UNIVERSITÉ DU QUÉBEC EN OUTAOUAIS

PROBLÈME DE COMMÉRAGE POUR LES AGENTS MOBILES

MÉMOIRE
PRÉSENTÉ
COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN INFORMATIQUE

PAR

JULIAN ANAYA

AVRIL 2014

UNIVERSITÉ DU QUÉBEC EN OUTAOUAIS

Département d'informatique et d'ingénierie

Ce mémoire intitulé :

PROBLÈME DE COMMÉRAGE POUR LES AGENTS MOBILES

présenté par
Julian Anaya

pour l'obtention du grade de maître ès science (M.Sc.)

a été évalué par un jury composé des personnes suivantes :

Dr. Jurek Czyzowicz ..... Directeur de recherche
Dr. Andrzej Pelc ..... Codirecteur de recherche
Dr. Karim El Guemhioui ..... Président du jury
Dr. Mohand S. Allili ..... Membre du jury

Mémoire accepté le : 4 Avril 2014

À ma femme. À ma famille, au Québec et en Colombie.

Table des matières

Liste des figures

Liste des tableaux

Liste des Algorithmes

Résumé

Ce mémoire étudie le problème de commérage par les agents mobiles. L'environnement dans lequel les agents opèrent est une droite (le cas de l'espace cartésien deux-dimensionnel est aussi discuté), chaque agent a une vue complète de la position des autres agents incluant la sienne et le système de coordonnées est commun à tous les agents. Les agents sur la ligne ont une séparation égale à 1. Chaque agent a une information initiale qui doit être communiquée à tout autre agent. La communication entre deux agents a lieu quand les agents se trouvent au même point de l'environnement. Le but de l'algorithme de commérage est de planifier les mouvements d'agents afin de minimiser la distance maximale parcourue. Dans ce travail on présente un algorithme général de commérage sur la ligne et on fait la preuve de l'optimalité de cet algorithme pour 3 et 4 agents. On donne aussi une borne inférieure pour n'importe quelle nombre d'agents. Un aperçu du problème avec des distances inégales entre les agents sur la ligne est illustré. Dans le cas des agents sur le plan, l'algorithme présenté est une 4-approximation de l'algorithme de commérage optimal. Un simulateur qui illustre le fonctionnement de l'algorithme sur la ligne est aussi inclus dans ce mémoire.

Abstract

This thesis studies the problem of gossiping by mobile agents. The environment in which the agents work is a line (the Cartesian space with two dimensions is also discussed). Every agent knows its own position and the position of the other agents, and the coordinate system is the same for all of them. The agents on the line have a separation equal to 1. Every agent has an initial piece of information that must be communicated to the others agents. The communication takes place when two agents are at the same location. The purpose of the gossip algorithm is to plan the movements of the agents to minimize the maximun travel distance. In this thesis we give a general gossiping algorithm on the line and we prove the optimality of this algorithm for 3 and 4 agents. We also give a lower bound on the cost of gossiping for and arbitrary number of agents. The case of different distances between the agents on the line is discussed as well. In the plane our algorithm is a 4-approximation of the optimal gossip algorithm. A simulator of a working algorithm in the line is also included.

Chapitre 1

Introduction

La communication est l'échange d'information entre un sujet émetteur et un sujet récepteur au moyen de signes. La communication d'aujourd'hui a un impact important dans notre vie quotidienne. Écouter la radio, regarder la télévision et/ou même utiliser Facebook, sont des exemples démontrant le problème de la communication omniprésent dans notre vie.

Les composants des réseaux actuels (commutateurs, routeurs, points d'accès, ordinateurs, mobiles) sont très puissants au moment de faire des calculs, mais le temps dédié à communiquer avec les autres éléments est habituellement supérieur au temps de calcul. Alors la construction des protocoles de communication efficients et optimaux est nécessaire.

L'un des problèmes fondamentaux du calcul distribué est la communication entre les processeurs éparpillés à travers le réseau. De cette façon, les efforts individuels de tous les processeurs peuvent être coordonnés afin de réaliser la tâche confiée à l'ensemble de processeurs. Habituellement, il y a trois types d'échanges de messages inter-processeurs : (1) *la diffusion* (ang. broadcast) où l'information possédée par un processeur doit être communiquée à tous les autres processeurs du réseau, (2) *le ramassage* (ang. convergecast) où l'union des informations provenant de tous les processeurs doit parvenir à un processeur, et (3) *le commérage* (ang. gossiping) où chaque processeur à la fin possède l'union des informations de tous les autres processeurs. Les messages individuels peuvent être transférés uniquement entre deux processeurs liés par un lien direct dans le réseau (c-à-d les voisins). Évidemment, pour que la communication soit réalisée de façon efficace, les processeurs qui collaborent profitent des possibilités offertes par le modèle. Par exemple, souvent les processeurs peuvent agréger dans un seul message pas seule-

ment leur information initiale mais l'ensemble des connaissances possédées au moment du transfert.

Beaucoup de recherches récentes dans le calcul distribué concernent les processeurs ayant la capacité de se promener à travers le réseau. De tels processeurs appelés *agents mobiles* (ou simplement agents) utilisent les mêmes liens du réseau pour leurs déplacements qui servaient comme des liens transférant les messages dans les problèmes de communication mentionnés ci-dessus. Les agents mobiles, à part de se déplacer, et de communiquer, possèdent souvent la capacité de percevoir les éléments de l'environnement de leur travail (en particulier ils peuvent détecter la présence d'autres agents), ainsi que la capacité de calculer et de mémoriser.

Les agents mobiles sont utilisés dans plusieurs problèmes du domaine du calcul distribué et de la robotique. Des centaines de papiers scientifiques sont consacrés au problème de l'exploration où l'ensemble des agents doit visiter tous les sommets d'un réseau afin de les connaitre où d'y trouver un intrus (statique ou mobile). Dans le cas de la recherche d'un intrus mobile, il s'agit d'un jeu de recherche. Les agents mobiles essayent de capturer l'intrus le plus vite possible et l'intrus, à son tour, tente de s'échapper le plus longtemps possible. Un autre problème impliquant les agents mobiles concerne la formation d'une configuration particulière, par exemple afin de réaliser une tâche nécessitant cette configuration au départ. La plus étudiée de cette catégorie est la question de rendez-vous, ou les deux (ou plusieurs) agents doivent se retrouver en même temps au même point du réseau.

Le sujet de ce mémoire se trouve à la frontière de deux domaines importants du calcul distribué : le problème de commérage et le problème de rendez-vous.

Dans le problème de commérage, les nœuds du réseau possèdent les informations qui doivent être communiquées aux autres nœuds. À la fin, chaque nœud du réseau doit connaître l'information initiale de chacun des autres nœuds. Les nœuds sont statiques (n'évoluent pas durant le temps) et le transfert d'information doit se faire selon un protocole bien précis. Souvent, le modèle est synchrone, où durant chaque ronde certains nœuds peuvent transférer les informations à leurs voisins dans le réseau. L'objectif est de réaliser le commérage efficacement (par exemple en utilisant le nombre de rondes le plus petit possible).

Le problème du rendez-vous concerne les agents mobiles, ayant la capacité de se promener à travers le réseau. Le but du problème est de faire rencontrer les agents quelque part sur leurs trajectoires, c'est-à-dire d'arriver en même temps au même point

du réseau. Habituellement, l'agent ne connaît pas la position de l'autre agent, et souvent, ils ne connaissent pas le réseau dans lequel ils opèrent. Cependant, les agents essayent de coordonner leurs efforts afin de se rencontrer le plus rapidement possible.

Dans ce texte, on étudie le problème de commérage par les agents mobiles. Les agents se promènent à travers le réseau et ils communiquent (c'est-à-dire, ils peuvent échanger l'information) dès qu'ils se retrouvent en même temps au même point du réseau.

Le modèle utilisé est : un ensemble de N agents mobiles placés sur une ligne. Chaque agent possède son information initiale. Les agents peuvent se déplacer sur la ligne en marchant dans une des deux directions. Si deux agents se trouvent en même temps au même point de la ligne (on dit qu'un rendez-vous est arrivé) ils peuvent échanger toute information qu'ils possèdent à ce moment, même celle obtenue précédemment via les autres agents. Les agents connaissent le nombre total d'agents localisés sur la ligne, ainsi que leurs positions. Chaque agent se déplace et échange les informations avec les agents rencontrés. La distance initiale entre les agents voisins est quelconque. Nous étudions plus particulièrement le cas des distances égales.

Le coût $c(a_i)$ représente la longueur de la trajectoire parcourue par l'agent a_i. Le coût de l'algorithme est $C = max\{c(a_1), c(a_2), c(a_3), ..., c(a_n)\}$, soit le coût maximal parcouru par les agents jusqu'à ce que le commérage soit effectué.

À chaque moment de l'algorithme de commérage, on dénote par $m(a_i)$ l'ensemble des informations initiales connues par l'agent a_i.

Le but des algorithmes est de transmettre le message de chaque agent à tous les autres agents en essayant de minimiser le coût C.

L'objectif de minimiser le coût maximal (c'est-à-dire la longueur maximale de trajectoire) parmi les agents de l'ensemble est dicté par les applications pratiques. On essaie de produire les masses d'agents très simples, peu chers, ayant des capacités très limitées. Par exemple, la diminution du coût maximal permet d'établir une borne sur la puissance de batterie que les agents doivent posséder afin de réaliser le commérage, puisque la distance traversée est proportionnelle à l'usage de la batterie.

D'autres mesures d'efficacité (par exemple le coût total correspondant à la somme des longueurs des trajectoires d'agents) peuvent aussi être prises en considération.

Chapitre 2

État des connaissances

2.1 Introduction

Le rendez-vous entre deux ou plusieurs agents est défini comme la rencontre dans une place spécifique en même temps. Le lieu pour la rencontre n'est pas choisi avant l'exécution de l'algorithme.

Le commérage entre les nœuds (les nœuds sont statiques) d'un réseau se fait par la transmission des messages. Les messages sont transmis par un nœud vers ses nœuds voisins. À la fin chaque nœud doit avoir les informations de tous les nœuds.

Ce travail de recherche consiste à utiliser les deux concepts ci-dessus pour accomplir le commérage entre les agents mobiles. Ce sujet n'a pas été traité auparavant.

La revue de la littérature est donc divisée en deux parties : celle concernant le rendez-vous des agents et celle concernant le commérage entre les nœuds statiques.

2.2 Rendez-vous aléatoire

2.2.1 Rendez-vous et regroupement dans les graphes

Le livre [1] est la monographie principale sur le rendez-vous, se concentrant surtout sur le cas aléatoire.

[1] donne comme exemple un anneau avec n sommets et arcs sans aucun sens d'orientation prédéfini. Les deux agents sont anonymes et sont placés dans une configuration symétrique, n est connu par les agents. Notez que ce type de configuration est impossible à résoudre avec des algorithmes déterministes. L'algorithme aléatoire est : l'agent lance

une pièce, si le résultat est face, alors il va parcourir l'anneau $\lceil n/2 \rceil$ sommets vers la droite, ou $\lceil n/2 \rceil$ sommets vers la gauche si le résultat est pile. La complexité est $O(n)$ et le rendez-vous sera effectué soit dans un sommet soit dans un arc.

Dans [21] on considère le rendez-vous asynchrone aléatoire des agents anonymes dans un graphe quelconque connexe. Le résultat principal est un algorithme qui réalise le rendez-vous avec probabilité 1, mais le coût de cet algorithme est exponentiel.

2.2.2 Rendez-vous sur le terrain

[1] présente le problème de rendez-vous comme une recherche qui est effectuée par deux ou plusieurs agents pour trouver les autres agents. [1] donne comme exemple de la vraie vie la recherche d'un compagnon du sexe opposé, dans les espèces avec une densité spatiale basse, ou le regroupement des animaux.

Avec la ligne comme environnement de rendez-vous pour deux agents, [1] présente un premier et simple algorithme qui permet aux deux agents le rendez-vous. Le modèle est : les deux agents sont placés sur une ligne avec une séparation $D = 2$ entre eux. $D = 2$ a été choisi par les auteurs pour simplifier le fonctionnement de leur algorithme. Les agents n'ont pas de système de coordonnées communes et la communication n'est pas possible entre les agents. Seulement la distance $D = 2$ entre eux est connue par les agents au début de l'algorithme. Il faut remarquer que cet algorithme est synchrone. Au début, chaque agent choisit une direction aléatoire et se déplace d'une distance 1. Par la suite, il se déplace d'une distance 2 dans la direction opposée. Répéter avec une direction aléatoire à chaque fois jusqu'à ce que le rendez-vous soit accompli. [1] montre que le cout de ce type de rendez-vous, symétrique, a une borne supérieure de 5, $R^s \leq 5$.

Un problème semblable au rendez-vous sur le terrain est présenté dans [21]. Les auteurs modélisent les agents comme des points dans le plan et chacun est équipé d'un compas. Les agents doivent s'approcher à une distance 1 l'un de l'autre. Les auteurs présentent un algorithme aléatoire pour accomplir l'approche avec probabilité 1. Le même algorithme peut-être utilisé dans l'espace en trois dimensions, où chaque agent aura un compas qui travaille en trois dimensions.

2.3 Rendez-vous déterministe

2.3.1 Rendez-vous et regroupement dans les graphes

Agents avec étiquettes

Le modèle utilisé dans [12] et [32] est le suivant : 2 agents placés dans un graphe connexe non orienté, inconnu et anonyme doivent se rencontrer dans un nœud du graphe. Le mouvement des agents se fait en rondes synchrones ; à chaque ronde l'agent peut rester dans le nœud courant ou visiter un nœud voisin. Il y a deux situations initiales possibles : les agents démarrent en même temps ou avec un délai entre eux. Le rendez-vous n'est pas possible dans les arcs, seulement dans les nœuds. Les étiquettes dans [12] et [32] sont deux entiers écrits comme une suite binaire en commençant par 1. L'agent connait seulement son étiquette.[12] donne comme résultat $O(n + \log\ l)$ pour le rendez-vous dans les arbres où n est le nombre de nœuds et l est l'étiquette plus petite. Le principe de l'algorithme est le suivant : faire l'exploration de l'arbre, trouver le nœud ou l'arête centrale et essayer de faire la rencontre. Dans le premier cas, le rendez-vous est fait quand le dernier agent arrive dans le nœud central. Dans le deuxième cas, le rendez-vous a lieu dans l'une des extrémités de l'arête centrale.

La procédure Extend-Labels [12] transforme l'étiquette en ajoutant des bits (10) et chaque bit est répété deux fois ; cette transformation se répète de façon indéfinie. Le but est de forcer un des agents à rester dans un des nœuds pendant que l'autre visite ce nœud. Les bits de la nouvelle étiquette (L^*) sont énumérés en partant du bit le moins significatif. La procédure parcour la nouvelle étiquette L^* par étape et dépendemment de la valeur du bit l'agent se déplace (bit égal à 1) ou reste immobile (bit égal à 0). Une présentation plus formelle de la procédure Extend-Label est :

- En étape 1 *se déplace*
- En étape 2 *reste immobile*
- En étapes $3 \leq i \leq 2\lfloor \log\ L \rfloor + 4$, se déplace si bit $\lceil (i-2)/2 \rceil$ de L est égal à 1, sinon reste immobile.
- En étapes $i \geq 2\lfloor \log\ L \rfloor + 4$, le comportement est égal à étape $1 + (i-1)(\ \mathrm{mod}\ 2\lfloor \log\ L \rfloor + 4)$

Un algorithme pour le rendez-vous dans l'anneau décrit dans [12] a un coût de $O(D\ \log\ l)$ où D est la distance initiale entre les agents et l est la plus petite étiquette entre les deux agents. Les deux agents commencent en même temps. La transformation

des étiquettes est faite encore une fois pour savoir quand se déplacer, rester dans le nœud ou retourner au nœud d'origine. S'il y a un délai τ entre les temps de départs des agents, et n est connu, le coût devient $O(n \log l)$. Le rendez-vous pour n'importe quel type de graphe connexe est aussi étudié dans [12] et [32]. [12] utilise une variation de leur algorithme pour l'anneau. L'agent effectue des phases où il fait des parcours en largeur du graphe et des phases où il reste passif. La borne de l'algorithme décrite par [12] est $O(n^5\sqrt{\tau l} + n^{10}l)$. L'algorithme décrit par [32] est la réponse à la question posée dans [12] dans ses conclusions : existe-il un algorithme déterministe pour le rendez-vous dans n'importe quel graphe connexe avec un coût polynomial en n et l (ou même en n et $\log l$), mais indépendante de τ ?. La réponse est oui, les auteurs présentent un algorithme déterministe avec une borne $O(n^5 l)$ unités de temps après l'activation du deuxième agent. L'algorithme décrit par [32] fait l'utilisation des UTS (Universal Traversal Sequences) et des SUTS (Strongly Universal Traversal Sequences) pour définir le parcours que doit faire l'agent dans sa phase active.

Le rendez-vous asynchrone dans [11] et [7] peut être accompli dans les nœuds ou dans les arcs. [11] commence avec un algorithme qui travaille sur une ligne infinie avec la distance D entre les deux agents, connue par les agents. Le coût est la somme des déplacements des agents. Le coût est de l'ordre $O(D|L_{min}|^2)$. Le fonctionnement est basé sur les déplacements à droite et à gauche dépendamment de l'étiquette transformée de chaque agent. La transformation de l'étiquette est faite de la façon suivante : L^* (étiquette transformée) est l'enchaînement d'une suite de $|L|$ zéros, un 1, et l'étiquette L. L'exécution de l'algorithme est la suivante : si $L^*(i) = 0$ l'agent fait $2iD$ pas vers la gauche, $(4i + 1)D$ pas vers la droite et finalement $(2i + 1)D$ pas vers la gauche ; si $L^*(i) = 1$ l'agent fait $2iD$ pas vers la droite, $(4i + 1)D$ pas vers la gauche et finalement $(2i + 1)D$ pas vers la droite. Il faut souligner que la direction droite et gauche sont des connaissances locales et elle peuvent être différentes pour les agents. [11] donne un autre algorithme qui fonctionne quand la distance D entre les agents n'est pas connue et il a un coût de $O((D + L_{max})^3)$. La transformation de l'étiquette est faite avec l'insertion d'un nouveau bit avant chaque bit de L, en alternance de 0 et 1, puis on ajoute la suite 11110 à la fin de l'étiquette trouvée dans le pas antérieur. Finalement l'étiquette L^* est l'enchaînement infini de l'étiquette résultante. L'exécution de cet algorithme est la suivante : si $L^*(i) = 0$, l'agent fait $2i^2$ pas vers la gauche, $(2i^2 + i)$ pas vers la droite et finalement $(i^2 + i)$ pas vers la gauche ; si $L^*(i) = 1$ l'agent fait $2i^2$ pas vers la droite, $(2i^2 + i)$ pas vers la gauche et finalement $(i^2 + i)$ pas vers la droite. Suite à l'algorithme

pour la ligne infinie, [11] propose un algorithme optimal dans l'anneau qui a un ordre de magnitude $O(n|L_{max}|)$ quand n n'est pas connu. La transformation de l'étiquette est : on construit l'étiquette $L^{'}$ avec l'enchaînement d'une suite de $|L|$ zéros, un 1, et l'étiquette L ; l'étiquette L^{*} est obtenue en remplacent chaque 0 par 01 et chaque 1 par 10. L'exécution de l'algorithme se fait en phases $a = 1, 2, 3, ...$ L'agent parcourt 3^{a} pas vers la gauche (si le bit est 0) ou vers la droite dans le cas contraire. À chaque phase a toute l'étiquette L^{*} est lue de gauche vers la droite au complet. L'algorithme s'arrête quand le rendez-vous est effectué.

Finalement, on trouve dans [11] un algorithme qui travaille pour un graphe quelconque, mais avec tous les ports étiquettés et le graphe, ainsi que les positions des agents dans le graphe connues par chaque agent. L'algorithme a un ordre de grandeur de $O(D|L_{min}|)$; l'ordre obtenu est optimal. Son fonctionnement est le suivant : chaque agent calcule la distance D entre eux et essaie de trouver le chemin le plus court entre eux. Ainsi, les deux agents identifient le même cycle de longueur 2D en partant de leurs positions initiales. Les agents utilisent l'algorithme pour le rendez-vous dans l'anneau sur ce cycle. La taille du cycle est connue.

Dans la conclusion de [11], une question est soulevée : Le rendez-vous asynchrone et déterministe est-t-il complètement faisable dans n'importe quel graphe de taille inconnue ? La réponse positive à cette question est donnée dans [7]. [7] arrive à cette conclusion avec la notion de l'existence des tunnels, qui permet de faire le rendez-vous. [7] applique le même algorithme dans le scénario géométrique. Un ordre de grandeur n'a pas été donné pour l'algorithme dans [7].

Agents sans étiquettes

Le problème abordé par [6] est le rendez-vous synchrone de deux agents sans identificateurs différents, et la quantité de mémoire optimale pour réaliser le rendez-vous. L'algorithme utilise les parcours synchrones générés à partir de sa position initiale et un UXS (séquence d'exploration universelle) pour créer une étiquette. Les étiquettes générées sont différentes si et seulement si les nœuds initiaux ont des vues différentes du graphe, c'est-à-dire ne sont pas symétriques. La borne $O(\log\ n)$ sur la grandeur de mémoire est donnée dans [6] et elle est optimale.

Le rendez-vous déterministe asynchrone des agents sans étiquettes est abordé par [21] dans le scénario où les agents sont placés dans un graphe quelconque, inconnu et connexe. Les agents ont la capacité de différencier les ports sur un nœud, mais les

nœuds sont anonymes. La vitesse est contrôlée par l'adversaire et le rendez-vous peut être réalisé sur les nœuds ou les arêtes. L'algorithme utilise le résultat obtenu par [7] mais les agents utilisent des étapes numérotées de 1 à n. À chaque étape, l'agent fait un parcours DFS (parcours en profondeur) avec une profondeur n et utilise la vue obtenue comme "étiquette" pour faire fonctionner l'algorithme décrit par [7]. La condition pour que l'algorithme [21] fonctionne est : les positions initiales des agents sont connectées par un chemin palindrome où la vue de chaque agent est différente. Les auteurs montrent que c'est une condition nécessaire pour le rendez-vous.

Le regroupement asynchrone des agents sur une grille a été étudié dans [9]. Dans ce travail les agents (robots) sont placés sur une grille anonyme. Les robots fonctionnent en cycles asynchrones Regarder-Calculer-Bouger, selon le modèle CORDA. Les robots dans la phase Regarder ont une vue de la grille en termes de nœuds occupés, tous leurs calculs sont basés seulement sur la dernière vue de la grille qu'ils ont eu. La particularité de l'article [9] est de permettre aux robots d'accomplir la tâche de regroupement sans avoir la capacité de détection de plusieurs robots sur un même nœud. L'article [9] décrit plusieurs algorithmes dépendamment de la configuration de la grille :

- Grille Impaire x Impaire : Les robots peuvent accomplir la tâche de regroupement peu importe la configuration initiale des robots sur la grille. Pour ce faire, ils doivent trouver le nœud central de la grille $M\left[\lceil\frac{m}{2}\rceil, \lceil\frac{n}{2}\rceil\right]$ où m est le nombre de lignes et n est le nombre de colonnes.
- Grille Impaire x Pair : Avec cette configuration, le regroupement n'est pas toujours faisable. Le regroupement n'est pas faisable si les robots sont placés d'une façon périodique, ou bien en symétrie par rapport à un axe qui passe à travers une arête de la grille. Le regroupement, quand il est possible, est accompli par les robots en utilisant une division en deux plans de l'ensemble des robots, avec l'utilisation d'un parcours minimal des nœuds séparant chaque robot dans la grille. Le *nord* est le plan avec le plus de robots. Si le nombre de robots est égal, les robots utiliseront d'autres méthodes pour trouver le nord. L'idée est que chaque robot se déplace vers le nord de la séparation calculée par les robots. Une fois que tous les robots se retrouvent au nord calculé, ils se déplacent vers le nœud choisi par les agents pour faire le regroupement.
- Grille Pair x Pair : Le regroupement avec cette configuration de grille est possible seulement si la disposition des robots n'est pas périodique, ou si une axe de symétrie verticale ou horizontale est introuvable. Une autre situation où le re-

groupement n'est pas possible est la grille 2x2. Cette configuration a le même comportement qu'un anneau de 4 nœuds, et dans l'article [25], il est prouvé que le regroupement est impossible avec deux ou quatre nœuds occupés même avec la capacité de détection des multiples robots sur un même nœud. Pour ce type de grille, seulement la configuration avec 3 robots dans les nœuds et la capacité de détection des multiplicités permet le regroupement des robots. Si aucune de ces restrictions ne sont trouvées, le regroupement est possible ; deux algorithmes sont donnés, le premier où il y a un coin occupé par un robot et le deuxième quand il y a trois coins occupés par des robots.

Le regroupement des agents sur des graphes bipartis est exploré dans l'article [22]. Les agents fonctionnent selon le modèle CORDA [30] et chaque agent est une entité anonyme, sans mémoire, qui fonctionne en cycles asynchrones Regarder-Calculer-Bouger. La différence avec les autres travaux est que l'agent n'a pas la possibilité de regarder la configuration complète des autres agents sur le graphe. Dans ce travail, les agents sont capables de voir seulement leurs voisins (les nœuds adjacents). La capacité de détection de multiplicité est supposée dans l'article, l'agent peut savoir si le nœud est occupé par 0 agents, 1 agent ou plusieurs agents.

Le résultat de l'article [22] est que seulement la configuration des agents de type étoile (un agent A avec tous les autres agents adjacent à lui) et de taille d'au moins 3 peut accomplir la tâche de regroupement.

L'algorithme pour cette configuration (étoile), (la seule qui permet le regroupement avec les conditions initiales décrites) est le suivant : l'agent qui doit se promener est nommé A. Si A se trouve dans un nœud avec multiplicité ou A a plus que un voisin occupé, l'agent A ne doit pas bouger. Sinon, A doit aller au seul voisin occupé.

Regroupement des robots dans un anneau asynchrone et anonyme.

Le modèle utilisé dans l'article [25] est le suivant : les robots sont placés dans un anneau dans lequel il n'y a pas d'étiquettes pour les sommets ou les liens. Initialement, quelques sommets sont occupés par exactement un robot. Comme tâche, les robots doivent se regrouper dans un sommet de l'anneau et s'arrêter. Les robots ont les mêmes caractéristiques décrites par [5], dont la capacité de détecter si dans un sommet il y a plus d'un robot. Par contre leurs mouvements sont instantanés. Les robots ne peuvent pas laisser de marques ni dans les sommets, ni dans les liens, et toute communication entre eux est interdite.

Les robots travaillent en cycles asynchrones **regarder**, **calcul** et **mouvement** jusqu'à ce que le regroupement soit fait. Les robots dans la phase de **regarder** peuvent voir l'ensemble des robots et détecter la multiplicité sur les sommets. Avec cette information, les robots construisent la *configuration* de leur environnement. La *configuration* est formée de deux séquences $((a_1, \ldots, a_r), (b_1, \ldots, b_s))$, où les valeurs de a_i sont les distances entre les sommets qui ont au moins un robot en sens horaire. b_j sont les distances entre les sommets qui ont une multiplicité ; encore une fois, on écrit la distance entre les sommets en faisant la tournée en sens horaire.

La séquence $(a_1, \ldots, a_r)$ est périodique si elle est composée d'au moins deux copies d'une sous-séquence p, comme la séquence $C = (2, 3, 1, 2, 3, 1)$ pour cette configuration. Une configuration est appelée *symétrique* s'il existe un axe de symétrie dans l'anneau, deux robots sont appelés *voisins* si au moins un des segments de l'anneau entre eux ne contient pas de robot. Une configuration est appelée *rigide* s'il n'y a pas de multiplicité et que la vue de chacun des robots est différente.

L'algorithme de [25] cherche à former un seul sommet avec multiplicité et à utiliser ce sommet pour regrouper les robots. S'il y a déjà une seule multiplicité dans le sommet v, le robot le plus proche du sommet v se déplace vers v, après le deuxième le plus proche et ainsi de suite ; à la fin, tous les robots seront dans un seul sommet. Le sommet v a la multiplicité initiale.

L'algorithme qu'on vient de décrire fonctionne avec une seule multiplicité, donc l'idée sera, dans les configurations sans multiplicité, de créer une seule multiplicité et de profiter de cet algorithme.

Pour les configurations *rigides*, l'idée est de faire l'élection d'un seul robot et de le déplacer vers un de ses voisins pour créer un sommet avec multiplicité, et utiliser l'algorithme qui travaille avec une seule multiplicité. Premièrement, les robots choisissent une paire de robots, les plus éloignés. Avec cette paire de robots, ils choisissent le robot avec le voisin le plus proche. Ce dernier va se déplacer vers son voisin proche (le robot avec le voisin le plus proche parmi les deux éloignés) pour générer la seule multiplicité. Par la suite, on applique l'algorithme pour une seule multiplicité.

Lorsque le nombre de robots est impair et qu'il n'y a pas une configuration périodique, le regroupement peut être fait de la façon suivante. Les robots peuvent se trouver dans deux configurations possibles. Ils sont dans une configuration *rigide* et donc, on applique l'algorithme décrit pour ce type de configuration, ou la configuration est symétrique. Comme le nombre de robots est impair, il y a un seul robot sur l'axe de symétrie donc

on déplace ce robot vers un sommet adjacent. Il y a 3 situations dans lesquelles on peut se retrouver : (1) une multiplicité a été construite donc on utilise l'algorithme déjà décrit pour résoudre le regroupement, (2) la configuration est *rigide*, on applique alors l'algorithme pour les configurations *rigides* ou (3) un autre axe de symétrie a été créé, on déplace donc le robot sur l'axe de symétrie vers un sommet adjacent et l'une des trois situations peut se répéter. [25] prouve qu'après avoir fini un certain nombre d'étapes, seulement les situations (1) ou (2) peuvent se présenter et le regroupement est possible.

Comme conclusion, les auteurs montrent que le regroupement des robots n'est pas possible quand le nombre de robot est 2, ou la configuration est périodique ou la symétrie arête-arête est présente. Par contre, quand le nombre de robot est impair le regroupement est faisable pour toutes les configurations non-périodiques.

L'article [8] utilise le même modèle que [25]. L'algorithme décrit par [8] ferme plusieurs questions laissées ouvertes par d'autres articles (exemple [24]) et permet de faire le regroupement des robots dans l'anneau si et seulement si toutes les conditions suivantes sont respectées dans la disposition des robots sur l'anneau : il y a plus que deux robots dans l'anneau, la configuration $SP4$ (4 robots dans des configurations pas résolues) n'est pas présente, la configuration n'est pas périodique et il n'y a aucune symétrie du type arête-arête. L'algorithme utilise la vue obtenue par le robot pour déterminer si le regroupement est réalisable et s'il est réalisable, effectuer un des sept sous-algorithmes pour accomplir la tâche. Comme la vue peut changer en cours de route pour les robots, le processus de sélection du sous-algorithme à utiliser s'exécute jusqu'au moment où les robots détectent qu'ils se trouvent dans une configuration avec une multiplicitée ou une configuration qui rendra le regroupement possible. Les sous-algorithmes utilisés lorsqu'il y a quatre robots sont décrits dans l'article [26]. Si le nombre de robots est six, les robots utiliseront les algorithmes décrit dans [10], pour les autres configurations où le regroupement est possible, d'autres sous-algorithmes sont utilisés par les robots.

L'utilisation des jetons par les agents mobiles est explorée par [14] pour résoudre le problème de regroupement, ou rendez-vous de plusieurs agents mobiles. Le modèle décrit par [14] est composé d'agents placés sur les nœuds d'un anneau. Le comportement des agents est synchrone, ils sont anonymes et ils partagent une orientation commune. [14] indique que n (nombre de nœuds) ou k (nombre des agents) doit être connu par les agents pour rendre le regroupement faisable. [14] suppose que k est connu par les agents. Un algorithme proposé par [14], en utilisant $O(k \log(n))$ bits de mémoire est : Laisser son jeton dans le nœud initial, choisir une direction et parcourir tout l'anneau, calculer

la distance entre chacun des jetons ; si les distances entre les jetons sont périodiques, s'arrêter (le rendez-vous n'est pas faisable), sinon, utiliser la séquence des distances pour trouver les déplacements à effectuer et, finalement, accomplir le rendez-vous. On remarque que si la suite des distances est périodique, la symétrie n'est peut pas être brisée, donc le rendez-vous est impossible.

2.3.2 Rendez-vous sur le terrain

Impossibilité de regroupement pour un ensemble des robots asynchrones et sans mémoire

Le problème de regroupement des robots est défini comme suit ; n robots sont placés de façon arbitraire sur le plan, sans qu'il n'y ait pas deux robots avec la même position. Il faut qu'ils se rencontrent sur un même point dans un nombre de cycles fini. Le problème a été considéré comme une extension du problème de rendez-vous, où il y a deux agents. La nature des robots qui feront le regroupement, était modélisée différemment dans différentes études, qui cherchent les qualités minimales dans chaque robot pour y accomplir le regroupement, [31], [15] et [5] décrits plus tard.

Le modèle utilisé pour les différentes solutions au problème de regroupement utilise des robots avec des capacités simples. Ils sont anonymes, de fonctionnement asynchrone, sans mémoire (les calculs sont effectués seulement avec des données actuelles). Chaque robot a un système de coordonnées et d'unités de mesure du plan qui peut être différente de celui des autres. Aussi, la communication entre eux est interdite ; [31] prouve qu'avec les propriétés décrites, le regroupement est impossible pour n'importe quel algorithme, asynchrone ou synchrone.

Les robots peuvent avoir un comportement de type *asynchrone* : le temps utilisé pour chaque étape(regarder, calculer et son mouvement) est fini mais pas connu ; ou un type **atomique** ou chaque étape est exécuté de façon atomique et synchrone. [31] prouve aussi que le problème de regroupement avec un comportement asynchrone est contenu dans l'ensemble des problèmes de type atomique, c'est-à-dire, si une configuration peut être regroupée en mode synchrone. elle peut aussi être regroupée en mode asynchrone, donc $\mathbb{AS} \subset \mathbb{AT}$, (groupe asynchrone $\mathbb{AS}$ et groupe atomique $\mathbb{AT}$). [31] décrit en premier l'impossibilité de regroupement pour le comportement synchrone et donc, aussi pour le type asynchrone.

Le fonctionnement de chaque robot r est découpé en 4 étapes : **repos** : un robot ne peut rester en repos infiniment, **regarder**, où le robot prend une vue de toutes les positions de chacun des robots dans son système de coordonnées (les autres robots sont vus comme des points), **calcul**, où le robot exécute l'algorithme $\mathcal{A}$ (qui a comme sortie le prochain point), et **mouvement**, où le robot se déplace vers le nouveau point calculé (le temps qu'il prend pour arriver à sa destination peut être variable mais il reste fini). Le robot peut ne pas arriver à sa destination pour des raisons diverses (par exemple épuisement d'énergie).

Pour $n = 2$, le problème n'a pas de solution. Pourquoi ? Imaginons deux robots sur le plan, l'adversaire s'active et fait exécuter l'algorithme au même moment pour les deux robots. Résultat : les deux robots ont calculé de se déplacer vers l'autre robot ; au milieu du parcours (les deux robots ont la même vitesse), les robots vont se croiser sans le savoir et ils vont arriver à la destination calculée, l'ancienne position de l'autre robot. Le cycle va se répéter indéfiniment. Dû au manque de détection de multiplicité (savoir quand il y a plus d'un robot sur le même point) dans les robots, ils peuvent se retrouver au même point que le deuxième robot sans le savoir.

Cette faille est exploitée par [31] pour démontrer l'impossibilité d'un algorithme qui fait le regroupement des robots pour n'importe quelle valeur de n sans changer les propriétés initiales dans les robots.

L'idée qui a été développée par [31] et qui permet de prouver l'impossibilité est la suivante : l'ensemble des robots est réparti dans deux groupes : l'ensemble noir et l'ensemble blanc. L'ensemble noir est composé de $n - 1$ robots qui ont le même système de coordonnées et unités de mesure ; le robot qui reste est l'ensemble blanc qui a comme différence un système de coordonnées inverse au niveau de l'orientation des axes x et y. Tous les robots exécutent l'algorithme $\mathcal{A}_g$. Maintenant l'adversaire place l'ensemble des robots noirs sur un même point et le robot blanc dans un autre point quelconque. L'algorithme $\mathcal{A}_g$ est exécuté par l'ensemble des robots et on trouve un comportement pareil à celui décrit pour $n = 2$ robots. Les robots ne sont pas capables de se regrouper en même temps dans un point. À noter, ici le problème est vu de façon synchrone, et le déplacement des robots est atomique (instantané).

Les deux algorithmes proposés dans [5] et [15] décrits dans les sections suivantes, contiennent des modifications apportées aux robots pour accomplir la tâche de regroupement.

Algorithme de regroupement pour des robots asynchrones et sans mémoire avec détection des multiplicités

Le premier algorithme déterministe qui permet le regroupement des robots pour n'importe quelle configuration initiale est décrit par [5]. Les robots sont anonymes ; il n'y a pas de système commun de coordonnées entre eux, sont oublieux (sans mémoire) et la communications entre eux est interdite. Avec ce faible ensemble de caractéristiques, il est impossible pour les robots de faire le regroupement [31]. Donc, on conclut qu'il faut donner aux robots une nouvelle compétence, soit la capacité de détecter la multiplicité.

La vue locale de chaque robot est composée de l'ensemble des points dans le plan occupés par au moins un robot. Le robot, à partir de l'ensemble de points peut détecter la multiplicité. La vue locale de chaque robot inclut une unité de mesure, une origine et un système de coordonnées. Il n'y a pas d'accord sur l'unité de mesure, l'origine et un système de coordonnées. Les robots exécutent les étapes : **repos**, **regarder**, **calcul**, et **mouvement** déjà décrits. Ces étapes sont exécutées de façon asynchrone jusqu'à ce que le regroupement soit accompli pour tous les robots.

Le fonctionnement de l'algorithme est le suivant : tous les robots se trouvent dans différentes positions, donc il n'y a pas deux robots sur le même point. Les robots prennent une vue du plan et font l'exécution de l'algorithme. Le robot regarde si l'ensemble des robots a une *configuration biangulaire*. Une configuration biangulaire se trouve s'il y a un point b, le centre et une suite des robots et deux angles α et $\beta > 0$ tel que chaque deux robots adjacents forment un angle α où β quant à b et les angles s'alternent. Si oui, les robots prennent le centre de la configuration biangulaire b comme leur nouvelle destination. Au moment où deux robots arrivent sur le point b, un point avec multiplicité est formé et les autres agents le prennent comme point pour le regroupement. Si, au contraire, l'ensemble de robots ne forme pas une configuration biangulaire, l'algorithme des robots calcule le plus petit cercle qui enveloppe l'ensemble des robots. Une fois le cercle calculé, l'algorithme utilise l'ensemble des angles entre les robots et le centre c du cercle calculé pour déterminer quels robots vont se déplacer vers le centre c pour construire le point de multiplicité. Une fois que l'unique point de multiplicité est construit, les autres robots se déplacent vers lui pour accomplir le regroupement.

Le mouvement des robots à chaque étape se fait de manière à éviter la création des autres points de multiplicité différents de ceux du premier calcul. L'algorithme présenté par [5] travaille pour $n > 5$ et n'importe quelle configuration initiale des robots.

Algorithme avec des robots qui possèdent une orientation commune, une visibilité réduite, qui sont asynchrones et sans mémoire

L'article [15] ajoute aux robots une caractéristique qui a été déjà présentée par [31], un système de coordonnées commun entre les robots, l'accord est défini sur les axes et les directions (positives ou négatives), mais pas nécessairement sur l'origine ou l'unité de mesure. Le modèle utilisé dans [15] donne à chaque robot la possibilité de voir le plan de façon partielle : un cercle avec le robot comme origine et un rayon V. Les étapes suivies par chaque robot sont les mêmes que celles décrites par [5] et [31], mais le calcul effectué par le robot dans cet algorithme est commandé par instructions suivantes : le robot qui fait le calcul est nommé r.

- 1. Si r voit des robots à sa gauche ou au-dessus de son axe vertical, pas de mouvement.
- 2. Si r voit des robots seulement en bas de son axe vertical, se déplacer vers le robot le plus proche.
- 3. Si r voit des robots seulement à sa droite, se déplacer horizontalement vers l'axe vertical du robot le plus proche.
- 4. Si r voit des robots en bas de son axe et à sa droite, faire un déplacement en diagonale vers la droite et le bas. L'angle pour le déplacement est calculé selon les indications dans [15].

Il faut remarquer qu'avant d'exécuter l'algorithme, tous les robots doivent avoir dans leur champ de vision au moins un autre robot pour permettre à l'algorithme d'être convergent. Pour prouver la nécessité de cette contrainte, imaginons deux robots dans le plan qui sont en dehors de champ de vision de l'autre. Puisqu'ils ont le même algorithme, ils vont faire les mêmes déplacements, donc la distance entre eux va se maintenir constante. L'adversaire, dans ce cas, donnera la même vitesse de déplacement aux deux robots. [15] prouve aussi que si cette contrainte est respectée au début, il y aura toujours au moins un robot dans le champ de vision de tout autre robot, pendant le roulement de l'algorithme.

2.4 Commérage entre nœuds d'un réseau

L'échange des messages entre les nœuds d'un réseau (appelé aussi commérage) a un grand intérêt dans la littérature. Ce problème touche l'optimisation des communications

entre les différents processeurs (dans un ordinateur multiprocesseur) où chaque processeur peut être vu comme un nœud et ses interconnexions comme les arcs d'un graphe. On peut remarquer aussi l'immense groupe des routeurs qui forment le réseau Internet, considéré aussi comme un graphe avec plusieurs nœuds qui veulent effectuer des tâches de communication. Le commérage peut être utilisé dans n'importe quel processus qui a besoin de faire le ramassage d'information sur les processeurs, comme découvrir la topologie d'un réseau ou la prise de décision, [28].

Le livre [23] présente le modèle des processeurs et leur interconnexion comme un graphe $G = (V, E)$ ou $V(G)$ est l'ensemble des nœuds et $E(G)$ comprend l'ensemble des arêtes.

Dans le modèle présenté par [23] chaque nœud a une information qui doit être partagée. Avec quels nœuds l'information doit être partagée donne comme résultat trois concepts qui sont liés entre eux, *Diffusion*, *Accumulation* et *Commérage*.

Le livre [23] utilise le concept de *stratégie de communication* comme l'algorithme qui doit être suivi par les nœuds pour accomplir la tâche de *Diffusion*, *Accumulation* ou *Commérage*.

Diffusion : Considérons le graphe $G = (V, E)$ et $v \in V$ un nœud de G. Le nœud v a une information $I(v)$ qui n'est pas connue par les autres nœuds. La stratégie de communication a comme objectif de faire connaître le morceau d'information $I(v)$ à tous les nœuds de G.

Accumulation : Considérons le graphe $G = (V, E)$ et $v \in V$ un nœud de G. Chaque nœud de $u \in V$ a une information $I(u)$. Le but est que le nœud v reçoive toutes les informations $I(u)$.

Commérage : Considérons le graphe $G = (V, E)$. $I(v)$ est une information en v pour tout $v \in V$. Le but est que chaque nœud en V apprenne l'ensemble des informations contenues dans tous les autres nœuds.

Dans ce qui suit nous nous concentrons sur la littérature concernant le **commérage** entre nœuds.

La stratégie de communication [23] est un ensemble de séquences de communication simples appelées *rondes de communication*. Dans chaque ronde de communication il est décrit plus précisément quels arcs sont utilisés pour partager l'information et le mode de communication à utiliser.

Dans la littérature, il y a deux modes classiques, le mode à un chemin et le mode à deux chemins.

Mode à un chemin : ou mode de communication de télégraphe, dans lequel chaque nœud a un seul arc actif parmi ses arcs adjacents, pour recevoir ou (exclusivement) envoyer un message en une ronde spécifique.

Mode à deux chemins : ou mode de communication de téléphone, dans lequel chaque nœud a un seul arc actif parmi ses arcs adjacents, pour envoyer et recevoir un message en une ronde spécifique.

Les stratégies de communication sont mesurées par le nombre de rondes nécessaires pour accomplir le partage d'information désiré. Dans le cas de commérage on cherche une stratégie de communication avec le nombre minimal de rondes qui permet de partager l'information entre les nœuds.

Un simple algorithme de commérage peut être décrit de la façon suivante. Chaque nœud a une étiquette (numéro entier). L'algorithme consiste en deux phases chacune de durée de $n-1$ rondes. Dans la *i-ème* ronde de la première phase le nœud i transmet au nœud $i+1$. Dans la *i-ème* ronde de la deuxième phase le nœud n transmet au nœud i. Cette algorithme est correct, mais il prend le temps $2n-2$.

Dans la littérature, un algorithme de commérage est dit *non-adaptatif* si l'algorithme utilise seulement l'étiquette et le nombre total de nœuds pour prendre les décisions par rapport au moment où un nœud peut transmettre son information. Par contre, si un algorithme utilise d'autres paramètres comme le contenu des messages reçus ou s'il y a eu des collisions, il est appelé un algorithme *adaptatif* [28].

Deux tableaux qui résument les bornes supérieures et inférieures (commérage) pour les types de graphes les plus connus sont présentés ci-dessous ([23]) :

Le types des graphes trouvés dans les tableaux 2.2 et 2.1 sont :

- K_n = Graphe complet de n arcs.
- H_k = Graphe Hypercube de dimension k.
- P_n = Graphe type Ligne de dimension n.
- C_n = Graphe Cycle de n nœuds.
- CCC_k = Graphe cycles de cubes connectés de dimension k.
- SE_k = Graphe shuffle-exchange network de dimension k.
- BF_k = Graphe réseau papillon de dimension k.
- DB_k = Réseau $deBruijn$ de dimension k.

Nous introduisons maintenant un nouveau modèle de communication appelé modèle *réseau radio*, dans lequel, lorsque le nœud fait la transmission de son message, le signal atteint tous ses voisins. Le message est bien obtenu par un nœud si et seulement si un

Graphe	Num. nœuds	Diamètre	Borne inférieure	Borne inférieure
K_n	n	1	$\lceil \log_2 n \rceil + odd(n)$	$\lceil \log_2 n \rceil + odd(n)$
H_k	2^k	k	k	k
P_n	n	$n-1$	$n - pair(n)$	$n - pair(n)$
C_n	n	$\lfloor n/2 \rfloor$	$\lceil n/2 \rceil + impair(n)$	$\lceil n/2 \rceil + odd(n)$
CCC_k	$k \cdot 2^k$	$\lfloor 5k/2 \rfloor - 2$	$\lceil 5k/2 \rceil - 2$	$\lceil 5k/2 \rceil - 2$, k pair $\lceil 5k/2 \rceil + 1$, k impair
SE_k	2^k	$2k-1$	$2k-1$	$2k+5$
BF_k	$k \cdot 2^k$	$\lfloor 3k/2 \rfloor$	$1.9770k$	$2.25 \cdot k + o(k)$
DB_k	2^k	k	$1.5965k$	$2k+5$

TABLE 2.1 – Temps pour le commérage dans des réseaux communs et mode de communication deux-chemins

seul voisin transmet pendant la ronde de communication (sinon il y a une *collision*). Utilisé par [18], [28], [19] et [29] ce modèle a connu une popularité croissante dû au développement des téléphones cellulaires.

Le modèle utilisé pour les réseaux radio est un graphe non dirigé $G(V, E)$ de connections. Chaque nœud a un numéro différent dans l'intervalle $[1, ..., N = O(n^c)]$ pour une constante $c \geq 1$. Deux nœuds en V sont connectés par une arête en E s'ils peuvent communiquer directement. Ces nœuds sont appelés *voisins*. La taille du réseau est déterminée par le nombre de nœuds qui le composent $|V| = n$. Deux paramètres amplement utilisés sont le *degré maximal* $\triangle$ et le *diamètre* D. Un autre ajout au modèle est l'incapacité des nœuds de distinguer entre une collision et le bruit de fond. Les nœuds peuvent connaître la topologie du réseau ou non. Dans le deuxième cas on parle des réseaux de type *Ad Hoc*.

L'article [19] présente un algorithme qui permet le commérage entre tous les nœuds à l'intérieur de G. Dans son modèle, la topologie est connue par les nœuds et chaque séquence de communication est calculée à l'avance.

Graphe	Num. nœuds	Diamètre	Borne inférieure	Borne inférieure
K_n	n	1	$1.44 \log_2 n$	$1.44 \log_2 n$
H_k	2^k	k	$1.44k$	$1.88k$
P_n	n	$n-1$	$n + impair(n)$	$n + impair(n)$
C_n	n pair n impair	$\lfloor n/2 \rfloor$ $\lfloor n/2 \rfloor$	$n/2 + \lceil \sqrt{2n} \rceil - 1$ $\lceil n/2 \rceil + \lceil \sqrt{2n} - 1/2 \rceil - 1$	$n/2 + \lceil \sqrt{2n} \rceil - 1$ $\lceil n/2 \rceil + \lceil 2\sqrt{\lceil n/2 \rceil} \rceil - 1$
CCC_k	$k \cdot 2^k$	$\lfloor 5k/2 \rfloor - 2$	$\lceil 5k/2 \rceil - 2$	$\lceil 7k/2 \rceil + \lceil 2\sqrt{\lceil k/2 \rceil} \rceil - 2$
SE_k	2^k	$2k-1$	$2k-1$	$3k+3$
BF_k	$k \cdot 2^k$	$\lfloor 3k/2 \rfloor$	$1.9770k$	$\lceil 5k/2 \rceil + \lceil 2\sqrt{\lceil k/2 \rceil} \rceil - 1$
DB_k	2^k	k	$1.5965k$	$3k+3$

TABLE 2.2 – Temps pour le commérage dans des réseaux communs et mode de communication un-chemin

L'algorithme décrit par [19] a une borne supérieure de $O(D + \triangle \log n)$. C'est la meilleure borne supérieure connue pour les algorithmes où la topologie est connue.

La base de l'algorithme est la construction des arbres *BFS* (algorithme de parcours en largeur) où les nœuds sont classés par rapport aux enfants. Chaque feuille v a une $classe(v) = 1$. Les autres nœuds (pas les feuilles) obtiennent leur classe avec la procédure suivante : pour un nœud v et les classes des enfants $r_1, ..., r_k$, $r_{max} = max(r_i)$. Si le nœud v a un seul enfant, sa classe sera $classe(v) = r_{max}$. Si le nœud a plus d'un enfant sa classe sera $classe(v) = r_{max} + 1$. L'algorithme obtient deux sous-ensembles de nœuds avec l'information de sa classe comme base, un sous-ensemble de *transmission rapide* et une autre *transmission lente*.

L'algorithme trouve les nœuds qui peuvent communiquer sans collision pour assembler un arbre libre de collision, *Arbre de commérage couvrant*, un nouveau concept introduit par l'article [19], et assigner, a des rondes différentes, les nœuds dans l'ensemble de transmission rapide ou transmission lente qui peuvent transmettre leur message.

L'aperçu effectué par [18] pour les algorithmes de commérage permet d'extraire les résultats obtenus par différents chercheurs. Pour l'instant, continuons avec les algorithmes

où les nœuds connaissent la topologie complète du réseau. Dans ce type de problème la complexité est donnée par rapport au diamètre D et au degré maximal $\triangle$.

Dans ses remarques sur le commérage, [18] indique qu'il a été prouvé que le commérage dans un réseau de taille n est réalisable en un temps au moins n avec l'utilisation du *principe de réduction à 2-nœuds* [20].

Une variante du commérage introduit une nouvelle contrainte : les messages envoyés par les nœuds sont limités à une taille constante. Cette variante a été étudiée par [17]. Ses résultats pour la ligne et l'anneau de taille n sont $3n+\Theta(1)$ et $2n+\Theta(1)$ respectivement. Pour les arbres, les bornes trouvées par [17] sont $\Omega(n \log n)$ et $O(n \log^2 n)$.

Comme autre variante, on retrouve aussi le problème de multidiffusion **M2M** (multi-à-multi) présenté par [16]. C'est un problème proche du commérage et il est sur la frontière entre les problèmes de type Ad Hoc et ceux où la topologie est connue. Dans cette variante, k nœuds de n sont réveillés dans une ronde quelconque et font la transmission de leur message. La topologie est connue par tous les nœuds du graphe, mais les nœuds sont incapables de savoir où se trouvent les $k-1$ nœuds qui font la transmission de leur message. L'algorithme présenté par [16] donne comme résultat une borne en temps $O(d \log^2 n + k \log^4 n)$.

Le deux prochains travaux de recherche, [28] et [29], utilisent la ligne (ou graphe à une dimension), comme contrainte dans la disposition des nœuds, un modèle partagé avec ce mémoire.

Le travail [28] a comme modèle un ensemble P de n processeurs $p_1, ..., p_n$ avec un ordre de gauche à droite. La séparation entre chaque processeur est fixe et ils sont placés sur un espace d'une dimension. Chaque processeur utilise un algorithme localement installé. Chaque processeur sait lorsqu'un intervalle de temps commence et si l'intervalle de temps est suffisamment large pour permettre au processeur de transmettre son message (tout ce qu'il a reçu) au complet. La topologie est connue par les processeurs.

À chaque intervalle de temps, au moins un processeur va transmettre son message et le message sera écouté par les processeurs qui se trouvent à une distance inférieure à R, définie comme le *rayon de transmission*. Les processeurs qui ont entendu le message envoyé par le processeur p sont appelés les *processeurs voisins* de p. Le diamètre D est défini comme la plus courte séquence de transmissions qui, en partant du processeur p_1, atteint le processeur p_n.

Trois algorithmes sont développés par [28], *l'algorithme singleton*, *l'algorithme libre de collision* et *l'algorithme libre de restrictions*.

Dans l'algorithme singleton, le commérage est obtenu après $n + D - 2$ rondes. L'algorithme commence avec la transmission de chaque processeur de droite à gauche en commençant par le processeur p_n jusqu'au processeur p_3. Quand le processeur p_3 a transmit son message, le processeur p_2 a les messages de $p_2, p_3, p_4, ..., p_n$. Finalement, les processeurs p_1 jusqu'à p_{n-1} transmettent leur information. L'algorithme est correct parce qu'à la fin de son exécution, chaque processeur possède l'information initiale contenue dans chacun des autres processeurs.

Un algorithme dans lequel il y a des processeurs qui peuvent se déplacer dans une ligne (un modèle très proche de notre modèle de recherche) est décrit par [29]. Le but de l'algorithme [29] est de permettre aux processeurs de réaliser la tâche de commérage.

Dans le modèle utilisé par [29], il y a un ensemble de processeurs où chaque processeur $p_i \in \{p_1, ..., p_n\}$ a un message m_i qu'il veut partager avec les autres processeurs (commérage). Les processeurs bougent avec des trajectoires continues dans un espace à une dimension (ligne) et la distance maximale qu'ils peuvent voyager par intervalle de temps est σ. Les processeurs communiquent sans-fil sur une chaîne de communication partagée. Un processeur p reçoit correctement un message provenant d'un processeur q pendant l'intervalle de temps t si et seulement si q est à une distance R (rayon de transmission) de p pendant tout l'intervalle de temps t et qu'aucun autre processeur transmettant q' ne soit à une distance R' (rayon d'interférence) de p pendant un instant quelconque dans l'intervalle de temps t. Chaque processeur a un identificateur unique (ID).

L'algorithme suppose que le réseau est dense, c'est a dire qu'il y a une constante $L \leq R$ tel qu'il ne peut y avoir deux processeurs voisins d'une distance supérieure à L. [29] déclare des constantes K et m, divise l'espace en segments de longueur K et divise l'ensemble des segments en m classes de segments. Ensuite, il divise l'ensemble des intervalles de temps en phases de longueur $m-1$. En tout temps, chaque processeur connaît sa position sur la ligne et le segment où il se trouve. Les valeurs de K et m sont choisies en utilisant les trois contraintes suivantes : (C1) $K > (m-1)\sigma$, (C2) $R + R' \leq (K - 2\sigma)(m-1)$ et (C3) $L \leq [R - 3(m-1)\sigma - 3K]/2$. La contrainte C1 implique qu'un processeur peut passer la frontière d'un segment en une seule phase. C2 implique que deux processeurs sont suffisamment éloignés pour transmettre dans le même intervalle de temps sans collision. C3 implique que, entre le processeur extrême gauche et le processeur extrême droite, il n'y aura pas un intervalle de longueur $[R - 3(m-1)\sigma - 3K]/2$ sans processeur, assurant qu'il y aura toujours un processeur pour propager un message.

L'algorithme s'appuie sur le travail de [13] introduisant le concept de *programme EWS*. Le programme EWS est l'algorithme développé par [13] pour maintenir à jour la position de chaque processeur voisin. Pour y arriver, le programme EWS fait l'élection d'un processeur chef par segment par intervalle de temps, celui avec l'ID plus petit. De cette façon tous les processeurs chefs peuvent transmettre leurs messages sans collision en utilisant le même intervalle de temps.

Dans l'algorithme de [29], chaque processeur suit l'exécution du programme EWS. Quand le chef de son segment est élu, il lui transmet toute l'information qu'il connaît (les informations déjà reçues, son propre message et l'information concernant les trajectoires).

[29] élit le chef différement de [13] : chaque processeur, au commencement de chaque phase, vérifie s'il a reçu les messages des autres processeurs dans son segment et s'il a transmis son propre message. Si oui, il choisit comme chef le processeur avec l'ID plus petit. Sinon, l'élection du chef se fait parmi les processeurs qui n'ont pas transmis ou reçu de messages. Le processeur avec l'ID plus petit dans ce groupe est élu comme chef. L'algorithme se termine à la fin de la phase numéro $6n + 13$, et sa complexité algorithmique est $O(n)$.

Chapitre 3

Préliminaires

Dans cette section, nous présentons les définitions et les observations préliminaires qui sont à la base du cadre formel de ce mémoire.

Dans tout ce mémoire, sauf dans les chapitres 7 et 9, on considère N agents initialement situés sur une ligne, chacun à distance 1 des agents voisins.

Par *schéma de commérage* $\mathcal{S}$, on appelle une séquence de *mouvements d'agents mobiles*, tel que durant chacun de ses mouvements un seul agent mobile marche en parcourant une distance spécifiée dans l'une des deux directions de la droite. Chaque mouvement sera dénoté par une paire (a_j, d) ou a_j est le numéro d'agent et d la distance parcourue (d sera négatif si l'agent se déplace dans la direction négative de la droite).

On suppose que les mouvements d'agents se font en un ordre croissant des étapes (voir l'exemple sur figure 3.1), c'est-à-dire en ordre de leurs apparitions dans la sequence S.

À chaque mouvement d'agent, on peut associer un numéro d'étape durant lequel le mouvement a lieu. Le numéro représente l'indice du mouvement d'agent dans le schéma de commérage.

Le *coût* du schéma de commérage est égal au maximum de la somme des distances parcourues (dans les deux directions) par le même agent.

On suppose que les agents se trouvant en même temps au même point de la droite échangent automatiquement toute information possédée. Nous disons que le schéma de commérage est *correct* si après avoir complété tous les mouvements d'agents mobiles, chaque agent possède l'information étant l'union des informations possédées initialement par tous les agents.

Le schéma de commérage est *optimal* si son coût est inférieur ou égal à celui de tous les autres schémas de commérage corrects.

Par *l'algorithme de commérage,* on comprend une procédure, qui, ayant en entrée le nombre d'agents, produit un schéma de commérage correct.

L'exemple suivant sert à donner l'intuition comment l'algorithme de commérage se déroule. Cet exemple illustre les concepts définis ci-dessus.

On suppose que l'on possède l'information que le commérage est faisable avec un coût de $C = 1.25$.

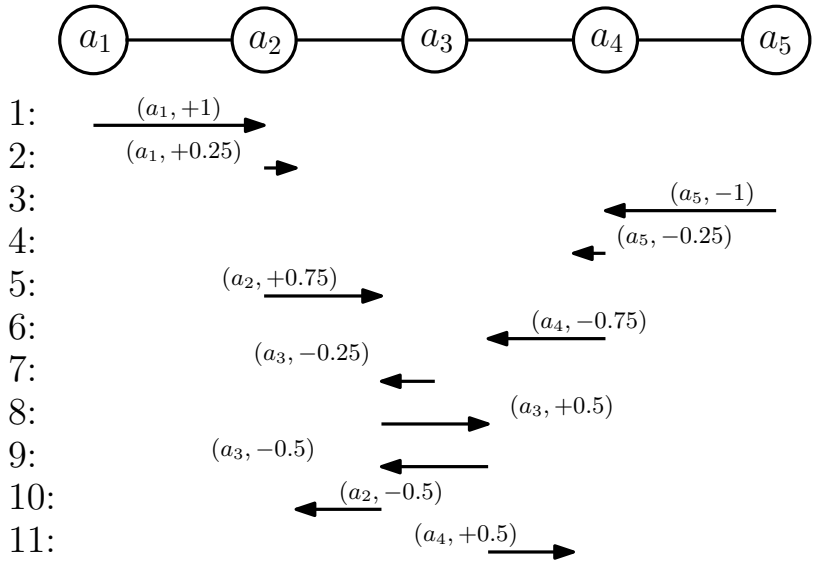

FIGURE 3.1 – 5 agents sur la ligne - déplacement de tous les agents

Nous observons que les agents extrêmes dans l'algorithme optimal se déplacent dans une seule direction (vers le milieu) donc, ils dépasseront leurs voisins immédiats.

Les agents a_2 et a_4 feront deux segments de parcours, le premier pour avancer et chercher les messages et le deuxième pour revenir vers les agents extrêmes et partager les messages avec eux.

L'agent du milieu, a_3, se promène dans les deux directions pour faire un pont entre les agents a_2 et a_4.

Il est préférable de représenter le schéma de commérage en forme de tableau (voir Table 3.1) - cela permet plus facilement de prouver son exactitude en tenant compte de l'état mémoire de chaque agent au bout de chaque mouvement du schéma.

Étape	Agent	Distance à parcourir	État Mémoire	Coût
1	a_1	$+1$	$m(a_1) = m(a_2) = \{1, 2\}$	$c(a_1) = 1$
2	a_1	$+0.25$	$m(a_1) = \{1, 2\}$	$c(a_1) = 1.25$
3	a_5	-1	$m(a_5) = m(a_4) = \{4, 5\}$	$c(a_5) = 1$
4	a_5	-0.25	$m(a_5) = \{4, 5\}$	$c(a_5) = 1.25$
5	a_2	$+0.75$	$m(a_2) = \{1, 2\}$	$c(a_2) = 0.75$
6	a_4	-0.75	$m(a_4) = \{4, 5\}$	$c(a_4) = 0.75$
7	a_3	-0.25	$m(a_3) = m(a_2) = \{1, 2, 3\}$	$c(a_3) = 0.25$
8	a_3	$+0.5$ (consolidation)	$m(a_3) = m(a_4) = \{1, 2, 3, 4, 5\}$	$c(a_3) = 0.75$
9	a_3	-0.5	$m(a_3) = m(a_2) = \{1, 2, 3, 4, 5\}$	$c(a_3) = 1.25$
10	a_2	-0.5	$m(a_2) = m(a_1) = \{1, 2, 3, 4, 5\}$	$c(a_2) = 1.25$
11	a_4	$+0.5$	$m(a_4) = m(a_5) = \{1, 2, 3, 4, 5\}$	$c(a_4) = 1.25$

TABLE 3.1 – Schéma de commérage pour 5 agents

Avant de présenter les algorithmes qui feront le commérage entre les agents, on fait les observations suivantes qui permettront de mieux comprendre le déroulement des algorithmes.

Observation 1. Dans un schéma de commérage optimal, chaque agent extrême se déplace dans une direction, vers le milieu, à une distance maximale.

Observation 2. Il existe un point R sur la ligne (*point de consolidation*) et un temps t_R (*moment de consolidation*) ou un agent a_i (comme premier de l'ensemble) présent au point R commence à posséder l'union des informations initiales de tous les agents.

Observation 3. Avant le temps t_R, l'algorithme exécute la *phase de consolidation* : l'information de chaque agent se dirige vers le point R (si l'information circule dans l'autre sens, on peut l'ignorer).

Observation 4. Après le temps t_R, l'algorithme exécute la *phase de distribution* : l'information consolidée est répartie à travers l'ensemble des agents.

Observation 5. Les agents critiques sont les agents extrêmes. Effectivement, si durant la phase de consolidation l'information initiale d'un agent extrême se rend au point de consolidation R alors l'information des autres agents se trouvant sur son chemin sera aussi transportée au point R. De façon similaire, si l'information complète est transportée vers l'agent extrême, alors tous les autres agents rencontrés durant ce transfert l'obtiennent aussi.

Par *diagramme de commérage* (voir figure 3.1) on va comprendre une image qui contient les positions initiales de chaque agent suivi par une suite de lignes numérotées, chaque ligne i contenant une paire (a_i, d) correspondant au mouvement de l'agent a à

distance d durant le i-ème étape du schéma de commérage ainsi qu'une flèche dont la longueur est la direction représentent ce mouvement.

Chapitre 4

Borne inférieure générale et optimalité du commérage pour les cas de trois et quatre agents

4.1 Borne inférieure générale.

Ce chapitre présente la borne inférieure générale pour un ensemble de n agents, c'est-à-dire le coût minimal que doit avoir chaque algorithme correct de commérage.

Théorème 4.1.1. *Considérons n agents sur la ligne $a_1, a_2, ..., a_n$ tel que la position initiale de l'agent a_i est le point i de la ligne. La borne inférieure pour le coût de n'importe quel algorithme qui fait le commérage des n agents sur la ligne avec une distance égale à 1 entre agents est : $2 - \frac{4}{n}$*

Preuve 4.1.1. . *En partant des observations faites dans le chapitre 3, nous analyserons le comportement que doivent avoir les agents.*

Considérons d'abord les agents de la tête et la queue de notre ensemble d'agents (agents extrêmes). Quand un tel agent visite son voisin pour lui transmettre son message, il n'a simplement pas besoin de revenir après : il n'y a pas un autre agent avec qui partager les messages reçus. Donc le segment entre un agent extrême et son voisin doit être parcouru au moins une fois.

Prenons maintenant n'importe quel segment $[i, i+1]$ entre deux agents consécutifs a_i et a_{i+1} pour $3 \leqslant i \leqslant n-2$.

Ce segment doit être parcouru deux fois, une fois durant la phase de consolidation et la deuxième fois durant la phase de distribution.

Finalement considérons les segments $[2,3]$ *et* $[n-2,n-1]$*. Observons que les trajectoires des agents parcourues de ces segments totalisent la distance de 2 au moins. Par symétrie, prenons le segment* $[2,3]$*. Deux cas sont possibles dépendemment si l'agent extrême* a_1 *traverse le point 2 ou non. Dans le premier cas, lorsque l'agent* a_1 *arrive au point* x*, tel que* $2 < x < 3$ *alors le segment* $[2,x]$ *est traversé par chacun des agents* a_1, a_2 *et le segment* $[x,3]$ *est traversé une fois durant la phase de consolidation et une autre fois durant la phase de distribution.*

Si l'agent a_1 *ne traverse jamais le point 2 alors le segment* $[2,3]$ *est traversé deux fois, une fois durant la phase de consolidation et une autre fois durant la phase de distribution (pour porter l'information consolidée à l'agent* a_1 *attendant à gauche du point 2).*

Conséquemment, deux segments extrêmes sont traversés complètement au moins une fois et tous les autres segments inter-agents (au nombre de $n-3$*) sont parcourus entièrement au moins deux fois. Alors la somme des longueurs des trajectoires de tous les agents effectuant le commérage, est au moins* $2(n-3)+2$*. Dans ce cas l'agent qui doit parcourir la plus grande distance doit traverser au moins leur moyenne arithmétique égale à* $\frac{2(n-3)+2}{n} = 2 - \frac{4}{n}$*, ce qui termine la preuve du théorème.*

Dans ce chapitre, nous définissons les schémas de commérage pour les cas de trois agents et quatre agents. Nous prouvons que les schémas proposés sont corrects. Nous prouvons aussi que ces deux schémas sont optimaux c'est-à-dire qu'il n'existe aucun schéma de commérage avec un coût inférieur.

4.2 Le cas de trois agents

4.2.1 Schéma de commérage pour trois agents et son coût

Il est évident que le commérage peut être facilement achevé avec le coût égal à 1, les agents a_1 et a_3 arrivent à la position de l'agent a_2.

Cependant il existe un schéma de commérage au coût de $\frac{5}{6}$ donné sur le diagramme de commérage illustré dans la figure 4.1.

Calcul du coût par agent (tous les coûts sont égaux) :

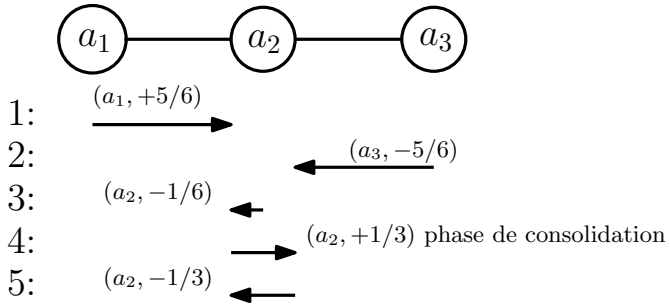

FIGURE 4.1 – 3 Agents sur la ligne et leurs trajectoires

$$
\begin{aligned}
c(a_1) &= d_1 \\
c(a_3) &= d_2 \\
c(a_2) &= d_3 + d_4 + d_5 \\
d_3 &= 1 - d_1 \\
d_4 &= d_3 + 1 - d_2 \\
d_5 &= d_4 \\
c(a_2) &= 5(1 - d_1) \\
c(a_1) &= c(a_2) \\
d_1 &= 5(1 - d_1) = 1 - \frac{1}{6} = \frac{5}{6} \\
c(a_1) &= c(a_2) = c(a_3)
\end{aligned}
$$

Coût de l'algorithme : $C = max\{c(a_1), c(a_2), c(a_3)\} = max\{\frac{5}{6}, \frac{5}{6}, \frac{5}{6}\} = \frac{5}{6}$

Étape	Agent	Distance à parcourir	État Mémoire	Coût
1	a_1	$+5/6$	$m(a_1) = \{1\}$	$c(a_1) = 5/6$
2	a_3	$-5/6$	$m(a_3) = \{3\}$	$c(a_3) = 5/6$
3	a_2	$-1/6$	$m(a_2) = m(a_1) = \{1, 2\}$	$c(a_2) = 1/6$
4	a_2	$+1/3$ consolidation	$m(a_2) = m(a_3) = \{1, 2, 3\}$	$c(a_2) = 1/2$
5	a_2	$-1/3$	$m(a_2) = m(a_1) = \{1, 2, 3\}$	$c(a_2) = 5/6$

TABLE 4.1 – Schéma de commérage pour le cas de 3 agents

4.2.2 Preuve d'exactitude

À la fin de l'étape 2, les agents a_1 et a_3 se retrouvent à une distance $1/6$ de a_2. À la fin de l'étape 3, l'agent a_2 rencontre a_1 et chacun d'eux possède l'union des informations initiales de a_1 et a_2. À la fin de l'étape 4, a_2 rencontre a_3 au point de consolidation et à ce moment a_2 et a_3 possèdent l'information totale $\{1, 2, 3\}$. L'étape 5 résulte du transfert de cette information à l'agent a_1 et l'état de mémoire de chaque agent est donc $\{1, 2, 3\}$.

4.2.3 Preuve d'optimalité

Par observation 1 (chapitre 3), dans chaque algorithme optimal on peut supposer que les agents a_1 et a_3 font des mouvements vers le milieu à une certaine distance x. Supposons que $x < 5/6$. À ce moment l'agent a_2 est le seul responsable à compléter le commérage et sa distance d à a_1 et a_3 est $d > 1/6$. Alors a_2, doit d'abord chercher l'information d'un de ses voisins (supposons a_1, pour symétrie), en parcourant la distance d. Ensuite, a_2 doit aller vers a_3 (au point de consolidation) en parcourant la distance $2d$. Finalement a_2 doit retourner vers a_1 pour réaliser la phase de distribution, en parcourant encore une fois la distance $2d$. La distance totale parcourue par a_2 est donc $d+2d+2d = 5d > 5/6$. Alors l'hypothèse que $x < 5/6$ conduit à un algorithme moins efficace.

4.3 Le cas de 4 agents

4.3.1 Schéma de commérage et son coût pour quatre agents

Le schéma est donné sur le diagramme de commérage de la figure 4.2.

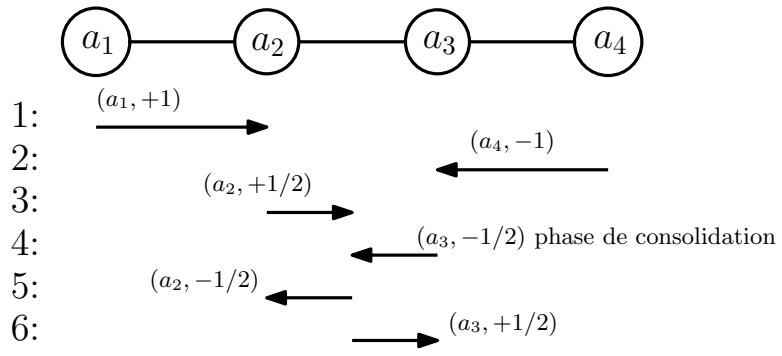

FIGURE 4.2 – 4 Agents sur la ligne et leurs trajectoires

Calcul du coût par agent (tous les coûts sont égaux) :

$$\begin{aligned} c(a_1) &= d_1 = 1 \\ c(a_4) &= d_2 = 1 \\ c(a_2) &= d_3 + d_5 = 1/2 + 1/2 = 1 \\ c(a_3) &= d_4 + d_6 = 1/2 + 1/2 = 1 \end{aligned}$$

Coût de l'algorithme : $C = max\{c(a_1), c(a_2), c(a_3), c(a_4)\} = max\{1, 1, 1, 1\} = 1$

Étage	Agent	Distance à parcourir	État Mémoire	Coût
1	a_1	$+1$	$m(a_1) = m(a_1) = \{1, 2\}$	$c(a_1) = 1$
2	a_4	-1	$m(a_4) = m(a_3) = \{3, 4\}$	$c(a_4) = 1$
3	a_2	$+1/2$	$m(a_2) = \{1, 2\}$	$c(a_2) = 1/2$
4	a_3	$-1/2$ consolidation	$m(a_2) = m(a_3) = \{1, 2, 3, 4\}$	$c(a_3) = 1/2$
5	a_2	$-1/2$	$m(a_2) = m(a_1) = \{1, 2, 3, 4\}$	$c(a_2) = 1$
6	a_3	$+1/2$	$m(a_3) = m(a_4) = \{1, 2, 3, 4\}$	$c(a_3) = 1$

TABLE 4.2 – Schéma de commérage pour 4 agents

4.3.2 Preuve d'exactitude

L'agent a_1 parcourt la distance d_1, $(d_1 = 1)$, de la gauche vers la droite puis s'arrête. Les agents a_1 et a_2 partagent leurs messages $m(a_1) = m(a_2) = \{1, 2\}$.

L'agent a_4 parcourt la distance d_2, $(d_2 = 1)$, de la droite vers la gauche, et s'arrête. Les agents a_3 et a_4 partagent leurs messages $m(a_3) = m(a_4) = \{3, 4\}$. L'agent a_2 se déplace à distance $d_3 = 1/2$ vers la droite pour rencontrer l'agent a_3. L'agent a_3 se déplace vers la gauche à distance $d_4 = 1/2$ pour rencontrer l'agent a_2. C'est le point de consolidation : les agents a_2 et a_3 partagent leurs messages, $m(a_2) = m(a_3) = \{1, 2, 3, 4\}$. L'agent a_3 se déplace vers la droite pour visiter l'agent a_4 à distance $d_5 = 1/2$, les agents a_3 et a_4 partagent leurs messages, $m(a_3) = m(a_4) = \{1, 2, 3, 4\}$. L'agent a_2 se déplace ensuite vers la gauche à distance $d_6 = 1/2$, transmet ses messages à l'agent 1, et s'arrête, $m(a_1) = m(a_2) = m(a_3) = m(a_4) = \{1, 2, 3, 4\}$.

4.3.3 Preuve d'optimalité

La preuve d'optimalité suit la preuve de la borne inférieure générale. Effectivement d'après le théorème 4.1.1 le coût de chaque schéma de commérage pour n agents est au moins $2 - 4/n$ ce qui coïncide avec le coût du schéma de la table 4.2 pour $n = 4$.

Chapitre 5

Mouvement d'agents réalisant le commérage en cas général

Ce chapitre introduit les types de mouvements que doivent exécuter les agents, dépendement de leurs positions sur la ligne. L'algorithme produisant le schéma de commérage qui effectue ces mouvements sera présenté dans le chapitre 6.

Les agents qui se promènent sur la ligne pour échanger leurs informations utiliseront un des trois types de mouvement suivants : *Aller*, *Aller-Retour*, *Serpentine*. L'utilisation d'un des types de mouvement pour un agent dépend de sa position sur la ligne, ainsi que du point où se trouve l'information qu'il a ramassée de son voisin.

Dans le cas du mouvement de type *Aller* il s'agit d'un mouvement dans lequel l'agent bouge dans une seule direction, et il est utilisé uniquement pour les agents extrêmes. La direction du déplacement coïncide avec la position de son voisin. Le coût prévu du mouvement *Aller* indique à l'agent s'il doit faire le mouvement en une seule étape, (si le coût est inférieur à 1), ou en deux étapes, (si le coût est supérieur à 1). Dans le cas du mouvement *Aller* en 2 étapes, pendant la première étape l'agent extrême visite son voisin, et leurs messages sont partagés. Par la suite l'agent extrême continu en dépassant son agent voisin jusqu'à ce que son coût soit épuisé.

Le type de mouvement *Aller-Retour*, (par exemple voir les agents a_2 et a_4 dans la figure 3.1), est utilisé par un agent chaque fois que l'agent a été dépassé par son voisin dans la phase de consolidation.

Le dernier type de mouvement *Serpentine*, (par exemple voir l'agent a_3 dans la figure 3.1) est utilisé par les agents qui n'ont pas été dépassés par les agents voisins pendant la phase de consolidation. L'agent qui réalise ce type de mouvement doit "chercher" l'in-

formation se trouvant chez son voisin, la ramener le plus loin possible tout en conservant assez de coût (puissance de sa batterie) afin de revenir à la position de ce voisin durant la phase de distribution. Dans ce type de mouvement, il y aura toujours un morceau de chemin qui sera parcouru trois fois.

L'exemple de la figure 3.1 illustre les trois types de mouvements : *Aller* (agents a_1 et a_5), *Aller-Retour* (agents a_2 et a_4) et *Serpentine* (agent a_3). Afin de traiter le cas pour le nombre n quelconque d'agents on peut observer que dans le schéma de commérage optimal chaque agent réalise un mouvement avec un coût identique. Tous les agents internes (non extrêmes) participent dans la phase de consolidation alors chacun d'eux doit suivre le mouvement de type *Aller-Retour* ou *Serpentine*. Le nombre d'agents en *Aller-Retour* et *Serpentine* dépend de n, mais il est évident que tous les agents symétriques ont des trajectoires symétriques (l'une est la copie miroir de l'autre). En conséquence, il suffit de considérer le mouvement de la moitié (droite ou gauche) d'agents. Premièrement on supposera que l'agent central de la suite ne bouge pas du tout. Autrement dit, la suite d'agents $a_1, ..., a_n$ doit transporter l'information de l'agent a_1 à l'agent immobile a_n et ensuite les agents $a_{n-1}, a_{n-2}, ..., a_2$ doivent conserver suffisamment d'énergie (ou coût) pour transférer l'information obtenue de l'agent a_n à l'agent a_1. Le comportement d'une telle suite d'agents est présenté dans la section 5.1.

Dans la suite, en fonction du nombre total d'agents, on va calculer le coût, le nombre d'agents qui font le mouvement *Aller-Retour* et le nombre d'agents qui feront le mouvement *Serpentine*. Deux fonctions sont présentées, une fonction quand le nombre d'agents est pair et une autre lorsque le nombre d'agents est impair. L'ensemble des données trouvées est utilisé par l'algorithme général pour produire le schéma de commérage.

Selon le nombre d'agents, la symétrie sur la ligne va se trouver soit dans un agent, dans le cas impair ou au milieu de la ligne entre deux agents centraux pour un cas pair.

Deux distances D_1 et D_2 seront calculées. La distance D_1 est mesurée à partir du point 0 de la ligne (position de l'agent extrême gauche) vers le centre. La distance D_1 indique à quel point sur la ligne l'information a été apporté par le dernier agent qui fait le mouvement *Aller-Retour* pendant la phase de consolidation. La deuxième distance a calculer, D_2, est parcourue par l'ensemble d'agents avec le mouvement *Serpentine* pendant la phase de consolidation.

Seulement quand la condition $D_1 + D_2 \geq \lfloor \frac{n}{2} \rfloor$, (cas impair) ou $D_1 + D_2 \geq \frac{n}{2} - 1$, (cas pair), est vrai, le schéma de commérage est correct. Intuitivement on peut dire que

l'information a été "poussée" par les agents *Aller-Retour* à un point ou les agents qui font le mouvement *Serpentine* ont réussi à la prendre et la transporter au milieu.

Le nombre d'agents qui feront le mouvement *Aller-Retour*, incluant l'agent extrême, sera dénoté par α. Le nombre d'agents faisant le mouvement *Serpentine* sera dénoté par γ.

5.1 Mouvement d'agents avec le dernier agent immobile

Dans cette section nous allons calculer quelle est la proportion d'agents qui exécutent le mouvement de type *Aller-Retour* (les autres agents feront un mouvement en *Serpentine*) en supposant que le dernier agent de la séquence doit rester immobile. Il s'avère que cette proportion change en fonction du nombre total d'agents n. On supposera que tous les agents ont le même coût.

L'algorithme décrit ci-dessous permet d'obtenir un coût égal à $C = 2 - \frac{1}{2^{(n-2)}}$.

Le coût obtenu n'est pas optimal, mais il décrit complètement le mouvement d'agents qui suivent un mouvement *Aller-Retour*. Les résultats de cet algorithme sont utilisés dans la suite pour déterminer le nombre maximal d'agents qui peuvent faire le mouvement *Aller-Retour* avec un coût C' donné au départ.

L'agent a_1 visite son voisin à droite, partage son message avec ce dernier et continue d'avancer sur une distance $d_2 < 1$ et s'arrête. L'agent a_2 visite son voisin à droite, partage son message avec lui $m(2) = m(3) = \{1, 2, 3\}$, puis il se déplace en avançant d'une distance d_4 et s'arrête. L'agent a_3 visite son voisin à droite, partage son message avec lui $m(3) = m(4) = \{1, 2, 3, 4\}$; il avance ensuite d'une distance d_6 et s'arrête. Ce comportement se répète jusqu'à l'agent a_{n-2} ayant en mémoire $m(n-2) = \{1, 2, 3, \ldots, n-1\}$ à la fin de son parcours. L'agent a_{n-1} a en mémoire $m(n-1) = \{1, 2, 3, \ldots, n-1\}$ avant de visiter l'agent a_n. Une fois la visite effectuée, l'agent a_n possède tous les messages $m(n) = m(n-1) = \{1, 2, 3, \ldots, n\}$. L'agent a_n n'a pas bougé. La phase de consolidation est terminée et le reste des mouvements constituent la phase de distribution. L'agent a_{n-1} après avoir visité son voisin de gauche (a_{n-2} dans sa position actuelle), partage son message avec celui-ci et s'arrête $m(n-2) = m(n-1) = \{1, 2, 3, \ldots, n\}$. Le reste des agents attend la visite de leurs voisins de droite avant de visiter leurs voisins de gauche.

Quand l'agent a_2 visite l'agent a_1, l'algorithme se termine ; les agents ont reçu tous les messages $m(1) = m(2) = m(3) = \ldots = m(n) = \{1, 2, 3, \ldots, n\}$.

La figure 5.1 et le tableau 5.1 illustrent le mouvement pour le cas de 8 agents avec a_8 restant immobile.

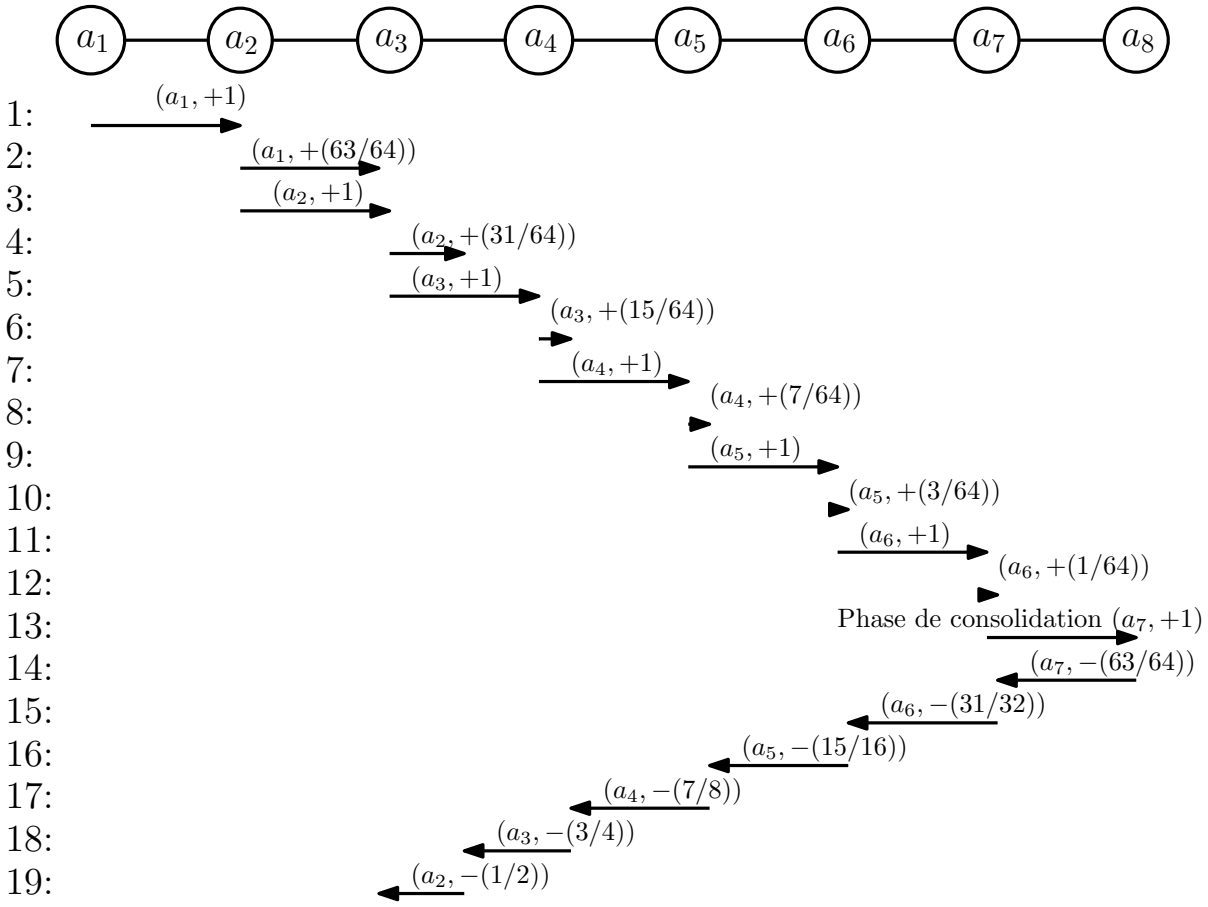

FIGURE 5.1 – Mouvement de 8 agents sur la ligne et leurs trajectoires en supposant l'agent a_8 immobile

Chaque agent a effectué un parcours supplémentaire (étapes 2,4,6,8,10,et 12), permettant d'améliorer la valeur du coût maximal. Le calcul effectué pour chacun de ces agents se présente ainsi :

Étage	Agent	Dist à parcourir	État Mémoire	Coût
1	a_1	$+1$	$m(a_1) = m(a_1) = \{1, 2\}$	$c(a_1) = 1$
2	a_1	$+(63/64)$	$m(a_1) = \{1, 2\}$	$c(a_1) = \frac{127}{64}$
3	a_2	$+1$	$m(a_2) = m(a_3) = \{1, 2, 3\}$	$c(a_2) = 1$
4	a_2	$+(31/64)$	$m(a_2) = \{1, 2, 3\}$	$c(a_2) = \frac{113}{64}$
5	a_3	$+1$	$m(a_3) = m(a_4) = \{1, 2, 3, 4\}$	$c(a_3) = 1$
6	a_3	$+(15/64)$	$m(a_3) = \{1, 2, 3, 4\}$	$c(a_3) = \frac{79}{64}$
7	a_4	$+1$	$m(a_4) = m(a_5) = \{1, 2, 3, 4, 5\}$	$c(a_4) = 1$
8	a_4	$+(7/64)$	$m(a_4) = \{1, 2, 3, 4, 5\}$	$c(a_4) = \frac{71}{64}$
9	a_5	$+1$	$m(a_5) = m(a_6) = \{1, 2, 3, 4, 5, 6\}$	$c(a_5) = 1$
10	a_5	$+(3/64)$	$m(a_5) = \{1, 2, 3, 4, 5, 6\}$	$c(a_5) = \frac{67}{64}$
11	a_6	$+1$	$m(a_6) = m(a_7) = \{1, 2, 3, 4, 5, 6, 7\}$	$c(a_6) = 1$
12	a_6	$+(1/64)$	$m(a_6) = \{1, 2, 3, 4, 5, 6, 7\}$	$c(a_6) = \frac{65}{64}$
13	a_7	$+1$ consolidation	$m(a_7) = m(a_8) = \{1, 2, 3, 4, 5, 6, 7, 8\}$	$c(a_7) = 1$
14	a_7	$-(63/64)$	$m(a_7) = m(a_6) = \{1, 2, 3, 4, 5, 6, 7, 8\}$	$c(a_7) = \frac{127}{64}$
15	a_6	$-(31/32)$	$m(a_6) = m(a_5) = \{1, 2, 3, 4, 5, 6, 7, 8\}$	$c(a_6) = \frac{127}{64}$
16	a_5	$-(15/16)$	$m(a_5) = m(a_4) = \{1, 2, 3, 4, 5, 6, 7, 8\}$	$c(a_5) = \frac{127}{64}$
17	a_4	$-(7/8)$	$m(a_4) = m(a_3) = \{1, 2, 3, 4, 5, 6, 7, 8\}$	$c(a_4) = \frac{127}{64}$
18	a_3	$-(3/4)$	$m(a_3) = m(a_2) = \{1, 2, 3, 4, 5, 6, 7, 8\}$	$c(a_3) = \frac{127}{64}$
19	a_2	$-(1/2)$	$m(a_2) = m(a_1) = \{1, 2, 3, 4, 5, 6, 7, 8\}$	$c(a_2) = \frac{127}{64}$

TABLE 5.1 – Schéma de commérage avec 8 agents, valeurs exactes

$$
\begin{aligned}
c(a_1) &= 1 + d_2 \\
c(a_2) &= 1 + d_4 + d_{19} \\
c(a_3) &= 1 + d_6 + d_{18} \\
c(a_4) &= 1 + d_8 + d_{17} \\
c(a_5) &= 1 + d_{10} + d_{16} \\
c(a_6) &= 1 + d_{12} + d_{15} \\
c(a_7) &= 1 + d_{14} \\
c(a_8) &= 0
\end{aligned}
$$

Afin de minimiser le coût maximal, on va balancer le parcours de chaque agent, soit $c(a_1) = c(a_2) = c(a_3) = \ldots = c(a_{n-1})$:

$$\begin{aligned}
d_{19} &= d_4 + 1 - d_2 \\
d_{18} &= d_6 + 1 - d_4 \\
d_{17} &= d_8 + 1 - d_6 \\
d_{16} &= d_{10} + 1 - d_8 \\
d_{15} &= d_{12} + 1 - d_{10} \\
d_{14} &= 1 - d_{12} \\
c(a_1) &= 1 + d_2 \\
c(a_2) &= 1 + d_4 + d_{19} = 1 + d_4 + d_4 + 1 - d_2 = 2 + 2d_4 - d_2 \\
d_4 &= d_2 - \frac{1}{2} \\
c(a_3) &= 1 + d_6 + d_{18} = 1 + d_6 + d_6 + 1 - d_4 = 2 + 2d_6 - d_1 \\
d_6 &= d_2 - \frac{3}{4} \\
d_8 &= d_2 - \frac{7}{8} \\
d_{10} &= d_2 - \frac{15}{16} \\
d_{12} &= d_2 - \frac{31}{32} \\
c(a_1) &= c(a_7) \\
1 + d_2 &= 1 + d_{14} \\
d_2 &= d_{14} \\
d_2 &= \frac{63}{64} \\
d_2 &= \frac{2^{(n-2)} - 1}{2^{(n-2)}} \\
c(a_1) &= 1 + d_2 = \frac{127}{64} \\
c(a_2), ..., c(a_7) &= \frac{127}{64} \\
c(a_8) &= 0
\end{aligned}$$

Le coût $C = max\{c(a_1), c(a_2), \ldots, c(a_8)\} = \{\frac{127}{64}, \frac{127}{64}, \ldots, 0\} = \frac{127}{64}$.

5.2 Calcul des nombres d'agents en mouvement *Aller-Retour* et en mouvement *Serpentine*

Afin d'évaluer la proportion d'agents en mouvement *Aller-Retour* par rapport aux agents en mouvement *Serpentine* on appliquera un raisonnement inverse. On supposera que la valeur de coût C est donné au départ. En fonction de C on va calculer quel est le nombre maximal d'agents pouvant faire un mouvement *Aller-Retour* avec le coût C. Ensuite on va calculer quel est le nombre d'agents en mouvement *Serpentine*. Par symétrie on prendra en considération la moitié d'agents se trouvant à gauche de l'axe de symétrie.

5.2.1 Agents en mouvement *Aller-Retour*

Dans l'algorithme de la section 5.1, on a eu comme résultat un coût égal à $C = 2 - 1/2^{(\alpha-2)}$. Cette équation, est réécrite pour y avoir α comme résultat, on obtient le nombre d'agents qui pourront faire le mouvement *Aller-Retour* :

$$\begin{aligned} C &= 2 - 1/2^{(\alpha-2)} \\ \alpha &= \lfloor \log_2(1/(2-C)) + 2 \rfloor \end{aligned} \tag{5.1}$$

Note : α peut prendre seulement des valeurs réelles positives.

Relation entre le nombre d'agents en mouvement *Aller-Retour* et le coût C

Avec le nombre d'agents qui peuvent faire le mouvement *Aller-Retour*, on calcule la distance réelle D_1 que vont parcourir les α agents sur la ligne. Le déplacement total fait par les α agents sur la ligne est donné par l'équation :

$$\begin{aligned} D_1 &= C + (1/2) + (3/4) + (7/8) + (15/16) + \ldots + \left(\frac{2^{(\alpha-1)} - 1}{2^{(\alpha-1)}}\right) \\ D_1 &= C + \sum_{i=1}^{\alpha-1} \left(\frac{2^i - 1}{2^i}\right) = C + \alpha - \left(\frac{1 - (1/2)^\alpha}{1 - (1/2)}\right) \end{aligned} \tag{5.2}$$

Distance D_1 parcourue par les agents *Aller-Retour* sur la ligne

Il faut remarquer que l'agent extrême est inclus dans le compte d'agents α.

5.2.2 La distance totale parcourue par les agents en mouvement *Serpentine*

Le mouvement de *Serpentine* a été décrit dans l'algorithme optimal pour 3 agents (voir 4.2.1). L'algorithme démontre que si l'agent revient en arrière pour chercher l'information qui n'est pas arrivée à lui, le coût de l'algorithme est amélioré. Mais combien d'agents doivent faire ce type de mouvement dans la ligne et avec un coût C donné ? Il s'agit du nombre d'agents qui n'ont pas bougé encore, soit $\lfloor \frac{n}{2} \rfloor + 1 - \alpha$ quand la valeur de n est impair et $\frac{n}{2} - \alpha$ quand n est pair. Cette valeur sera notée comme γ. Il faut trouver une distance sur la ligne, D_2, qui sera parcourue par les agents de type *Serpentine*. La distance D_2 est mesurée à partir de la position de l'agent du milieu vers sa gauche.

La figure suivante sera utilisée dans le cas pair et impair, mais l'ensemble des équations est légèrement différent dans chaque cas.

Pour le cas impair, le nombre d'agents donné dans l'exemple sur la figure 5.2 est 11, et pour le cas pair, il y a 12 agents.

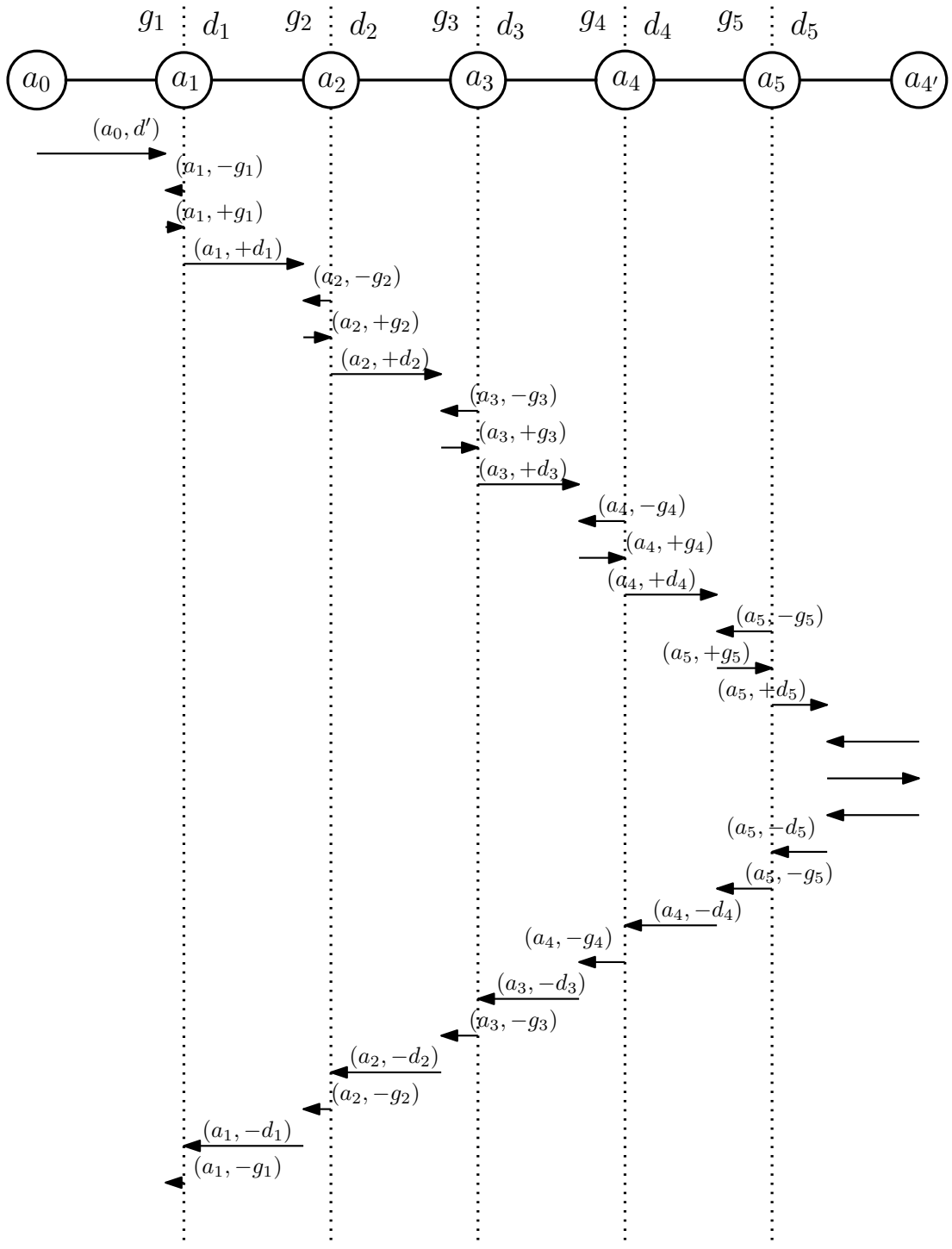

FIGURE 5.2 – Mouvement d'agents de type *Serpentine*

Dans le cas impair, les équations qui décrivent les mouvements d'agents dans la figure 5.2 (11 agents) sont :

$$\begin{aligned}
3(g_1) + 2(d_1) &= C \\
3(g_2) + 2(d_2) &= C \\
3(g_3) + 2(d_3) &= C \\
3(g_4) + 2(d_4) &= C \\
3(g_5) + 2(d_5) &= C \\
\text{Avec } d_k = (1 - g_{k+1}) &\text{ et } g_5 = d_5 \text{ on obtient :} \\
3(g_1) + 2(1 - g_2) &= C \\
3(g_2) + 2(1 - g_3) &= C \\
3(g_3) + 2(1 - g_4) &= C \\
3(g_4) + 2(1 - g_5) &= C \\
5(g_5) &= C
\end{aligned}$$

En résolvant le système d'équations :

$$\begin{aligned}
g_5 &= \frac{C}{5} \\
g_4 &= \left(\frac{C-2}{3}\right) + \left(\frac{2}{3}\right) = \left(\frac{C-2}{3}\right) + \left(\frac{2}{3}\right)\left(\frac{C}{5}\right) \\
g_3 &= \left(\frac{C-2}{3}\right) + \left(\frac{2}{3}\right) g_4 = \left(\frac{C-2}{3}\right)\left(1 + \left(\frac{2}{3}\right)\right) + \left(\frac{2}{3}\right)^2\left(\frac{C}{5}\right) \\
g_2 &= \left(\frac{C-2}{3}\right) + \left(\frac{2}{3}\right) g_3 = \left(\frac{C-2}{3}\right)\left(1 + \left(\frac{2}{3}\right) + \left(\frac{2}{3}\right)^2\right) + \left(\frac{2}{3}\right)^3\left(\frac{C}{5}\right) \\
g_1 &= \left(\frac{C-2}{3}\right) + \left(\frac{2}{3}\right) g_2 = \left(\frac{C-2}{3}\right)\left(1 + \left(\frac{2}{3}\right) + \left(\frac{2}{3}\right)^2 + \left(\frac{2}{3}\right)^3\right) + \left(\frac{2}{3}\right)^4\left(\frac{C}{5}\right)
\end{aligned}$$

Dans notre exemple il y a eu 5 agents qui ont bougé avec le mouvement de type *Serpentine*. L'équation pour la distance parcourue sur la ligne par les agents avec le mouvement *Serpentine* pour n impair est :

$$(5-1) + \left(\frac{C-2}{3}\right)\left(1 + \left(\frac{2}{3}\right) + \left(\frac{2}{3}\right)^2 + \left(\frac{2}{3}\right)^3\right) + \left(\frac{2}{3}\right)^4\left(\frac{C}{5}\right) \tag{5.3}$$

Le portion d'équation $(5-1)$ (équation (5.3)) est la distance déjà parcourue par l'information de droite vers gauche par les agents avec le mouvement *Serpentine*. Si l'équation pour la distance parcourue est en fonction de γ (nombre d'agents avec le mouvement *Serpentine*) on obtient :

$$\begin{aligned} D_2 &= (\gamma-1) + \left[\left(\frac{C-2}{3}\right)\left(\sum_{i=0}^{\gamma-2}\left(\frac{2}{3}\right)^i\right) + \left(\frac{2}{3}\right)^{(\gamma-1)}\left(\frac{C}{5}\right)\right] \\ D_2 &= (\gamma-1) + \left[\left(\frac{C-2}{3}\right)\left(\frac{1-(2/3)^{\gamma-1}}{1-(2/3)}\right) + \left(\frac{2}{3}\right)^{(\gamma-1)}\left(\frac{C}{5}\right)\right] \end{aligned} \tag{5.4}$$

L'équation (5.4) présente la distance parcourue par les agents *Serpentine* sur la ligne, pour n impair.

Dans le cas ou le nombre d'agents est pair, les agents $a_{\frac{n}{2}}$ et $a_{\frac{n}{2}+1}$ auront comme point de rencontre le milieu de la ligne. Cette contrainte définit un nouveau système d'équations pour décrire le mouvement d'agents qui font le mouvement de *Serpentine*.

Les équations dans le cas pair sont :

$$\begin{aligned} 3(g_1) + 2(d_1) &= C \\ 3(g_2) + 2(d_2) &= C \\ 3(g_3) + 2(d_3) &= C \\ 3(g_4) + 2(d_4) &= C \\ 3(g_5) + 2(d_5) &= C \\ \text{Avec } d_k = (1-g_{k+1}) \text{ et } d_5 &= \frac{1}{2} \text{ on obtient :} \\ 3(g_1) + 2(1-g_2) &= C \\ 3(g_2) + 2(1-g_3) &= C \\ 3(g_3) + 2(1-g_4) &= C \\ 3(g_4) + 2(1-g_5) &= C \\ 3(g_5) &= C-1 \end{aligned} \tag{5.5}$$

En résolvant le système d'équations, on obtient :

$$
\begin{aligned}
g_5 &= \frac{C-1}{3} \\
g_4 &= \left(\frac{C-2}{3}\right) + \left(\frac{2}{3}\right) = \left(\frac{C-2}{3}\right) + \left(\frac{2}{3}\right)\left(\frac{C-1}{3}\right) \\
g_3 &= \left(\frac{C-2}{3}\right) + \left(\frac{2}{3}\right) g_4 = \left(\frac{C-2}{3}\right)\left(1 + \left(\frac{2}{3}\right)\right) + \left(\frac{2}{3}\right)^2\left(\frac{C-1}{3}\right) \\
g_2 &= \left(\frac{C-2}{3}\right) + \left(\frac{2}{3}\right) g_3 = \left(\frac{C-2}{3}\right)\left(1 + \left(\frac{2}{3}\right) + \left(\frac{2}{3}\right)^2\right) + \left(\frac{2}{3}\right)^3\left(\frac{C-1}{3}\right) \\
g_1 &= \left(\frac{C-2}{3}\right) + \left(\frac{2}{3}\right) g_2 = \left(\frac{C-2}{3}\right)\left(1 + \left(\frac{2}{3}\right) + \left(\frac{2}{3}\right)^2 + \left(\frac{2}{3}\right)^3\right) + \left(\frac{2}{3}\right)^4\left(\frac{C-1}{3}\right)
\end{aligned}
\tag{5.6}
$$

Dans notre exemple il y a eu 5 agents qui ont bougé avec le mouvement de *Serpentine*. L'équation pour la distance parcourue sur la ligne par les agents avec le mouvement de *Serpentine* pour n pair est :

$$
(5-1) + \left(\frac{C-2}{3}\right)\left(1 + \left(\frac{2}{3}\right) + \left(\frac{2}{3}\right)^2 + \left(\frac{2}{3}\right)^3\right) + \left(\frac{2}{3}\right)^4\left(\frac{C-1}{3}\right) \tag{5.7}
$$

Le portion d'équation $(5-1)$ (équation 5.7)) est la distance déjà parcourue par l'information de droite vers gauche par les agents avec le mouvement *Serpentine*. Si l'équation pour la distance parcourue en fonction de γ (nombre d'agents avec le mouvement *Serpentine*) on obtient :

$$
\begin{aligned}
D_2 &= (\gamma-1) + \left[\left(\frac{C-2}{3}\right)\left(\sum_{i=0}^{\gamma-2}\left(\frac{2}{3}\right)^i\right) + \left(\frac{2}{3}\right)^{(\gamma-1)}\left(\frac{C-1}{3}\right)\right] \\
D_2 &= (\gamma-1) + \left[\left(\frac{C-2}{3}\right)\left(\frac{1-(2/3)^{\gamma-1}}{1-(2/3)}\right) + \left(\frac{2}{3}\right)^{(\gamma-1)}\left(\frac{C-1}{3}\right)\right]
\end{aligned}
\tag{5.8}
$$

L'équation (5.8) illustre la distance parcourue par les agents *Serpentine* sur la ligne pour n pair. La distance D_2 calculée est mesurée à partir de la position initiale de l'agent du milieu le plus à gauche.

Il est important de noter que le calcul de distances de cette section implique trois observations :

Observation 6

Pendant la phase de *consolidation*, chaque agent a_i, pour $i = 2, 3, ..., \lfloor \frac{n}{2} \rfloor$ commence son mouvement à partir du point où est arrivé son voisin de gauche a_{i-1}. Également, par symétrie, chaque agent a_i, pour $i = \lfloor \frac{n}{2} \rfloor + 1, ..., n$ commence son mouvement à partir du point d'arrivée de son voisin de droite a_{i+1}.

Observation 7

Pendant la phase de *distribution*, chaque agent a_i, pour $i = 2, 3, ..., \lfloor \frac{n}{2} \rfloor - 1$ commence son mouvement à partir du point où est arrivé son voisin de droite a_{i+1}. Également, par symétrie, chaque agent a_i pour $i = \lceil \frac{n}{2} \rceil + 1, ..., n$ commence son mouvement à partir du point d'arrivée de son voisin gauche a_{i-1}.

Observation 8

1. Pour n pair, les deux agents du centre, $a_{\frac{n}{2}}$ et $a_{\frac{n}{2}+1}$ terminent la phase de consolidation en obtenant l'union des informations initiales de tous les agents et ensuite ils commencent la phase de distribution.
2. Pour n impair à la fin de la phase de consolidation l'agent du centre $a_{\frac{n+1}{2}}$ se rend vers l'un de ses voisins (disons par symétrie, voisin gauche $a_{\frac{n-1}{2}}$) pour obtenir l'information initiale de tous les agents à sa gauche. Ensuite $a_{\frac{n+1}{2}}$ se rend à son voisin de droite $a_{\frac{n+3}{2}}$ terminant la phase de consolidation. Les agents $a_{\frac{n+1}{2}}$ et $a_{\frac{n+3}{2}}$ commencent ensuite la phase de distribution.
3. Dans les deux cas ci-dessus la phase de consolidation se termine donc au point de commencement de la phase de distribution.

5.3 Coût du commérage par un ensemble de n agents, pour n impair

Les équations D_1 et D_2 peuvent être fusionnées dans une seule équation et utilisées dans un algorithme qui va chercher la valeur du coût optimal.

La fusion des deux équations donne :

$$
\begin{aligned}
\alpha + \gamma &= \left\lfloor \frac{n}{2} \right\rfloor + 1 \\
\gamma - 1 &= \left\lfloor \frac{n}{2} \right\rfloor - \alpha \\
\alpha &= \left\lfloor \log\left(\frac{1}{2-C}\right) + 2 \right\rfloor \\
\left\lfloor \frac{n}{2} \right\rfloor &\leqslant D_1 + D_2 \\
\left\lfloor \frac{n}{2} \right\rfloor &\leqslant C + \alpha - \left(\frac{1-\left(\frac{1}{2}\right)^{\alpha}}{1-\left(\frac{1}{2}\right)}\right) + (\gamma - 1) + \left[\left(\frac{C-2}{3}\right)\left(\frac{1-\left(\frac{2}{3}\right)^{\gamma-1}}{1-\left(\frac{2}{3}\right)}\right) + \left(\frac{2}{3}\right)^{(\gamma-1)}\left(\frac{C}{5}\right)\right] \\
\left(\frac{2}{3}\right)^{\left\lfloor \frac{n}{2} \right\rfloor} &\leqslant \frac{(2^{\alpha})(2-C)-1}{(3^{\alpha})\left(1-\frac{2C}{5}\right)} \\
\left\lfloor \frac{n}{2} \right\rfloor &\leqslant \frac{\log\left[(2^{\alpha})(2-C)-1\right] - \left[(\alpha)\log(3) + \log\left(1-\frac{2C}{5}\right)\right]}{\log\left(\frac{2}{3}\right)} \\
\left\lfloor \frac{n}{2} \right\rfloor &\leqslant \frac{\log\left[(2^{\left\lfloor \log\left(\frac{1}{2-C}\right)+2 \right\rfloor})(2-C)-1\right] - \left[\left\lfloor \log\left(\frac{1}{2-C}\right) + 2 \right\rfloor \log(3) + \log\left(1-\frac{2C}{5}\right)\right]}{\log\left(\frac{2}{3}\right)}
\end{aligned}
\tag{5.9}
$$

L'équation (5.9) va nous permettre de vérifier si un coût trouvé va vraiment faire bouger les agents correctement afin de partager les messages entre les agents. Donc, l'équation (5.9) peut être utilisée pour répondre à la question : avec un coût C donné, le commérage entre n agents peut-il être fait ? La réponse est oui ou non.

On pourrait utiliser l'équation (5.9) en utilisant la recherche binaire sur la valeur de C, et, trouver un C_{opt} qui ne sera pas exact, mais, l'erreur sera d'au plus ϵ par rapport à la valeur optimale, pour n'importe quelle valeur $\epsilon > 0$.

Pour trouver une valeur de coût sans erreur, on va faire la recherche binaire plutôt sur un plan discret. Dans notre cas la recherche binaire se fait sur le nombre d'agents α qui font le mouvement *Aller-Retour*.

Dans le développement de l'équation (5.9), on trouve une équation qui donne une relation entre le nombre d'agents n, et le nombre d'agents qui feront le mouvement *Aller-Retour* α et le coût C.

En partant de l'équation (5.9) :

$$\begin{aligned} \frac{(2^{\alpha})(2-C)-1}{(3^{\alpha})\left(1-\frac{2C}{5}\right)} &\geqslant \left(\frac{2}{3}\right)^{\lfloor \frac{n}{2} \rfloor} \\ C &\geqslant \frac{(3)^{\alpha}\left(\frac{2}{3}\right)^{\lfloor \frac{n}{2} \rfloor}+1-2\,(2)^{\alpha}}{(3)^{\alpha}\left(\frac{2}{3}\right)^{\lfloor \frac{n}{2} \rfloor}\left(\frac{2}{5}\right)-(2)^{\alpha}} \end{aligned} \tag{5.10}$$

Voici l'algorithme pour trouver le coût optimal et le nombre d'agents qui font le mouvement allez-retour, lorsque le nombre n d'agents est impair :

Fonction CoûtEtAgentsAllerRetourImpair

Data : nombre d'agents n

Result : Le coût optimal C_{opt}, nombre d'agents faisant aller-retour α

begin

$\alpha \leftarrow 0$;

$C_{\text{opt}} \leftarrow 0.5$;

while $\left(\frac{\log\left[(2^{\left\lfloor \log\left(\frac{1}{2-C_{opt}}\right)+2\right\rfloor})(2-C_{opt})-1\right]-\left[\left\lfloor \log\left(\frac{1}{2-C_{opt}}\right)+2\right\rfloor \log(3)+\log\left(1-\frac{2C_{opt}}{5}\right)\right]}{\log\left(\frac{2}{3}\right)} \leqslant \lfloor \frac{n}{2} \rfloor \right)$ **do**

$\alpha \leftarrow \alpha + 1$;

$C_{\text{opt}} \leftarrow \frac{(3)^{\alpha}\left(\frac{2}{3}\right)^{\lfloor \frac{n}{2} \rfloor}+1-2(2)^{\alpha}}{(3)^{\alpha}\left(\frac{2}{3}\right)^{\lfloor \frac{n}{2} \rfloor}\left(\frac{2}{5}\right)-(2)^{\alpha}}$;

end

return α;

return C_{opt};

end

5.4 Coût du commérage par un ensemble de n agents, pour n pair

Dans ce cas, les équations D_1 et D_2 peuvent aussi être fusionnées dans une seule équation pour être utilisées dans un algorithme qui va chercher le valeur du coût optimal.

La fusion des deux équations donne :

$$
\begin{aligned}
\frac{n}{2} &= \alpha + \gamma \\
\frac{n}{2} - \alpha &= \gamma \\
\left\lfloor \log\left(\frac{1}{2-C}\right) + 2 \right\rfloor &= \alpha \\
D_1 + D_2 &\geqslant \frac{n}{2} - 1 \\
C + \alpha - \left(\frac{1-\left(\frac{1}{2}\right)^{\alpha}}{1-\left(\frac{1}{2}\right)}\right) + (\gamma) + \left[\left(\frac{C-2}{3}\right)\left(\frac{1-\left(\frac{2}{3}\right)^{\gamma-1}}{1-\left(\frac{2}{3}\right)}\right) + \left(\frac{2}{3}\right)^{(\gamma-1)}\left(\frac{C-1}{3}\right)\right] &\geqslant \frac{n}{2} \\
\frac{(2^{\alpha})(2-C)-1}{(3^{\alpha})\left(\frac{5-2C}{4}\right)} &\geqslant \left(\frac{2}{3}\right)^{\frac{n}{2}} \\
\frac{\log\left[(2^{\alpha})(2-C)-1\right] - \left[(\alpha)\log(3) + \log\left(\frac{5-2C}{4}\right)\right]}{\log\left(\frac{2}{3}\right)} &\geqslant \frac{n}{2} \\
\frac{\log\left[(2^{\left\lfloor \log\left(\frac{1}{2-C}\right)+2\right\rfloor})(2-C)-1\right] - \left[\left\lfloor \log\left(\frac{1}{2-C}\right)+2\right\rfloor \log(3) + \log\left(\frac{5-2C}{4}\right)\right]}{\log\left(\frac{2}{3}\right)} &\geqslant \frac{n}{2}
\end{aligned}
\tag{5.11}
$$

L'équation (5.11) va nous permettre de vérifier si un coût trouvé va vraiment faire bouger les agents correctement afin de partager les messages parmi les agents.

On a besoin d'une équation en relation avec le nombre d'agents qui font l'aller-retour pour retourner comme réponse des valeurs de coûts, (valeurs qui seront validées par l'équation (5.11)) et définir la valeur exacte du coût qui permettra de partager les messages parmi les agents.

En partant de l'équation (5.11) :

$$
\begin{aligned}
\frac{(2^{\alpha})(2-C)-1}{(3^{\alpha})\left(\frac{5-2C}{4}\right)} &\geqslant \left(\frac{2}{3}\right)^{\frac{n}{2}} \\
C &\geqslant \frac{(3)^{\alpha}\left(\frac{2}{3}\right)^{\frac{n}{2}}\left(\frac{5}{4}\right) + 1 - 2\,(2)^{\alpha}}{(3)^{\alpha}\left(\frac{2}{3}\right)^{\frac{n}{2}}\left(\frac{1}{2}\right) - (2)^{\alpha}}
\end{aligned}
\tag{5.12}
$$

L'algorithme pour trouver le coût optimal quand le nombre d'agents est pair :

Fonction CoûtEtAgentsAllerRetourPair

Data : nombre d'agents n pair

Result : Le coût optimal C_{opt}, nombre d'agents faisant aller-retour α

begin

$\alpha \leftarrow 0$; $C_{\text{opt}} \leftarrow 0.5$;

while $\left(\frac{\log\left[(2^{\left\lfloor \log\left(\frac{1}{2-C_{opt}}\right)+2\right\rfloor})(2-C_{opt})-1\right]-\left[\left\lfloor \log\left(\frac{1}{2-C_{opt}}\right)+2\right\rfloor \log(3)+\log\left(\frac{5-2C_{opt}}{4}\right)\right]}{\log\left(\frac{2}{3}\right)} \leqslant \frac{n}{2} \right)$

do

$\alpha \leftarrow \alpha + 1$;

$C_{\text{opt}} \leftarrow \frac{(3)^{\alpha}\left(\frac{2}{3}\right)^{\frac{n}{2}}\left(\frac{5}{4}\right)+1-2(2)^{\alpha}}{(3)^{\alpha}\left(\frac{2}{3}\right)^{\frac{n}{2}}\left(\frac{1}{2}\right)-(2)^{\alpha}}$;

end

end

5.5 Preuve d'exactitude générique

Dans cette section nous présentons un théorème permettant de prouver l'exactitude des schémas de commérage produit par nos algorithmes pour n pair et impair.

Soit donné un schéma de commérage :

$$S = ((a_{i_1}, d_{i_1}), (a_{i_2}, d_{i_2}), ..., (a_{i_m}, d_{i_m}))$$

On appelle la *consolidation gauche* une sous-suite S_{cg} de S, telle que

$$S_{cg} = ((a_{k_1}, d_{k_1}), (a_{k_2}, d_{k_2}), ..., (a_{k_{cg}}, d_{k_{cg}}))$$

et

1. $a_{k_1} = 1$ (le premier agent)
2. Pour chaque $i = 1, ..., cg - 1$ nous avons
   (a) $k_i < k_{i+1}$
   (b) Tous les mouvements d_{k_i} sont de gauche à droite.

(c) Le mouvement d_{k_i} de l'agent a_{k_i} se termine au point du début du mouvement $d_{k_{i+1}}$ de l'agent $a_{k_{i+1}}$

On appelle la *consolidation droite* une sous-suite S_{cd} de S, telle que

$$S_{cd} = ((a_{j_1}, d_{j_1}), (a_{j_2}, d_{j_2}), ..., (a_{j_{cd}}, d_{j_{cd}}))$$

et

1. $a_{j_1} = n$ (le numéro du dernier agent)
2. Pour chaque $i = 1, ..., cd - 1$ nous avons
   (a) $j_i < j_{i+1}$
   (b) Tous les mouvements d_{j_i} sont de droite à gauche
   (c) Le mouvement d_{j_i} de l'agent a_{j_i} se termine au point du début du mouvement $d_{j_{i+1}}$ de l'agent $a_{j_{i+1}}$

On appelle la *distribution gauche* une sous-suite S_{dg} de S, telle que

$$S_{dg} = ((a_{p_1}, d_{p_1}), (a_{p_2}, d_{p_2}), ..., (a_{p_{cd}}, d_{p_{cd}}))$$

et

1. Pour chaque $i = 1, ..., dg - 1$ nous avons
   (a) $p_i < p_{i+1}$
   (b) Tous les mouvements d_{p_i} sont de droite à gauche
   (c) Le mouvement d_{p_i} de l'agent a_{p_i} se termine au point du début du mouvement $d_{p_{i+1}}$ de l'agent $a_{p_{i+1}}$.
2. Le mouvement $d_{p_{dg}}$ se termine au point ou se trouve en ce moment l'agent le plus à gauche sur la ligne.

On appelle la *distribution droite* une sous-suite S_{dd} de S, telle que

$$S_{dd} = ((a_{q_1}, d_{q_1}), (a_{q_2}, d_{q_2}), ..., (a_{q_{dd}}, d_{q_{dd}}))$$

et

1. Pour chaque $i = 1, ..., dd - 1$ nous avons
   (a) $q_i < q_{i+1}$
   (b) Tous les mouvements d_{q_i} sont de gauche à droite
   (c) Le mouvement d_{q_i} de l'agent a_{q_i} se termine au point du début du mouvement $d_{q_{i+1}}$ de l'agent $a_{q_{i+1}}$.
2. Le mouvement $d_{q_{dd}}$ se termine au point ou se trouve en ce moment l'agent le plus à droite sur la ligne.

Théorème 5.5.1. *Supposons que S est un schéma de commérage et $S_{cg}, S_{cd}, S_{dg}, S_{dd}$, les quatre sous-suites S constituant, respectivement, la consolidation gauche, consolidation droite, distribution gauche et distribution droite.*

Si

1. *La fin de mouvement $d_{k_{cg}}$ coïncide avec la fin de mouvement $d_{j_{cd}}$, et*
2. *$k_{cg} < p_1$ et $j_{cd} < q_1$ (c'est à dire les consolidations se terminent avant que les distributions commencent)*

Alors S est un schéma de commérage correct.

Preuve 5.5.1. . *Prenons n agents $a_1, ..., a_n$ ayant les positions initiales sur la ligne $1, ..., n$, respectivement.*

Notons par C le point étant la destination de $d_{k_{cg}}$. Vu que $a_{k_1} = a_1$ et que les mouvements $d_{k_1}, ..., d_{k_{cg}}$ se font dans cet ordre, avec le point de départ de $d_{k_{i+1}}$ étant le point d'arrivé de d_{k_i} , pour $i = 1, 2, ..., cd-1$, à la fin de l'exécution de la consolidation gauche, l'information initiale de tous les agents initialement dans les positions dans l'intervalle $[0, C]$ se trouve dans le point C. Par symétrie, l'information initiale de tous les agents initialement placés dans l'intervalle $[C, n]$ se trouve au point C à la fin de la distribution droite. Par conséquence, l'agent $a_{k_{cg}}$ (et $a_{j_{cd}}$, si différent) possède toute information initiale.

Vu que $k_{cg} < p_1$ et $j_{cd} < q_1$ les phases de distributions commencent avec les informations de tous les agents. Vu que les mouvements de la distribution gauche $d_{p_1}, ..., d_{p_{dg}}$ se font dans cet ordre, avec le point de départ de $d_{p_{i+1}}$ étant le point d'arrivé de d_{p_i}, pour $i = 1, 2, ..., dg - 1$, finalement atteignant l'agent le plus à gauche sur la ligne, cette information est transférée à tous les agents à gauche du point C. Par symétrie, la distribution droite permet de transférer cette information à tous les agents à droite du point

C. Conséquemment, l'information provenant de tous les agents atteint tous les agents de l'ensemble et le commérage se termine correctement.

Chapitre 6

Commérage de n agents sur la ligne (n quelconque)

Dans ce chapitre nous présentons l'algorithme générant le schéma de commérage, ainsi que la preuve de son exactitude.

6.1 Algorithme de commérage

L'algorithme qui sera décrit dans cette section permet d'obtenir le schéma de commérage (mouvements et déplacements de chaque agent) pour accomplir la tâche de commérage entre les agents.

Le schéma généré aura deux phases. La première sera la phase de consolidation où les messages sont transmis vers l'agent central (ou les agents centraux), selon le nombre d'agents pair ou impair.

La deuxième phase sera la phase de distribution, dans laquelle les messages, une fois tous obtenus par un agent, sont retransmis aux autres agents.

Dans chaque phase, consolidation et distribution, il y aura deux sous-phases : une sous-phase pour les agents qui font un mouvement du type *Allez-Retour* et une autre sous-phase pour les agents qui font le mouvement de type *Serpentine*.

Les mouvements à effectuer par les agents de milieu seront calculés dans une autre sous-phase à part, pour mieux comprendre où se situe le point R (point de distribution). L'algorithme de commérage génère des mouvements effectués par l'agent du milieu (nombre d'agents impair) et les agents du milieu (nombre d'agents pair).

Algorithme 1: Algorithme *SC*

Data : nombre des agents n
Result : Schéma de commérage $\mathcal{S}$
begin
  $C_{\text{optCal}}, a_{\text{AR}} \leftarrow$ CoûtEtAgentsAllerRetour(n) ;
  $\mathcal{S} \leftarrow \{\}$; *Etape* $\leftarrow 1$;
  /* Phase de Consolidation */
  for $i = 1$; $i \leqslant a_{AR}$; $i{+}{+}$ **do** /* *Aller-Retour*, section (1.1) */
    MouvementsAgentsAllerRetour(C_{optCal},i) ;
  end
  if *Pair(n)* **then** /* sous-phase agents *Serpentine* */
    for $i = a_{AR} + 1$; $i < (n/2)$; $i{+}{+}$ **do** /* section (1.2) */
      MouvementsAgentsSerpentPair(C_{optCal},i) ;
    end
  else
    for $i = a_{AR} + 1$; $i \leqslant |(n/2)|$; $i{+}{+}$ **do** /* section (1.3) */
      MouvementsAgentsSerpentImpair(C_{optCal},i) ;
    end
  end
  if *Pair(n)* **then** /* sous-phase agents agents du milieu */
    MouvementsAgentsMillieuPair(C_{optCal},i) ;/* section (1.4) */
  else
    MouvementsAgentsMillieuImpair(C_{optCal},i) ;/* section (1.5) */
  end
  /* Phase de Distribution */
  if *Pair(n)* **then** /* sous-phase agents serpents */
    for $i = (n/2)$; $i > a_{AR}$; $i - -$ **do** /* section (1.6) */
      MouvementsAgentsSerpentPairDist(C_{optCal},i) ;
    end
  else
    for $i = |(n/2)| - 1$; $i > a_{AR}$; $i - -$ **do** /* section (1.7) */
      MouvementsAgentsSerpentImpairDist(C_{optCal},i) ;
    end
  end
  for $i = a_{AR}$; $i > 2$; $i - -$ **do** /* section (1.8) */
    MouvementsAgentsAllerRetourDist(C_{optCal},i) ;
  end
end

Les fonctions données ci-dessus sont utilisées par l'*Algorithme SC* comme procédures auxiliaires dans la construction du schéma de commérage pour un nombre d'agents n donné.

La fonction *CoûtEtAgentsAllerRetour* utilise les fonctions *CoûtEtAgentsAllerRetour-Pair* (section 5.4) quand le nombre d'agents est pair et *CoûtEtAgentsAllerRetourImpair* (section 5.3) quand le nombre d'agents est impair. Le résultat qui est la fonction *CoûtEtAgentsAllerRetour* trouve le coût dont aura besoin chaque agent pour effectuer la tâche de commérage et le nombre d'agents qui auront le mouvement de type *Aller-Retour*.

Fonction CoûtEtAgentsAllerRetour(n)

Data : nombre des agents n
Result : Coût optimal (C_{optCal}) et le nombre des agents qui feront le mouvement du type *Allez-Retour* (a_{AR})

$a_{\text{AR}} \leftarrow 0$;
$C_{\text{opt}} \leftarrow 0.5$;
if *Pair(n)* **then**
  while $\left(\frac{\log\left[(2^{\left\lfloor \log\left(\frac{1}{2-C_{opt}}\right)+2\right\rfloor})(2-C_{opt})-1\right]-\left[\left\lfloor \log\left(\frac{1}{2-C_{opt}}\right)+2\right\rfloor \log(3)+\log\left(\frac{5-2C_{opt}}{4}\right)\right]}{\log\left(\frac{2}{3}\right)} \leqslant \frac{n}{2} \right)$
  do
    $a_{\text{AR}} \leftarrow a_{\text{AR}} + 1$;
    $C_{\text{opt}} \leftarrow \frac{(3)^{a_{\text{AR}}}\left(\frac{2}{3}\right)^{\frac{n}{2}}\left(\frac{5}{4}\right)+1-2(2)^{a_{\text{AR}}}}{(3)^{a_{\text{AR}}}\left(\frac{2}{3}\right)^{\frac{n}{2}}\left(\frac{1}{2}\right)-(2)^{a_{\text{AR}}}}$;
  end
else
  while $\left(\frac{\log\left[(2^{\left\lfloor \log\left(\frac{1}{2-C_{opt}}\right)+2\right\rfloor})(2-C_{opt})-1\right]-\left[\left\lfloor \log\left(\frac{1}{2-C_{opt}}\right)+2\right\rfloor \log(3)+\log\left(1-\frac{2C_{opt}}{5}\right)\right]}{\log\left(\frac{2}{3}\right)} \leqslant \left\lfloor \frac{n}{2} \right\rfloor \right)$ **do**
    $a_{\text{AR}} \leftarrow a_{\text{AR}} + 1$;
    $C_{\text{opt}} \leftarrow \frac{(3)^{a_{\text{AR}}}\left(\frac{2}{3}\right)^{\lfloor \frac{n}{2} \rfloor}+1-2(2)^{a_{\text{AR}}}}{(3)^{a_{\text{AR}}}\left(\frac{2}{3}\right)^{\lfloor \frac{n}{2} \rfloor}\left(\frac{2}{5}\right)-(2)^{a_{\text{AR}}}}$;
  end
end
return C_{opt}, a_{AR}

La fonction suivante, *MouvementAgentsAllerRetour*, va générer les mouvements d'agents du type *Aller-Retour* sur la ligne. Ce groupe d'agents, inclut les agents extrêmes, même si les agents extrêmes ne font pas le mouvement de retour. La fonction *MouvementAgentsAllerRetour* utilise les résultats trouvés dans l'équation 5.2.

Fonction MouvementAgentsAllerRetour(C_{opt}, i)

Data : C_{opt}, i (agent à bouger)
Result : Ajout des mouvements sur le schéma de commérage $\mathcal{S}$
if $i == 1$ **then** ; /* Agent extrême */

  if $C_{opt} > 1$ **then** ; /* section (2.1) */

    $D_{suiv} \leftarrow C_{\text{opt}} - 1$;
    $\mathcal{S}_{Etape} \leftarrow (a_i, 1)$; $Etape \leftarrow Etape + 1$;
    $\mathcal{S}_{Etape} \leftarrow (a_i, D_{suiv})$; $Etape \leftarrow Etape + 1$;
    $\mathcal{S}_{Etape} \leftarrow (a_{n+1-i}, -1)$; $Etape \leftarrow Etape + 1$;
    $\mathcal{S}_{Etape} \leftarrow (a_{n+1-i}, -D_{suiv})$; $Etape \leftarrow Etape + 1$;
  else; /* section (2.2) */

    $\mathcal{S}_{Etape} \leftarrow (a_i, D_{suiv})$; $Etape \leftarrow Etape + 1$;
    $\mathcal{S}_{Etape} \leftarrow (a_{n+1-i}, -D_{suiv})$; $Etape \leftarrow Etape + 1$;
  end
else
  $D_{suiv} \leftarrow C_{\text{opt}} + i - \left(\frac{1-(1/2)^i}{1-(1/2)}\right)$;
  if $D_{suiv} > i$ **then** ; /* section (2.3) */

    $D_{suiv} \leftarrow D_{suiv} - i$;
    $\mathcal{S}_{Etape} \leftarrow (a_i, 1)$; $Etape \leftarrow Etape + 1$;
    $\mathcal{S}_{Etape} \leftarrow (a_i, D_{suiv})$; $Etape \leftarrow Etape + 1$;
    $\mathcal{S}_{Etape} \leftarrow (a_{n+1-i}, -1)$; $Etape \leftarrow Etape + 1$;
    $\mathcal{S}_{Etape} \leftarrow (a_{n+1-i}, -D_{suiv})$; $Etape \leftarrow Etape + 1$;
  else; /* section (2.4) */

    $D_{suiv} \leftarrow (1 - i + D_{suiv})$;
    $\mathcal{S}_{Etape} \leftarrow (a_i, D_{suiv})$; $Etape \leftarrow Etape + 1$;
    $\mathcal{S}_{Etape} \leftarrow (a_{n+1-i}, -D_{suiv})$; $Etape \leftarrow Etape + 1$;
  end
end

Les fonctions *MouvementAgentsSerpentPair* et *MouvementAgentsSerpentImpair* ont comme objectif de générer les mouvements d'agents pour aller chercher l'information chez leurs voisins, en la transportant le plus loin possible, tout en conservant assez de coût nécessaire pour accomplir la phase de distribution. Les agents de type *Serpen-*

tine font 3 déplacements chacun. Ces mouvements sont basés sur l'équation 5.8 pour *MouvementAgentsSerpentPair* et 5.4 pour *MouvementAgentsSerpentImpair*.

Fonction MouvementAgentsSerpentPair(C_{opt}, i)

Data : $C_{\text{opt}}, Etape, i$ (agent a bouger)
Result : Ajout des mouvements sur le schéma de commérage $\mathcal{S}$
$D_{suiv} \leftarrow \left[\left(\frac{C_{opt}-2}{3}\right)\left(\frac{1-(2/3)^{\lfloor\frac{n}{2}\rfloor-i}}{1-(2/3)}\right)+\left(\frac{2}{3}\right)^{(\lfloor\frac{n}{2}\rfloor-i)}\left(\frac{C_{opt}-1}{3}\right)\right]$;
$\mathcal{S}_{Etape} \leftarrow (a_i, -D_{suiv})$; $Etape \leftarrow Etape + 1$; `/* section (3.1) */`
$\mathcal{S}_{Etape} \leftarrow (a_i, +\left(\frac{C_{\text{opt}}-D_{suiv}}{2}\right))$; $Etape \leftarrow Etape + 1$;
$\mathcal{S}_{Etape} \leftarrow (a_{n+1-i}, +D_{suiv})$; $Etape \leftarrow Etape + 1$; `/* section (3.2) */`
$\mathcal{S}_{Etape} \leftarrow (a_{n+1-i}, -\left(\frac{C_{\text{opt}}-D_{suiv}}{2}\right))$; $Etape \leftarrow Etape + 1$;

Fonction MouvementAgentsSerpentImpair(C_{opt}, i)

Data : $C_{\text{opt}}, Etape, i$ (agent à bouger)
Result : Ajout des mouvements sur le schéma de commérage $\mathcal{S}$
$D_{suiv} \leftarrow \left[\left(\frac{C_{opt}-2}{3}\right)\left(\frac{1-(2/3)^{(\lfloor\frac{n}{2}\rfloor+1-i)}}{1-(2/3)}\right)+\left(\frac{2}{3}\right)^{(\lfloor\frac{n}{2}\rfloor+1-i)}\left(\frac{C_{opt}}{5}\right)\right]$;
$\mathcal{S}_{Etape} \leftarrow (a_i, -D_{suiv})$; $Etape \leftarrow Etape + 1$; `/* section (4.1) */`
$\mathcal{S}_{Etape} \leftarrow (a_i, +\left(\frac{C_{\text{opt}}-D_{suiv}}{2}\right))$; $Etape \leftarrow Etape + 1$;
$\mathcal{S}_{Etape} \leftarrow (a_{n+1-i}, +D_{suiv})$; $Etape \leftarrow Etape + 1$; `/* section (4.2) */`
$\mathcal{S}_{Etape} \leftarrow (a_{n+1-i}, -\left(\frac{C_{\text{opt}}-D_{suiv}}{2}\right))$; $Etape \leftarrow Etape + 1$;

Le point R de la phase de consolidation n'est pas le même quand le nombre d'agents est pair ou impair. Quand le nombre d'agents est pair, le point R se situe au milieu de la ligne ($n/2$), mesuré à partir de l'agent 1, et il aura deux agents ($a_{\frac{n}{2}}$ et $a_{\frac{n}{2}+1}$) qui auront toutes les informations au point R. Pour le cas impair (nombre d'agents), le point R va se situer à une distance $\frac{n-1}{2} - \frac{C_{opt}}{5}$ à partir de l'agent 1.

Nous observons avec les fonctions *MouvementAgentsSerpentPair* et *MouvementAgentsMillieuPair* qu'on ne peut pas utiliser la fonction *MouvementAgentsSerpentImpair* pour l'agent du milieu parce que la symétrie est faite sur un agent, et donc il n'y a qu'un seul agent qui bouge. Pour toutes les autres fonctions, il y a un agent de chaque côté qui se promène.

Les fonctions *MouvementAgentsSerpentPairDist* et *MouvementAgentsSerpentImpairDist* transportent l'information à travers les agents faisant le mouvement de type *Serpentine* vers les agents de type *Aller-Retour*, en effectuent le seul mouvement qui reste à

Fonction MouvementAgentsMillieuPair(C_{opt}, i)

Data : $C_{\text{opt}}, Etape, i$ (agent à bouger)
Result : Ajout des mouvements sur le schéma de commérage $\mathcal{S}$
$D_{suiv} \leftarrow \left[\left(\frac{C_{opt}-2}{3}\right)\left(\frac{1-(2/3)^{\lfloor\frac{n}{2}\rfloor-i}}{1-(2/3)}\right)+\left(\frac{2}{3}\right)^{(\lfloor\frac{n}{2}\rfloor-i)}\left(\frac{C_{opt}-1}{3}\right)\right]$;
$\mathcal{S}_{Etape} \leftarrow (a_i, -D_{suiv})$; $Etape \leftarrow Etape + 1$;
$\mathcal{S}_{Etape} \leftarrow (a_i, +\left(\frac{C_{\text{opt}}-D_{suiv}}{2}\right))$; $Etape \leftarrow Etape + 1$;
$\mathcal{S}_{Etape} \leftarrow (a_{n+1-i}, +D_{suiv})$; $Etape \leftarrow Etape + 1$;
`/* Point` R `de la phase de Consolidation */`
$\mathcal{S}_{Etape} \leftarrow (a_{n+1-i}, -\left(\frac{C_{\text{opt}}-D_{suiv}}{2}\right))$; $Etape \leftarrow Etape + 1$;

Fonction MouvementAgentsMillieuImpair(C_{opt}, i)

Data : $C_{\text{opt}}, Etape, i$ (agent à bouger)
Result : Ajout des mouvements sur le schéma de commérage $\mathcal{S}$
$D_{suiv} \leftarrow \left(\frac{C_{opt}}{5}\right)$;
$\mathcal{S}_{Etape} \leftarrow (a_i, -D_{suiv})$; $Etape \leftarrow Etape + 1$;
`/* Point` R `de la phase de Consolidation */`
$\mathcal{S}_{Etape} \leftarrow (a_i, +\left(\frac{C_{\text{opt}}-D_{suiv}}{2}\right))$; $Etape \leftarrow Etape + 1$;
$\mathcal{S}_{Etape} \leftarrow (a_i, -\left(\frac{C_{\text{opt}}-D_{suiv}}{2}\right))$; $Etape \leftarrow Etape + 1$;

faire. Pour les agents *Aller-Retour*, il reste à faire aussi un seul mouvement, à l'exception d'agents extrêmes qui n'ont pas d'autres mouvements à effectuer.

La fonction *MouvementAgentsSerpentPairDist* utilise l'équation 5.8 pour le calcul de la distance à traverser.

Fonction MouvementAgentsSerpentPairDist(C_{opt}, i)

Data : $C_{\text{opt}}, Etape, i$ (agent à bouger)
Result : Ajout des mouvements sur le schéma de commérage $\mathcal{S}$
$D_{suiv} \leftarrow \left[\left(\frac{C_{opt}-2}{3}\right)\left(\frac{1-(2/3)^{\lfloor\frac{n}{2}\rfloor-i}}{1-(2/3)}\right)+\left(\frac{2}{3}\right)^{(\lfloor\frac{n}{2}\rfloor-i)}\left(\frac{C_{opt}-1}{3}\right)\right]$;
$\mathcal{S}_{Etape} \leftarrow (a_i, -\left(\frac{C_{\text{opt}}-D_{suiv}}{2}\right))$; $Etape \leftarrow Etape + 1$;
$\mathcal{S}_{Etape} \leftarrow (a_{n+1-i}, +\left(\frac{C_{\text{opt}}-D_{suiv}}{2}\right))$; $Etape \leftarrow Etape + 1$;

La fonction *MouvementAgentsSerpentImpairDist* utilise l'équation 5.4 pour le calcul de la distance à bouger.

Fonction MouvementAgentsSerpentImpairDist(C_{opt}, i)

Data : $C_{\text{opt}}, Etape, i$ (agent à bouger)
Result : Ajout des mouvements sur le schéma de commérage $\mathcal{S}$
$D_{suiv} \leftarrow \left[\left(\frac{C_{opt}-2}{3}\right)\left(\frac{1-(2/3)^{(|\frac{n}{2}|+1-i)}}{1-(2/3)}\right) + \left(\frac{2}{3}\right)^{(|\frac{n}{2}|+1-i)}\left(\frac{C_{opt}}{5}\right)\right]$;
$\mathcal{S}_{Etape} \leftarrow (a_i, -\left(\frac{C_{\text{opt}}-D_{suiv}}{2}\right))$; $Etape \leftarrow Etape + 1$;
$\mathcal{S}_{Etape} \leftarrow (a_{n+1-i}, +\left(\frac{C_{\text{opt}}-D_{suiv}}{2}\right))$; $Etape \leftarrow Etape + 1$;

La fonction *MouvementAgentsAllerRetourDist* génère le mouvement qui sert a transférer l'information provenant du segment d'agents de type *Serpentine* vers les agents extrêmes. L'équation 5.2 est utilisée pour calculer la distance qui reste à parcourir.

Fonction MouvementAgentsAllerRetourDist(C_{opt}, i)

Data : C_{opt}, i (agent à bouger)
Result : Ajout des mouvements sur le schéma de commérage $\mathcal{S}$
$D_{suiv} \leftarrow \left(\frac{(2)^{(i-1)}-1}{(2)^{(i-1)}}\right)$ /*Équation 5.2 */;
$\mathcal{S}_{Etape} \leftarrow (a_i, -D_{suiv})$; $Etape \leftarrow Etape + 1$;
$\mathcal{S}_{Etape} \leftarrow (a_{n+1-i}, +D_{suiv})$; $Etape \leftarrow Etape + 1$;

6.2 Preuve d'exactitude de l'algorithme de commérage

Théorème 6.2.1. *L'algorithme SC calcule correctement un schéma de commérage pour n'importe quelle valeur de n agents.*

Preuve 6.2.1. . *Nous prouvons d'abord que les phases de distribution gauche et droite sont générées correctement par l'algorithme SC. Observons que la procédure MouvementAgentsAllerRetour génère le déplacement de deux agents extrêmes (dans le cas $i = 1$ agent gauche a_i et agent droit a_{n+1-i})-voir étape (2.1). Pour chaque agent a_i (et a_{n+1-i}) pour $i > 1$ le mouvement de la phase de consolidation est précédé par un mouvement préparatoire pour se rendre à la position de l'arrivée de l'agent a_{i-1} (respectivement a_{n+1-i}). Alors, par induction sur i on observe que l'agent a_i commence son mouvement de la position finale -durant la phase de consolidation- de l'agent a_{i-1} (respectivement l'agent a_{n+1-i} commence son mouvement de la position de l'agent a_{n+2-i}). Conséquemment, la partie de consolidation gauche et droite est accomplie correctement pour tous les agents exécutant les mouvements de type Aller-Retour.*

Considérons maintenant les agents qui font les mouvements de type Serpentine, c'est à dire les agents avec numéros $a_{AR+1}, a_{AR+2}, ..., n - a_{AR}$. La moitié de ces agents (pour n pair), $a_{AR+1}, a_{AR+2}, ..., a_{n/2}$ assurent la consolidation gauche et l'autre moitié la consolidation droite (voir l'étape (1.2) de l'algorithme SC). Le cas de n impair est presque identique, mais la formule du coût (voir fonction CoûtEtAgentsAllerRetour) est différente à cause de la symétrie sur l'agent central. Tous ces agents font d'abord un mouvement préparatoire ((3.1) et (4.1) pour les agents faisant la consolidation gauche ainsi que (3.2) et (4.2) pour ceux responsables de la consolidation droite). Observons que par le calcul de C_{opt} de la fonction CoûtEtAgentsAllerRetour par observation 6 la valeur de D_{suiv} provenant des formules (6.3) et (6.6) assure que chaque agent a_i (respectivement a_{n+1-i}) se rends à l'endroit ou son voisin a_{i-1} (respectivement a_{n+2-i}) est arrivé. Ceci assure que la deuxième partie de la sequence de consolidation gauche (respectivement droite) respecte la condition (2.c) de son exactitude.

Ceci implique que les séquences de consolidation exigées par le Théorème 5.5.1 sont générées correctement.

Le mouvement généré au point (1.4) ou (1.5) de l'algorithme SC assure que, par l'observation 8, les deux consolidations (gauche et droite) se terminent au même point étant la destination de $d_{k_{cg}}$ et $d_{j_{cd}}$ du Théorème 5.5.1.

Considérons maintenant les phases de distribution gauche et droite. Observons que les mouvements d'agents en Serpentine sont ici inverse aux mouvements de ces agents, effectués durant la phase de consolidation. Cela implique que les parties des distributions gauche et droite effectuées par les agents en mouvement de type Serpentine sont correctes.

Prenons maintenant les agents en mouvement de type Aller-Retour (point (1.8) de l'algorithme SC). Avec la formule (6.2) et l'observation 7 la valeur de D_{suiv} de la fonction MouvementAgentsAllerRetourDist assure que chaque agent a_i (respectivement a_{n+1-i}) retourne au point ou se trouve son voisin a_{i-1} (respectivement a_{n+2-i}). Ceci implique que la condition (1.c) de la séquence de distribution (gauche ou droite) et bien respectée.

Observons aussi que la formule (6.2) assure que la distribution se termine exactement au point où se trouve l'agent extrême a_1 (respectivement a_n) vérifiant la condition 2 de la séquence de distribution.

Vu que toutes les séquences de consolidation et distribution analysées ci-dessus respectent les conditions de leurs définitions et que les conditions du Théorème 5.5.1 sont vérifiées ceci conclut la preuve de l'exactitude de notre algorithme.

6.3 Tableau des résultats pour n agents

Voici un tableau comparatif entre la borne inférieure et le coût de l'algorithme SC lorsque le nombre d'agents n est pair et impair :

Valeur n	Algorithme SC	Agents *Aller-Retour*	Borne inférieure
3	0,833333333	1	0,666666667
4	1	1	1
5	1,25	2	1,2
6	1,375	2	1,333333333
7	1,477272727	2	1,428571429
8	1,5625	3	1,5
9	1,647727273	3	1,555555556
10	1,696428571	3	1,6
11	1,739864865	3	1,636363636
12	1,776785714	4	1,666666667
13	1,815878378	4	1,692307692
14	1,839673913	4	1,714285714
15	1,861869748	4	1,733333333
16	1,876358696	5	1,75
17	1,897321429	5	1,764705882
18	1,910530822	5	1,777777778
19	1,923173592	5	1,789473684
20	1,931305066	5	1,8
21	1,940139075	6	1,80952381
22	1,948031388	6	1,818181818

TABLE 6.1 – Performance de l'algorithme SC et bornes inférieures sur le coût de commérage

Chapitre 7

Commérage en 2-dimensions

Dans cette section nous considérons le problème de commérage pour un ensemble d'agents sur un plan (espace à deux dimensions).

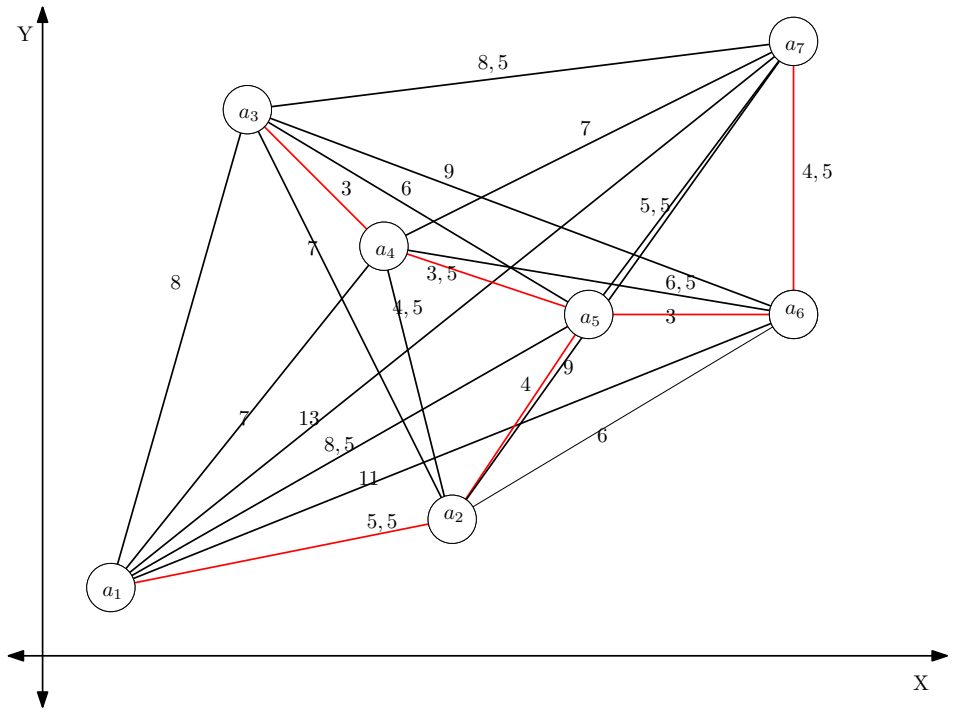

FIGURE 7.1 – Exemple commérage 2-dimensions

Le modèle du problème est : l'ensemble d'agents est placé dans un espace à deux dimensions, et le système de coordonnées est commun à tous les agents. Chaque agent a une vue complète des positions des autres agents, incluant la sienne.

Les agents sont capables de se déplacer dans n'importe quelle direction sur le plan. La vitesse de déplacement n'a pas été mise en évidence, constante dans notre modèle, parce que la valeur de recherche est le coût de déplacement C_{max}. Le coût est défini comme dans le cas de la ligne.

Le commérage est accompli à la fin de l'exécution de l'algorithme, si chaque agent a reçu l'information originale de tous les autres agents.

Chaque agent a_i a une position dans le plan représentée par la valeur $P(a_i)$. La position $P(a_i)$ est une paire $(X(a_i), Y(a_i))$ représentant la position de l'agent dans le plan cartésien.

L'ensemble d'agents est $A = \{a_1, a_1, a_1, ..., a_n\}$ où n est le nombre d'agents sur le plan.

L'échange d'informations entre deux agents est fait quand les deux agents sont sur le même point en même temps.

L'algorithme *commérage-dans-le-plan* est décrit comme suit.

À partir du positionnement initial d'agents, on commence par créer un graphe complet G avec les distances entre chaque agent par rapport aux autres agents. Ensuite, on trouve l'arbre couvrant de poids minimal A_G sur G, en utilisant l'algorithme de *Prim* ou *Kruskal* [27], en prenant la distance entre chaque pair d'agents comme poids sur l'arête entre les deux (distance $d_{i,j} = Dist(P(a_i), P(a_j))$). On suppose que l'arbre obtenu (voir image 7.1 arêtes rouges) est connu par l'ensemble d'agents, et que chaque agent connaît sa position dans l'arbre.

Une fois que l'arbre couvrant de poids minimal a été trouvé, l'algorithme est divisé en deux phases, la *Phase de consolidation* et la *Phase de distribution.*

Dans la Phase de consolidation, l'information est transportée par les agents vers un ou deux agents centraux (dépendant de la topologie de l'arbre) de la façon suivante. On considère cet arbre comme enraciné dans l'un des sommets centraux. Chaque agent de type feuille (agent qui a le degré 1 dans l'arbre) se déplace vers son parent pour partager son information. Chaque agent parent attend tous ses enfants avant de se déplacer vers son parent et ainsi de suite. Le point R, est le nœud de l'arbre auquel pour une première fois un agent a reçu toutes les informations des autres agents. Après un certain temps, on se trouve avec deux cas possibles :

- Il y a un seul agent central : dans ce cas, le point R se trouve sur la position de l'agent racine. L'agent central va recevoir de ses enfants les informations initiales de tous les autres agents.
- Il y a deux agents centraux : le point R est au milieu de l'arc qui sépare les deux agents centraux.

Une fois le point R atteint, on commence la phase de distribution, qui a pour objectif de partager l'ensemble des informations connues par l'agent (ou agents) a_R (agent ou agents qui possèdent toutes les informations) aux autres agents.

Dans la phase de distribution, les agents vont parcourir le chemin inverse à celui parcouru pendant la phase de consolidation. Comme à la phase de consolidation il y a deux cas à considérer :

- Il y a un seul agent central : l'agent central a reçu les messages de chacun de ses agents enfants. L'agent central et ses enfants se trouvent au même point sur le plan et l'agent central partage l'ensemble des informations avec ses enfants. Par la suite, les agents enfants se déplaceront vers leurs positions initiales.
- Il y a deux agents centraux : une fois le point R déterminé, chaque agent central fait un demi-tour pour revenir sur sa position initiale et partager l'ensemble de l'information avec ses enfants.

Chaque agent attend son agent parent pour recevoir l'ensemble des informations de sa part, avant de se déplacer vers sa position initiale. Seulement les agents feuilles n'ont pas besoin de revenir vers leurs positions initiales.

L'algorithme prend fin lorsque chaque agent feuille reçoit l'ensemble des informations des autres agents.

Puisque chaque agent parcourt au plus une arête de A_G en deux directions, nous avons :

Théorème 7.0.1. *Soit δ la longueur de la plus longue arête de A_G. Le coût de l'Algorithme commérage-dans-le-plan est au plus égal à 2δ.*

7.0.1 Borne inférieure pour le commérage en 2-dimensions

Nous allons prouver que l'algorithme de commérage de la section précédente est une 4-approximation de l'algorithme de commérage optimal.

Théorème 7.0.2. *Soit δ la longueur de la plus longue arête de l'arbre A_G, l'arbre couvrant minimal du graphe G. Chaque algorithme de commérage a un coût d'au moins $\frac{\delta}{2}$.*

Preuve 7.0.1. *Supposons que $\triangle$ est la plus grande distance telle qu'il existe une partition de l'ensemble de positions initiales d'agents en deux sous-ensembles non-vides A et B, tel que $dist(A, B) = \triangle$. Observons, pour que l'information soit transférée entre A et B la distance d'au moins $\frac{\triangle}{2}$ doit être parcourue, par au moins un agent, car deux agents, l'un de l'ensemble A et l'autre de B doivent se rencontrer. Puisque l'arbre couvrant minimal A_G doit posséder au moins une arête de longueur au moins $\triangle$, nous avons $\triangle \geq \delta$, ce qui conclue la preuve.*

Corollaire 7.0.1. *Le coût de l'algorithme commérage-dans-le-plan est au plus 4 fois plus grand que le coût de l'algorithme optimal.*

Chapitre 8

Programmes de simulation

8.0.2 Simulation avec iOS

Le logiciel de simulation a été développé pour le système d'exploitation iOS de Apple. Il peut être exécuté dans un iPod, un iPhone ou une tablette iPad, doté d'un système d'exploitation supérieur à la version 5 d'iOS.

L'écran principal pour le logiciel de simulation se trouve sur la figure 8.1.

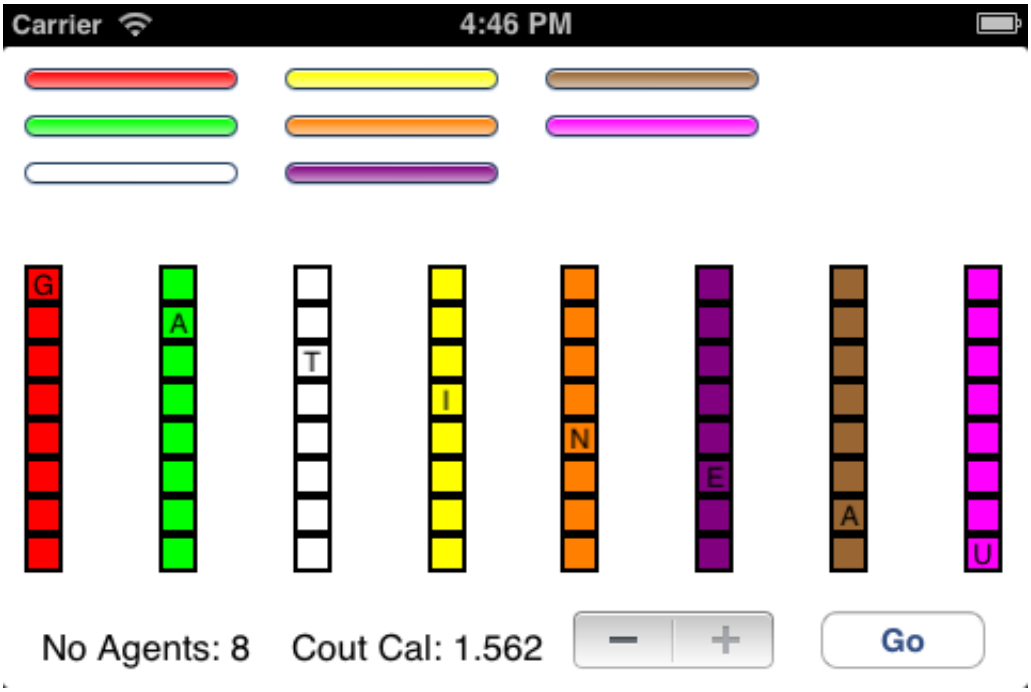

FIGURE 8.1 – Simulation iOS 8 agents

Sur l'écran principal on peut choisir le nombre d'agents à faire bouger. Le bouton *Go* active la simulation.

Les barres d'état en haut de l'écran indiquent le niveau de batterie qu'il reste a chacun des agents pendant la simulation, la correspondance est illustrée par la couleur de l'agent.

Au dessus de l'écran on trouve le nombre d'agents, le coût requis pour partager les messages et les boutons pour augmenter et/ou diminuer le nombre d'agents. Le nombre maximal d'agents qui peuvent être simulés a été fixé à 8.

Avec le bouton *Jouer*, on change le mode qui permet à l'utilisateur de trouver le coût par essai et erreur. Le logiciel tente d'effectuer la tâche de commérage avec le coût donné par l'utilisateur. Si avec le coût donné le commérage est réalisable, une simulation sera produite avec schéma de commérage calculé. Par contre, si le coût donné pour le commérage n'est pas possible, un message d'erreur sera affiché à l'écran.

8.0.3 Simulation avec Matlab et Octave

Les algorithmes, pour trouver le coût quand le nombre d'agents est pair où impair ont été écrits et testés avec Matlab sur Windows et Octave dans l'environnement Mac. Une copie de chaque logiciel se trouve en annexe.

Même s'il est possible d'utiliser les routines écrites pour Octave sur iOS, avec des fichiers .mex pré-compilés, cette méthode n'a pas été retenue à cause de sa complexité. Par contre, le code pour la simulation sous iOS de la fonction qui calcule le coût et très similaire à celle écrite sur Octave-Matlab.

Chapitre 9

Conclusion

Dans ce mémoire, nous avons considéré la minimisation du coût maximal par agent de la tâche du commérage (échange d'information) parmi plusieurs agents mobiles. Nous avons trouvé la borne inférieure pour le coût du commérage des agents mobiles qui se trouvent sur la ligne à distance égale entre eux. Nous avons proposé aussi un algorithme général qui permet de trouver le coût requis et le schéma de commérage (une suite de mouvements à faire pour accomplir le commérage) pour n'importe quelle nombre n d'agents. L'optimalité du coût calculé pour des valeurs de n égales à 3 et 4 a été prouvée. Pour les valeurs de n plus grandes, le problème de savoir si notre algorithme est optimal reste ouvert.

Une indication que notre algorithme est possiblement sub-optimal découle de l'exemple suivant : sur une ligne on a des agents placés à distances inégales, a_0 $Pos(a_0) = 0$, a_1 $Pos(a_1) = 5.5$, a_2 $Pos(a_2) = 6.5$, et a_3 $Pos(a_3) = 11.5$. L'algorithme décrit dans ce mémoire essaie de pousser l'information le plus loin possible, à chaque fois.

Cette stratégie a été prouvée sub-optimale par [4] pour la configuration ci-dessous (figure 9.2) en proposant un algorithme à coût plus bas. On ne sait pas si un tel exemple montrant la sub-optimalité de notre algorithme peut être construit pour les distances égales entre les agents.

Pour effectuer l'algorithme proposé par [4] on doit évaluer la position de tous les agents sur la ligne. Notre algorithme par contre utilise des informations moins détaillées :

- Où est rendu l'information.
- Où se trouve son prochain voisin pour partager l'information.

Suite à l'exemple ci-dessous deux questions restent ouvertes : (1), prouver ou réfuter l'optimalité de l'algorithme de commérage présenté dans ce mémoire dans le cas des

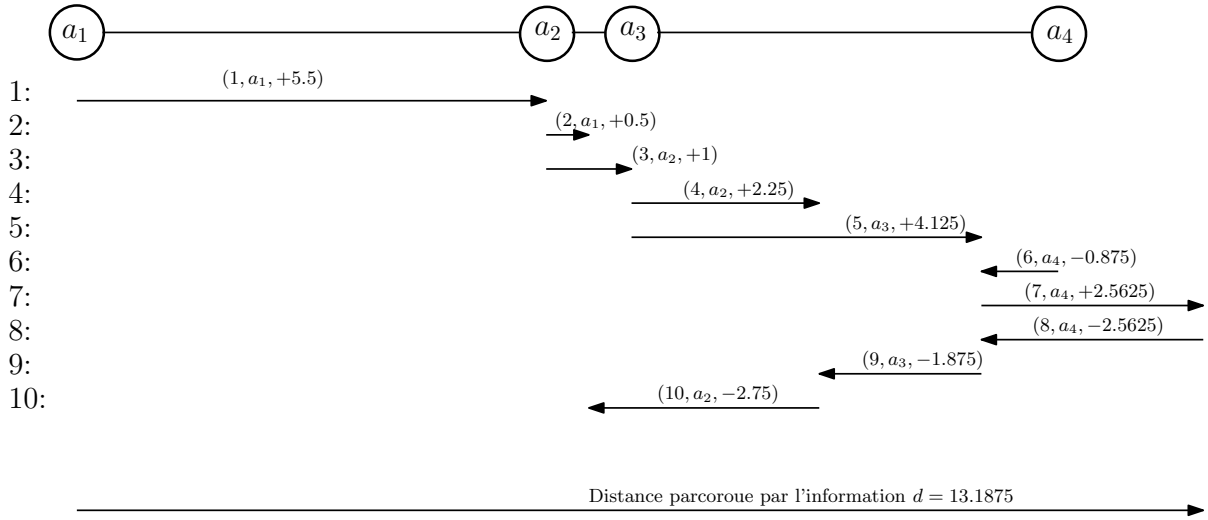

FIGURE 9.1 – Exemple avec distance inégales

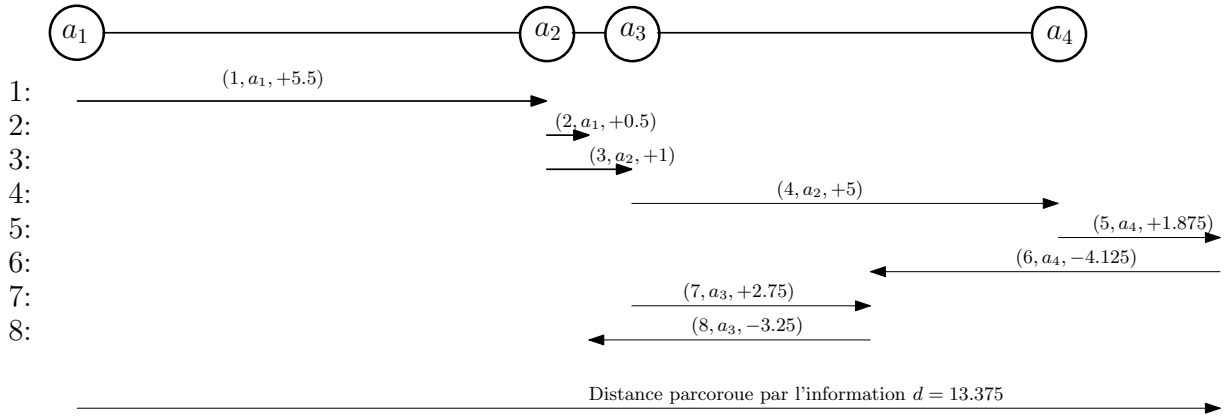

FIGURE 9.2 – Exemple avec distance inégales - contrexemple

distances égales entre les agents et (2), trouver les configurations avec distances inégales entre les agents sur la ligne pour lesquelles notre algorithme est optimal.

Un autre résultat présenté dans ce mémoire c'est un algorithme qui permet de faire le commérage dans un plan à deux dimensions. On obtient un coût qui est dans le pire des cas, 4 fois plus grand que le coût optimal.

L'amélioration de cet algorithme pour les agents dans le plan est un autre problème ouvert.

Ce mémoire a inspiré deux articles écrit conjointement avec d'autres chercheurs. L'article [2] considère le problème de ramasser l'information de tous les agents dans un

seul agent et l'article [3] considère en plus le problème de diffusion de l'information d'un agent à tous les autres. Dans les deux cas le coût est défini comme dans ce mémoire et le but est aussi d'optimiser le coût maximal encouru par les agents.

Chapitre 10

Annexes

10.1 Code logiciel iOS

10.1.1 ViewController.m

```
1  //
2  //  ViewController.m
3  //  Agent02
4  //
5  //  Created by Ayanami-Mac on 2013-05-22.
6  //  Copyright (c) 2013 Ayanami-Mac. All rights reserved.
7  //  Simulation des résultats
8  //  Mémoire intitulé: Problème de commérage pour les agents
       mobiles.
9  //  Julian Anaya: ingjuliananaya@gmail.com
10
11 #import "ViewController.h"
12
13 @interface ViewController ()
14
15 @end
16
17 @implementation ViewController
18
```

```
19 - (void)viewDidLoad
20 {
21     [super viewDidLoad];
22         // Do any additional setup after loading the view,
               typically from a nib.
23     NSInteger containnerLargeur =
           self.view.frame.size.height - 40;
24     agentContainer = [[UIView alloc]
           initWithFrame:CGRectMake(10, 10, containnerLargeur,
           200)];
25    [agentContainer setTag:12345]; //pour pouvoir récupérer
        le ń subview. ż
26
27     //Tag sur le view pour le retrouver facilement plus
           tard.
28     [[self.view viewWithTag:12345] removeFromSuperview];
29
30     //netoyage de subview master.
31     for(UIImageView *subview in [agentContainer subviews]){
32         [subview removeFromSuperview];
33     }
34
35     [self ActValeurAgents];
36 }
37
38 - (void)didReceiveMemoryWarning
39 {
40     [super didReceiveMemoryWarning];
41     // Dispose of any resources that can be recreated.
42 }
43
44 -(IBAction)ActValeurAgents{
45
```

```
46      //On vide les agents qui peuvent être contenus dans le
            container.
47      noAgents = (int)noAgentStepper.value;
48      noAgentLabel.text = [[NSString alloc]
            initWithFormat:@"No␣Agents:␣%d",noAgents];
49      CoutCalLabel.text = [[NSString alloc]
            initWithFormat:@"Cout␣Cal:␣%.3f",[self coutBrute]];
50      schemaCommerageCalcule = [self calculSchemaCommerage];
51      NSInteger tailleBlock = 18;
52      NSInteger barreTaille = 100;
53      NSInteger barreGap = 22;
54
55      //Tableaux de couleurs.
56      NSMutableArray *tableColeurs = [[NSMutableArray alloc]
            initWithObjects:[UIColor blackColor], [UIColor
            redColor], [UIColor greenColor], [UIColor
            whiteColor], [UIColor yellowColor], [UIColor
            orangeColor], [UIColor purpleColor], [UIColor
            brownColor], [UIColor magentaColor],nil];
57
58      //Chaine de caractères utilisé comme information à
            partager par les agents.
59      NSString *motTest = @"GATINEAU1234567890";
60
61      [[self.view viewWithTag:12345] removeFromSuperview];
62
63      for(UIImageView *subview in [agentContainer subviews]){
64          [subview removeFromSuperview];
65      }
66
67      //Création de chaque agent.
68      for (int i=0; i<noAgents; i++) {
69
70          //création de la barre de batterie.
```

```
71
72          UIProgressView *barreTest = [[UIProgressView alloc]
                initWithFrame:CGRectMake((barreTaille +
                barreGap)*(i/3), barreGap*(i%3), barreTaille,
                10)];
73          barreTest.progressTintColor = [tableColeurs
                objectAtIndex:i+1];;
74
75          [barreTest
                setProgressViewStyle:UIProgressViewStyleBar];
76          [barreTest setProgress:1.0f animated:YES];
77
78          [agentContainer addSubview:barreTest];
79
80          UIImageView *lettresContainer = [[UIImageView
                alloc] initWithFrame: CGRectMake(
                (agentContainer.frame.size.width
                /(noAgents-1))*i,
                agentContainer.frame.size.height
                -(tailleBlock*(noAgents-2)), tailleBlock,
                tailleBlock)];
81
82          //création de chaque lettre dans agent par séparé
83          for(int j=0; j<noAgents; j++) {
84              UILabel *textTest = [[UILabel alloc]
                    initWithFrame:CGRectMake(0,(tailleBlock*j),
                    tailleBlock, tailleBlock)];
85              if(i==j) textTest.text = [motTest
                    substringWithRange:NSMakeRange(j, 1)];
86              else textTest.text = @"␣";
87              textTest.textAlignment = NSTextAlignmentCenter;
88              textTest.layer.borderWidth = 2.0f;
89              textTest.font = [UIFont systemFontOfSize:13];
```

```
90              textTest.layer.borderColor = [UIColor
                    blackColor].CGColor;
91              textTest.backgroundColor = [tableColeurs
                    objectAtIndex:i+1];
92              [lettresContainer addSubview:textTest];
93          }
94
95          [agentContainer addSubview:lettresContainer];
96      }
97          [self.view addSubview:agentContainer];
98  }
99
100
101 -(float)coutBrute{
102     //Fonction qui retourne le cout telque calculé par
            l'algorithme.
103     NSInteger AgentsAllezRetour = 0;
104     float C_opt = 0.5;
105     float distanceAux = 0.0;
106
107     if((noAgents%2) !=0) {
108     while ((distanceAux-floorf(noAgents/2.0))<-0.00001)
            //le nombre des agents est impair.
109     {
110         AgentsAllezRetour++;
111         C_opt = (powf(3.0,
                AgentsAllezRetour)*powf((2.0/3.0),(int)(noAgents/2.0))
                + 1.0 - 2.0*powf(2.0, AgentsAllezRetour))
                /(powf(3.0, AgentsAllezRetour) *powf((2.0/3.0),
                (int)(noAgents/2.0)) *(2.0/5.0) -powf(2.0,
                AgentsAllezRetour));
112
113         distanceAux = log2f(powf(2.0,
                floorf(log2f(1.0/(2-C_opt))+2))*(2.0-C_opt)-1.0);
```

```
114         distanceAux = distanceAux -
                floorf(log2f(1.0/(2-C_opt))+2)*log2f(3.0);
115         distanceAux = distanceAux -
                log2f(1.0-2.0*(C_opt/5.0));
116         distanceAux = distanceAux / log2f(2.0/3.0);
117
118     }
119     }else {
120         //le nombre des agents est pair.
121
122         while ((distanceAux-floorf(noAgents/2.0))<-0.00001)
123         {
124             AgentsAllezRetour++;
125
126             C_opt = (powf(3.0, AgentsAllezRetour)
                    *powf((2.0/3.0),(noAgents/2.0)) *(5.0/4.0) +
                    1.0 - 2.0*powf(2.0, AgentsAllezRetour))
                    /(powf(3.0, AgentsAllezRetour)
                    *powf((2.0/3.0), (noAgents/2.0))*(1.0/2.0)
                    -powf(2.0, AgentsAllezRetour));
127
128             distanceAux = log2f(powf(2.0,
                    floorf(log2f(1.0/(2.0-C_opt))+2.0001))
                    *(2.0-C_opt) -1.0);
129             distanceAux = distanceAux -
                    floorf(log2f(1.0/(2-C_opt))+2.0001)*log2f(3.0);
130             distanceAux = distanceAux -
                    log2f((5.0-2*C_opt)/4.0);
131             distanceAux = distanceAux / log2f(2.0/3.0);
132
133         }
134
135     }
136
```

```
137     return C_opt;
138 }
139
140
141 -(NSMutableArray*)calculSchemaCommerage{
142     //retourne l'ensemble des mouvements a effectuer pour
            les agents.
143     NSMutableArray *schemaDeCommerage = [NSMutableArray
            arrayWithCapacity:0];
144     NSMutableArray *mouvement = [NSMutableArray
            arrayWithCapacity:0];
145     float coutCal = [self coutBrute];
146     float pointInf = 0.00; //variable qui garde jusque à
            quel point l'information a arrivé.
147     float posAgent[noAgents];
148
149     //Les agents extrêmes son un case spécial.
150     posAgent[0] = 0; //actualisation de la position de
            l'agent.
151     if (coutCal<1) {
152         //on fait un simple déplacement.
153
154         mouvement = [NSMutableArray arrayWithCapacity:0];
155         [mouvement addObject:[NSNumber
                numberWithInteger:0]]; //premier agent.
156         [mouvement addObject:[NSNumber
                numberWithFloat:coutCal]]; //premier agent.
157         [schemaDeCommerage addObject:mouvement];
158         posAgent[0] = posAgent[0] + coutCal;
                //actualisation de la position de l'agent.
159
160         mouvement = [NSMutableArray arrayWithCapacity:0];
```

```
161         [mouvement addObject:[NSNumber
                numberWithInteger:(noAgents-1)]]; //dernier
                agent.
162         [mouvement addObject:[NSNumber
                numberWithFloat:(-1*coutCal)]]; //dernier agent.
163         [schemaDeCommerage addObject:mouvement];
164     } else {
165         //si les agents extrêmes dépassent ses agents
                voisins.
166         mouvement = [NSMutableArray arrayWithCapacity:0];
167         [mouvement addObject:[NSNumber
                numberWithInteger:0]]; //premier agent.
168         [mouvement addObject:[NSNumber numberWithFloat:1]];
                //visite son agent voisin.
169         [schemaDeCommerage addObject:mouvement];
170         posAgent[0] = posAgent[0] + 1; //actualisation de
                la position de l'agent.
171
172
173         mouvement = [NSMutableArray arrayWithCapacity:0];
174         [mouvement addObject:[NSNumber
                numberWithInteger:0]]; //premier agent.
175         [mouvement addObject:[NSNumber
                numberWithFloat:(coutCal-1)]]; //continue jusque
                épuisé sa batterie.
176         [schemaDeCommerage addObject:mouvement];
177         posAgent[0] = posAgent[0] + coutCal - 1;
                //actualisation de la position de l'agent.
178
179
180         mouvement = [NSMutableArray arrayWithCapacity:0];
181         [mouvement addObject:[NSNumber
                numberWithInteger:(noAgents-1)]]; //dernier
                agent.
```

```
182         [mouvement addObject:[NSNumber
                numberWithFloat:-1]]; //dernier agent.
183         [schemaDeCommerage addObject:mouvement];
184
185         mouvement = [NSMutableArray arrayWithCapacity:0];
186         [mouvement addObject:[NSNumber
                numberWithInteger:(noAgents-1)]]; //dernier
                agent.
187         [mouvement addObject:[NSNumber
                numberWithFloat:-(coutCal-1)]]; //dernier agent.
188         [schemaDeCommerage addObject:mouvement];
189
190     }
191     pointInf = coutCal;
192
193     for (NSInteger i=1; i<noAgents/2; i++) {
194         posAgent[i] = i; //actualisation de la position de
                l'agent.
195
196         //Question pour savoir si l'agent sera de type
                aller-retour ou serpentine.
197         if(i<=pointInf){
198             //l'agent est aller-retour.
199             //question pour savoir si l'agent va dépasser
                    l'agent voisin.
200             float distance = (coutCal+pointInf-i)/2;
201
202             if((distance) > 1){
203                 //on déplace l'agent, il faut le faire en
                        deux étapes.
204                 mouvement = [NSMutableArray
                        arrayWithCapacity:0];
205                 [mouvement addObject:[NSNumber
                        numberWithInteger:i]]; //agent droite.
```

```
206                 [mouvement addObject:[NSNumber
                        numberWithFloat:1]]; //agent droite.
207                 [schemaDeCommerage addObject:mouvement];
208                 posAgent[i]=posAgent[i]+1; //actualisation
                        de la position de l'agent.
209
210
211                 mouvement=[NSMutableArray
                        arrayWithCapacity:0];
212                 [mouvement addObject:[NSNumber
                        numberWithInteger:i]]; //agent droite.
213                 [mouvement addObject:[NSNumber
                        numberWithFloat:distance - 1]]; //agent
                        droite.
214                 [schemaDeCommerage addObject:mouvement];
215                 posAgent[i]=posAgent[i]+distance-1;
                        //actualisation de la position de
                        l'agent.
216
217                 mouvement=[NSMutableArray
                        arrayWithCapacity:0];
218                 [mouvement addObject:[NSNumber
                        numberWithInteger:(noAgents-1-i)]];
                        //agent gauche.
219                 [mouvement addObject:[NSNumber
                        numberWithFloat:-1]]; //agent gauche.
220                 [schemaDeCommerage addObject:mouvement];
221
222                 mouvement=[NSMutableArray
                        arrayWithCapacity:0];
223                 [mouvement addObject:[NSNumber
                        numberWithInteger:(noAgents-1-i)]];
                        //agent gauche.
```

```
224                     [mouvement addObject:[NSNumber
                            numberWithFloat:-(distance - 1)]];
                            //agent gauche.
225                     [schemaDeCommerage addObject:mouvement];
226
227
228                 } else {
229                     //l'agent n'arrive pas a dépasser l'agent
                            voisin.
230                     mouvement=[NSMutableArray
                            arrayWithCapacity:0];
231                     [mouvement addObject:[NSNumber
                            numberWithInteger:i]]; //agent droite.
232                     [mouvement addObject:[NSNumber
                            numberWithFloat:distance]]; //agent
                            droite.
233                     [schemaDeCommerage addObject:mouvement];
234                     posAgent[i]=posAgent[i]+distance;
                            //actualisation de la position de
                            l'agent.
235
236                     mouvement=[NSMutableArray
                            arrayWithCapacity:0];
237                     [mouvement addObject:[NSNumber
                            numberWithInteger:(noAgents-1-i)]];
                            //agent gauche.
238                     [mouvement addObject:[NSNumber
                            numberWithFloat:-(distance)]]; //agent
                            gauche.
239                     [schemaDeCommerage addObject:mouvement];
240                 }
241                 pointInf = pointInf + (powf(2.0, i)-1) /
                        powf(2.0, i);
242             }
```

```
243         else {
244             //l'agent est de type serpent.
245             //il faut y allez chercher l'information qui se
                    trouve sur le point pointInf.
246             float distanceArriere = i - pointInf;
247             mouvement = [NSMutableArray
                    arrayWithCapacity:0];
248             [mouvement addObject:[NSNumber
                    numberWithInteger:i]]; //agent droite.
249             [mouvement addObject:[NSNumber
                    numberWithFloat:-distanceArriere]]; //agent
                    droite.
250             [schemaDeCommerage addObject:mouvement];
251             posAgent[i] = posAgent[i] - distanceArriere;
                    //actualisation de la position de l'agent.
252
253             float
                    distanceAvance=(coutCal-3*distanceArriere)/2;
254             //pour apporter l'information plus loin.
255             mouvement = [NSMutableArray
                    arrayWithCapacity:0];
256             [mouvement addObject:[NSNumber
                    numberWithInteger:i]]; //agent droite.
257             [mouvement addObject:[NSNumber
                    numberWithFloat:distanceArriere +
                    distanceAvance]]; //agent droite.
258             [schemaDeCommerage addObject:mouvement];
259             posAgent[i] = posAgent[i] + distanceArriere +
                    distanceAvance; //actualisation de la
                    position de l'agent.
260
261             mouvement=[NSMutableArray arrayWithCapacity:0];
```

```
262                 [mouvement addObject:[NSNumber
                        numberWithInteger:(noAgents-1-i)]]; //agent
                        gauche.
263                 [mouvement addObject:[NSNumber
                        numberWithFloat:distanceArriere]]; //agent
                        gauche.
264                 [schemaDeCommerage addObject:mouvement];
265
266                 mouvement=[NSMutableArray arrayWithCapacity:0];
267                 [mouvement addObject:[NSNumber
                        numberWithInteger:(noAgents-1-i)]]; //agent
                        gauche.
268                 [mouvement addObject:[NSNumber
                        numberWithFloat:-(distanceArriere +
                        distanceAvance)]]; //agent gauche.
269                 [schemaDeCommerage addObject:mouvement];
270
271                 pointInf=pointInf+distanceAvance+distanceArriere;
272             }
273         }
274         //Si le nombre des agents est impair il manque faire
                l'agent du milieu.
275         if((noAgents%2) !=0)
276         {
277             posAgent[noAgents/2] = noAgents/2;
278             float distance = (noAgents/2) - pointInf;
279             mouvement = [NSMutableArray arrayWithCapacity:0];
280             [mouvement addObject:[NSNumber
                    numberWithInteger:((noAgents/2))]]; //agent
                    gauche.
281             [mouvement addObject:[NSNumber
                    numberWithFloat:-distance]]; //agent gauche.
282             [schemaDeCommerage addObject:mouvement];
283             posAgent[noAgents/2]=posAgent[noAgents/2]-distance;
```

```
284
285             mouvement = [NSMutableArray arrayWithCapacity:0];
286             [mouvement addObject:[NSNumber
                    numberWithInteger:((noAgents/2))]]; //agent
                    droite.
287             [mouvement addObject:[NSNumber
                    numberWithFloat:+2*distance]]; //agent droite.
288             [schemaDeCommerage addObject:mouvement];
289             posAgent[noAgents/2]=posAgent[noAgents/2]+2*distance;
290
291             mouvement = [NSMutableArray arrayWithCapacity:0];
292             [mouvement addObject:[NSNumber
                    numberWithInteger:((noAgents/2))]]; //agent
                    gauche.
293             [mouvement addObject:[NSNumber
                    numberWithFloat:-2*distance]]; //agent gauche.
294             [schemaDeCommerage addObject:mouvement];
295             posAgent[noAgents/2]=posAgent[noAgents/2]-2*distance;
296         }
297         //phase de distribution:
298
299         for (NSInteger i=(noAgents/2)-1; i>0; i--) {
300             mouvement = [NSMutableArray arrayWithCapacity:0];
301             [mouvement addObject:[NSNumber
                    numberWithInteger:i]]; //agent droite.
302             [mouvement addObject:[NSNumber
                    numberWithFloat:-(posAgent[i] -
                    posAgent[i-1])]]; //agent droite.
303             [schemaDeCommerage addObject:mouvement];
304
305             mouvement = [NSMutableArray arrayWithCapacity:0];
306             [mouvement addObject:[NSNumber
                    numberWithInteger:(noAgents-1-i)]]; //agent
                    gauche.
```

```
307             [mouvement addObject:[NSNumber
                    numberWithFloat:+(posAgent[i]-posAgent[i-1])]];
                    //agent gauche.
308             [schemaDeCommerage addObject:mouvement];
309         }
310         //Log du schéma de commérage.
311         NSLog(@"Schéma de commérage pour (%d) agents.",noAgents);
312
313         for(NSInteger i=0; i<[schemaDeCommerage count]; i++){
314             NSLog(@"S_étape(%d, %2f2)",[[[schemaDeCommerage
                    objectAtIndex:i] objectAtIndex:0]
                    integerValue],[[[schemaDeCommerage
                    objectAtIndex:i] objectAtIndex:1] floatValue]);
315
316         }
317         return schemaDeCommerage;
318 }
319
320 - (IBAction)buttonAction:(id)sender{
321
322         index = 0;
323         [self BougerAgents];
324 }
325
326 -(void) BougerAgents{
327
328         //fonction qui prendre le schéma de commérage calculé
                et fait bouger les agents sur l'écran.
329         //Calcul du schéma de commérage.
330         NSInteger agentAbouger = [[[schemaCommerageCalcule
                objectAtIndex:index] objectAtIndex:0] integerValue];
331         float distance = [[[schemaCommerageCalcule
                objectAtIndex:index] objectAtIndex:1] floatValue];
332         //agent a bouger.
```

```
333     UIImageView *agentAbougerFrame = [[[self.view
            viewWithTag:12345] subviews]
            objectAtIndex:agentAbouger*2+1];
334
335     //Batterie a bouger.
336     UIProgressView *batterieAbougerFrame = [[[self.view
            viewWithTag:12345] subviews]
            objectAtIndex:agentAbouger*2];
337
338     agentAbougerFrame.layer.zPosition = -1;
339     CGRect agentFrameFinal = agentAbougerFrame.frame;
340     agentFrameFinal.origin.x = agentFrameFinal.origin.x +
            (int)(distance*[self.view
            viewWithTag:12345].frame.size.width/(noAgents-1));
341
342     //On généré le mouvement.
343    //animation de la batterie utilise.
344     float coutCal = [self coutBrute];
345     if (distance<0) {
346         distance = distance * -1.0f;
347     }
348
349     [batterieAbougerFrame
            setProgress:(float)((float)batterieAbougerFrame.progress
           - (float)distance/(float) coutCal) animated:YES];
350
351     if(batterieAbougerFrame.progress<0.00010){
352         [batterieAbougerFrame setProgress:0.0f];
353     }
354
355     [UIView animateWithDuration:1.5
356                           //delay:1.5
357                          //option:UIViewAnimationCurveEaseOut
```

```
358                         animations:^{agentAbougerFrame.frame =
                                agentFrameFinal;}
359                         completion:^(BOOL finished){
360                             NSLog(@"Done agent %d, index
                                    %d",agentAbouger,index);
361                             NSLog(@"Progress bar %d,
                                    %3f10",agentAbouger,
                                    batterieAbougerFrame.progress);
362                             [self
                                    InterchangeInformation:agentAbouger];
363                                 }
364     ];
365 }
366
367 -(void) InterchangeInformation:(NSInteger)agentEnMouvement1{
368     //Fonction qui fait l'inter change d'information entre
            les agents qui se trouvent dans un point commune.
369
370     //on doit trouver les deux agents qui sont involucrés
            dans l'inter change d'information.
371
372     //agent a bouger.
373     UIImageView *agentAbougerFrame = [[[self.view
            viewWithTag:12345] subviews]
            objectAtIndex:agentEnMouvement1*2+1];
374
375     //Frame pour l'agent le plus proche.
376     UIImageView *agentAbougerFrameProche;
377
378     //On parcours les agents voisins pour savoir si on est
            dans la même position.
379     NSInteger distanceEntreAgents = 1024;
380     for(NSInteger i=0; i<noAgents; i++){
381     //on prend les informations des autres agents.
```

```
382         if(i!=agentEnMouvement1){
383         UIImageView *agentAbougerFrameTest = [[[self.view
                viewWithTag:12345] subviews]
                objectAtIndex:i*2+1];
384
385         NSInteger distanceEntreAgentsTest =
                agentAbougerFrame.frame.origin.x -
                agentAbougerFrameTest.frame.origin.x;
386
387             if(distanceEntreAgentsTest<0){
388                 distanceEntreAgentsTest=distanceEntreAgentsTest*-1;
389             }
390             if(distanceEntreAgentsTest<distanceEntreAgents){
391                 //Deux agents proches.
392                 //On a le voisine sur le même point.!!
393                 distanceEntreAgents=distanceEntreAgentsTest;
394                 agentAbougerFrameProche=agentAbougerFrameTest;
395             }
396         }
397     }
398
399     //on vérifie s?il y a eu un agent proche.
400
401     if(distanceEntreAgents<5){
402         NSLog(@"Distance %d",distanceEntreAgents);
403         //On a le voisine sur le même point.!!
404         //on fait bouger chaque élément pour partager
                l'information.
405         for (NSInteger i=0; i<noAgents; i++) {
406             //on bouge chaque lettre dans l'agent.
407             UILabel *lettreFrameAux = [[agentAbougerFrame
                    subviews] objectAtIndex:i];
```

```
408                 UILabel *lettreFrameVoisinAux =
                        [[agentAbougerFrameProche subviews]
                        objectAtIndex:i];
409                 NSLog(@"Agent1:␣␣%@,␣Agnt␣voisin:␣
                        %@",lettreFrameAux.text,lettreFrameVoisinAux.text);
410                 if ([lettreFrameAux.text isEqual: @"␣"]) {
411                     lettreFrameAux.text=lettreFrameVoisinAux.text;
412
413                 }
414                 if ([lettreFrameVoisinAux.text isEqual:@"␣"]) {
415                     lettreFrameVoisinAux.tex
                            =lettreFrameAux.text;
416                 }
417                 NSLog(@"Agent1:␣␣%@,␣Agnt␣voisin:␣
                        %@",lettreFrameAux.text,lettreFrameVoisinAux.text);
418             }
419             }
420             if(index<([schemaCommerageCalcule count]-1)){
421             //on n'a pas fini de parcourir le schéma de
                    commérage calculé
422             index++;
423             agentAbougerFrame.layer.zPosition = 1;
424             //On fait l'inter change d'information s'il y a
                    lieu..!!
425             [self BougerAgents];
426         }
427 }
428 @end
```

10.1.2 ViewController.h

```
1
2
3 //
4 //  ViewController.h
```

```
5  //  Agent02
6  //
7  //  Created by Ayanami-Mac on 2013-05-22.
8  //  Copyright (c) 2013 Ayanami-Mac. All rights reserved.
9  //
10 //  Julian Anaya ingjuliananaya@gmail.com
11
12 #import <UIKit/UIKit.h>
13 #import "UIImage+Sprite.h"
14 #import "QuartzCore/QuartzCore.h"
15 #import "math.h"
16
17
18 @interface ViewController : UIViewController{
19
20
21     UIView *agentContainer; //  ń View ż qui contient
          l'ensemble des agents.
22     NSMutableArray *schemaCommerageCalcule; //  ń array ż
          avec le schéma de commérage calculé
23     NSInteger noAgents; //Nombre des agents
24     NSInteger index; //indexe utilisé pendant le partage
          d'information.
25     IBOutlet UILabel *noAgentLabel; //Label sur l'écran qui
          montre le nombres agents actuel.
26     IBOutlet UILabel *CoutCalLabel; //Label pour le coût
          calculé ou choisi par l'utilisateur.
27     IBOutlet UIStepper *noAgentStepper; //sélecteur pour
          choisir le nombre des agents.
28 }
29
30 @end
```

10.2 Code pour logiciels de Matlab et Octave

10.2.1 distance _ beta.m

```
1
2  function C_opt = distan_beta(N)
3  %Fonction qui retourne le cout totaux qui devra parcourir
4  %un agent pour passer le message entre les agents et
       combien des agents
5  %utilisent le mouvement aller-retour.
6  %testé sur octave-03.4.0
7
8  if(mod(N,2)==0)
9          C_opt = distan_beta_pa00(N); %Si N est pair on
               utilise ce sub-routine
10 else
11         C_opt = distan_beta_im00(N); %Si non on utilise
               celui pour N impair.
12
13 end
14
15 end
```

10.2.2 distance _ beta _ pa00.m

```
1
2  function C_opt_sortie = distan_beta_pa00( N )
3  %Fonction qui retourne le cout totaux qui devra parcourir
4  %un agent pour passer le message entre les agents.
5  %fonctionne seulement pour des valeurs pair des agents.
6  %la recherche binaire est fait sur le nombre des agent qui
       font
7  %le mouvement aller-retour.
8  %testé sur octave-03.4.0
9
```

```
10 lim_bas = 0; %nombre des agents avec le mouvement
       aller-retour.
11 C_opt = 0.5; %valeur initial pour le cout.
12 distance_par = 0.01;
13
14 while (abs(double(distance_par) -
       double(fix(N./2)))>0.0000000001)
15     lim_bas= lim_bas + 1;
16
17         A = ((3)^lim_bas)*(2/3)^(fix(N/2));
18         B = A;
19         A = A*(5/4) + 1 - 2*(2^lim_bas);
20         B = B*(1/2) - (2^lim_bas);
21         C_opt = A/B;
22
23     distance_par = log2((2-C_opt)*(2^fix(log2(1/(2-C_opt))
           +2))-1);
24         distance_par = distance_par -fix(log2(1/(2-C_opt))
               +2)*log2(3);
25         distance_par = distance_par - log2((5-2*C_opt)/4);
26         distance_par = distance_par/log2(2/3);
27 end
28
29 C_opt_sortie = zeros(1,2);
30 C_opt_sortie(1,1) = C_opt;
31 C_opt_sortie(1,2) = lim_bas;
32
33 end
```

Bibliographie

[1] Alpern, S., and Gal, S. *The theory of search games and rendezvous.* Boston, MA : Kluwer Academic Publishers, 2003.

[2] Anaya, J., Chalopin, J., Czyzowicz, J., Labourel, A., Pelc, A., and Vaxès, Y. Collecting information by power-aware mobile agents. In *DISC*, vol. 7611 of *Lecture Notes in Computer Science*. 2012, pp. 46–60.

[3] Anaya, J., Chalopin, J., Czyzowicz, J., Labourel, A., Pelc, A., and Vaxès, Y. Convergecast and broadcast by power-aware mobile agents, manuscript, 2013.

[4] Chalopin, J. Exemple des agents avec distances inégales. information privée, 2012.

[5] Cieliebak, M., Flocchini, P., Prencipe, G., and Santoro, N. Solving the robots gathering problem. In *Proceedings of the 30th International Conference on Automata, Languages and Programming*, ICALP'03. 2003, pp. 1181–1196.

[6] Czyzowicz, J., Kosowski, A., and Pelc, A. How to meet when you forget : log-space rendezvous in arbitrary graphs. In *Proceeding of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC '10. 2010, pp. 450–459.

[7] Czyzowicz, J., Labourel, A., and Pelc, A. How to meet asynchronously (almost) everywhere. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10. 2010, pp. 22–30.

[8] D'Angelo, G., Di Stefano, G., and Navarra, A. How to gather asynchronous oblivious robots on anonymous rings. In *Distributed Computing*. Springer, 2012, pp. 326–340.

[9] D'Angelo, G., Stefano, G., Klasing, R., and Navarra, A. Gathering of robots on anonymous grids without multiplicity detection. In *Structural Information*

and Communication Complexity, G. Even and M. M. Halldórsson, Eds., vol. 7355 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2012, pp. 327–338.

[10] D'Angelo, G., Stefano, G., and Navarra, A. Gathering of six robots on anonymous symmetric rings. In *Structural Information and Communication Complexity*, A. Kosowski and M. Yamashita, Eds., vol. 6796 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2011, pp. 174–185.

[11] De Marco, G., Gargano, L., Kranakis, E., Krizanc, D., Pelc, A., and Vaccaro, U. Asynchronous deterministic rendezvous in graphs. *Theor. Comput. Sci. 355* (April 2006), 315–326.

[12] Dessmark, A., Fraigniaud, P., Kowalski, D. R., and Pelc, A. Deterministic rendezvous in graphs. *Algorithmica 46* (September 2006), 69–96.

[13] Ellen, F., Subramanian, S., and Welch, J. Maintaining information about nearby processors in a mobile environment. In *Distributed Computing and Networking*, S. Chaudhuri, S. Das, H. Paul, and S. Tirthapura, Eds., vol. 4308 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2006, pp. 193–202.

[14] Flocchini, P., Kranakis, E., Krizanc, D., Santoro, N., and Sawchuk, C. Mobile agent rendezvous in a ring. In *LATIN 2004 : Theoretical Informatics*, vol. 2976. 2004, pp. 599–608.

[15] Flocchini, P., Prencipe, G., Santoro, N., and Widmayer, P. Gathering of asynchronous oblivious robots with limited visibility. In *Proceedings of the 18th Annual Symposium on Theoretical Aspects of Computer Science*, STACS '01. Springer-Verlag, London, UK, 2001, pp. 247–258.

[16] Gąsieniec, L., Kranakis, E., Pelc, A., and Xin, Q. Deterministic m2m multicast in radio networks. In *Automata, Languages and Programming*. Springer, 2004, pp. 670–682.

[17] Gasieniec, L., and Potapov, I. Gossiping with unit messages in known radio networks. In *Proceedings of the IFIP 17th World Computer Congress-TC1 Stream/2nd IFIP International Conference on Theoretical Computer Science : Foundations of Information Technology in the Era of Networking and Mobile Computing*. 2002, pp. 193–205.

[18] Gąsieniec, L. On efficient gossiping in radio networks. In *Structural Information and Communication Complexity*, S. Kutten and J. Zerovnik, Eds., vol. 5869 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2010, pp. 2–14.

[19] Gąsieniec, L., Peleg, D., and Xin, Q. Faster communication in known topology radio networks. *Distributed Computing 19* (2007), 289–300.

[20] Gąsieniec, L., Potapov, I., and Xin, Q. Time efficient centralized gossiping in radio networks. *Theoretical Computer Science 383*, 1 (2007), 45 – 58. Structural Information and Communication Complexity (SIROCCO 2004).

[21] Guilbault, S., and Pelc, A. Asynchronous rendezvous of anonymous agents in arbitrary graphs. In *Principles of Distributed Systems*, A. Fernandez Anta, G. Lipari, and M. Roy, Eds., vol. 7109 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2011, pp. 421–434.

[22] Guilbault, S., and Pelc, A. Gathering asynchronous oblivious agents with local vision in regular bipartite graphs. *Theoretical Computer Science 509*, 0 (2013), 86 – 96.

[23] Hromkovic, J., Klasing, R., Pelc, A., Ruzicka, P., and Unger, W. *Dissemination of Information in Communication Networks*. Texts in Theoretical Computer Science An EATCS Series. Springer-Verlag, Berlin/Heidelberg, 2005.

[24] Klasing, R., Kosowski, A., and Navarra, A. Taking advantage of symmetries : Gathering of asynchronous oblivious robots on a ring. In *Proceedings of the 12th International Conference on Principles of Distributed Systems*, OPODIS '08. Springer-Verlag, Berlin, Heidelberg, 2008, pp. 446–462.

[25] Klasing, R., Markou, E., and Pelc, A. Gathering asynchronous oblivious mobile robots in a ring. *Theor. Comput. Sci. 390* (January 2008), 27–39.

[26] Kореń, M. Gathering small number of mobile asynchronous robots on ring. *Zeszyty Naukowe 18* (2010), 325–331.

[27] Lynch, N. A. *Distributed algorithms*. Morgan Kaufmann, 1996.

[28] Miller, A. Gossiping in jail. In *Algorithmic Aspects of Wireless Sensor Networks*, S. Dolev, Ed., vol. 5804 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2009, pp. 242–251.

[29] Miller, A. Gossiping in one-dimensional synchronous ad hoc wireless radio networks. In *Proceedings of the 4th International Workshop on Theoretical Aspects of Dynamic Distributed Systems*, TADDS '12. ACM, New York, NY, USA, 2012, pp. 32–43.

[30] Prencipe, G. Corda : Distribuited coordination os a set of autonomous mobile robots. In *Proceedings of the 4th European Research Seminar on Advances in Distribuited Systems*, ERSADS '01. 2001, pp. 185–189.

[31] Prencipe, G. Impossibility of gathering by a set of autonomous mobile robots. *Theor. Comput. Sci. 384* (October 2007), 222–231.

[32] Ta-Shma, A., and Zwick, U. Deterministic rendezvous, treasure hunts and strongly universal exploration sequences. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2007, pp. 599–608.