supérieure, il a été décidé de limiter le nombre d'éléments dans chacune des branches de la racine. Pour la limite inférieure, un calcul de la taille du groupe est effectué et la racine décide de dissoudre le groupe si la taille est inadéquate.

2.3.2 Représentation locale d'un arbre sans cycle

L'algorithme prend en compte la gestion d'une table de voisinage qui est la représentation locale de l'arbre faite par les agents. Cette table contient les information sur les voisins, leur role dans la relation et des informations temporaires nécessaires pour le comptage (voir section 2.4). Le fait que les agents n'ont besoin que d'une connaissance locale du système est

un bon atout pour la scalabilité qui sera étudiée ultérieurement dans ce mémoire.

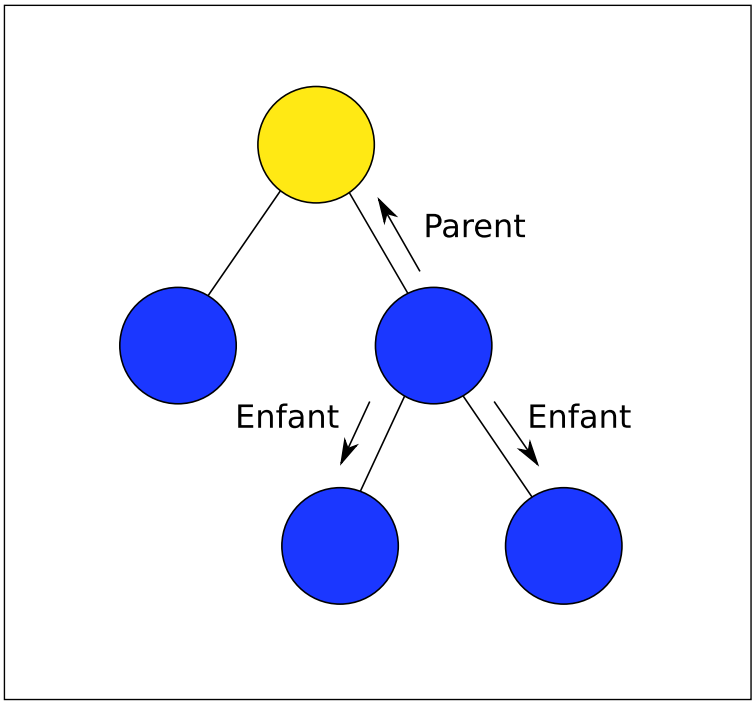


FIG. 2.2 – Représentation d'un arbre au niveau des éléments

2.3.3 Création d'une structure en arbre

Créer une structre en arbre entre les agents est la première étape de l'algorithme. Pour créer un arbre, il est nécessaire qu'une racine soit désignée. Chaque agent est potentiellement une racine et peut le devenir lorsqu'un groupe est formé. Les autres agents, requêtant l'admission dans un arbre, reçoivent par message l'autorisation de la racine. Le rôle d'un agent dans un arbre est défini par les rôles que celui-ci a attribué à ses voisins.

Former un groupe revient à immobiliser les agents lorsqu'ils rencontrent d'autres agents. La reconnaissance de proximité d'un agent se fait par la réception de messages. Ainsi, lorsqu'un agent reçoit un message, c'est qu'il est à proximité d'un autre agent. Ce comportement d'immobilisation tend à former des groupes et permet aux agents voisins dans un groupe de communiquer.

2.3.4 Limitation du nombre d'éléments dans un arbre

Limiter la taille de l'arbre peut se réduire à limiter la taille de chacune des branches. En connaissant la limite supérieure du nombre de descendants qu'il peut y avoir, un agent envoie

à chacun de ses enfants une partition de cette quantité. Chacun de ses enfants fera de même et lorsqu'un agent ne peut plus avoir de descendants, il envoie à ses enfants un message leur signalant qu'ils doivent quitter l'arbre. La racine, déclencheur du processus de limitation, prend en compte la taille maximale que le groupe peut atteindre.

2.3.5 Comptage du nombre d'élements dans un arbre

Dans l'arbre créé, les extrémités de l'arbre sont les éléments qui n'ont pas d'enfants. Ces extrémités envoient un message à leur parent pour leur dire qu'ils n'ont pas de descendants. Un parent, lorsque tous ses enfants ont envoyé ce message, additionne les quantités reçues, ajoute une unité pour se prendre en compte et envoye le total à son parent.
Ce comportement simple permet à la racine de connaitre la taille du groupe. Sur base de cette taille, la racine décide de conserver ou de dissoudre le groupe.

2.4 Table de voisinage

L'algorithme inclut la gestion d'une table de voisinage. Dans cette table de voisinage, chaque entrée représente un voisin pour lequel des paramètres sont définis. Voici les champs que la table de voisinage inclut pour chaque entrée.

Role	:	role du voisin dans l'arbre.
NbreEnfants	:	nombre d'enfants directs du voisin.
DateCréation	:	date d'ajout ou du dernier message du voisin dans la table.
NbreDescendants	:	nombre d'agents dans la branche.
NbreRessourceADistribuer	:	quantité de ressources à distribuer.

Le champ Role définit le rôle du voisin dans l'arbre. Ce champ peut prendre les valeurs suivantes :

NONE	:	Le voisin n'a pas de relation directe
CHILD	:	Le voisin est un enfant
PARENT	:	Le voisin est LE parent

Il ne peut y avoir qu'un seul voisin dont le rôle est PARENT pour respecter la structure en arbre.

Il faut bien distinguer le champ NbreEnfants et le champ NbreDescendants. Le champ NbreEnfants représente le nombre d'enfants directs que le voisin possède tandis que le champ NbreDescendants représente le nombre de descendants dans la branche issue du voisin. Le champ DateCréation est utile pour l'élimination de voisins dans la table. Si la date d'ajout ou de dernière réception de message est dépassée depuis un intervalle de temps défini, l'algorithme élimine le voisin de la table. Dès qu'un message est reçu, l'agent identifie la source, l'ajoute et le met à jour dans la table de voisinage.

2.5 Communication

2.5.1 Communication entre agents

La méthode s'applique à des agents pouvant communiquer par messages structurés mais la façon dont ces agents émettent et recoivent les messages n'est pas un facteur étudié dans ce mémoire. Dans les expériences effectuées, la communication entre les agents a été réduite à son plus simple modèle. En l'occurence, si un agent est dans le rayon d'émission d'un autre agent émettant un message, le premier agent reçoit le message. La communication, le déplacement des agents, le comportement sont des facteurs importants pour la dynamique du système. Si la communication n'est pas adaptée, il est probable que l'information ne voyagera pas assez rapidement des feuilles de l'arbre jusque la racine et donc l'image de la taille que recoit la racine peut etre obsolète ainsi que les décisions que celle-ci devra prendre.

2.5.2 Structure des messages

AgentSource	AgentDestinataire	AgentExclus	Commande
AgentTierce	Paramètre		

FIG. 2.3 – Structure des messages et noms des champs

Les messages comportent les champs suivants :

AGENTSOURCE	:	agent source du message.
AGENTDESTINATAIRE	:	spécifie la destination du message.
AGENTEXCLUS	:	spécifie l'agent à qui le message n'est pas destiné.
COMMANDE	:	identifiee la commande du message.
AGENTTIERCE	:	identifie un tierce agent.
PARAMÈTRE	:	une zone numérique.

2.5.3 Envoi et réception des messages

La structure des messages permet de cibler les agents qui recevront les messages et ceux qui ne devront pas les recevoir. Quand un agent est dans la zone d'émission d'un message, il le lit quel que soit son contenu. L'agent lit la zone AGENTDESTINATAIRE et s'il se reconnait,

traite le contenu du message. Pour envoyer un message en broadcast, l'agent laisse vide la zone AgentDestinataire. Mais dans le cas où la zone AgentExclus contient son identifiant, il ne le traite pas. La zone Commande sert à identifier ce que l'agent doit faire avec les zones AgentTierce et Paramètre.

2.6 Protocole

Le système décrit ici est divisé en deux étapes. La première est la création d'un arbre virtuel entre les agents et la deuxième est le processus de comptage et de prise de décision pour atteindre la taille désirée. L'algorithme utilise conjointement deux protocoles pour effectuer ces deux étapes. Cette section décrit les deux protocoles ainsi que leur jeu de commandes.

2.6.1 TSP : Création de l'arbre

2.6.1.1 Description et commandes

Le protocole de création de l'arbre nommé TSP pour Tree Setup Protocol se base sur un jeu de commandes spécifiques.

Voici la liste des commandes que le protocole TSP utilise :

ALONE Commande est envoyée par les agents qui ne font pas partie d'un groupe.

DRP Requête aux autres agents pour savoir s'ils appartiennent à un arbre et dans ce cas s'ils peuvent confirmer l'entrée dans l'arbre.

ADRP Autorisation donnée à la source d'un message DRP d'entrer dans l'arbre.

MIR Reinitialisation les agents la recevant et ayant la même racine que celle spécifiée dans la zone AgentTierce.

ARID Commande envoyée par les agents dans l'arbre de manière asynchrone. Elle permet de gérér l'apparition de racines multiples dans un même voisinage.

IAS Commande envoyée par les agents dans l'arbre de manière asynchrone. Le message est destiné au parent de l'agent dans l'arbre. Son but est de permettre au parent de savoir que l'agent source du message est son enfant dans le voisinage et le nombre d'enfants que celui-ci a.

2.6.1.2 Scénario d'utilisation du protocole TSP

Les agents sont disposés de manière aléatoire dans l'arène. Ils émettent des messages dont le champ COMMANDE a la valeur ALONE. Les messages ALONE servent uniquement à permettre aux autres agents de 'voir' la source.

Valeurs des champs d'un message ALONE émis par un agent A1 :

AGENTSOURCE	:	A1
AGENTDESTINATAIRE	:	*vide*
AGENTEXCLUS	:	*vide*
COMMANDE	:	ALONE
AGENTTIERCE	:	*vide*
PARAMÈTRE	:	*vide*

Quand un agent reçoit un message, celui-ci est à proximité d'un autre agent et fait donc partie d'un groupe. L'agent se met alors à émettre des messages dont le champ COMMANDE a la valeur DRP.

Valeurs des champs d'un message DRP émis par un agent A1 :

AGENTSOURCE	:	A1
AGENTDESTINATAIRE	:	*vide*
AGENTEXCLUS	:	*vide*
COMMANDE	:	DRP
AGENTTIERCE	:	*vide*
PARAMÈTRE	:	*vide*

A ce moment, l'agent peut soit devenir racine, soit être accepté par un autre agent déjà dans un arbre.
Un agent racine, même seul, fait partie d'un arbre. Il peut donc accepter d'autres agents en repondant aux messages DRP par des messages ADRP.

Valeurs des champs d'un message ADRP émis par un agent A1 ayant reçu un message DRP d'un agent A2 (l'agent est dans un arbre dont la racine est R1) :

AGENTSOURCE	:	A1
AGENTDESTINATAIRE	:	A2
AGENTEXCLUS	:	*vide*
COMMANDE	:	ADRP
AGENTTIERCE	:	R1
PARAMÈTRE	:	Age de R1

L'age de la racine R1 est important. En effet ce sera ce paramètre qui décidera lequel des deux arbres dans un même voisinage sera gardé (voir section 2.6.1.3).

Lorsqu'un agent est accepté dans un arbre, il envoye des messages dont le champ COMMANDE à la valeur IAS. destiné à son parent. Cela permet au parent de tenir à jour sa table de voisinage.

Valeurs des champs d'un message IAS émis par un agent A1 à son parent A2 :

AGENTSOURCE	:	A1
AGENTDESTINATAIRE	:	A2
AGENTEXCLUS	:	*vide*
COMMANDE	:	IAS
AGENTTIERCE	:	*vide*
PARAMÈTRE	:	Nombre d'enfants de A1

Le nombre d'enfants de A1 est calculé sur base de sa table de voisinage. Cette information est importante pour le parent lors de la phase de limitation (voir section 2.6.2).

2.6.1.3 Evitement de l'apparition de racines multiples dans un même voisinage

Comme décrit ci-dessus, le protocole TSP permet de générer une structure en arbre sans cycle dans un groupe d'agents. Mais cela n'empêche pas que deux racines puissent apparaître dans un même groupe, ce qui constitue un problème. En effet, l'algorithme permet de contrôler la taille d'un arbre mais si deux arbres existent au sein d'un même groupe, la taille valide de ce groupe sera le double de celle autorisée.
Pour pallier à ce problème, nous avons ajouté les messages ARID et MIR au panel des commandes du protocole TSP. L'utilisation de ces messages se fait comme suit. Les agents appartenant émettent des messages dont le champ COMMANDE a la valeur ARID.

Valeurs des champs d'un message ARID émis par un agent A1 appartenant à un arbre dont la racine est R1 :

AGENTSOURCE	:	A1
AGENTDESTINATAIRE	:	*vide*
AGENTEXCLUS	:	*vide*
COMMANDE	:	ARID
AGENTTIERCE	:	R1
PARAMÈTRE	:	Age de la racine R1

Quand un agent reçoit un message ARID, il compare l'identifiant de la racine de l'arbre auquel il appartient et l'identifiant du champ AGENTTIERCE du message. Si ceux-ci sont identiques, c'est que les agents font partie du même arbre. Par contre, s'ils sont différents, l'agent qui a reçu le message doit prendre une décision, choisir lequel des deux arbres va persister. Ce choix se base sur l'âge des racines. Notre première solution était d'inclure le deuxième arbre dans le premier en inversant certaines relations mais nous avons observé qu'en dissolvant un arbre, celui-ci était rapidement intégré dans l'autre. L'agent qui observe

la rencontre de deux arbres choisit donc un des ces arbres et le dissout. Nous avons décidé que l'arbre le plus jeune serait dissout. En effet, il est aisé de comprendre qu'un arbre plus jeune soit moins grand.

Pour dissoudre un arbre, il envoye un message dont le champ COMMANDE a la valeur MIR.

Valeurs des champs d'un message MIR émis par un agent A1 appartenant à un arbre dont la racine est R1 et ayant choisi de dissoudre l'arbre dont la racine est R2 :

AGENTSOURCE	:	A1
AGENTDESTINATAIRE	:	*vide*
AGENTEXCLUS	:	*vide*
COMMANDE	:	MIR
AGENTTIERCE	:	R2
PARAMÈTRE	:	*vide*

Quand un agent reçoit un message MIR, il compare l'identifiant de la racine de l'arbre auquel il appartient et l'identifiant du champ AGENTTIERCE du message. Si ceux-ci sont différents, il ne fait rien. Par contre, s'ils sont identiques, il envoye le même message et quitte l'arbre.

Vu qu'il est quasiment impossible pour les agents d'avoir leurs horloges internes synchronisées, nous proposons une solution contournant le problème. Quand un agent devient racine, il initialise un compteur qui est incrémenté à chaque pas d'éxécution. Quand un agent A1 s'approche de la racine R1 et demande l'autorisation d'entrer dans l'arbre, la racine R1 envoye la valeur de ce compteur dans le champ PARAMÈTRE du message ADRP. L'agent A1 va lui aussi incrémenter ce compteur en interne à chaque pas d'éxécution. C'est ainsi que l'âge de la racine de l'arbre est propagé dans l'arbre.

Le protocole TSP, adapté pour la robotique en essaim permet donc de créer une structure d'arbre virtuel entre des robots communicants. Les robots n'ont besoin que d'une connaissance locale de leur environnement, c'est-à-dire les agents voisins dans l'arbre et leur rôle. Dans le cas où deux groupes de racine différente se rencontrent, le protocole TSP permet de dissoudre un des deux arbres et dans le cas d'une dynamique adaptée, les robots de l'arbre dissout peuvent rapidement s'intégrer à l'arbre persistant.

2.6.2 CMP : Contrôle de la taille du groupe

Comme décrit à la section 2.3.1, nous avons divisé le problème du contrôle de la taille de groupe en deux en fixant une limite supérieure et une limite inférieure. Le protocole CMP conçu lors de ce mémoire est constitué respectivement de deux parties distinctes. La première sert à limiter le nombre d'éléments dans l'arbre. La deuxième permet à la racine de connaître la taille du groupe et de réagir si celle-ci est inférieure à la limite fixée.

2.6.2.1 Description et commandes

Voici la liste des commandes que le protocole CMP utilise :

TAD

Commande destinée aux enfants de la source. Il spécifie la quantité de ressources que l'enfant devra à son tour distribuer.

DAN

Commande, envoyée en broadcast mais traitée seulement par les enfants, servant à faire quitter de l'arbre tous les agents dans la branche issue de la source.

CM

Cette commande, envoyée au parent, sert à spécifier le nombre de descendants dans la branche issue de la source.

2.6.2.2 Limite supérieure

Cette section décrit comment, à partir de deux commandes et d'une structure en arbre dans un ensemble d'agents, limiter le nombre d'agents dans un groupe. A partir de la racine, des branches issues de ses enfants se développent. La racine, connaissant la taille maximale que le groupe peut atteindre, répartit cette taille maximale entre chacun de ses enfants. La racine calcule donc une quantité par enfant et leur envoye respectivement la quantité calculée par des messages dont le champ COMMANDE a la valeur TAD.

Valeurs des champs d'un message TAD émis par une racine R1 à un des ses enfants, soit l'agent A1 :

AGENTSOURCE	:	R1
AGENTDESTINATAIRE	:	A1
AGENTEXCLUS	:	*vide*
COMMANDE	:	TAD
AGENTTIERCE	:	*vide*
PARAMÈTRE	:	Quantité calculée pour A1

Quand un agent reçoit un message TAD, celui-ci à son tour répartit cette quantité entre chacun de ses enfants.

Valeurs des champs d'un message TAD émis par un agent A1 à un des ses enfants, soit l'agent A2 :

AgentSource	:	A1
AgentDestinataire	:	A2
AgentExclus	:	*vide*
Commande	:	TAD
AgentTierce	:	*vide*
Paramètre	:	Quantité calculée pour A2

Si le champ Paramètre à la valeur nulle, alors l'agent doit faire quitter de l'arbre tous ses enfants. Pour ce faire, il envoye un message dont le champ Commande a la valeur DAN.

Valeurs des champs d'un message DAN émis par un agent A1 à un des ses enfants, soit l'agent A2 :

AgentSource	:	A1
AgentDestinataire	:	A2
AgentExclus	:	*vide*
Commande	:	DAN
AgentTierce	:	*vide*
Paramètre	:	Quantité calculée pour A2

A la réception d'un message DAN, l'agent envoye ce message à son tour à ses enfants et quitte l'arbre. Il est à noter qu'un message TAD dont le champ Paramètre a la valeur -1 a le même comportement qu'un message DAN.

On peut voir ce processus comme une distribution pyramidale de ressources. Chaque élément se voit attribuer une quantité de ressources qu'il répartit ensuite. Une récursivité apparaît dans cette procédure, ce qui facilite l'implémentation sur chacun des agents. Cette répartition, décrite plus en détails à la section 2.7, se base sur le nombre d'enfants et le nombre de petits-enfants.

2.6.2.3 Limite inférieure

Cette partie décrit comment, à partir d'une commande et d'une structure en arbre dans un ensemble d'agents, la racine peut avoir une image de la taille du groupe. Les extrémités de l'arbre sont les seuls agents à ne pas avoir d'enfants. Le parent, dont les descendants ne sont que ses enfants, peut additionner le nombre d'enfants qu'il a et envoyer à son parent le total calculé. Ainsi, chaque agent, quand chacun de ses enfants lui a communiqué le nombre de descendants, communique à son parent le nombre de descendants issus de sa branche. Quand la racine réceptionne le nombre de descendants de chacune de ses branches, elle les additionne et peut ainsi connaître la taille du groupe. Ce processus peut être vu comme une vague d'informations qui part des feuilles de l'arbre jusque la racine. Les messages utilisés sont des messages dont le champ Commande a la valeur CM.

Valeurs des champs d'un message CM émis par un agent A1 à son parent, soit l'agent A2 :

AgentSource	:	A1
AgentDestinataire	:	A2
AgentExclus	:	*vide*
Commande	:	CM
AgentTierce	:	*vide*
Paramètre	:	Nombre de descendants de A1, y compris A1

Un exemple simple est illustré sur la figure 2.4.

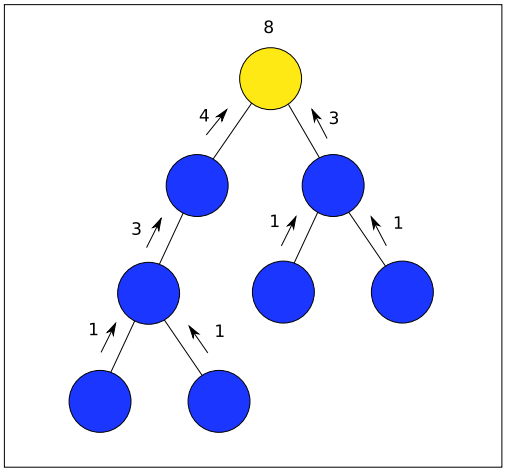


Fig. 2.4 – Mise en application du calcul de la taille du groupe.

2.7 Fonctionnement de l'algorithme

Après avoir décrit les concepts de base, cette section décrit de manière approfondie le fonctionnement de l'algorithme. Celui-ci se base sur une machine à états finie dont les états et les transitions sont expliqués. On peut considérer les agents comme des sytèmes à entrées-sorties. Les entrées sont les messages reçus et les sorties sont le déplacement et les messages envoyés. Une description détaillée d'un pas d'éxécution est également faite.

2.7.1 Machine à états finie

Voici la liste des états de notre machine à états finie.

EXPLORE	:	Exploration de l'arène en random walk
NEIGHBOUR	:	Dans le voisinage d'autres agents
INTREE	:	Element d'un arbre
ROOT	:	Racine d'un arbre

L'agent peut se retrouver dans un de ces états. En fonction de l'état, l'agent traitera les messages reçus et en enverra d'autres. Nous allons maintenant décrire chacun de ces états et les transitions vers les autres états. Certains états requièrent des méthodes de calcul pour produire le ou les messages à envoyer. Ces méthodes seront décrites plus loin.

2.7.1.1 Etat Explore

L'agent dans l'état EXPLORE, état initial des agents, explore l'arène en random walk en évitant les obstacles (autres agents et les bords de l'arène). Il émet en permanence un message ALONE. Quand deux agents se rencontrent, ils recoivent les messages ALONE l'un de l'autre. Ils passent alors dans l'état NEIGHBOUR. Remarquons que la commande du message reçu pour passer à l'état NEIGHBOUR peut être de n'importe quel type.

La figure 2.5 illustre le schéma de transition pour l'état EXPLORE.

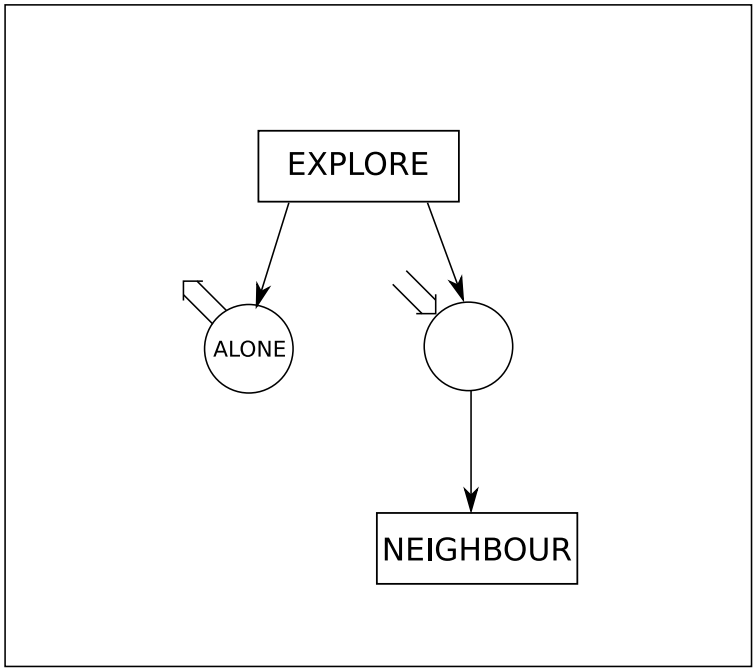


FIG. 2.5 – Etat EXPLORE

2.7.1.2 Etat Neighbour

L'agent dans l'état NEIGHBOUR est dans le voisinage d'autres agents. Il ne se déplace plus et émet sporadiquement des messages DRP, requêtes pour être accepté dans un arbre. Ils forment ensemble un groupe. C'est à partir d'ici qu'il faut créer la structure en arbre. Pour ce faire, l'agent peut soit passer à l'état ROOT pour devenir la racine d'un arbre, ou soit être accepté dans un arbre déjà existant et passer à l'état INTREE.

Les agents dans l'état NEIGBHOUR ont une certaine probabilité de passer à l'état ROOT à chaque pas d'éxécution. Pour passer à l'état INTREE, ils doivent recevoir un message ADRP. A la réception d'un message ADRP, ils gardent en mémoire qui est leur parent dans la table de voisinage et qui est leur racine. Ils stockent également le compteur pour l'âge de la racine.

La figure 2.6 illustre le schéma de transition pour l'état NEIGHBOUR.

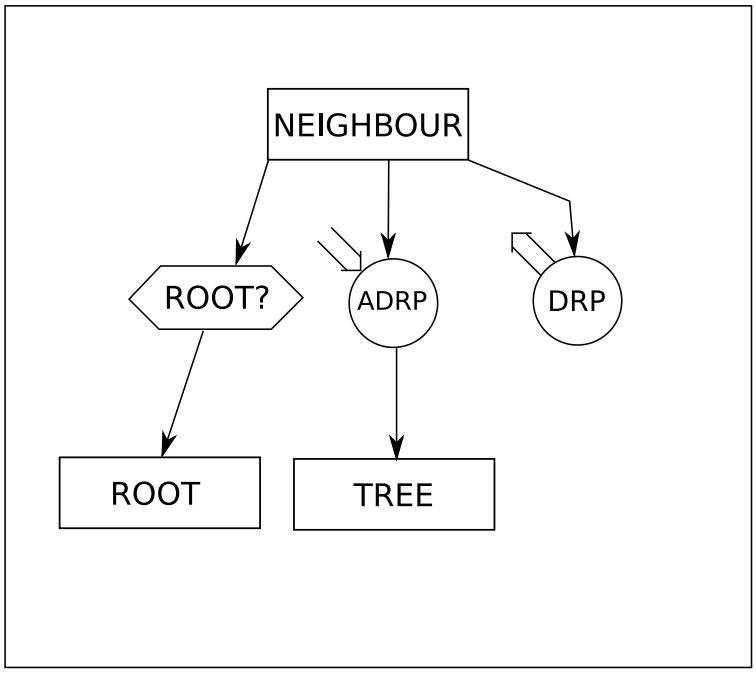


FIG. 2.6 – Etat NEIGHBOUR

2.7.1.3 Etat Root

Un agent dans l'état ROOT est la racine d'un arbre. Il émet sporadiquement des message ARID, pour détecter un éventuel arbre différent du sien dans son voisinage.

A la réception d'un message MIR, l'agent compare son identifiant avec l'identifiant dans le champ AGENTTIERCE du message.

```
Réception message MIR
Si Identifiant agent = identifiant agent tierce message
   Envoyer message MIR(identifiant agent)
   Passer à l'état EXPLORE
Si Identifiant agent != identifiant agent tierce message
   Pas de traitement
```

A la réception d'un message ARID, l'agent compare son identifiant avec l'identifiant dans le champ AGENTTIERCE du message.

```
Réception message ARID
Si Identifiant agent != identifiant agent tierce message
   Si Age racine > paramètre message
      Envoyer MIR(identifiant agent tierce message)
   Si Age racine < paramètre message
      Envoyer MIR(Identifiant agent)
      Passer à l'état EXPLORE
Si Identifiant agent = identifiant agent tierce message
   Pas de traitement
```

A la réception d'un message CM, l'agent vérifie si tous ses enfants ont envoyé un message CM. Si tel est le cas, Il envoye à son parent un message CM dont le champ PARAMÈTRE prend pour valeur la somme des message CM reçus et ajouté d'une unité.

```
Si pas d'enfants alors
   Envoyer CM(1) au parent
Réception message CM
Si tous les enfants ont envoyé CM
   Total = Somme des CM
   Ajouter 1 à Total
   Si dissoudre groupe
      Envoyer DAN aux enfants
```

A la réception d'un message DRP, l'agent envoye à la source du message un message ADRP dont le champ AGENTTIERCE a pour valeur l'identifiant de la racine de l'arbre et dont le champ PARAMÈTRE a pour valeur l'âge de la racine.

```
Réception message DRP
Envoyer ADRP(identifiant racine, age racine) à la source
```

Un agent racine peut déclencher le processus de limitation avec une certaine probabilité à chaque pas d'éxécution. Le processus de limitation est décrit à la section 2.7.3.

La figure 2.7 illustre le schéma de transition pour l'état ROOT.

2.7.1.4 Etat InTree

Un agent dans l'état INTREE est un élément d'un arbre. Il émet sporadiquement des messages ARID, pour détecter un éventuel arbre différent du sien dans son voisinage et des messages IAS pour signaler sa présence à son parent.

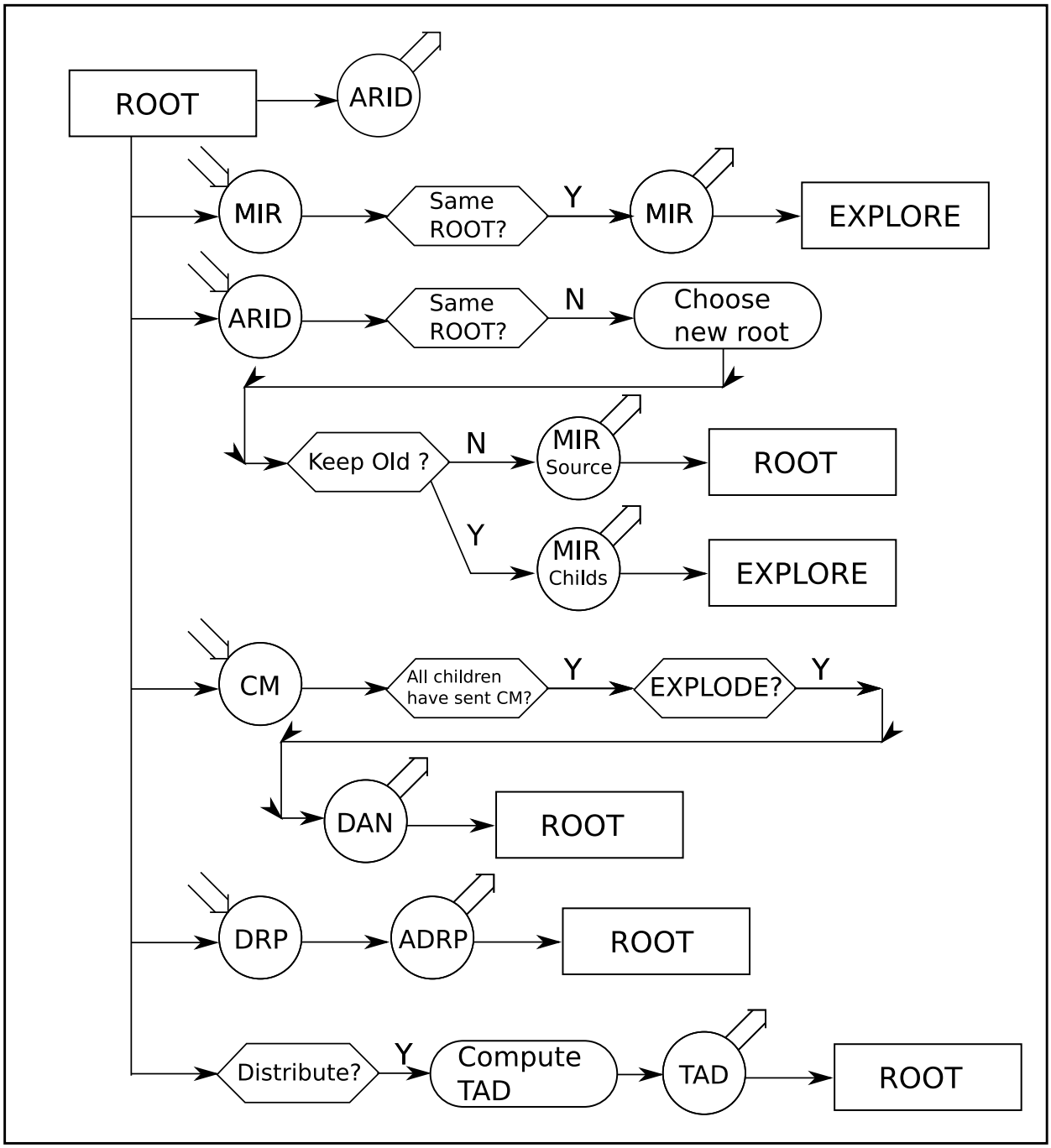


FIG. 2.7 – Etat ROOT

A la réception d'un message MIR, l'agent compare l'identifiant de sa racine avec l'identifiant dans le champ AGENTTIERCE du message.

```
Réception message MIR
Si Identifiant racine = identifiant agent tierce message
```

```
   Envoyer message MIR(identifiant racine)
   Passer à l'état EXPLORE
Si Identifiant racine != identifiant agent tierce message
   Pas de traitement
```

A la réception d'un message ARID, l'agent compare l'identifiant de sa racine avec l'identifiant dans le champ AgentTierce du message.

```
Réception message ARID
Si Identifiant racine != identifiant agent tierce message
   Si Age racine > paramètre message
      Envoyer MIR(identifiant agent tierce message)
   Si Age racine < paramètre message
      Envoyer MIR(Identifiant racine)
      Passer à l'état EXPLORE
Si Identifiant racine = identifiant agent tierce message
   Pas de traitement
```

A la réception d'un message DAN, l'agent ainsi que ses enfants doivent quitter l'arbre. Pour ce faire, il envoye à ses enfants un message DAN.

```
Réception message DAN
Envoyer DAN aux enfants
```

A la réception d'un message TAD, l'agent a trois possibilités en fonction de la valeur du champ Paramètre du message.

1. le paramètre vaut -1 : l'agent ainsi que ses enfants doivent quitter l'arbre.
2. le paramètre vaut 0 : l'agent reste dans l'arbre mais ses descendants doivent le quitter.
3. le paramètre vaut n : l'agent doit répartir la quantité entre ses enfants.

Le processus de répartition, quand le paramètre est supérieur à zéro, est décrit à la section 2.7.3.

La figure 2.7 illustre le schéma de transition pour l'état INTREE.

2.7.2 Création d'un arbre

La création d'un arbre se fait par la recontre d'agents et ensuite par l'établissement de relations ordonnées entre les agents. A l'instant initial, les agents sont dans l'état EXPLORE et voyagent aléatoirement dans l'arène. Quand ils se rencontrent, ils passent à l'état NEIGHBOUR. Dans cet état, ils peuvent soit devenir racine soit être accepté dans un arbre. Une fois qu'une racine est apparue, les agents dans le voisinage de la racine entrent dans l'arbre.

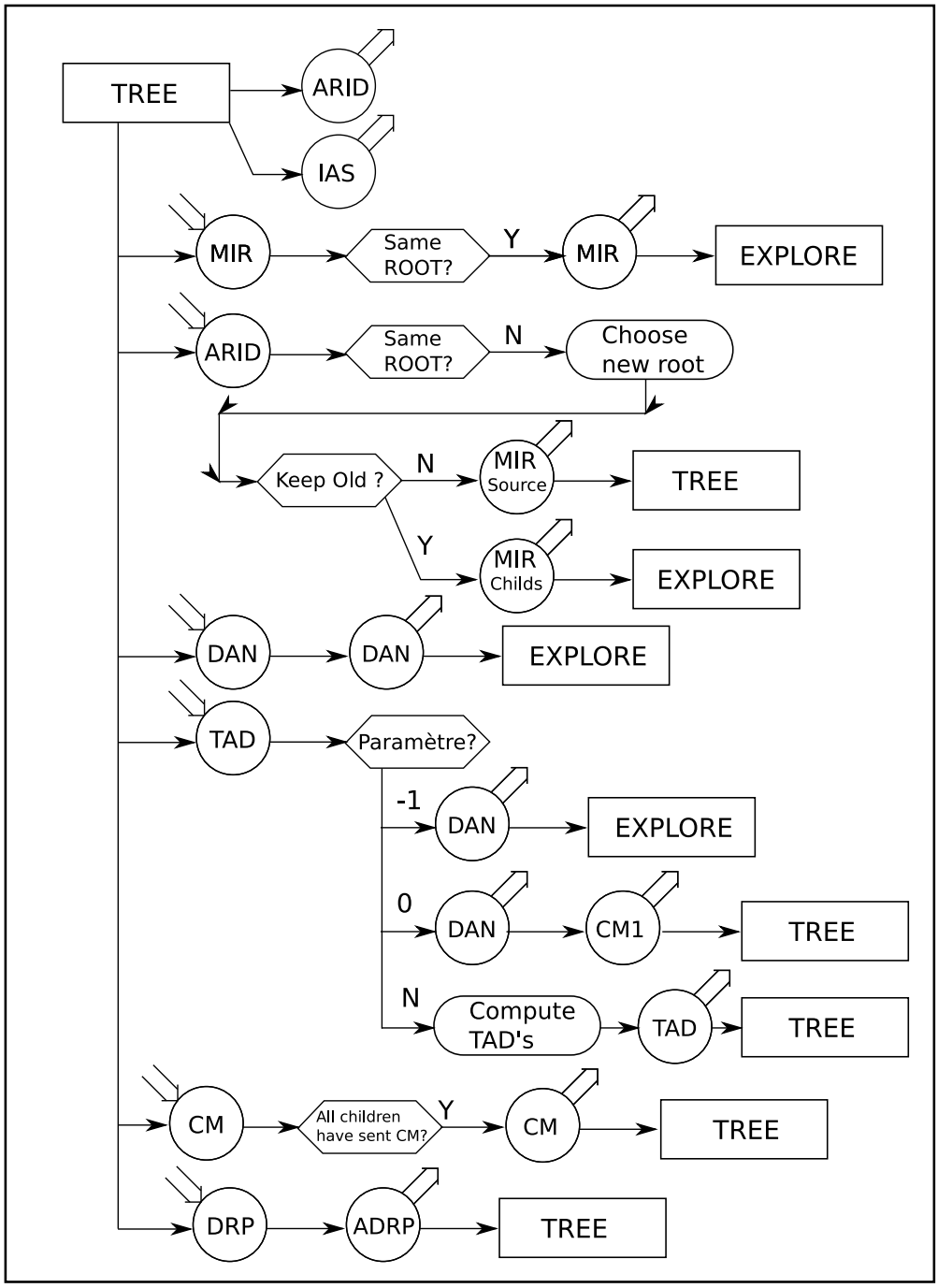


FIG. 2.8 – Etat TREE

La figure 2.9 illustre l'établissement d'un arbre entre deux agents.

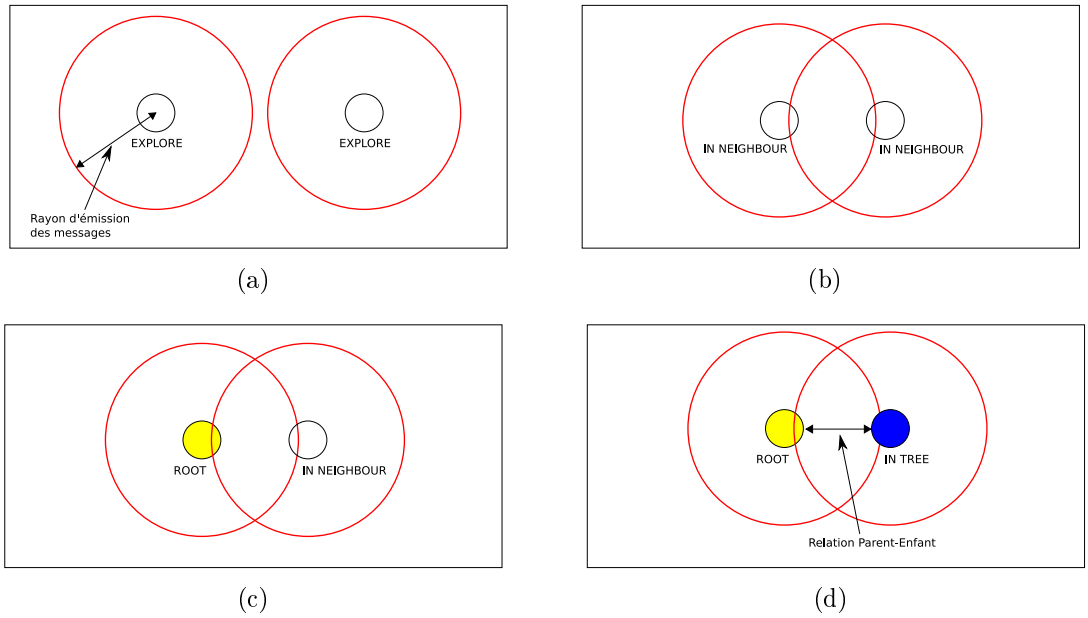


FIG. 2.9 – Etablissement d'un arbre entre deux agents. (a) Exploration de l'arène par les agents. (b) Rencontre des agents. (c) Apparition d'une racine. (d) Entrée d'un agent dans l'arbre.

2.7.3 Limitation

La limitation peut se voir comme une distribution de ressources diminuant avec les générations. La racine, de génération nulle, connait la quantité maximale d'éléments qui peuvent appartenir à l'arbre. Elle répartit ces ressources entre chacun de ses enfants et leur fait connaître ceci via des messages TAD. Quand un agent reçoit un message TAD, il répartit à son tour les ressources qui lui ont été attribuées entre ses enfants. Ce comportement récursif s'arrête dans une branche lorsqu'un agent reçoit une quantité nulle de ressource.

Le calcul de répartition entre enfants est effectué sur base du nombre d'enfants, sur base du nombre de petits-enfants et sur base de la quantité attribuée à l'agent. Par la suite, le symbole N représentera le nombre d'enfants de l'agent considéré, le symbole T représentera la quantité de ressources attribuée à cet agent.

Il y a trois cas possible pour la répartition.

1. $N < T$
2. $N = T$
3. $N < T$

Chacun de ces cas est décrit ci-dessous.

$N > T$ Il y a plus d'enfants que permis. L'agent fait savoir à T enfants qu'ils peuvent rester dans l'arbre mais que tous leurs descendants doivent le quitter. Aux autres enfants, il

dit qu'ils doivent quitter l'arbre ainsi que tous leurs descendants.

```
Si N > T
   Garder T enfants sans descendants : envoyer T fois message TAD(0)
   Rejetter N-T enfants : envoyer N-t fois message TAD(-1)
```

N = T Il y a autant d'enfants que de ressources reçues. Ils peuvent rester dans l'arbre mais tous leurs descendants doivent le quitter.

```
Si N == T
   Garder T enfants sans descendants : envoyer N fois message TAD(0)
```

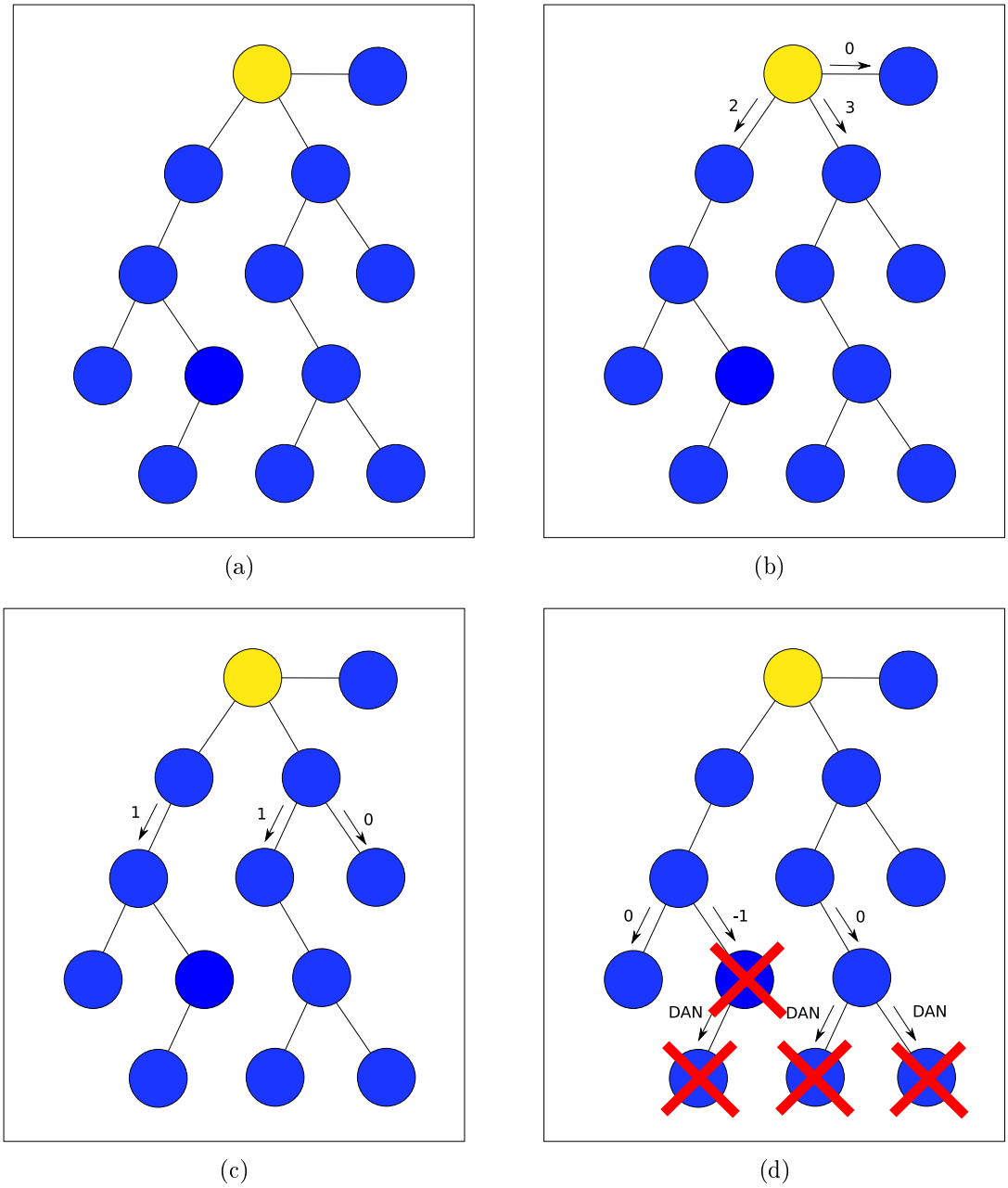
N < T Le nombre d'enfants est inférieur à la quantité d'agents permise. L'agent va devoir procéder à une répartition des ressources qui lui ont été confiées. Cette quantité prend déjà compte de l'existence de l'agent. La quantité reçue n'est valable que pour sa descendance. La méthode que nous avons choisie pour cette répartition est de trier les enfants par ordre décroissant de petits-enfants et de distribuer une par une les ressources aux enfants sauf à ceux qui n'ont aucune descendance. Comme décrit plus haut, quand un agent distribue, il prend en compte l'existence de ses enfants. Dans la table de voisinage, chaque entrée est munie d'un champ NbreRessourceADistribuer (nommée TS dans le pseudo-code suivant) prévu pour calculer la valeur du champ Paramètre du message TAD avant de l'envoyer.

```
Si N < T alors
   AD = T-N : quantité à distribuer en tenant compte des enfants
   Trier les enfants par ordre décroissant de petits-enfants
   Tant que AD > 0
        Si  petits-enfants > 0
            Incrémenter NbreRessourceADistribuer de l'enfant
            Décrémenter AD
   Pour  chaque enfant
        Envoyer un message TAD(NbreRessourceADistribuer)
```

Un exemple de limitation à la taille 9 est illustré à la figure 2.10.

2.7.4 Comptage et dissolution de groupe

Puisque la racine est supposée connaitre la taille minimum à atteindre pour le groupe, elle peut comparer la taille actuelle du groupe et la cible. Si elle le décide, elle peut, par des messages DAN, faire quitter tous les agents de l'arbre.



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

FIG. 2.10 – Exemple de limitation pour une taille maximale de 9 éléments. (a) Groupe en structure en arbre. (b) Répartition de la racine. (c) Répartition de la première génération. (d) Répartition de la deuxième génération.

Les expériences d'agrégation menées auparavant montrent que quand il y a compétition entre groupes, il faut une loi au minimum quadratique de probabilité de séparation de groupe pour qu'un groupe l'emporte. Quand la racine connait la taille du groupe, elle peut calculer le rapport taille observée sur limite inférieure et a une représentation de l'avancement de la

création du groupe. S'il le groupe est trop petit, la probabilité pour que la racine disolve le groupe est élevée. Mais au plus le groupe est grand, au plus cette probabilité diminue pour s'annuler quand le groupe a atteint la taille minimale acceptée.

La fonction de calcul de la probabilité pour dissoudre un groupe se fait comme suit.

$$\theta = \frac{taille}{cible}$$

$$\varphi = P(\frac{0}{cible})$$

$$P(\text{dissoudre groupe}) = -\frac{\varphi}{2} * \theta^2 - \frac{\varphi}{2} * \theta + \varphi$$

La figure 2.11 montre le graphique de cette probabilité pour un φ valant 0.2.

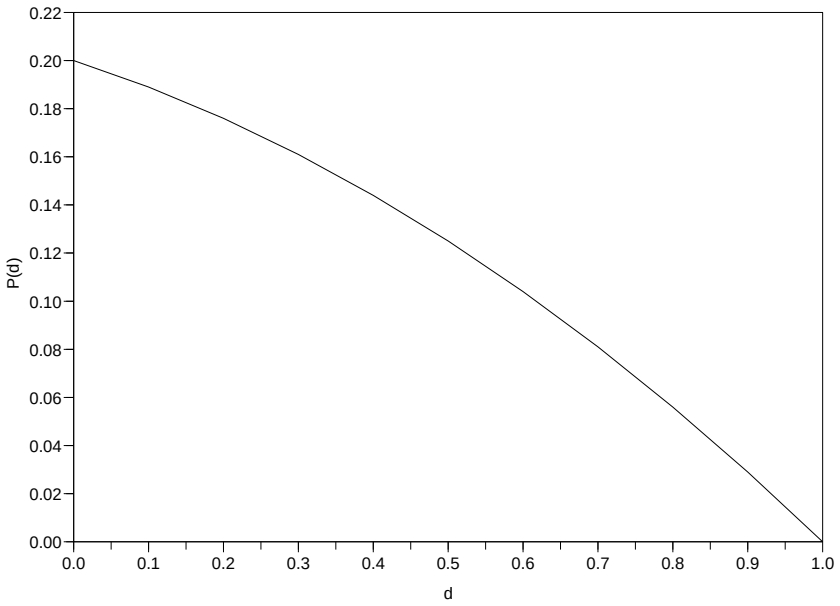


FIG. 2.11 – Probabilité de dissoudre un groupe en fonction du rapport entre la taille du groupe et la cible (φ=0.2)

Chapitre 3

Etude des résultats

Ce chapitre présentera l'environnement d'étude de l'algorithme ainsi que les résultats des expériences menées. Des propriétés intrinseques à l'algorithme seront de même exposées comme le phénomène de Load Balancing. Des paramères ont du être ajoutés lors de l'implémentation de l'algorithme. Ces paramètres seront décrits ainsi que l'effet de leur modification sur le comporetment du système. Par une série d'expériences, nous allons montrer à quel niveau la méthode est robuste et scalable. Pour la scalabilité du système, quelques tailles de populations et de groupes seront étudiées et comparées. Pour la robustesse, la désactivation d'un robot d'un groupe formé montrera comment le système réagit.

3.1 Description de l'environnement

3.1.1 Les agents

Développés par l'EPFL (Ecole Polytechnique Fédérale de Lausanne), les E-pucks sont des robots simples mais complets. Ils sont cylindriques et de petite taille. Ces robots ont été utilisés dans le cadre du Projet E-puck à l'IRIDIA duquel fait partie ce mémoire. Ils peuvent se déplacer aisément grace à deux roues motrices et possèdent huit senseurs de proximité (IR), permettant aussi la communication par infra-rouges. Voir [4] pour une description détaillée des E-pucks. Ces robots conviennent parfaitement à l'application de l'algorithme.

3.1.2 Le simulateur

Les expériences effectuées n'ont pas été menées sur les robots. C'est dans le simulateur Twodeepuck développé dans le cadre du Projet E-puck qu'elles ont été faites. Le simulateur, à l'utilisation intuitive, reproduit le comportement des E-pucks, avec entre autres, les moteurs et roues, les senseurs de proximité et le contrôleur.

Classes supplémentaires

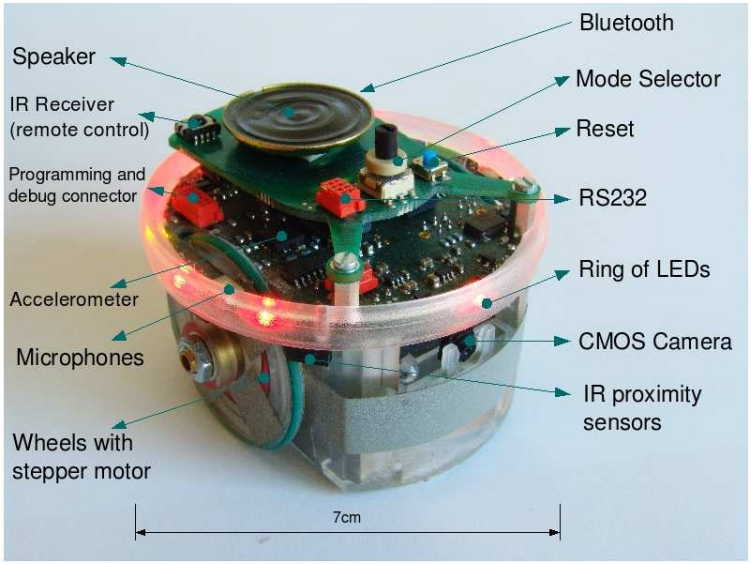


FIG. 3.1 – Vue générale d'un robot E-PUCK et mise en évidence des différents composants embarqués.

Le simulateur TWODEEPUCK implémente les classes nécessaires à de simples expériences comme l'agrégation. Pour tester notre algorithme, il a fallu implémenter quelques classes supplémentaires pour la communication et pour la gestion de la table de voisinage.
Comme décrit plus haut, la communication a été réduite à son plus simple modèle. Si un robot est dans le rayon d'émission d'un autre robot émettant, alors le premier reçoit le message. La classe MESSAGEACTUATOR sert à émettre un message tandis que la classe MESSAGESENSOR sert à lire les messages émis. Cette dernière classe met les messages reçus dans une liste accessible par le contrôleur via quelques méthodes spécifiques.
Pour obtenir un système causal, il a fallu insérer un champ supplémentaire dans les messages, en l'occurence, le numéro du pas d'émission. Pour qu'un robot reçoive un message, il faut désormais que le numéro du pas de réception soit supérieur à celui d'émission. Lorsqu'un robot est dans le rayon d'émission de plusieurs robots, il reçoit tous les messages mais une méthode de la classe de réception permet de choisir aléatoirement un de ces messages pour le traiter.
La classe NEIGHBOUR sert à la gestion de table de voisinage. Elle implémente des méthodes comme l'ajout de voisin, la modification des champs d'un voisin, la vérification de l'existence des voisins.
Nous avons aussi ajouté une classe MESSAGEQUEUE qui permet de gérér une file d'attente pour l'émission de messages. Ainsi lorsque un robot veut envoyer un message, il le place dans cette file d'attente et c'est le contrôleur qui à chaque pas d'éxécution envoye le premier message de la file d'attente et le supprime de la liste.

La figure 3.2 montre un rendu du simulateur lorsqu'un groupe de dix robots est formé. Les liaisons sont représentées par les lignes rouge et noir. La racine est l'E-PUCK mauve. Les

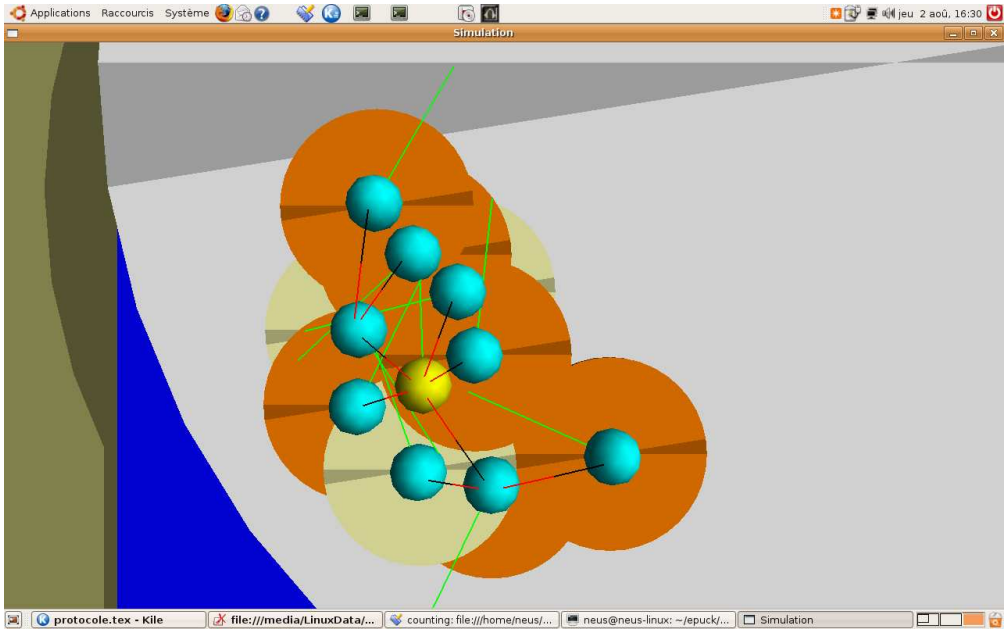


FIG. 3.2 – Arbre formé dans le simulateur

disques autour des robots représentent les messages en émission.

3.1.3 Le contrôleur

Nous allons ici décrire les modalités d'implémentation de notre algorithme, comme l'ajout de compteur pour relaxer les transitions entre états, l'ajout de tests supplémentaires,...

Un pas d'éxécution

A chaque pas d'éxécution, le contrôleur éxécute la liste suivante de traitements.

Réception des messages

Le MESSAGESENSOR lit tous les messages destinés au robot et les place dans une liste accessible par le contrôleur.

Vérification du message en émission

Les messages se voient dotés d'un temps d'émission exprimé en nombre de pas pendant lesquels le message est émis. Si le temps de vie du message en émission est dépassé, il faut arreter de l'émettre.

Gestion des voisins

Pour chaque message reçu, la source du message est ajoutée ou mise à jour dans la table de voisinage. De plus si le message est une commande IAS, alors le rôle CHILD est attribué au voisin. Un temps de vie est défini pour les entrées de la table de voisinage. Si ce temps de vie est dépassé sans réception de message, le voisin est retiré de la table.

Gestion des compteurs

Pour passer de l'état EXPLORE à l'état NEIGHBOUR, l'algorithme spécifie qu'il suffit de recevoir un message. Cela a pour conséquence que les robots s'arrêtent à une distance égale au rayon d'émission des robots, ce qui les empêche de former des groupes denses. Pour éviter ce problème, nous avons implémenté un compteur (appelé ToNeighbourCounter) qui compte le nombre de pas consécutifs pendant lesquels un message est reçu. Si ce compteur dépasse une valeur définie, le robot peut passer dans l'état NEIGHBOUR.
Pour relaxer encore plus le passage de l'état EXPLORE à l'état NEIGHBOUR, un compteur supplémentaire (appelé ToNeighbourRelaxCounter) a été ajouté. Le compteur ToNeighbourCounter compte les pas consécutifs pendant lesquels un message est reçu. Si pendant un pas d'éxécution, aucun message n'est reçu, ce compteur est réinitialisé. Si rien n'est fait, le robot risque de ne jamais rentré dans l'état NEIGHBOUR dans le cas d'une réception sporadique de messages. Le compteur supplémentaire ToNeighbourRelaxCounter est un compteur de pas d'éxécution sans message reçu. Une fois qu'il a atteint une valeur définie, le compteur ToNeighbourCounter peut être réinitialisé.
Un compteur général supplémentaire (appelé NoMessageCounter) sert à réinitialiser l'état du robot dans le cas où il se passe une certaine quantité de pas d'éxécution sans réception de message. Ce compteur permet d'éviter que des robots dans un état autre que EXPLORE ne restent immobiles alors qu'ils sont seuls.

Traitement en fonction de l'état

Le contrôleur, en fonction de l'état, éxécute ce que les schémas du chapitre précédent décrivent, c'est-à-dire lecture des messages reçus, traitement, modification des variables d'état, placement des messages dans la file d'attente.

Envoi asynchrone de messages

Lorsque le traitement en fonction de l'état ne produit pas l'envoi de message, on peut profiter de la bande passante libre pour envoyer des messages de manière asynchrone, comme des commandes ALONE, DRP, IAS et ARID en fonction de l'état du robot.

Envoi du premier message de la file d'attente

Dans le cas ou l'émetteur est libre, il faut envoyer le premier message de la file d'attente et le retirer.

Envoi de messages

Il a été montré que les champs AgentDestinataire et AgentExclus des messages permettent de cibler les robots devant recevoir les messages. Dans le cas des messages DAN (la branche doit quitter l'arbre), seuls les enfants doivent recevoir ce message. Pour éviter d'envoyer autant de messages DAN qu'il n'y a d'enfants, le message est envoyé en broadcast mais seuls les enfants de la source, effectuant un test de parenté avec la source, traitent le message.

3.2 Propriétés

Nous allons décrire ici quelques propriétés que nous avons obervées durant les expériences.

3.2.1 Paramètres et influence des variations

Les paramètres de l'algorithme sont nombreux. Certains définissent l'application, comme la taille de l'arène ou le nombre de robots, d'autres influent sur le comportement général de l'expérience. Le flux d'un groupe, c'est-à-dire le nombre de robots qui entrent dans un groupe et ceux qui en sortent, ainsi que la mise à jour de la table de voisinage des robots sont des facteurs importants. L'influence des paramètres sur ceux-ci est prise en compte dans cette section.

Les paramètres de définition sont ceux listés ci-dessous.

1. Taille de l'arène
2. Nombre de robots dans l'arène
3. Intervalle de tailles valides des groupes : limite inférieure et supérieure

A partir de ces paramètres, il est possible de connaître l'intervalle de nombre de groupes valides dans l'arène.

– Nombre de groupes valides :

$$\{\frac{\text{Nombre robots}}{\text{Limite supérieure}}, \frac{\text{Nombre robots}}{\text{Limite inférieure}}\}$$

Voici la liste de paramètres d'influence.

Rayon d'émission

Le rayon d'émission est la distance à laquelle un robot peut recevoir un message. Dans le simulateur, la puissance d'émission est identique sur toute la surface définie par le rayon d'émission et centrée sur le robot.

Rayon d'émission élevé : Les robots se rencontrent à grande distance. La cohésion des groupes formés est donc faible.

Rayon d'émission petit : Les robots ont moins de chance de rencontrer d'autres robots. La dynamique du système en est donc affectée et les temps de formation seront donc plus élevés.

Compteur EXPLORETONEIGHBOUR

Ce compteur, introduit à la section 3.1.3 , compte le nombre de messages reçus quand le robot est dans l'état EXPLORE. Quand celui-ci a atteint une certaine valeur, le robot passe à l'état NEIGHBOUR. C'est la valeur de la limite qui est importante ici. Cette limite influe sur le flux des groupes.

Limite basse : Les robots rejetés par le processus de limitation passent à l'état EXPLORE. Une limite basse ne permettra pas à ceux-ci de quitter le groupe. En effet, ils passeront rapidement à l'état NEIGHBOUR pour être ensuite intégrés de nouveau à l'arbre du groupe. Ce qui diminue le flux de sortie du groupe. De plus, vu que les robots dans le voisinage du groupe valide sont plus souvent acceptés dans l'arbre, les mises à jour de la table de voisinage des robots du groupe risquent de ne pas être assez fréquentes.

Limite élevée : Les robots s'approchant d'un autre risquent de ne pas passer à l'état NEIGHBOUR. Ce qui ralentit la formation de groupes.

Compteur de relaxation EXPLORETONEIGHBOURRELAX

Ce compteur compte le nombre de pas d'éxécution sans message lorsque le compteur précédent est lancé. Quand celui-ci atteint une certaine valeur, le compteur EXPLORETONEIGHBOUR est réinitialisé. Il a pour but d'éviter une réinitialisation trop fréquente du compteur précédent, ce qui ralentirait la formation de groupes.

Limite basse : Les robots ne forment pas de groupe si les messages envoyés ne sont pas fréquents.

Limite élevée : Les robots peuvent profiter de messages reçus avant la rencontre effective avec un groupe.

Compteur NEIGHBOURTOEXPLORENOMESSAGES

Ce compteur compte le nombre de pas d'éxecution sans message lorsque le robots est dans l'état NEIGHBOUR. Il est réinitialisé dès qu'un message est reçu. Si celui-ci atteint une certaine valeur, le robot retourne dans l'état EXPLORE.

Limite élevée : Les robots risquent de rester immobiles et seuls plutôt que d'essayer de rentrer dans un groupe.

Limite basse : Les robots risquent de quitter un groupe alors que leur présence dans celui-ci est justifiée.

Compteur NeighbourToExploreNoTree

Ce compteur compte le pas d'éxécution lorsque le robot est dans l'état Neighbour. Si ce compteur atteint une certaine limite, le robot retourne à l'état Explore. Ce compteur sert à empêcher les robots formant un voisinage sans créer de groupe de rester immobiles lorsqu'aucune racine n'apparaît.

limite élevée : Les robots forment un groupe mais sans structure en arbre.
Limite basse : Une racine n'a pas le temps d'apparaître dans un groupe.

Compteur RootInTreeToExplore

Ce compteur compte le nombre de pas d'éxécution sans message lorsuqe le robot est dans l'état Root ou InTree. Si celui-ci atteint une certaine valeur, le robot retourne dans l'état Explore.

Limite basse : Les robots utiles quittent le groupe alors que leur présence est justifiée.
Limite élevée : Les robots risquent de rester dans un état qui les rend immobiles alors qu'ils pourraient essayer de rejoindre un groupe.

Temps de vie des voisins : NeighbourTimeLife

Pour chacun des voisins, un compteur compte le nombre de pas d'éxécution sans message. Si celui-ci atteint une certaine valeur, le voisin est retiré de la table de voisinage.

Limite basse : Le voisin est retiré alors qu'il est toujours présent. Ce qui a pour conséquence d'empêcher le processus de limitation et de comptage de fonctionner correctement.
Limite élevée : Le comptage et la limitation se basent sur des informations obsolètes.

Temps de vie des enfants : ChildrenTimeLife

Le calcul de répartition lors du processus de limitation se base sur le nombre d'enfants du robot. Si celui-ci n'est pas correct, une limitation erronée peut avoir lieu. C'est pourquoi les voisins dont le rôle est Child ont un temps de vie spécifique. Pour chacun des voisins enfants, un compteur compte le nombre de pas d'éxécution sans réception de message IAS. Si ce compteur atteint une certaine valeur, le voisin est retiré de la table de voisinage.

Limite élevée : Table de voisinage obsolète.
Limite basse : Mises à jour de la table de voisinage très fréquentes.

Temps de vie des robots dans l'arbre : InTreeTimeLife

Dans le cas d'apparition d'une boucle dans la structure en arbre, si rien n'est fait, l'arbre peut perdurer sans contrôle de taille, même si la racine est retirée de l'arbre. C'est pourquoi nous avons décidé d'utiliser les message TAD comme preuve d'existence de la racine dans un arbre. En effet, seule la racine, qui déclenche le processus de limitation, peut décider d'émettre des message TAD dans l'arbre. Si un robot dans un arbre, ne reçoit pas de message TAD, c'est que la racine n'existe plus ou qu'il y a une boucle dans l'arbre. Le compteur InTreeTimeLife compte le nombre de pas d'éxécution sans réception de message TAD. Si celui-ci atteint une certaine valeur, le robot ainsi que ses descendants quittent l'arbre.

Limite basse : Les robots quittent l'arbre alors que leur présence est justifiée.
Limite élevée : Risque d'apparition de cycle sans réaction.

Probabilité NeighbourToRoot

Quand un robot est dans l'état Neighbour, la section 2.7 montre que le robot a une certaine probabilité de devenir racine. C'est de cette probabilité qu'il s'agit ici.

Probabilité basse : Formation rare de groupes.
Probabilité élevée : Formation fréquente de groupes mais risque plus élevé d'apparition de racines multiples.

Probabilité de limitation : Limitation

La racine d'un arbre a une certaine probabilité de déclencher le processus de limitation. C'est de cette probabilité qu'il s'agit ici.

Probabilité élevée : L'arbre est saturé de messages TAD. Moins de bande passante allouée au reste de messages.
Probabilité basse : Contrôle moins fréquent de la taille de groupe. Risque de départ de robots dont la présence est justifiée.

Paramètres utilisés lors des expériences

Voici un tableau récapitulatif des paramètres décrits ci-dessus que nous avons utilisés lors des expériences menées.

Rayon d'émission	:	0.155 m
ExploreToNeighbour	:	25 pas
ExploreToNeighbourRelax	:	5 pas
NeighbourToExploreNoMessages	:	125 pas
NeighbourToExploreNoTree	:	50 pas
RootInTreeToExplore	:	200 pas
NeighbourTimeLife	:	5 pas
ChildrenTimeLife	:	100 pas
InTreeTimeLife	:	250 pas
NeighbourToRoot	:	0.005 par pas
Limitation	:	0.05 par pas

3.2.2 Connaissance de l'environnement

Les robots ne doivent pas avoir une connaissance globale mais locale de leur environnement. Ils ne doivent avoir connaissance que de l'existence des voisins dans l'arbre, en particulier les voisins avec lesquels ils sont en relation. Le nombre de petits-enfants aussi est une connaissance qu'ils doivent avoir pour pouvoir effectuer une répartition correcte lors de la limitation de la taille de groupe.

3.2.3 Load Balancing

Le Load balancing (équilibrage de la charge en français) est un phénomène inhérent à la répartition (voir section 2.6.2).
Envisageons un cas simple comme décrit à la figure 3.3. On voit trois branches partant de la racine. Deux enfants de la racine ont deux enfants mais leurs branches ne comptent que trois agents. Tandis que la branche du milieu compte 1002 agents. Il y a donc 1009 agents dans ce cas. La racine sait que le groupe ne doit pas dépasser 1000 éléments. La taille du groupe est trop grande et certains agents devront quitter le groupe.
La racine envoye à chacun de ses enfants un message TAD dont le champ Paramètre a la valeur 333 vu que ses enfants comptent deux ou un petits-enfants. Les branches aux extrémités vont garder tous leur descendants mais la branche du milieu va devoir rejeter 701 agents. On peut observer que le load balancing qui tend à équilibrer la quantité d'agents dans chaque branche produit un rejet d'agents trop important. Après la limitation, il ne restera que 340 agents dans le groupe, ce qui est bien inférieur à la limite maximale ciblée.

Le phénomène de load balancing pourrait etre considéré comme un inconvénient mais il a ses avantages aussi. Le load balancing force les groupes à maintenir une certaine cohésion entre les agents du groupe, ce qui ralentit la formation d'un groupe de taille désirée.
Pour résoudre le problème posé par le phénomène de load balancing, une répartition basée sur le nombre d'éléments par branche peut être envisagé.

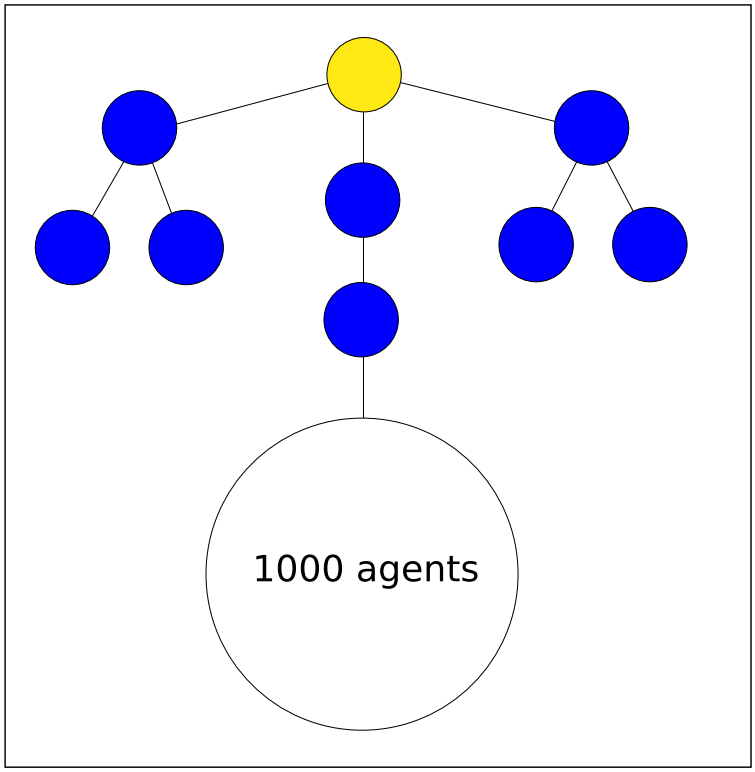


FIG. 3.3 – Load Balancing

3.2.4 Méthode sans chef

Le comportement identique de tous les robots nous assure qu'aucun robot n'est désigné comme chef de groupe. Tous les robots peuvent devenir temporairement chef de groupe, c'est-a-dire la racine de l'arbre formé au sein du groupe. Lors d'une compétition entre arbres, l'arbre le plus âgé est choisi, l'autre arbre étant dissout.

3.3 Expériences et résultats

Cette section traitera des expériences menées dans le cadre de l'étude de la méthode. Les expériences y seront décrites, les résultats traduits et interprétés. L'étude du comportement est menée sur l'ensemble des expériences et sur les observations générales que celles-ci ont engendrées. L'étude de la dynamique permettra de trouver les composantes temporelles principales de la méthode et les paramètres qui l'influent. L'étude de la robustesse se fera par l'analyse de la faculté à la recomposition de groupe lors de retraits brusques d'un agent pendant l'expérience. La scalabilité sera étudiée en calculant les temps de création de groupe pour des valeurs variables du nombre de robots dans l'arène ainsi que les tailles de groupes ciblées. Toutes les expériences ont été effectuées dans une arène ronde dont le diamètre peut être défini.

3.3.1 Dynamique du système

La dynamique du système représente les caractéristiques temporelles principales. Comme elle dépend du déplacement des agents, de la vitesse de la communication et de l'algorithme lui-même, les caractéristiques temporelles de la méthode pourraient varier du tout au rien. Comme la méthode ne definit aucune constante de temps, nos expériences sont basées sur le nombre de pas d'éxécution, la base de temps indivisible du système.

Les expériences menées sur la dynamique du système ont été faites à nombre de robots dans l'arène constant et taille de groupe désirée constante. Les deux méthodes de contrôle permettent de cibler un intervalle de taille valide. En égalisant la limite supérieure et la limite inférieure, cet intervalle est vide et seuls les groupes dont la taille vaut celle désirée seront valides. Le rapport entre le nombre de robots et la taille de groupe désirée est constant et vaut trois-demi. Ce rapport nous permet d'étudier l'existence d'un seul groupe viable à chaque moment. Ce groupe n'existera pas à tout moment mais peut apparaître et se dissoudre. Ce sont les temps de création et les temps de vie du groupe qui sont mesurés.

Les paramètres de l'expériences sont :

Taille de l'arène	:	1 mètre de rayon
Nombre de robots	:	15
Taille de groupe*valide*	:	10
Répétitions	:	500

La figure 3.4 montre respectivement l'histogramme des temps de création du groupe et l'histogramme des temps de vie du groupe.

L'histogramme des temps de création du groupe montre qu'une majorité des groupes se forment entre quarante et soixante pas d'éxécution. Cela s'explique par le rejet d'un agent dû au load balancing et puis à son insertion dans l'arbre dû à sa proximité du groupe.
L'histogramme des temps de vie du groupe montrent deux pics. Le premier entre zéro et dix, le deuxième entre deux cent quarante et deux cent soixante. Le premier pic ainsi le pic du temps de création montrent que le système oscille rapidement. Cela est dû au load balancing. En effet, un groupe dont la taille est une unité en dessous de la taille désirée est formé. Un agent s'en approche et est accepté dans le groupe. Le groupe a donc la taille désirée mais la limitation le rejette. Malgré son rejet, l'agent est encore à proximité du groupe et se fait rapidement reinséré dans l'arbre.

3.3.2 Robustesse

La robustesse d'un système représente sa capacité à réagir lors d'une perturbation. Nous allons simuler cette perturbation en désactivant un robot de certaine génération lorsqu'un groupe est formé. La désactivation d'un robot se fait en l'immobilisant et en l'empêchent

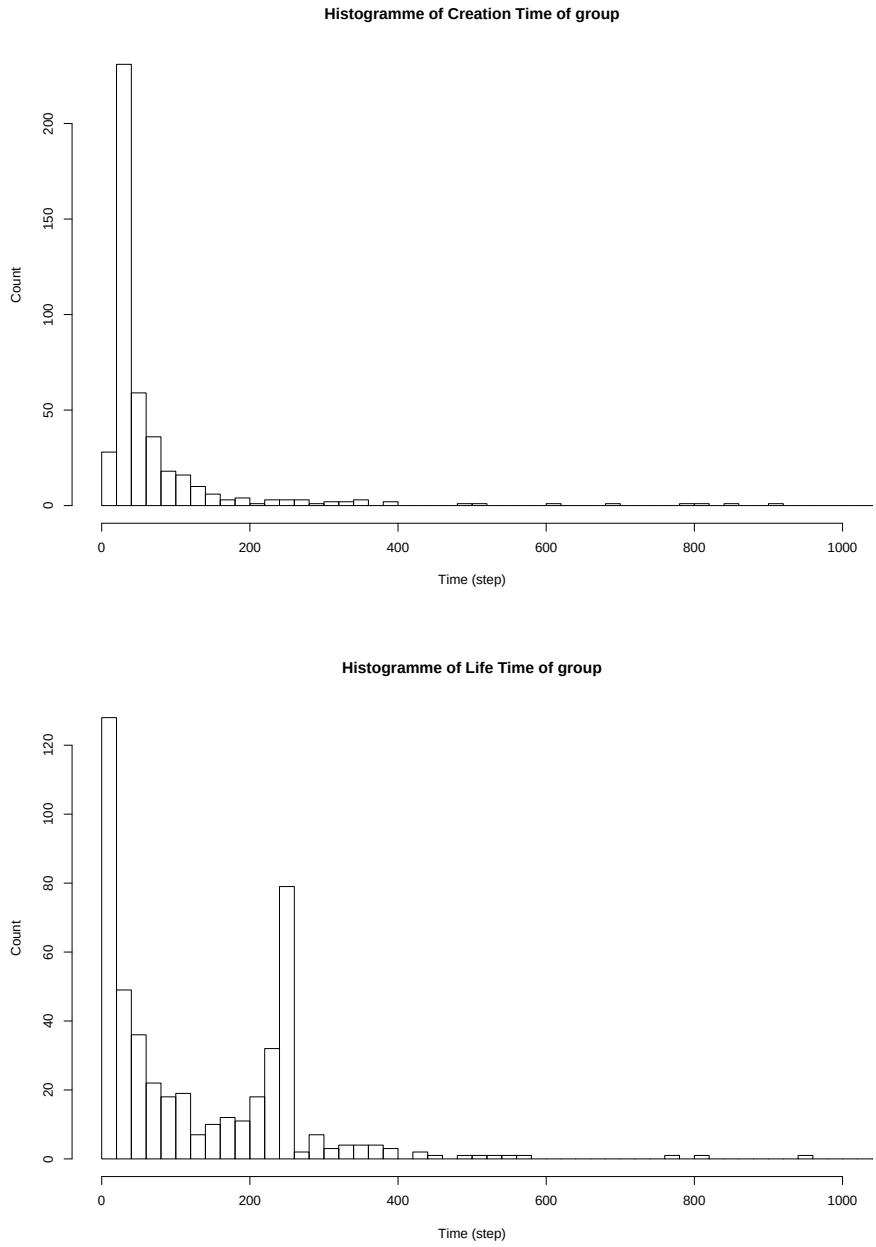


FIG. 3.4 – (a) Temps de creation d'un groupe. - (b) Temps de vie d'un groupe.

d'émettre des messages. Le robot est toujours dans l'arène mais ne réagit plus. Nous avons étudié le temps de reformation de groupe après la désactivation d'un robot choisi aléatoirement dans le groupe, de la racine, d'un robot de première génération et d'un robot de deuxième génération dans un groupe de dix robots.

Voici les détails des expériences menées pour le robustesse.

Taille de l'arène	:	1 mètre de rayon
Nombre de robots	:	15
Taille de groupe désirée	:	10
Répétitions	:	500 par génération

Dès qu'un groupe est formé, un robot est désactivé. Premièrement, un robot choisi aléatoirement est désactivé. Le temps pour que le groupe se reforme est mesuré. La figure 3.5 représente l'histogramme des temps mesurés. Celui-ci montre que le groupe se reforme assez rapidement. En moyenne, le groupe met 17418 pas d'éxécution pour se reformer. En effet, quelques unes des expériences ont montré un temps de reformation beaucoup plus élevé que les autres. Celles-ci ne sont pas montrées sur l'histogramme.

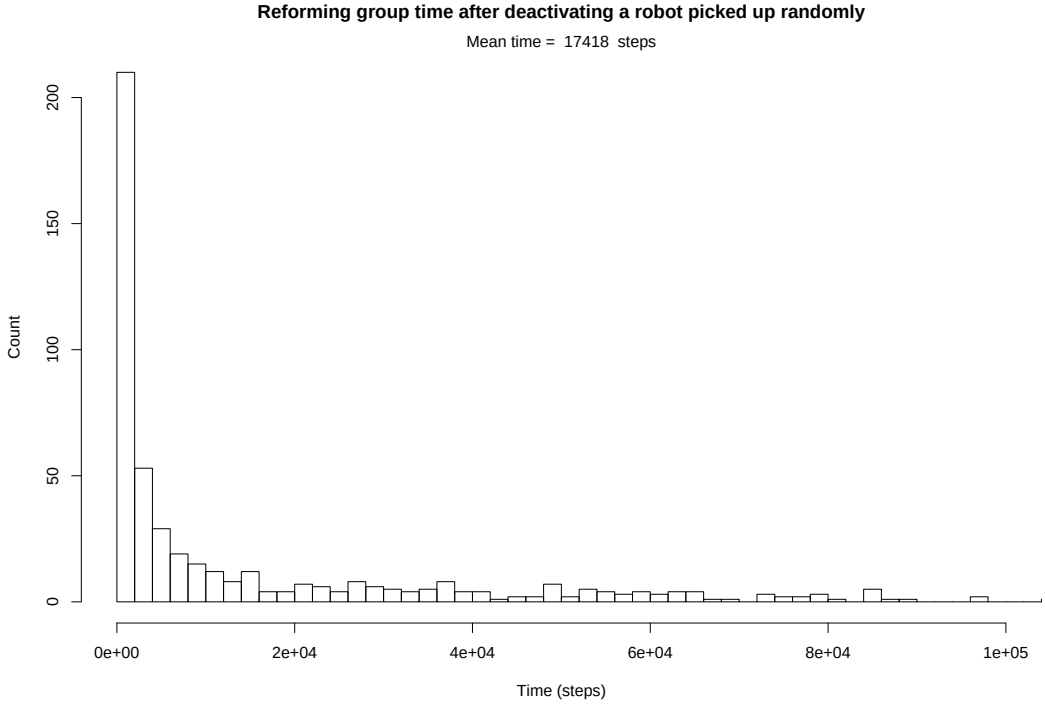


FIG. 3.5 – Temps de reformation de groupe après avoir désactivé un robot choisi aléatoirement dans le groupe.

Les résultats suivants proviennent d'expérience où un robot, dont la génération est connue, est désactivé. Comme ci-dessus, le temps que le groupe met pour se reformer est mesuré. Cinq cents répétitions ont été effectuées pour chacune des génération, depuis la racine de génération zéro aux robots de deuxième génération. La figure 3.6 montre les histogrammes des trois générations. Les conditions d'expérimentation sont identiques à celles utilisées pour l'étude de la désactivation d'un robot choisi aléatoirement dans le groupe.

Les histogrammes montrent que la désactivation d'une génération proche de la racine tend à augmenter le temps de reformation du groupe. En effet, lorsque la racine est désactivée, tous les robots quittent l'arbre dès qu'ils sont conscients que la racine n'est plus active. Ils leur faut donc plus de temps pour reformer un groupe de taille voulue. Lorsque c'est un robot de génération éloigné de la racine qui est désactivé, ce ne sont seulement que les descendants de ce robot qui quittent l'arbre. Mais ils sont rapidement insérés dans l'arbre dès qu'ils rencontrent les robots encore dans celui-ci.

La figure 3.7 montre la relation entre le temps moyen de reformation du groupe en fonction de la génération désactivée. Elle montre clairement que plus la génération désactivée est élevée, donc éloignée de la racine, plus le temps de reformation du groupe diminue.

3.3.3 Scalabilité

Cette section traite de l'étude de la scalabilité du système. Des mesures de formation de groupes de différentes tailles seront effectuées tout en gardant un rapport constant entre le nombre de robots dans l'arène et le nombre de robots dans un groupe valide. Ces expériences ne sont pas identiques à celle effectuée lors de l'étude de la dynamique du système. En effet, ici seul le temps de formation du premier groupe importe, c'est-à-dire le temps pour que les robots forment un groupe de la taille voulue à partir de leur placement aléatoire dans l'arène.

Les tailles considérées varient de six à vingt-et-un robots dans l'arène pour des groupes valides de quatre à quatorze robots. Les expériences ont été effectuées cinq cents fois par taille pour une densité constante. La densité est le rapport entre le nombre de robots dans l'arène et la surface de l'arène. Voici les tailles d'arène ronde que nous avons utilisées pour les expériences.

Robots dans l'arène	Taille des groupes	Rayon de l'arène (mètres)
6	4	0.632
9	6	0.774
12	8	0.894
15	10	1
18	12	1.095
21	14	1.183

La figure 3.8 montre la relation entre le temps moyen de formation du premier groupe d'un expérience et la taille de ce groupe à densité constante.

On voit que le temps de formation du groupe est proportionnel à la taille du groupe. En effet, il faut que plusieurs robots se rencontrent pour former un groupe. Quand un groupe est formé mais n'a pas la taille désirée, au plus cette taille est grande au plus il faut que des robots de l'arène s'incorporent dans le groupe.

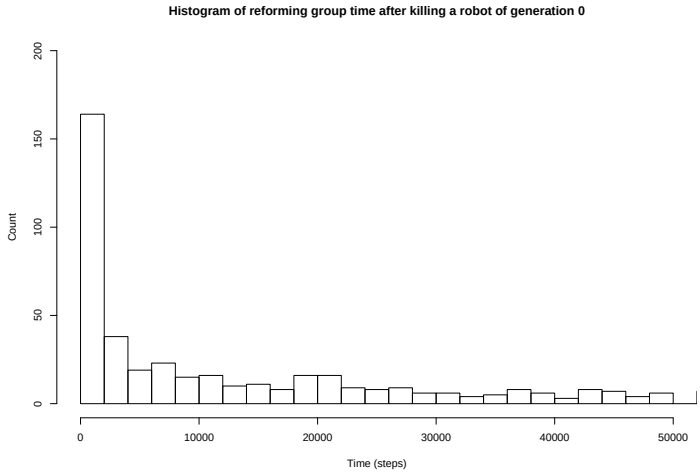
3.4 Perspectives

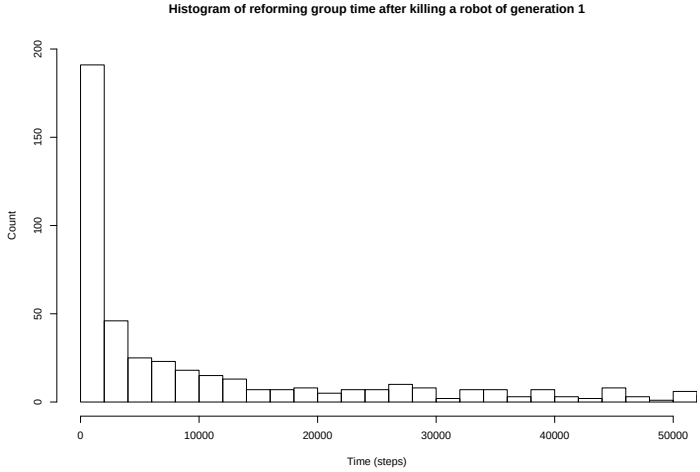
Nous avons proposé une méthode de contrôle de taille de groupe se basant sur la théorie des arbres. Le phénomène de load balancing a été introduit et une solution proposée que les personnes intéressées pourront étudier à l'aide du simulateur TWODEEPUCK.

Une application de cette méthode peut être envisagée dans le cadre des systèmes multi-tâches. En effet, en considérant une tâche complexe composée de tâches plus simples requiérant un certain nombre d'agents, la méthode permet de former des groupes dont la taille est comprise dans un intervalle fixé. La méthode ne peut par contre pas calculer le nombre d'agents requis par tâche.

Une étude plus approfondie de la méthode peut être par ailleurs réalisée en considérant des tailles plus grandes. L'algorithme conçu introduit un grand nombre de paramètres. Une étude détaillée de ces paramètres permettrait d'optimiser ceux-ci et d'améliorer le comportement des robots.

Ensuite, une comparaison avec la méthode de C. Melhuish [6] peut être effectuée en considérant la dynamique, la scalabilité ainsi que la robustesse du système.





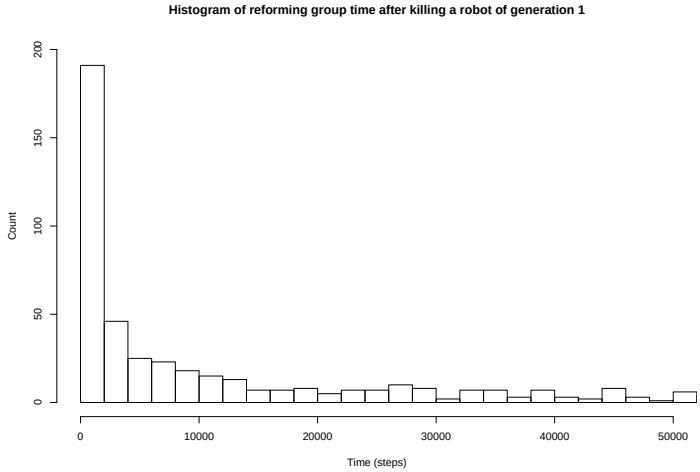


FIG. 3.6 – (a) Temps de reformation de groupe après avoir désactivé la racine du groupe. - (b) Temps de reformation de groupe après avoir désactivé un robot de première génération.- (c) Temps de reformation de groupe après avoir désactivé un robot de deuxième génération.

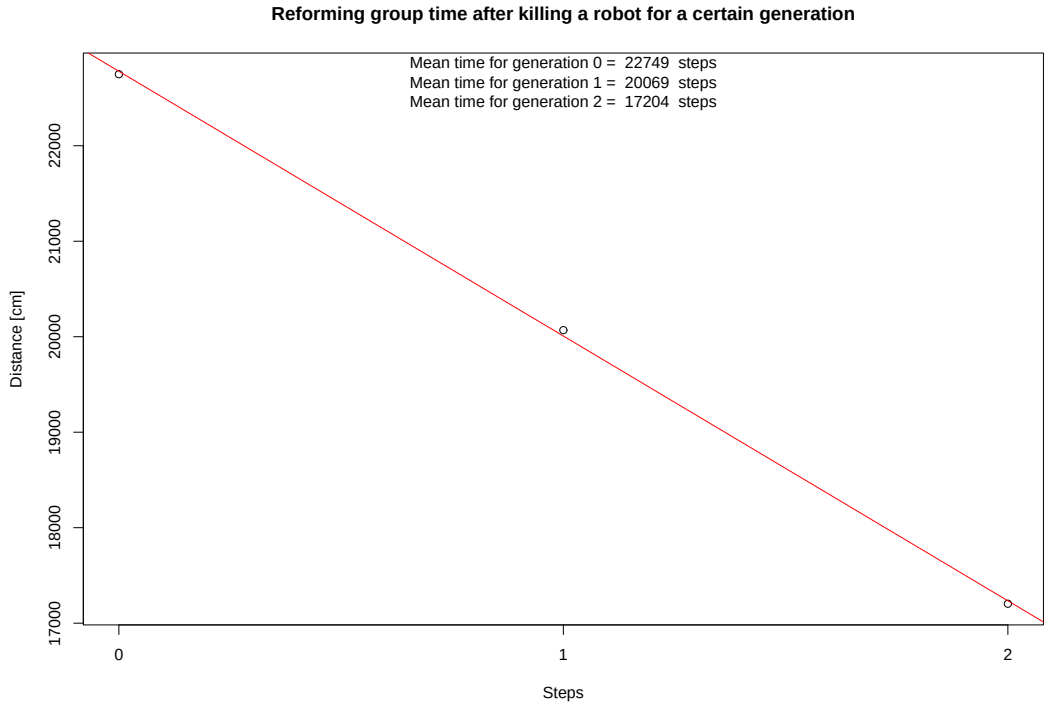


FIG. 3.7 – Temps de reformation de groupe en fonction de la génération désactivée

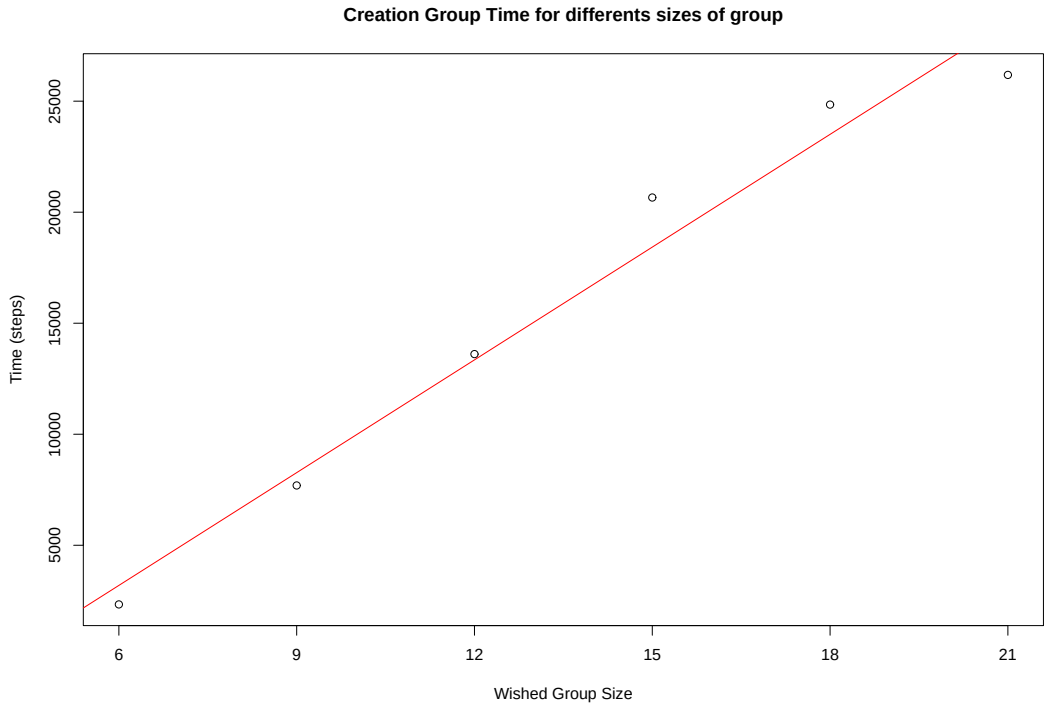


FIG. 3.8 – Temps moyen de formation de groupe en fonction de la taille

Chapitre 4

Conclusions

Nous avons présenté une méthode de contrôle de la taille de groupe en robotique en essaim s'inspirant de la communication entre ordinateurs et de la théorie des arbres.

La première étape a été de concevoir cette méthode. Pour cela, nous avons d'abord conçu un protocole de communication par messages structurés entre les agents concernés. Ensuite, nous avons mis en place un algorithme qui implémente ce protocole. Le protocole conçu est divisé en deux parties.

La première partie de ce protocole, appelée protocole TSP, sert à la création d'une structure virtuelle en arbre au sein d'un groupe. Cette structure est modélisée par des relations fictives entre robots connues par ceux-ci au niveau local. Cette connaissance au niveau local assure un niveau minimal de scalabilité de la méthode. De plus, l'apparition sporadique d'une racine couplée à une implémentation identique du contrôleur sur tous les robots permet d'affirmer qu'il n'existe aucun chef au sein de la population. La maintenance de la structure arborescente est assurée par la vérification de proximité des voisins. L'apparition d'arbres différents dans un même voisinage posant problème, le protocole TSP inclut une reconnaissance de racines multiples et un mécanisme de choix de l'arbre le plus âgé.

La deuxième partie de ce protocole, appelé protocole CMP, assure un contrôle de la taille du groupe. Profitant de la structure arborescente générée grâce au protocole TSP, le protocole CMP dans un premier temps limite le nombre de robots dans le groupe à une valeur maximale par répartition de celle-ci entre chacune des branches issues de la racine. Dans un deuxième temps, il s'assure que la taille de ce groupe n'est pas inférieure à une limite fixée en déclenchant un processus de comptage suivi d'une prise de décision concernant la dispersion des robots. Ce protocole permet donc de fixer un intervalle de tailles de groupe valides. En égalisant la limite supérieure et la limite inférieure, le protocole ne permet qu'aux groupes de taille unique de se maintenir et il est donc bien question de contrôle de taille de groupe.

L'algorithme conçu, basé sur le protocole, a été implémenté dans le simulateur TWO-DEEPUCK. Celui-ci nous a permis d'effectuer un grand nombre de répétitions d'expériences ciblées mettant en évidence le comportement collectif des robots. Les paramètres de définition et d'influence permettent d'élargir l'horizon d'application de la méthode. La cohésion des groupes ainsi que le flux d'entrée et de sortie des robots dans le groupe ont été déduits du fonctionnement intrinsèque de l'algorithme. De plus, un phénomène émergeant appelé Load Balancing a pu être observé et une solution aux inconvénients de celui-ci a été proposée. La dynamique du système, indépendante de toute base temporelle, a été étudiée et les résultats montrent que les groupes se forment assez rapidement malgré un manque relatif de stabilité de ceux-ci. L'étude de la robustesse de la méthode montre qu'un groupe se reforme rapidement malgré le retrait brusque d'un robot appartenant à celui-ci. Le choix de ce robot désactivé est d'abord aléatoire, puis nous avons ciblé celui-ci en fonction de sa génération dans l'arbre. L'étude de la scalabilité montre que le temps de formation est proportionnel à la taille de groupe désirée.

Bibliographie

[1] Melhuish C., Holland O., and Hoddell S. Convoying : using chorusing to form travelling groups of minimal agents. *Robotics and Autonomous Systems*, (28), 1999.

[2] Sumpter D. The principles of collective animal behaviour. In *Philosophical transactions-Royal Society of London. Biological sciences*, pages 5–22, London, UK, 2006. Royal Society of London.

[3] Bury L. Conception et implémentation en c++ d'un simulateur pour les robots e-puck et réalisation de tests de validation pour la cinématique de base. Master's thesis, ULB - Université Libre de Bruxelles, 2007.

[4] Dedriche O. La prise de décision au sein d'un groupe de robots : Conception et développement d'une plateforme de travail libre et gratuite pour les e-puck robots à destination de la communauté académique, et étude d'un comportement collectif auto-organisé via une tâche d'agrégation en robotique en essaim. Master's thesis, ULB - Université Libre de Bruxelles, 2007.

[5] Holland O. and Melhuish C. Stigmergy, self-organisation, and sorting in collective robotics. *Artificial Life*, 5, No 2 :173–202, 1999.

[6] Holland O., Melhuish C., and Hoddell S. Chorusing and controlled clustering for minimal mobile agents.

[7] Jeanson R., Rivault C., Deneubourg J. L., Blanco S., Fournier R., Jost C., and Theraulaz G. Self-organized aggregation in coakcroaches. *Animal Behaviour*, (69) :173–202, 2005.

[8] R Development Core Team. *R : A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2005. ISBN 3-900051-07-0.

[9] Garnier S., Jost C., Jeanson R., Gautrais J., Asadpour M., Caprari G., and Theraulaz G. Collective decision-making by a group of coackcroach-like robots.

[10] Garnier S., Jost C., Jeanson R., Gautrais J., Asadpour M., Caprari G., and Theraulaz G. Aggregation behaviour as a source of collective decision in a group of cockroach-like-robots. *Proceeding of the 8th European Conference on Artificial Life*, 3630, 2005.