UNIVERSITÉ DU QUÉBEC EN OUTAOUAIS

APPROCHE DÉTERMINISTE DES AGENTS MOBILES DANS LE PLAN

MÉMOIRE
PRÉSENTÉ
COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN INFORMATIQUE

PAR

ZACHARIE JONAS E.M TCHUIDJANG

MARS 2025

Prof. Andrzej Pelc ..... Directeur de recherche
Prof. Nadia Baaziz ..... Président du jury
Prof. Mohand Said Allili ..... Membre du jury

Table des matières

Liste des figures **3**

1 Introduction **1**

2 Revue de la littérature **5**
2.1 Introduction 5
2.2 Le rendez-vous synchrone 6
2.3 Le rendez-vous asynchrone 10
2.4 Autres études 14

3 Modèle, problèmes et notions préliminaires **23**
3.1 Le modèle 23
3.2 Problèmes 23
3.3 Notions préliminaires 25

4 Chasse au trésor dans le plan **26**

5 Approche avec la grandeur de l'espace des étiquettes connue **28**
5.1 Algorithme de rendezvous 28

6 Approche avec la grandeur de l'espace des étiquettes inconnue **33**
6.1 Algorithme de l'approche 33
6.2 Borne inférieure sur le temps de l'approche 39

7 Simulation de l'algorithme **40**
7.1 Informations techniques sur le logiciel 40
7.1.1 Langage et outils 40
7.1.2 Défis techniques 41

7.2 Fonctionnalités du logiciel 41
7.3 Capture de l'écran pendant la simulation 42

8 Conclusion **45**

Liste des figures

3.1 Problème de chasse au trésor : le point noir représente l'agent 24

4.1 mouvement de la spirale produit par l'agent lors de l'exécution de l'algorithme *Chasse au trésor* 27

6.1 Présentation d'un déphasage entre agents 38

7.1 Première capture : les agents sont en positions initiale 42
7.2 Deuxième capture : les agents sont en mouvement 43
7.3 Troisième capture : L'approche des agents. 44

Résumé

Dans ce mémoire nous étudions deux problèmes principaux. Le premier problème connu sous le nom de chasse au trésor présente un agent modélisé par un point qui navigue dans le plan et doit retrouver un trésor situé à distance au plus D de celui-ci, avec D inconnu. Le trésor est modélisé par un point immobile. Les mouvements de l'agent sont contraints : il peut se déplacer à distance quelconque positive mais seulement dans l'une des directions cardinales. De plus, à partir de sa position actuelle, l'agent a une visibilité limitée à une distance maximale de 1 unité autour de lui. L'objectif de l'agent est de voir le trésor, c'est-à-dire de s'approcher à une distance maximale de 1 unité du trésor.

Le deuxième problème, connu sous le nom de l'approche et qui est un cas plus géneral du premier, implique deux agents modélisés par des points mobiles dans le plan qui doivent se rapprocher l'un de l'autre. Les agents sont initialement situés à distance au plus D l'un de l'autre, D étant inconnu des deux agents. Chaque agent est identifié par une étiquette représentée par un entier positif unique. Les agents se déplacent de manière synchrone, en suivant les mêmes règles de mouvement que dans le premier problème. Ils commencent simultanément et procèdent en rondes synchrones qui durent une unité de temps. Pendant chaque ronde, un agent peut soit rester immobile soit se déplacer à une distance de 1 unité. Ils ont le même champ de vision restreint à une distance maximale de 1 unité autour d'eux. L'objectif est que les deux agents se rapprochent à une distance maximale de 1 unité l'un de l'autre, c'est-à-dire qu'ils se voient.

Pour chacun de ces problèmes, le temps est mesuré par le nombre de rondes qui s'écoulent entre la mise en mouvement et l'atteinte de l'objectif (trouver le trésor ou l'approche, selon le cas).

Ce mémoire démontre que ces problèmes sont résolubles grâce à des algorithmes déterministes. Pour chaque algorithme présenté, nous prouvons son exactitude et analysons sa complexité temporelle.

Chapitre 1

Introduction

Imaginez-vous dans une ville inconnue, où vous vous êtes retrouvé(e) séparé(e) d'un(e) ami(e) (ou d'un frère, d'une sœur, etc.) au cours de votre marche. Vous n'avez aucun moyen de communication avec cette personne : vous ne pouvez la joindre ni par téléphone, ni d'aucune autre manière, et il est interdit de crier dans les rues de cette ville. Comment vous y prendriez-vous pour retrouver cette personne ? La question se pose également pour la personne dont vous avez été séparé(e).

De même, avez-vous déjà réfléchi à ce que vous feriez dans une situation similaire, où au lieu de chercher une autre personne, vous seriez à la recherche d'un restaurant célèbre du coin, sans la possibilité d'utiliser votre téléphone ou de demander des renseignements aux passants ?

Le premier problème est l'un des problèmes étudiés intensément en informatique, notamment en calcul distribué. Il est connu sous le nom de problème de l'approche des agents. En fonction de l'environnement, ce problème concerne des agents mobiles dispersés dans un espace, soumis à différentes contraintes (représentant plus ou moins la réalité), qui doivent se rapprocher les uns des autres jusqu'à une distance qui leur permet de se voir. Un autre problème très similaire, appelé problème de la chasse au trésor, exige qu'un agent retrouve un bbutut immobile. On peut constater que le premier problème est une généralisation du deuxième. Dans chaque cas on veut accomplir la tâche aussitôt que possible.

L'environnement peut être représenté ou modélisé par un réseau, comme dans nos exemples, où les rues d'une ville peuvent faire partie d'un réseau. L'environnement peut également être modélisé par le plan. Dans ce cas on pense aux personnes qui se sont perdu dans une forêt ou dans le désert.

Les agents sont souvent des entités autonomes de calcul. Ils ne sont généralement pas distinguables ; toutefois, dans certaines applications, on peut leur assigner des étiquettes pour les différencier. Cependant, ils doivent exécuter la même procédure. Les agents peuvent fonctionner de manière synchrone : dans ce cas, ils agissent par rondes, chaque ronde étant régi par une horloge globale commune à tous les agents. Ils peuvent également fonctionner de manière asynchrone : dans ce cas la durée de chaque action de l'agent est finie mais indéterminée. Souvent, ils disposent d'un système de coordonnées global ; dans d'autres cas, chaque agent possède son propre système de coordonnées local et les systèmes peuvent ne pas être cohérents.

Les contraintes se résument à l'ensemble des actions que chaque agent peut ou ne peut pas faire, aux éléments contrôlés par les agents, ainsi qu'à ceux contrôlés par une entité extérieure, généralement appelée adversaire. Chacune de ces contraintes représente, d'une certaine manière, la réalité. Parmi les contraintes rencontrées dans la littérature, on peut citer :

1. L'absence de communication entre les agents : C'est le cas du premier problème évoqué ci-dessus. Imaginez que, pendant que vous cherchez l'un l'autre, vous passez dans deux rues voisines où vous pourriez crier vos noms. Malheureusement, il est interdit de crier dans cette ville, et vous passez ainsi à proximité l'un de l'autre sans le savoir.
2. Le fonctionnement asynchrone des agents : Cette contrainte rend le problème plus complexe. En général, un agent effectue des calculs basés sur les informations qu'il possède, obtenues via ses capteurs. Dans un fonctionnement asynchrone, les agents agissent avec des vitesses possiblement différentes, ce qui signifie qu'un agent peut effectuer des calculs basés sur des informations obsolètes. Cependant, ce type de fonctionnement reflète davantage la réalité.
3. La présence d'un agent malicieux : Un agent malicieux est un agent auquel tout est permis : il peut changer son étiquette (si celle-ci existe), se faire passer pour un autre agent, ou envoyer de fausses informations. Dans les situations où les agents communiquent entre eux, la présence d'un agent malicieux pose un problème important car un agent pourrait, comme dans le cas ci-haut, se retrouver a faire des calculs sur la base des informations fausses.

L'étude des problèmes de la chasse au trésor et de l'approche est très importante car elle n'a pas que des applications dans l'interaction humaine. On trouve aussi des

applications dans l'interaction animale, comme les groupes d'oiseaux migrateurs venant de différents endroits et devant se regrouper à une destination commune. Des applications importantes concernent le domaine de la robotique. Un robot mobile doit trouver une épave au fond de la mer : voici une application du problème de la chasse au trésor. Deux robots mobiles naviguent dans un terrain contaminé et ramassent des échantillons du sol. Ensuite ils doivent se rencontrer : le but de la rencontre est souvent de permettre un échange de données collectées ou de coordonner une tâche future. : voici une application du problème de l'approche dans le plan.

Parmi les applications pratiques du problème de la chasse au trésor on peut encore citer :

1. **Recherche et sauvetage** : Dans des situations de recherche et de sauvetage, comme après une catastrophe naturelle ou un accident, il est crucial de localiser rapidement des personnes ou des objets dans un environnement potentiellement dangereux. Le problème de la chasse au trésor offre un cadre pour développer des stratégies de recherche efficaces dans de telles situations.
2. **Surveillance et sécurité** : Dans le domaine de la sécurité, la surveillance de zones sensibles telles que les frontières, les installations militaires ou les sites industriels peut nécessiter la localisation rapide d'objets suspects ou de personnes indésirables. Les techniques développées pour la chasse au trésor peuvent être adaptées à ces scénarios pour améliorer la sécurité et la surveillance.
3. **Exploration de l'environnement** : Les drones et les robots terrestres sont de plus en plus utilisés pour explorer des environnements difficiles d'accès ou dangereux pour les humains, tels que des zones sinistrées, des régions polaires ou des environnements extraterrestres. La résolution du problème de la chasse au trésor peut aider à développer des stratégies d'exploration efficaces pour ces applications.
4. **Jeux et divertissements** : En dehors des applications pratiques, le problème de la chasse au trésor est également pertinent dans le domaine des jeux et des divertissements. Des jeux de réalité augmentée aux applications de géocaching, les concepts de recherche de trésors peuvent être utilisés pour créer des expériences ludiques et interactives pour les joueurs.

La résolution du problème de l'approche des agents présente aussi plusieurs motivations et applications potentielles dans divers domaines :

1. **Robotique** : Dans le domaine de la robotique, la capacité des agents de se déplacer efficacement et se rencontrer dans un environnement inconnu est essentielle. Résoudre le problème de l'approche des agents permet de développer des algorithmes de navigation et de coordination pour les robots autonomes.
2. **Systèmes multi-agents** : Dans les systèmes multi-agents, où plusieurs agents autonomes interagissent pour atteindre un objectif commun, la rencontre des agents est cruciale pour la planification et la coordination des actions des agents.
3. **Réseaux de capteurs** : Les réseaux de capteurs comprennent souvent des agents mobiles chargés de collecter des données dans les environnements dynamiques. La capacité des agents à se déplacer de manière efficace pour couvrir une zone donnée peut être améliorée en résolvant le problème de l'approche.
4. **Logistique et transport** : Dans les opérations de logistique et de transport, la résolution du problème de l'approche des agents est pertinente pour l'optimisation des itinéraires et la coordination des véhicules. Cela peut être particulièrement important dans les contextes tels que la gestion de flottes de véhicules de livraison ou la surveillance de zone à l'aide de drones.

Ces exemples mettent en évidence la pertinence et l'importance de ces problèmes dans divers domaines d'application. En exploitant les concepts et les techniques abordés dans ce mémoire, il est possible de développer des solutions innovantes pour résoudre des problèmes du monde réel.

Dans ce qui suit, nous aborderons d'abord ce qui a déjà été fait dans la littérature concernant les problèmes du rendez-vous et de l'approche, la chasse au trésor étant un cas particulier de ces problèmes où l'un des agents est immobile. Ensuite, nous présenterons les modèles pour les problèmes de chasse au trésor et de l'approche, suivis de quelques notions préliminaries importantes. Enfin, nous introduirons successivement l'algorithme pour résoudre le problème de chasse au trésor d'une part, et l'algorithme pour résoudre le problème de l'approche d'autre part. Ce mémoire sera conclu par une présentation d'un logiciel de simulation de notre algorithme d'approche.

Chapitre 2

Revue de la littérature

2.1 Introduction

Le rendez-vous est une tâche qui concerne deux agents mobiles (qui sont des entités équipés de capacité de calcul), initialement dispersés dans un environnement, qui peut être le plan ou un réseau modélisé par un graphe arbitraire. Ces agents mobiles doivent se rencontrer en un temps fini et à un endroit inconnu à priori. Quand il y a plus de deux agents qui doivent se rencontrer, on parle de *rassemblement*. Les problèmes du rendez-vous et du rassemblement ont été intensivement étudiés dans la littérature du calcul distribué. Ils ont été approchés dans différents contextes, chacun représentant un cas particulier d'une situation réelle. Dans tous les cas les agents mobiles sont modélisés par des points qui naviguent dans l'environnement et le rendez-vous ou le rassemblement demandent que les agents se rendent au même point de l'environnement au même moment. Dans le cas du plan, on diminue souvent l'exigence de la rencontre : au lieu de se rendre au même point du plan les agents doivent juste se rendre à distance au plus 1 l'un de l'autre. Dans ce cas, on parle de l'approche plutôt que du rendez-vous. C'est la où la littérature concernant le rendez-vous rejoint le sujet de ce mémoire.

Les deux principaux cadres d'étude dans la littérature du rendez-vous sont le cadre *synchrone* et le cadre *asynchrone*. Nous présenterons tour à tour les recherche qui ont été menées dans le cadre synchrone d'abord et ensuite dans le cadre asynchrone. Nous décrirons principalement les modèles utilisés, l'environnement d'étude (le plan, la grille, etc...) et les résultats.

Plusieurs auteurs se sont concentrés sur d'autres aspects du problème de rendez-vous comme la quantité minimale d'espace mémoire requise pour accomplir le rendez-vous ou

le fait que certains agents soient *byzantin* (ils agissent de façon tout à fait imprévisible). Nous parlerons de ces cas dans la section *autres études*.

2.2 Le rendez-vous synchrone

L'un des problèmes fondamentaux dans l'étude de la faisabilité et de l'éfficacité du rendez-vous déterministe dans le cadre synchrone est la rupture de la *symétrie*. En effet, si les agents sont identiques, s'ils exécutent le même algorithme déterministe et s'ils sont synchrones, alors dans beaucoup de réseaux dont tous les noeuds sont identiques, tel que par exemple l'anneau, leurs trajectoires seront identiques et donc la distance initiale entre les agents ne changera jamais au fur et à mesure de leurs mouvements, rendant ainsi le rendez-vous impossible. Une des façons de briser cette symétrie pour rendre le rendez-vous possible est l'utilisation des jetons pour différencier les noeuds dans le cas des graphes. La méthode la plus utilisée pour rompre la symétrie est l'utilisation des étiquettes différentes pour chaque agent comme c'est le cas dans ce mémoire. L'étiquettte devient alors un paramètre de l'algorithme de rendez-vous et permet de différencier le comportement des agents.

Antonio Di Luna *et al.* [11] ont étudié la faisabilité du problème de rassemblement dans des environnements d'anneaux dynamiques. Ils considèrent un anneau, noté $R = (v_0, \ldots, v_{n-1})$, qui est synchrone et dynamique. Dans un anneau dynamique, à n'importe quel moment $t \in \mathbb{N}$, un de ses liens peut être absent. Le choix du lien absent est de la responsabilité d'un adversaire et ce choix n'est soumis à aucune restriction.

Chaque nœud v_i est connecté à ses deux voisins v_{i-1} et v_{i+1} via les ports q_{i-} et q_{i+} respectivement. Le numérotage des ports des nœuds est arbitraire. Les nœuds sont anonymes, c'est-à-dire qu'il n'existe aucun moyen de les différencier. Chaque nœud possède un buffer d'entrée et un buffer de sortie.

Les agents, notés par un ensemble $A = (a_0, \ldots, a_{k-1})$, sont des entités de calcul identiques, capables de se déplacer et qui exécutent le même algorithme. Lorsqu'un agent est dans un nœud v_i, il peut soit se situer dans le nœud, soit dans l'un des buffers du nœud. Un agent peut déterminer le nombre d'agents présents dans un nœud et l'emplacement de ces différents agents. Initialement, les agents sont placés à des endroits différents dans l'anneau, appelés *domiciles*. Chaque agent connaît son domicile. Les agents peuvent ne pas avoir le même sens d'orientation. On parle de *chiralité* lorsque les agents ont le même sens d'orientation. Les agents n'ont aucun moyen de communication

entre eux, on dit qu'ils sont silencieux. Plusieurs agents peuvent traverser un lien dans n'importe quelle direction.

Le système possède une *détection croisée* lorsque les agents, traversant un lien dans des directions opposées, peuvent se détecter les uns les autres.

Le système fonctionne en rondes synchrones. Chaque agent est dans un état S (initial $Init$ ou terminal $Term$) et détermine le nombre et la position des autres agents au nœud v au début de chaque ronde. En utilisant cette information, son état et sa mémoire locale, l'agent décide de se déplacer (gauche, droite ou rester). Si le lien est absent, l'agent reste dans le buffer de sortie du nœud courant. Les agents dans le même nœud, état et direction forment un groupe qui ne se divise jamais et peut seulement croître.

Soit (R, A) un système défini comme ci-dessus. Dans (R, A), le rassemblement est atteint au tour r si tous les agents de A se trouvent dans le même nœud ou dans deux nœuds voisins à la ronde r ; dans le premier cas, on parle de rassemblement strict. Un algorithme résout le problème de rassemblement si, à partir de toute configuration où le rassemblement est possible, tous les agents sont, en temps fini, dans l'état terminal, rassemblés et conscients que le rassemblement a été réalisé. Un algorithme de solution est efficace si, à partir de toute configuration où le rassemblement n'est pas possible, tous les agents détectent cette impossibilité en temps fini.

D'après leur étude, Antonio Di Luna *et al.* ont pu montrer, en fournissant un algorithme pour le rassemblement dans un anneau dynamique, que l'obstacle créé par le dynamisme de l'anneau peut être surmonté grâce à la *chiralité* ou à la *détection croisée*.

Collins *et al.* [6] ont étudié le problème de rendez-vous de deux agents dans les réseaux où chaque agent est conscient de sa position initiale. Le réseau est représenté par un graphe (fini ou infini) dans lequel des agents mobiles visitent les nœuds en traversant les arêtes dans les deux directions. Cette approche s'étend également au cadre géométrique continu. Il est supposé que les agents se déplacent à vitesse uniforme. Plus précisément, un agent mobile nécessite une unité de temps pour traverser une arête du graphe ou un segment de longueur 1 dans le cadre géométrique. Chaque agent peut accéder à son horloge et les horloges des agents sont synchronisées pour marquer les mêmes moments dans le temps. Un agent peut compter le nombre de tours (tics d'horloge) et utiliser cette information pour planifier ses actions. Nous supposons que les deux agents commencent leurs actions simultanément, c'est-à-dire que les deux horloges indiquent la même heure. Les agents sont anonymes, mais chaque agent est conscient de son emplacement initial, qu'il peut utiliser comme une étiquette unique. Nous supposons également que les agents

sont pleinement conscients de la topologie du réseau. Les agents peuvent se rencontrer aux sommets du graphe et peuvent se croiser sur les arêtes dans des directions opposées sans se rencontrer. Les auteurs se sont concentrés sur les procédures de rendez-vous déterministe (c'est-à-dire sans randomisation). Le mouvement d'un agent est déterminé par son emplacement actuel et l'heure sur son horloge.

À l'issue de leur étude, les auteurs ont pu proposer plusieurs algorithmes asymptotiquement optimaux pour le problème de rendez-vous. Ils ont montré que, dans le cas particulier des lignes, des arbres et des grilles multidimensionnelles, les agents peuvent se rencontrer en temps $O(d)$, où d est la distance entre les positions initiales des agents.

Amnon Ta-Shma et Uri Zwick [17] ont étudié le problème de rendez-vous de deux robots placés dans un environnement inconnu. L'environnement est formellement modélisé par un graphe fini non-orienté $G = (V, E)$. Les liens incidents à un nœud u sont numérotés de $1, 2, \ldots, \deg(u)$, où $\deg(u)$ est le degré de u. Un robot, étant dans un nœud, peut voir le degré de celui-ci. Les robots peuvent être actifs ou inactifs. Un robot actif peut naviguer dans le graphe. Pour ce faire, un robot peut effectuer des opérations sur les informations qu'il possède. Ces informations peuvent comprendre l'étiquette du robot, le nombre de rondes ou de temps écoulé depuis la première activation, et le degré du nœud courant. La sortie de ces opérations est une instruction dont la valeur est un nombre de l'ensemble $\{0, 1, \ldots, \deg(u)\}$. Le nombre choisi représente la direction (par quel lien l'agent va aller au prochain nœud) que le robot va emprunter. La valeur 0 représente le fait que l'agent reste sur place.

Les robots sont placés dans l'environnement par un adversaire. Il peut les placer où il veut dans le graphe et il peut les activer quand il veut. Pour briser une possible symétrie, les auteurs ont donné des identifiants distincts aux robots. Le but est de donner les suites d'instructions qui leur permettront de se rencontrer, et ce, quel que soit l'environnement. Les auteurs ont supposé que les robots se déplacent de façon synchrone. Les deux robots se rencontrent seulement quand ils sont tous les deux actifs et dans le même nœud. Il n'y a donc pas de détection croisée ici. Étant donné deux robots placés initialement par un adversaire et activés à des moments possiblement différents, le coût de la rencontre est le nombre d'unités de temps écoulé depuis l'activation du second robot jusqu'à la rencontre des deux robots. Une solution au problème de rendez-vous est un algorithme tel que, quel que soit l'environnement, quels que soient les étiquettes des robots, quels que soient les moments d'activation et quelles que soient leurs positions initiales, les deux robots se rencontrent. Les auteurs ont construit un algorithme de rendez-vous dont le

coût est polynomial dans le nombre des noeuds du graphe et dans la longueur de la plus courte étiquette.

Miller et Pelc [16] ont étudié quant à eux, le problème de rendez-vous dans un réseau modélisé par une ligne infinie étiquettée. Ils ont considéré une ligne infinie dont chaque nœud a deux ports numérotés 0 et 1 : chaque nœud est donc de degré 2. Un agent qui est dans un nœud donné peut voir les ports de ce nœud. Les nœuds ont des étiquettes différentes qui sont des entiers positifs. L'étude des problèmes de rendez-vous dans les graphes étiquettés finis se réduisait à la tâche d'exploration où les agents exploraient le graphe entier, trouvaient le nœud avec la plus petite étiquette et adoptaient celui-ci comme point de rendez-vous. Cette pratique, étant consommatrice de temps dans les graphes finis, est impraticable dans les graphes infinis. Il est donc naturel de lier la tâche de rendez-vous à la distance initiale qui sépare les agents plutôt qu'à la taille du graphe. Du point de vue des auteurs, la distance initiale entre les agents et les étiquettes des nœuds de départ devraient être les uniques paramètres d'efficacité des algorithmes dans ces situations.

Les agents sont techniquement anonymes, cependant ils peuvent utiliser les étiquettes des nœuds de départ comme les leurs. Lorsqu'ils traversent un lien dans des directions opposées, ils ne se détectent pas. Il n'y a pas de limites de mémoire pour les agents. Les agents fonctionnent en rondes synchrones. Ils sont initialement *endormis* et peuvent être réveillés par un adversaire à des moments différents. Le coût du rendez-vous est le temps (nombre de rondes) depuis le tout premier réveil jusqu'à la rencontre.

Les auteurs ont considéré trois scénarios qui diffèrent les uns des autres selon la quantité d'information que les agents possèdent au début. Dans le premier scénario, chaque agent, débutant dans un nœud, connaît sa position initiale par rapport au nœud qui a la plus petite étiquette, de quel côté se trouve ce nœud par rapport à son nœud de départ et donc la direction vers laquelle l'agent doit se diriger pour l'atteindre. Pour ce scénario, les auteurs ont réussi à concevoir un algorithme qui travaille en temps optimal de complexité $O(D)$, où D est la distance initiale entre les agents. Dans le second scénario, chaque agent connaît seulement l'étiquette de son nœud de départ et la distance initiale D qui les sépare. Pour ce scénario, les auteurs ont conçu un algorithme qui travaille en temps $O(D \log^* l)$, où l est la plus grande étiquette des deux noeuds de départ. Cet algorithme est aussi optimal car pour ce scénario, ils ont pu prouver que la borne inférieure sur le temps de rendez-vous est de $\Omega(D \log^* l)$. Le troisième scénario, qui est le scénario le plus général, est celui où chaque agent connaît seulement l'étiquette de

son nœud de départ. Pour ce scénario, les auteurs ont réussi à concevoir un algorithme qui travaille en temps $O(D^2(\log^* l)^3)$. Il posent la question quel est le temps optimal de rendez-vous dans ce scénario le plus général.

2.3 Le rendez-vous asynchrone

L'aspect commun de tous les modèles asynchrones concernant le rendez-vous et le rassemblement est que les agents peuvent choisir la destination de leur mouvement mais ils n'ont pas de contrôle sur la vitesse de leurs déplacements. On suppose seulement que chaque agent va se rendre à la destination de son mouvement en temps fini, mais la durée de se déplacement peut être arbitraire : on dit qu'elle est décidée par un adversaire, c'est-à dire on considère le pire cas qui puisse arriver.

Les chercheurs ont étudié ce problème dans plusieurs environnements, dont les principaux sont : les grilles multidimensionnelles [1], le plan [13, 3], et les graphes arbitraires [15]. Un modèle important de l'asynchronie utilisé pour le rendez-vous dans les réseaux [1, 15, 10] repose sur un adversaire capable de faire varier la vitesse d'un agent, de l'arréter et même de le reculer, tant et aussi longtemps que le mouvement de l'agent soit continu et ne quitte pas l'arête choisie par celui-ci. Dans ce cas la rencontre dans un noeud du réseau est souvent impossible et on relaxe l'exigence en permenttant la rencontre aussi à l'intérieur d'un arête.

Flocchini *et al.* [13] ont abordé le problème de *rassemblement* d'un groupe de robots dans un plan, chaque robot ayant une visibilité limitée. Ils ont considéré un groupe de robots mobiles autonomes et identiques. Les robots exécutent le même algorithme. Chaque robot possède ce qu'on appelle un *cycle de fonctionnement* : initialement en mode attente, il peut, à n'importe quel moment et de manière asynchrone et indépendante des autres robots, observer tout ce qui l'entoure jusqu'à une distance maximale V (c'est l'aire de visibilité). À partir de cette observation, il effectue des calculs pour déterminer sa prochaine destination et ensuite se rend à cette destination à une vitesse incontrolable.

Les robots sont *amnésiques*, c'est-à-dire qu'ils ne se souviennent pas de ce qui s'est passé lors des cycles précédents. Ils sont également *anonymes* ou *identiques*, car il n'existe aucun moyen de les différencier. Le fait que les robots soient *asynchrones* et *identiques* constitue une caractéristique très importante, car, en pratique, elle permet la fabrication de robots à moindre coût et simplifiée. Les robots n'ont aucun moyen de communication

entre eux. Notons que l'asynchronie découle ici du fait que les agents effectuent leurs cycles avec des vitesses pssiblement différentes.

Au vu de leurs travaux, Flocchini *et al.* ont démontré, en fournissant un algorithme pour cela, que le rassemblement peut être réalisé par des robots très simples, en temps fini et avec très peu de contraintes.

Bampas *et al.* [1], Guilbault *et al.* [15] et Dieudonné *et al.* [10] ont étudié le problème de rendezvous dans des modèles asynchrones qui ont des points en communs.

L'étude principale de Bampas *et al.* [1] porte sur le rendezvous de deux agents mobiles dans une grille multidimensionnelle orientée infinie. Les agents sont anonymes mais chacun d'eux connaît son emplacement dans la grille (on dit souvent que les agents possèdent un GPS). Les agents sont placés par un adversaire sur deux noeuds arbitraires de la grille multidimensionnelle infinie de dimension δ denotée G_δ. L'ensemble des noeuds de G_δ sont les points de coordonnées entières de l'ensemble de l'espace Euclidien de dimension δ, que l'on notera F. Deux noeuds $u = (u_1, u_2, \ldots, u_\delta)$ et $v = (v_1, v_2, \ldots, v_\delta)$ de G_δ sont connectés si et seulement si il existe un $i \in \{1, 2, \ldots, \delta\}$ tel que $\forall j \neq i, u_j = v_j$ et $u_i = v_i \pm 1$. Chaque noeud possède des ports. Ce sont des sortis situées aux différentes extrémités d'un noeud et qui permettent de connecter les noeuds entre eux. Pour un noeud, on attribue à chacun de ses ports un numéro unique appartenant à l'ensemble $\{1, 2, \ldots, 2\delta\}$. En utilisant la supposition très forte de la présence du GPS, Bampas *et al.* [1] ont proposé un algorithme de rendez-vous qui fonctionne en temps presque optimal, notamment $O(D^\delta polylog(D))$.

Guilbault *et al.* [15] ont étudié le problème du rendez-vous dans un graphe arbitraire. Le modèle qu'ils ont employé décrit deux agents débutant sur des nœuds différents d'un graphe connexe et anonyme : les noeuds du graphe n'ont pas d'étiquettes visibles pour les agents. Une des raisons de cette supposition d'anonymat est de permettre d'effectuer le rendez-vous dans des conditions réalistes où par exemple, un noeud pourrait ne pas vouloir divulguer son étiquette pour des raisons de sécurité. Chaque agent est capable de différencier localement les ports d'un nœud, mais la numérotation des ports peut varier d'un nœud à l'autre. De plus, les agents eux-mêmes sont complètement anonymes. Cet anonymat pose un problème dans cette étude : le graphe étant anonyme, les agents étant identiques, cette situation ne permet pas à un agent de savoir quand le rendez-vous est possible. Nonobstant cette difficulté, les auteurs ont réussi à caractériser les situations où le rendez-vous est possible et ils ont fourni un algorithme déterministe pour y parvenir.

Dieudonné *et al.* [10] ont étudié le problème de rendez-vous asynchrone dans un réseau arbitraire. Le réseau est modélisé comme un graphe connexe fini et non orienté, où les nœuds ne sont pas étiquetés, mais les arêtes incidentes à un nœud v ont des étiquettes distinctes dans $\{0, \ldots, d-1\}$, d étant le degré de v. Chaque arête non orientée $\{u, v\}$ a donc deux étiquettes, appelées numéros de port à u et à v, sans relation entre eux. Sans numéros de port, les arêtes incidentes à un nœud seraient impossible à distinguer pour les agents, rendant souvent le rendez-vous impossible.

Il y a deux agents qui commencent à partir de différents nœuds du graphe et parcourent ses arêtes sans pouvoir marquer les nœuds visités ou les arêtes traversées. Les agents sont modélisés comme des points se déplaçant dans ce graphe. Ils ont des étiquettes distinctes qui sont des entiers strictement positifs, et chaque agent connaît seulement sa propre étiquette, qui est une entrée initiale de l'algorithme déterministe. Les agents ne connaissent pas la topologie du graphe ni aucune limite de sa taille, mais peuvent acquérir des informations sur le réseau : lorsqu'un agent entre dans un nœud, il apprend son degré et le port d'entrée. La mémoire des agents est illimitée. Le rendez-vous a lieu lorsque les deux agents sont en même temps au même noeud ou au même point d'une arête. Le coût d'un algorithme de rendez-vous est le nombre d'arêtes traversées par les deux agents avant le rendez-vous. Dieudonné *et al.* [10] fournissent un algorithme déterministe de rendez-vous asynchrone qui fonctionne dans un réseau arbitraire et dont le coût est polynomial en ℓ et n, où ℓ c'est la longueur de la plus petite étiquette et n est le nombre de noeuds du graphe.

Les points communs importants des trois études [1, 15, 10] sont les notions de *parcours* et d'*itinéraire*. Ces notions servent à caractériser l'asynchronie dans ces études. L' itinéraire d'un agent est la succession de noeuds par lesquels il passe depuis le début jusqu'au rendez-vous. Un agent, étant dans un noeud, décide du prochain port qu'il va emprunter. Un agent décide donc de son *itinéraire*.

Nous décrivons maintenant le parcours f d'un agent sur son itinéraire selon [10]. Soit $R = (e_1, e_2, \ldots)$ l'itinéraire d'un agent, où chaque e_i est une arête $\{v_{i-1}, v_i\}$. Soit $(t_0, t_1, t_2, \ldots)$, avec $t_0 = 0$, une séquence croissante de réels, choisie par l'adversaire, représentant des moments précis dans le temps. Soit $f_i : [t_i, t_{i+1}] \to [v_i, v_{i+1}]$ une fonction continue, choisie par l'adversaire, telle que $f_i(t_i) = v_i$ et $f_i(t_{i+1}) = v_{i+1}$. Pour tout $t \in [t_i, t_{i+1}]$, nous définissons $f(t) = f_i(t)$. L'interprétation du parcours f est la suivante : à l'instant t, l'agent est au point $f(t)$ de son itinéraire. Cette définition générale du parcours, et le fait qu'il soit contrôlé par l'adversaire, sont des moyens de formaliser

les caractéristiques asynchrones du processus. Le mouvement de l'agent peut être à une vitesse arbitraire, l'adversaire pouvant parfois arrêter l'agent ou le faire reculer, tant que le parcours sur chaque arête de l'itinéraire est continu et complet. Cette définition confère à l'adversaire un grand pouvoir, ce qui limite le contrôle des agents sur leurs mouvements, rendant une rencontre entre agents difficile à réaliser. Des agents avec des itinéraires R_1 et R_2 et des parcours f_1 et f_2 se rencontrent à l'instant t, si les points $f_1(t)$ et $f_2(t)$ coïncident. Une rencontre est garantie pour les itinéraires R_1 et R_2, si les agents utilisant ces itinéraires se rencontrent à un certain instant t, indépendamment des parcours déterminés par l'adversaire.

Le problème étudié par Bouchard *et al.* [3] concerne l'approche asynchrone des agents dans le plan. C'est donc le même problème que celui de notre mémoire, mais considéré dans le cadre asynchrone. Le système distribué étudié dans [3] consiste en deux agents mobiles initialement placés par un adversaire à des positions distinctes dans un plan. Chaque agent a un rayon de vision limité, noté δ, qui leur permet de percevoir leur environnement jusqu'à une distance de δ. Les agents connaissent la valeur de δ et doivent s'approcher l'un de l'autre jusqu'à être à une distance δ pour se voir. On suppose que $\delta = 1$.

Les positions initiales des agents sont séparées par une distance d inconnue et supérieure à 1. Chaque agent reçoit une étiquette unique, un entier non négatif, qui est l'unique entrée de l'algorithme déterministe qu'il exécute. Les agents sont équipés d'une boussole et d'une unité de longueur, identiques pour les deux.

Le mouvement de chaque agent est décrit par son itinéraire, une séquence de segments parcourus en étapes. Chaque étape commence à une position donnée, l'agent choisit une direction et une distance, et se déplace jusqu'à ce qu'il voie l'autre agent ou atteigne la fin du segment. L'adversaire contrôle le déplacement le long de chaque segment, pouvant varier la vitesse, arrêter ou faire reculer l'agent, mais en maintenant la continuité du mouvement.

L'utilisation d'étiquettes distinctes est essentielle, car sans elles, il n'y aurait pas de solution déterministe à l'approche, si l'adversaire synchronise parfaitement les agents. En fait, dans ce cas l'itinéraire d'un agent serait une translation de l'itinéraire de l'autre et donc l'approche n'aurait jamais lieu.

Pour l'approche asynchrone, la mesure d'efficacité d'un algorithme est son coût, défini par la longueur totale des trajectoires des deux agents jusqu'à l'approche.

Dans [3], Bouchard *et al.* ont prouvé que la tâche d'approche peut être résolue de manière déterministe dans le modèle asynchrone décrit, avec un coût polynomial en fonction de la distance initiale inconnue séparant les agents et de la longueur de la représentation binaire de l'étiquette la plus courte.

2.4 Autres études

Dans cette section, nous présenterons les études dont les résultats principaux ne sont pas centrés sur l'algorithme de résolution du problème du rendez-vous ou du rassemblement lui-même. Les résultats des études qui vont suivre sont beaucoup plus centrés sur d'autres aspects comme l'impact des mémoires persistantes, la quantité minimale de mémoire dont doivent disposer les agents pour garantir un rendez-vous ou un rassemblement, ou encore le rôle des agents malicieux dans l'accomplissement du rassemblement.

Czyzowicz *et al.* [7] se sont penchés sur l'étude de la quantité minimum de mémoire requise pour le rendez-vous déterministe de deux agents identiques dans un graphe arbitraire. Ils ont considéré le réseau modélisé par un graphe non-orienté dont les nœuds n'ont pas d'étiquettes. Une arête $\{u, v\}$ reliant les nœuds u et v possède deux étiquettes qui sont des numéros de port à u et à v respectivement. Les numéros de port d'un nœud v de degré $d(v)$ sont des entiers distincts de l'ensemble $\{0, \ldots, d(v) - 1\}$. Cette numérotation est locale à chaque nœud. En d'autres termes, il n'y a pas de relation entre la numérotation des ports du nœud u et la numérotation des ports du nœud v. Une notion importante dans cette étude est la notion de "vue" à partir d'un nœud qui est définie comme suit. Soit u un nœud de degré k du graphe. La "vue" à partir du nœud u est l'arbre infini avec des ports étiquettés, dénoté $\mathcal{V}(u)$, enraciné en u tel que : la racine u est dénotée x_0 et pour tout nœud u_i, $i = 1, \ldots, k$, adjacent à u, il existe un voisin x_i dans $\mathcal{V}(u)$ tel que le numéro de port à v correspondant au lien $\{u, u_i\}$ est le même que le numéro de port à x_0 correspondant au lien $\{x_0, x_i\}$, et le numéro de port à u_i correspondant à l'arête $\{u, u_i\}$ est le même que le numéro de port à x_i correspondant à l'arête $\{x_0, x_i\}$. Le nœud x_i, pour $i = 1, \ldots, k$ est la racine de la "vue" à partir de v_i. Une paire de nœuds distinct (u, v) est dite symétrique si $\mathcal{V}(u) = \mathcal{V}(v)$.

Les agents sont modélisés comme des machines abstraites à état $\mathcal{A} = (S, \pi, \lambda, s_0)$, où S est un ensemble d'états parmi lesquels un état initial est dénoté s_0. π est la fonction de transition d'état définie comme suit : $\pi : S \times \mathbb{Z}^2 \to S$ et $\lambda : S \to \mathbb{Z}$ est la fonction de sortie. Un agent est initialement positionné dans un nœud v_0 et cet agent est dans

un état initial $s_0 \in S$. Chaque agent effectue des actions pendant les rondes. Les rondes sont mesurées par des horloges internes. Pendant une ronde un agent peut soit rester sur le nœud courant soit se déplacer vers un nœud adjacent. Plus précisément, un agent étant dans l'état s_0 et positionné dans un nœud précis, décide de se déplacer ou pas selon la valeur de l'entier $\lambda(s_0)$. Si $\lambda(s_0) = -1$, alors l'agent décide de rester sur le nœud courant. Si $\lambda(s_0) \geq 0$ alors l'agent décide de quitter le nœud courant, appelons ce nœud v_0, par le port $k = \lambda(s_0) \bmod d(v_0)$. Lorsqu'un agent, étant dans un état $s \in S$, entre dans un nœud u_0 par le port i, l'agent scanne ce port par lequel il est entré et scanne aussi le degré d du nœud u_0. Ensuite l'agent passe à l'état $s' = \pi(s, (i, d))$. Et si ce même agent, étant dans un nœud w_0 de degré k, avait décidé de rester sur place au cours de l'action précédente alors le nouvel état serait $s' = \pi(s, (-1, k))$. Dans tous les cas, s', le nouvel état, permettra de déterminer la prochaine action par l'entier $\lambda(s')$ qui sera soit -1, qui décrit l'action de rester sur place, soit un entier supérieur ou égal à 0 qui décrit l'action de se déplacer par le port $\lambda(s') \bmod k$. L'agent effectue ces opérations indéfiniment jusqu'au rendez-vous.

Les agents sont anonymes. En d'autres termes ils sont des copies A et A' de la même machine d'état abstraite $\mathcal{A}$ présentée plus haut. Les agents sont tous les deux synchrones. Plus précisément, les horloges internes des agents fonctionnent à la même fréquence et ces horloges se déclenchent quand les agents commencent à exécuter leurs actions. Les agents commencent tous les deux à partir de leurs positions initiales avec un *délai* $\tau \geq 0$, contrôlé par un adversaire. Cela signifie que l'agent qui commence tardivement le fait τ rondes après celui qui commence en premier. Les agents ne savent pas qui commence en premier ou non et ils ne connaissent pas la valeur de τ.

L'étude de Czyzowicz *et al.* [7] a pu révéler que, pour des positions initiales non-symétriques des agents, le nombre minimum de bits de mémoire requise pour un rendez-vous déterministe dans un graphe avec n noeuds est de $\Theta(\log n)$ et ceci peu importe le délai entre les mises en actions des deux agents. Les auteurs ont aussi prouvé que la borne inférieure pour cette quantité de mémoire nécessaire pour le rendez-vous est de $\Omega(\log n)$, donc leur résultat est optimal.

Czyzowicz *et al.* [8] ont étudié le problème de rendez-vous dans les arbres. Dans leur étude, les arbres sont des graphes connexes sans cycles avec les nœuds non-étiquettés et les nœuds ont des ports étiquettés. Les ports d'un nœud de degré d sont numérotés de 0 à $d-1$. Le nombre de nœuds d'un arbre est appelé *l'ordre de cet arbre*. Un *isomorphisme* entre deux arbres $T = (V, E)$ et $T' = (V', E')$ est une bijection $f : V \to V'$ telle que

pour tous nœuds $u, v \in V, u$ est adjacent à v si et seulement si $f(u)$ est adjacent à $f(v)$. En plus, on dit qu'un isomorphisme préserve la numérotation des ports si pour tous nœuds $u, v \in V$, le numéro de port correspondant à l'arête $\{u, v\}$ au nœud u est égal au numéro de port correspondant à l'arête $\{f(u), f(v)\}$ au nœud $f(u)$. Un *automorphisme* est un isomorphisme d'un arbre sur lui-même. Une paire de nœuds (u, v) d'un arbre est *symétrique* s'il existe un automorphisme f préservant la numérotation des ports et tel que $f(u) = v$.

Les agents sont modélisés comme des machines abstraites à état $\mathcal{A} = (S, \pi, \lambda, s_0)$ exactement comme dans [7] (voir la première étude de cette section). Les agents débutent aussi avec un délai $\tau \geq 0$ et ce délai est contrôlé par un adversaire comme dans [7].

Pour tout arbre T, on construit une séquence d'arbres $T_0 = T$, et T_{i+1} est obtenu en enlevant toutes les feuilles de T_i. On définit $\hat{T}$ comme T_j pour le plus petit j tel que T_j a au plus deux nœuds. Si $\hat{T}$ a un nœud, ce nœud est appelé le nœud central de T. Si $\hat{T}$ a deux nœuds, l'arête les reliant est appelée l'arête centrale de T. Un arbre avec des étiquettes de port est *symétrique* s'il existe un automorphisme non triviale qui préserve la numérotation des ports. Un arbre avec un nœud central ne peut pas être symétrique. Dans un arbre non symétrique, chaque paire de nœuds est non symétrique, rendant le rendez-vous possible pour toutes les positions initiales des agents.

Une *marche de base* dans un arbre de n nœuds, à partir du nœud v, est une traversée de toutes les arêtes de l'arbre se terminant au nœud de départ v telle que lorsqu'on entre dans un nœud par le port i, on en sort par le port $(i + 1) \bmod d$, où d est le degré du nœud. Cette marche est périodique avec une période de longueur $2(n - 1)$.

Un algorithme déterministe résout le problème du rendez-vous dans les arbres si, pour tout arbre de la classe des arbres décrite plus haut et pour toutes positions initiales non symétriques, les agents se retrouvent finalement au même nœud en même temps, indépendamment de leurs moments de départ respectifs (ou à condition qu'ils commencent en même temps).

Le résultat principal de Czyzowicz *et al.* [8] est le compromis entre le temps optimal pour effectuer le rendez-vous dans les arbres et la quantité de mémoire des agents. Il en ressort que pour les agents avec k bits de mémoire, le temps optimal de rendez-vous est $\Theta(n + n^2/k)$, avec n le nombre de noeuds de l'arbre. L'algorithme de rendez-vous qu'ils proposent fait appel à la marche de base.

Flocchini *et al.* [14] ont étudié l'impact de la mémoire persistante dans le problème de rendez-vous de deux robots dans le plan. Ils considèrent deux robots autonomes qui

sont ici des entités de calcul modélisées par des points de $\mathbb{R}^2$. Chaque robot a son propre système de coordonnées et chacun considère qu'il est situé à l'origine de son système de coordonnées local. Les robots ont aussi leur propre mesure de distance. Ils fonctionnent en cycles et chaque cycle fonctionne en trois phases : LOOK, COMPUTE et MOVE. Dans la phase LOOK, un robot observe tout ce qui est autour de lui et repère la position de l'autre robot. Dans la phase COMPUTE, il calcule son point de destination et dans la phase MOVE, il se déplace dans la direction de son point de destination calculé dans la phase COMPUTE. Les robots sont anonymes : ils sont identiques et ils exécutent le même algorithme dans la phase COMPUTE. Ils sont aussi amnésiques : les entrées de cet algorithme qu'ils exécutent dans la phase COMPUTE d'un cycle sont les informations récoltées uniquement dans la phase LOOK de ce même cycle.

Les auteurs se sont basés sur un modèle de *lumière* pour ressortir deux configurations du problème. Dans ce modèle de *lumière*, chaque robot possède une quantité constante de mémoire appelée lumière ; la valeur de la lumière est appelée couleur ou état et il est mis à jour par le robot durant chaque phase COMPUTE. Chacune des configurations du problème renseigne sur le degré de communication, s'il y a lieu, entre les robots. Dans la première configuration, les robots sont qualifiés de *silencieux à état fini*. Plus précisément, la lumière d'un robot n'est visible que par le robot lui-même. Dans ce cas, cette lumière représente l'état interne d'un robot. Dans la deuxième configuration, dite *à communication finie*, la lumière d'un robot dans un cycle donné est visible uniquement par les autres robots. Dans ce cas, les robots communiquent avec d'autres robots à travers les couleurs de leurs lumières dans le cycle courant mais ne peuvent pas se souvenir des couleurs des autres robots dans les autres cycles. Dans tous les cas de configuration, cette couleur qu'un robot voit est passée en entrée de l'algorithme.

Les auteurs ont abordé le problème dans les modèles asynchrone et semi-synchrone. Dans le modèle asynchrone, les robots sont activés indépendamment et la durée de chaque phase LOOK, COMPUTE et MOVE est finie mais imprévisible. Comme conséquence, les robots ont des notions de temps différentes et un robot peut effectuer ses calculs sur la base des informations déjà obsolètes. Dans le modèle semi-synchrone, les robots peuvent ne pas être tous actifs au même moment. En effet, cette activation suit un schéma divisé en rondes globales et pendant chaque ronde, un ou les deux robots peuvent être activés. Les auteurs supposent que chaque robot est activé un nombre infini de fois.

Les auteurs ont aussi pris en compte les cas où l'adversaire contrôle les mouvements des robots. Les systèmes dans lesquels l'adversaire contrôle les mouvements des robots,

souvent en les stoppant avant qu'ils atteignent leurs destinations, sont appelés *systèmes non-rigides*. Dans ces systèmes, il existe une constante $\delta > 0$ telle que si le point de destination est à une distance inférieure à δ, alors le robot va atteindre ce point et sinon, il va avancer en direction de ce point d'une distance d'au moins δ. Les systèmes dans lesquels l'adversaire n'a pas ce pouvoir sont appelés *systèmes rigides*.

On dit que le problème de rendez-vous est résolu si les deux robots se dirigent vers le même point et y restent en un temps fini.

Flocchini *et al.* ont pu prouver, en donnant un algorithme pour cela à la fin de leur étude, que les robots à état fini peuvent effectuer un rendez-vous dans le modèle semi-synchrone et les robots à communication finie peuvent le faire dans le modèle asynchrone.

Cielebak *et al.* [5] ont étudié le problème du rassemblement de n robots avec $n > 2$. Le système est composé d'un ensemble de robots mobiles anonymes, autonomes et silencieux. Ces robots possèdent chacun leur propre système de coordonnées locales, mais ces systèmes peuvent être incohérents les uns par rapport aux autres.

Chaque robot peut observer son environnement et les positions des autres robots grâce à ses capteurs. Cependant, ils ne peuvent pas communiquer directement entre eux. Les robots exécutent tous le même algorithme déterministe, basé sur les positions observées des autres robots, pour calculer un point de destination vers lequel se déplacer.

Le modèle est asynchrone, ce qui signifie que les robots n'ont pas de notion commune du temps, et la durée de chaque cycle de calcul et de mouvement est imprévisible. Un cycle de calcul se compose de trois phases pour un robot actif : LOOK-COMPUTE-MOVE. Chacune de ces phases se déroule exactement comme dans [14].

Leur étude démontre, avec un algorithme à l'appui, que le problème de rassemblement peut être résolu pour tout $n > 2$, quelle que soit la position initiale, et sous les contraintes décrites ci-dessus.

Di Luna *et al.* [12] ont étudié le problème de rencontre dans un polygone. Ils considèrent un ensemble de $k \geq 2$ robots mobiles et autonomes appelé ici *chercheurs*, modélisé par des points géométriques localisés dans un polygone P qui peut contenir des *trous*. L'objectif est que au moins deux de ces chercheurs puissent se voir. Les chercheurs fonctionnent en cycles LOOK-COMPUTE-MOVE et dans chacune de ces phases, les chercheurs se comportent exactement de la même façon que décrite dans [14].

Les chercheurs sont sévèrement limités, ce qui rend le problème de la rencontre plus difficile à résoudre.

- Ils ne connaissent pas la forme de P à l'avance, ni leur position à l'intérieur de P.

- Ils sont anonymes, ce qui implique qu'ils exécutent tous le même algorithme pour déterminer leurs points de destination.

- Ils sont amnésiques, ce qui signifie que chaque point de destination est calculé uniquement en fonction de la dernière prise de vue (LOOK), tandis que les anciennes prises de vue sont oubliés, et aucune mémoire n'est conservée entre les cycles.

- Ils sont déterministes, ce qui signifie qu'ils ne peuvent pas recourir au hasard dans leurs calculs.

- Ils sont asynchrones, en ce sens qu' aucune hypothèse n'est faite sur la rapidité avec laquelle chaque chercheur termine un cycle LOOK-COMPUTE-MOVE par rapport aux autres. Ces paramètres sont dynamiques et sont contrôlés par un adversaire.

- Ils sont désorientés, ce qui signifie qu'ils n'ont pas de boussoles magnétiques, d'appareils GPS ou d'accords de quelque nature que ce soit. Chaque chercheur a sa propre orientation locale indépendante, son unité de longueur et sa latéralité.

- Ils sont silencieux, en ce sens qu'ils ne peuvent pas communiquer entre eux de quelque manière que ce soit. En particulier, il n'y a pas de mémoire partagée, et les informations contenues dans une prise de vue ne peuvent être consultées que par le chercheur qui l'a exécutée.

- Ils ont des emplacements initiaux arbitraires à l'intérieur de P.

À l'issue de leur étude, Di Luna *et al.* ont pu minimiser le nombre de chercheurs k nécessaire pour résoudre le problème de rencontre dans le polygone P. Plus précisément, si P a une symétrie de rotation d'ordre σ (avec $\sigma = 1$ indiquant aucune symétrie), ils ont prouvé que $k = \sigma + 1$ chercheurs sont suffisants, et cette borne est optimale. De plus, un algorithme amélioré est proposé qui résout de manière optimale le problème de la rencontre avec $k = 2$ chercheurs dans les polygones dont le barycentre n'est pas dans un trou, ce qui inclut les polygones sans trous.

Le polygone quant à lui est anonyme (ce qui signifie la même chose que dans les cas des graphes anonymes puisqu'un polygone est un graphe).

Dieudonné *et al.* [9] ont étudié le problème de rassemblement d'agents en présence d'agents malicieux, appelés *byzantins*. L'environnement est modélisé comme un graphe non-orienté connexe. Les nœuds du graphe sont anonymes et les arêtes qui lient les nœuds ainsi que les ports sont décrits de la même façon que dans [7]. L'ensemble de tous les agents forme un groupe ou une équipe.

Les agents commencent à partir de différents nœuds du graphe (donc il y a au plus n agents dans un graphe de n nœuds) et traversent ses arêtes en rondes synchrones. Ils

ne peuvent pas marquer les nœuds visités ni les arêtes traversées. L'adversaire réveille certains des agents à des moments différents. Un agent dormant est réveillé par le premier agent qui visite son nœud de départ. Chaque agent commence à exécuter l'algorithme dès son réveil. Les agents ne connaissent pas la topologie du graphe. Les auteurs ont considéré deux scénarios : l'un où les agents connaissent le nombre de nœuds n du graphe, appelé sa taille, et l'autre où ils ne connaissent ni n ni aucune borne supérieure de celui-ci. Chaque agent a une étiquette qui est un entier positif différent et le connaît, mais il ne connaît pas à priori les étiquettes des autres agents ni la taille de l'équipe. À chaque tour, un agent peut effectuer des calculs locaux et se déplacer vers un nœud adjacent par un port choisi ou rester au nœud actuel. Lorsqu'un agent entre dans un nœud, il apprend son degré et le port d'entrée. Lorsque plusieurs agents sont au même nœud au même tour, ils voient les étiquettes des autres agents et peuvent échanger toutes les informations qu'ils ont. Cet échange se fait en un tour et toutes les informations échangées deviennent des connaissances communes pour les agents présents au nœud. Cependant, les agents qui se croisent sur une arête en se déplaçant simultanément dans des directions différentes ne remarquent pas ce fait. Ici, la mémoire des agents est illimitée.

Les auteurs supposent qu'un maximum de f agents peuvent être byzantins. Tous les autres agents sont appelés bons. Le paramètre f est connu de tous les agents. Ils ont défini deux niveaux de comportement byzantin. Un agent *fortement byzantin* peut choisir un port arbitraire lorsqu'il se déplace et peut transmettre des informations arbitraires aux autres agents, tandis qu'un agent *faiblement byzantin* peut faire de même, sauf changer son étiquette. Ici, la tâche est celle du rassemblement f-byzantin (avec terminaison), formulée comme suit. L'adversaire réveille au moins un bon agent, et tous les bons agents doivent finalement se retrouver dans le même nœud au même tour, déclarer simultanément la terminaison et s'arrêter, à condition qu'il y ait au plus f agents byzantins. Le temps d'un algorithme de rassemblement est le nombre de rondes entre le réveil du premier bon agent et l'arrêt de tous les bons agents. Selon les auteurs, il n'était pas possible d'exiger que les bons agents apprennent combien d'entre eux sont bons, car les agents byzantins peuvent choisir de ne révéler aucun comportement défectueux. Nous ne pouvons pas non plus exiger que les bons agents apprennent la taille de l'ensemble de l'équipe : les agents byzantins peuvent éviter d'être vus par les bons agents indéfiniment.

Pour les agents faiblement byzantins, les auteurs montrent que chaque nombre de bons agents permet de résoudre le problème pour des réseaux de taille connue. Si la taille est inconnue, alors ce nombre minimum est $f + 2$. Plus précisément, ils construisent un

algorithme déterministe polynomial qui rassemble tous les bons agents dans un réseau arbitraire, à condition qu'il y en ait au moins $f+2$. Ils fournissent également une borne inférieure correspondante : ils prouvent que si le nombre de bons agents est au plus $f+1$, alors ils ne peuvent pas se rassembler de manière déterministe avec terminaison dans certains réseaux.

Pour les agents fortement byzantins, ils donnent une borne inférieure de $f+1$, même lorsque le graphe est connu : ils montrent que f bons agents ne peuvent pas se rassembler de manière déterministe en présence de f agents byzantins, même dans un anneau de taille connue. Du côté positif, ils construisent des algorithmes de rassemblement déterministes pour au moins $2f+1$ bons agents lorsque la taille du réseau est connue et pour au moins $4f+2$ bons agents lorsque la taille est inconnue.

Bouchard *et al.* [2] ont étudié le problème de rassemblement en présence des agents byzantins et ont amélioré les résultats de [9]. Pour être plus précis, Bouchard *et al.* ont déterminé le nombre exact minimum de bons agents qui peuvent garantir le rassemblement en présence de f agents fortement byzantins.

Ils ont prouvé que dans le scénario où les agents connaissent la taille du graphe, le nombre minimum de bons agents pouvant garantir un f-rassemblement est $f+1$ et dans le scénario où les agents n'ont pas la connaissance de la taille du graphe, ce nombre est $f+2$. Pour chacun de ces scénarios, ils fournissent un algorithme déterministe pour effectuer le rassemblement et montrent qu'un nombre inférieur de bons agents ne permet pas de le faire.

Bouchard *et al.* [4] ont étudié le même problème dans le même modèle que [2] à part le fait qu'ils ont utilisé la notion de connaissance globale dénotée par $\mathcal{GK}$ et qui constitue l'ensemble des informations communes aux agents. Dans leur modèle, ils ont aussi ajouté la notion de *groupe fort* qui est un groupe dans lequel le nombre de bons agents est une valeur prescrite et qui est quadratique en f, où f est le nombre d'agents fortement byzantins. La nouvelle question posée dans [4] et celle du temps de fonctionnement d'un algorithme de rassemblement avec le nombre minimum de bons agents. Ils ont aussi cherché à utiliser la connaissance globale dont la représentation binaire est petite. Ils ont obtenu un résultat positif et un résultat négatif.

Leur résultat positif montre, en donnant un algorithme pour le faire, que le problème de rassemblement en présence de f agents fortement byzantins dans un groupe fort peut se résoudre dans n'importe quel graphe de taille au plus n, pour tout entier n et f, en temps polynomial en n et en $|l_{min}|$ où $|l_{min}|$ est la longueur de la représentation binaire

de la plus petite étiquette d'un bon agent. L'algorithme utilise une connaissance globale de taille $\mathcal{O}(\log\log\log n)$.

Leur résultat négatif montre qu'il n'existe pas d'algorithme déterministe qui résout ce problème dans n'importe quel graphe de taille au plus n, en temps polynomial en n et en $|l_{min}|$ et en utilisant une connaissance globale de taille $o(\log\log\log n)$.

Chapitre 3

Modèle, problèmes et notions préliminaires

3.1 Le modèle

Nous considérons des agents mobiles, localisés initialement à des positions différentes et inconnues dans le plan infini, qui peuvent se déplacer en rondes synchrones dont chacune dure une unité de temps. Les agents se réveillent à la même ronde et commencent l'exécution d'un algorithme déterministe. Pendant chaque ronde, chaque agent peut soit se déplacer à une distance unitaire, dans une des directions cardinales (Nord, Est, Sud et Ouest), soit rester immobile. Les agents sont modélisés par des points en mouvement. Chaque agent a un champ de vision d'un rayon unitaire : il peut voir tout ce qui se trouve à distance maximale d'une unité à partir de sa position courante.

3.2 Problèmes

Dans ce mémoire nous considérons deux problèmes :

- Le premier problème, connu sous le nom de *la chasse au trésor*, implique un agent mobile et un trésor, tous deux positionnés dans le plan. Le trésor est un point immobile et se trouve à une distance maximale D de l'agent. Une illustration de la situation est donnée à la figure 3.1. L'agent ne connaît ni la position du trésor ni la valeur de D. Son objéctif est de voir le trésor, c'est à dire de s'approcher de lui à distance au plus 1.

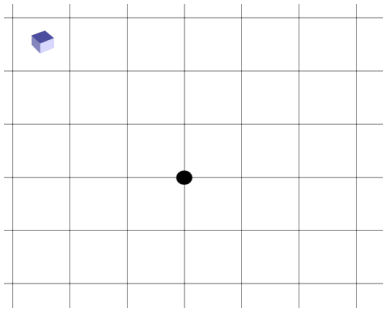

FIGURE 3.1 – Problème de chasse au trésor : le point noir représente l'agent

– Le deuxième problème, connu sous le nom de *l'approche des agents*, implique deux agents mobiles dans le plan qui se trouvent initialement à distance au plus D l'un de l'autre. Les agents ne connaissent ni la position de l'autre agent ni la valeur de D. Chaque agent possède une étiquette différente, representée par un entier positif appartenant à l'ensemble $\{1, \ldots, L\}$. Chaque agent connaît uniquement son étiquette. Les agents sont réveillés à la même ronde et l'objectif est pour les agents de s'approcher, c'est à dire de se trouver à distance au plus 1 l'un de l'autre.

Pour chacun des deux problèmes, on définit *le temps de sa solution* comme la durée entre le réveil des agents et la découverte du trésor ou l'approche, selon le cas. Le but de ce mémoire est de fournir les algorithmes déterministes pour résoudre ces problèmes de façon efficace. Le critère d'efficacité est le temps de solution. Dans le cas de la chasse au trésor, le temps est mesuré en fonction de la distance initiale maximale D entre l'agent et le trésor et dans le cas de l'approche le temps est mesuré en fonction de la distance initiale maximale D entre les agents et en fonction de la grandeur L de l'espace des étiquettes.

Remarques :

– On peut constater que les deux problèmes sont liés. En effet, le problème de la chasse au trésor peut être considéré comme le problème de l'approche dans le cas spécial où un des agents est immobile. Nous allons considérer la chasse au trésor comme un problème auxiliaire pour le problème de l'approche.

– Dans le cas du problème de l'approche, si les deux agents connaissaient les deux étiquettes, il y a un simple algorithme pour le résoudre. L'agent avec l'étiquette plus petite reste immobile et l'autre agent le cherche, suivant un algorithme pour le problème de la chasse au trésor. Ainsi, si les deux étiquettes sont connues par

les deux agents, le problème de l'approche peut être réduit au problème de la chasse au trésor.

3.3 Notions préliminaires

Avant de commencer le développement, il est important de définir certaines notions et terminologies qui seront utilisées tout au long du processus. En établissant ces définitions initiales, nous clarifions les concepts clés et établissons un langage commun pour faciliter l'exposition et la compréhension du sujet traité.

Dans tous le mémoire nous utilisons la notation $\log n$ pour le logarithme à base 2 de n.

Par la suite, nous utiliserons la notation $va(dir)$ pour décrire l'action d'un agent se déplaçant à une distance 1 dans la direction spécifiée par *dir*, à partir de sa position actuelle. Dans cette notation, le terme *dir* peut représenter une des quatre directions cardinales : N pour le Nord, S pour le Sud, E pour l'Est et O pour l'Ouest. À partir de cette instruction élémentaire, on définit la procédure $va(dir, x)$, où x est un entier positif, qui instruit l'agent d'aller à distance x unités dans la direction dir. Voici le pseudo-code de cette procédure :

Procédure $va(dir, x)$
for $i := 1$ **to** x **do**
    $va(dir)$

On utilisera également l'instruction *reste* pour signifier qu'un agent reste immobile pendant une ronde (une unité de temps). À partir de cette instruction élémentaire, on définit la procédure $reste(x)$, qui instruit l'agent de rester immobile pendant x unités de temps.

Voici le pseudo-code de cette procédure :

Procédure $reste(x)$
for $i := 1$ **to** x **do**
    $reste$

Chapitre 4

Chasse au trésor dans le plan

Au départ l'agent est dans sa position initiale. Dans l'approche que nous proposons, l'agent suit un schéma de mouvement spécifique pour couvrir progressivement le plan. Ce schéma peut être décrit comme une spirale, où l'agent alterne les déplacements vers l'Est, le Sud, l'Ouest et le Nord, tout en augmentant la distance parcourue. Le but est que le champ de vision de l'agent (qui est de rayon 1) puisse couvrir progréssivement le plan jusqu'à la découverte du trésor. Pour être plus explicite, à la première ronde, l'agent se déplace à distance 1 vers la direction l'Est ; à la deuxième ronde, il se dirige à distance 1 vers le Sud ; à la troisième et à la quatrième ronde, il parcourt une distance 2 vers l'Ouest ; à la cinquième et à la sixième ronde, il parcourt une distance 2 vers le Nord et ainsi de suite. L'agent poursuit ces mouvements jusqu'au moment de voir le trésor.

L'algorithme 1 donne le pseudo-code de la procédure Chasse au trésor. Elle est formulée comme une procédure infinie qui sera interrompue aussitôt que l'agent voit le trésor dans son champs de vision. La raison pour cette formulation est que la distance initiale du trésor est inconnue, on ne peut donc pas prédire combien de tours de la boucle l'agent va exécuter. On ne peut non plus utiliser une condition du type "**while** *trésor non vu*" parce que l'interruption peut avoir lieu au milieu d'une ronde et pas forcément à sa fin, selon la position du trésor par rapport à la position initiale de l'agent.

Théorème 1. *La procédure Chasse au trésor trouve un trésor situé à une distance au plus D de la position initiale de l'agent en $O(D^2)$ rondes.*

Démonstration. Soit S la partie de la spirale traversée pendant les D premiers tours de la boucle **repeat forever**. Les segments externes de S sont à l'extérieur du disc de rayon D centré dans la position initiale de l'agent (voir Fig. 4.1). Puisque les segments

Algorithme 1 : Procédure *Chasse au trésor*

```
1 x ← 1;
2 repeat forever
3 |   va(E, x);
4 |   va(S, x);
5 |   x ← x + 1;
6 |   va(O, x);
7 |   va(N, x);
8 |   x ← x + 1;
9 ;
```

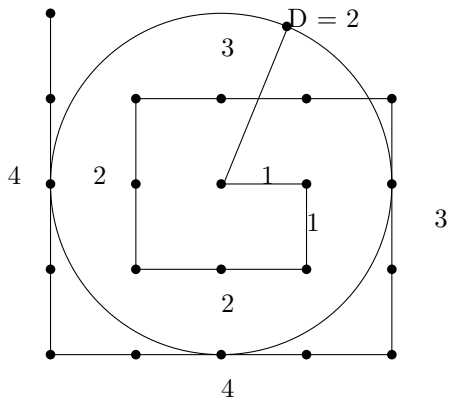

FIGURE 4.1 – mouvement de la spirale produit par l'agent lors de l'exécution de l'algorithme *Chasse au trésor*

parallèles consécutifs de S sont à distance 1, chaque point de ce disc est à distance au plus 1 d'un point de S. Il s'ensuit qu'avant la fin des D premiers tours de la boucle **repeat forever** le trésor sera trouvé.

Pour ce qui concerne le nombre de rondes des D premiers tours de la boucle **repeat forever**, il est calculé comme suit :

$$\sum_{i=1}^{2D} 2i = 2 \cdot \frac{(2D)(2D+1)}{2} = 4D^2 + 2D$$

qui est de l'ordre de $O(D^2)$. □

Dans ce qui suit nous utiliserons la notation $Borne(D) = 4D^2 + 2D$. Donc $Borne(D)$ est une borne supérieure sur le temps de trouver un trésor situé à distance au plus D de la position initiale de l'agent.

Chapitre 5

Approche avec la grandeur de l'espace des étiquettes connue

Dans le Chapitre 4, nous avons étudié le cas particulier du problème de l'approche, où un agent était mobile et l'autre immobile (le trésor). Dans ce chapitre, nous aborderons le cas plus général du problème, où les deux agents sont mobiles, chacun d'eux a une étiquette différente et connaît seulement sa propre étiquette, mais la grandeur de l'espace des étiquettes des agents est connue des deux agents.

5.1 Algorithme de rendezvous

Nous commençons par définir une procédure de modification de l'étiquette. Rappelons que l'espace des étiquettes des agents est $\{1, \ldots, L\}$ et les étiquettes des deux agents sont différentes. Chaque agent connaît son étiquette (et peut l'utiliser comme paramètre dans l'algorithme), mais il ne connaît pas l'étiquette de l'autre agent. Dans ce chapitre nous supposons que les deux agents connaissent la valeur de L.

Soit $k = \lfloor \log L \rfloor + 1$. k est un entier positif représentant le nombre de bits suffisant pour coder chaque entier de l'ensemble $\{1, \ldots, L\}$ en système binaire. Soit x l'étiquette d'un agent et soit $Bin(x)$ sa représentation binaire. Soit n le nombre de bits de $Bin(x)$ ($n = |Bin(x)|$). La suite $NewBin(x)$ est obtenu en ajoutant $k - n$ zéros avant $Bin(x)$. Donc le nombre de bits de $NewBin(x)$ est k. Par exemple, si $x = 5$ et $L = 17$, alors $Bin(x) = 101$, $n = 3$ et $m = 5$. Donc $NewBin(x) = 00101$.

Lemme 2. *Soient x et y deux entiers de l'ensemble $\{1, \ldots, L\}$. Si $x \neq y$ alors $NewBin(x) \neq NewBin(y)$.*

Démonstration. Soient x et y deux entiers pris dans $\{1, \ldots, L\}$. Supposons que $x \neq y$, alors $Bin(x) \neq Bin(y)$. Soient $n = \lfloor \log x \rfloor + 1$, $m = \lfloor \log y \rfloor + 1$ et $k = \lfloor \log L \rfloor + 1$ les nombres de bits suffisant pour coder les entiers x, y et L respectivement. Supposons $Bin(x) = (x_1, \ldots, x_n)$ et $Bin(y) = (y_1, \ldots, y_m)$ et $x_i, y_j \in \{0, 1\}$ pour tout $1 \leq i \leq n$ et $1 \leq j \leq m$. Puisque $NewBin(x)$ et $NewBin(y)$ sont obtenus par complétion par des zéros avant $Bin(x)$ et $Bin(y)$ respectivement, nous allons considérer deux cas :

1. cas où $n = m$: $Bin(x) \neq Bin(y)$ implique qu'il existe au moins un x_i de $Bin(x)$ et un y_i de $Bin(y)$, tels que $x_i \neq y_i$. Après avoir complété par $k - n$ zéros avant $Bin(x)$ et par $k - m$ zéros avant $Bin(y)$, le bit $x_{i+(k-n)}$ dans $NewBin(x)$ aura la même valeur que le bit x_i dans $Bin(x)$ et le bit $y_{i+(k-m)}$ aura la même valeur que le bit y_i dans $Bin(y)$. On aura alors $x_{i+(k-n)} \neq y_{i+(k-m)}$, donc $NewBin(x) \neq NewBin(y)$.
2. cas où $n \neq m$:
   (a) cas où $n = max(n, m)$: on sait $x_1 = 1$ dans $Bin(x)$ quelque soit x dans $\{1, \ldots, L\}$ et donc $x_{k-n+1} = 1$ dans $NewBin(x)$. Cependant, $y_{k-n+1} = 0$ dans $NewBin(y)$. On aura alors $x_{k-n+1} \neq y_{k-n+1}$ et donc $NewBin(x) \neq NewBin(y)$.
   (b) cas où $m = max(n, m)$: le raisonnement est symétrique.

□

Pour chaque entier $D \geq 1$, soit $S(D)$ la spirale parcourue par l'agent qui exécute les D premiers tours de la boucle **repeat forever** de l'algorithme 1. L'algorithme se déroule en phases $1, 2, 3, \ldots$. Au début de chaque phase, l'agent est à sont point de départ. L'idée générale de l'algorithme est la suivante. Soit $k = \lfloor \log L \rfloor + 1 = |NewBin(x)|$. Pendant chaque phase $i \geq 1$, l'agent traite les bits consécutifs de $NewBin(x)$. Pendant la $i$ème phase le traitement de chaque bit dure $2Borne(2^i)$ rondes et procède comme suit :

- si le bit est 1, l'agent parcourt la spirale $S(2^i)$ et ensuite parcourt cette spirale en sens inverse, retournant à son point de départ : la durée de ce trajet est de $2Borne(2^i)$ rondes.
- si le bit est 0, l'agent reste sur son point de départ pendant $2Borne(2^i)$ rondes.

.

De façon formelle, le mouvement de spirale réalisé pendant la $i$ème phase peut être décrit par une variante de l'algorithme 1 qui consiste à effectuer les 2^i premiers tours de la boucle **repeat forever** de cet algorithme :

Algorithme 2 : Procedure $Spirale(i)$

```
1 x ← 1;
2 for j := 1 to 2^i do
3     va(E, x);
4     va(S, x);
5     x ← x + 1;
6     va(O, x);
7     va(N, x);
8     x ← x + 1;
```

Pour le retour à la position initiale, le mouvement de spirale inverse, peut être décrit ainsi :

Algorithme 3 : Procedure $SpiraleInverse(i)$

```
1 x ← 2^(i+1);
2 for j := 1 to 2^i do
3     va(S, x);
4     va(E, x);
5     x ← x − 1;
6     va(N, x);
7     va(O, x);
8     x ← x − 1;
```

Rappelons que $Borne(D) = 4D^2 + 2D$ est une borne supérieure sur le temps de trouver un trésor situé à distance au plus D de la position initiale de l'agent. C'est le nombre de rondes que l'agent passe en exécutant les D premiers tours de la boucle **repeat forever** de l'algorithme 1. Donc $Borne(2^i)$ est la durée de chacune des procédures $Spirale(i)$ et $SpiraleInverse(i)$.

L'algorithme de rendezvous pour un agent avec l'étiquette $x \in \{1, \ldots, L\}$ peut être maintenant formulé de la façon suivante. Tout comme l'algorithme 1, il est formulé comme une procédure infinie qui sera interrompue aussitôt que l'approche aura lieu,

c'est-à-dire quand les agents seront à distance au plus 1.

Algorithme 4 : Procédure $Rendez - vous(x)$

```
1  k ← ⌊log L⌋ + 1;
2  NewBin(x) ← (a_1, a_2, ..., a_k);
3  i ← 1 //Pour compter les phases;
4  repeat forever
5      for j := 1 to k do
6          if a_j = 1 then
7              Spirale(i);
8              SpiraleInverse(i);
9          end
10         if a_j = 0 then
11             reste(2 · Borne(2^i))
12         end
13     end
14     i ← i + 1;
15 ;
```

Le théorème suivant qui est le résultat principal de ce chapitre établit l'exactitude et la complexité du rendezvous sous la supposition que la grandeur de l'espace des étiquettes est connue.

Théorème 3. *Supposons que deux agents sont initialement situés à distance au plus D l'un de l'autre, qu'ils ont des étiquettes différentes de l'ensemble $\{1, \ldots, L\}$ et qu'ils connaissent l'entier L. Alors la procédure Rendezvous accomplit l'approche de ces agents en $O(D^2 \log L)$ rondes.*

Démonstration. Considérons les agents avec les étiquettes x et y, avec $x \neq y$. Pendant chaque tour de la boucle **repeat forever** dans la Procédure $Rendez - vous(z)$, pour $z \in \{x, y\}$, l'agent avec l'étiquette z traite tous les bits de $NewBin(z)$. Rappelons que le nombre de bits de $NewBin(z)$ est $k = \lfloor \log L \rfloor + 1$. En vue du Lemme 2, on sait que $NewBin(x) \neq NewBin(y)$. Soit $j \leq k$ tel que le j-ème bit de $NewBin(x)$ diffère du j-ème bit de $NewBin(y)$. Sans perte de généralité supposons que le j-ème bit de $NewBin(x)$ est 1 et le j-ème bit de $NewBin(y)$ est 0.

Soit $t = \lceil \log D \rceil$. Considérons la période quand les deux agents traitent le j-ème bit de $NewBin(x)$ et $NewBin(y)$ respectivement pendant le t-ème tour de la boucle **repeat forever**. Alors l'agent avec l'étiquette x effectue la $Spirale(t)$, alors que l'agent avec l'étiquette y reste immobile sur sa position initiale. La situation de l'agent avec l'étiquette y est la même que s'il était un trésor à distance au plus D du point initial de l'agent avec l'étiquette x. Puisque $2^t \geq D$, il s'ensuit que l'agent avec l'étiquette x trouverait le trésor au plus tard pendant le t-ème tour de la boucle **repeat forever**. Donc l'approche aura lieu avant la fin du t-ème tour de la boucle **repeat forever**. Il s'ensuit que la procédure *Rendezvous* accomplit l'approche de ces agents.

Il reste à estimer le nombre de rondes. Le i-ème tour de la boucle **repeat forever** dure $k \cdot 2 \cdot Borne(2^i)$ et $Borne(2^i) = 4 \cdot 2^{2i} + 2 \cdot 2^i \leq 5 \cdot 2^{2i}$. Puisque $2^t \leq 2D$, les t premiers tours de la boucle **repeat forever** dans la Procédure $Rendez-vous(\cdot)$ durent

$$\sum_{i=1}^{t} k \cdot 2Borne(2^i) \leq 10k \cdot \sum_{i=1}^{t} 2^{2i} = 10k \cdot \sum_{i=1}^{t} 4^i \leq 10k \cdot 4^{t+1} \leq 160kD^2$$

rondes. Puisque $k \in O(\log L)$, il s'ensuit que le temps de l'approche est de l'ordre $O(D^2 \log L)$. □

Chapitre 6

Approche avec la grandeur de l'espace des étiquettes inconnue

Dans le chapitre 5 nous avons étudié le cas particulier de l'approche avec l'espace des étiquettes connu. Dans cette section, nous allons étudier le cas le plus général où l'espace des étiquettes est inconnu.

6.1 Algorithme de l'approche

Avant de commencer, il est important de noter qu'on ne peut malheureusement plus utiliser la technique de complétion de l'étiquette par un préfixe de zéros. Rappelons qu'au chapitre 5, l'espace des étiquettes était connu et grâce à cela, on pouvait se permettre cette complétion car on connaissait la valeur de L (la plus grande étiquette dans l'espace connu). Dans cette section, cette information est inconnue, d'où le fait qu'on ne peut plus compléter l'étiquette par un préfixe de zéros pour obtenir des étiquettes de longueur égale. On aura donc deux représentations binaires de longueurs différentes ce qui engendrera une asynchronie entre les différentes phases : pendant certains moments l'un des agents (celui avec la plus petite étiquette) sera dans une phase j et l'autre agent (celui avec la plus grande étiquette) sera en phase $j-1$.

Nous commençons par définir une nouvelle procédure de modification de l'étiquette. Rappelons que l'espace des étiquettes des agents est $\{1, \ldots, L\}$ et les étiquettes des deux agents sont différentes. Chaque agent connaît son étiquette (et peut l'utiliser comme paramètre dans l'algorithme), mais il ne connaît pas l'étiquette de l'autre agent. Dans ce chapitre nous supposons que les deux agents ne connaissent pas la valeur de L.

Soit x l'étiquette d'un agent et soit $Bin(x)$ sa représentation binaire. Pour chacun des bits de $Bin(x)$ on effectue la transformation suivante : remplacer chaque bit 1 par la suite de bits 10 et remplacer chaque bit 0 par la suite de bits 01. Appelons la nouvelle suite binaire obtenue $BinOne(x)$ et voyons un exemple de transformation : si $x = 5$, alors $Bin(x) = 101$ et $BinOne(x) = 100110$. Cependant la modification de l'étiquette ne s'arrête pas là. Après avoir obtenu $BinOne(x)$, il faut encore concaténer à cette suite les suites de bits 0 et 11 respectivement au début et à la fin de $BinOne(x)$. Appellons cette nouvelle suite $TransBin(x)$. Avec notre exemple précédent, on obtient : si $x = 5$, alors $Bin(x) = 101$, $BinOne(x) = 100110$ et $TransBin(x) = 010011011$.

Remarque 1 || Pour un entier positif arbitraire x, la suite $TransBin(x)$ ne contient jamais trois zéros consécutifs.

Pour chaque entier $D \geq 1$, soit $S(D)$ la spirale parcourue par l'agent qui exécute les D premiers tours de la boucle **repeat forever** de l'algorithme 1. L'algorithme d'approche que nous présentons dans cette section se déroule en phases $1, 2, 3, \ldots$. Au début de chaque phase, l'agent est à sont point de départ. L'idée générale de l'algorithme est de créer, grâce à notre nouvelle transformation de l'étiquette, des périodes d'immobilité suffisamment longues pour l'un des agents, ainsi que des périodes de mobilité suffisamment longues pour l'autre agent, et ce, plus ou moins au même moment. Pendant chaque phase $i \geq 1$, l'agent traite les bits consécutifs de $TransBin(x)$. Pendant la i-ème phase le traitement de chaque bit dure $2Borne(2^i)$ unités temps (voir chapitre 4) et procède comme suit :

- si le bit est 1, l'agent parcourt la spirale $S(2^i)$ et ensuite parcourt cette spirale en sens inverse, retournant à son point de départ : la durée de ce trajet est de $2Borne(2^i)$ unités de temps.
- si le bit est 0, l'agent reste sur son point de départ pendant $2Borne(2^i)$ unités de temps.

.

Les mouvements de spirale et de spirale en sens inverse sont décris par les algorithmes 2 et 3 du chapitre 5.

L'algorithme d'approche pour un agent peut être maintenant formulé de la façon suivante. Il est très similaire à l'algorithme $rendez - vous$ du chapitre 5 à l'exception des deux premières lignes.

Algorithme 5 : Procédure *Approche*(x)

```
1  k ← 2(⌊log x⌋ + 1) + 3;
2  TransBin(x) ← (a_1, a_2, ..., a_k);
3  i ← 1 //Pour compter les phases;
4  repeat forever
5      for j := 1 to k do
6          if a_j = 1 then
7              Spirale(i);
8              SpiraleInverse(i);
9          end
10         if a_j = 0 then
11             reste(2 · Borne(2^i))
12         end
13     end
14     i ← i + 1;
15 ;
```

Avant d'énoncer le théorème qui établit l'exactitude et la complexité du rendezvous sous la supposition de la grandeur de l'espace inconnue, énonçons deux lemmes qui nous aideront dans la preuve de ce théorème.

Lemme 4. *Considérons deux phases i et j, avec $j = i + 1$ et $i \geq 1$. Soient t_1^i et t_1^j les instants où un agent débute le traitement d'un bit de son étiquette transformée dans la Procédure Approche en phases i et j respectivement. Soient t_2^i et t_2^j les instants où un agent finit ce traitement. Alors $t_2^j - t_1^j \geq 3\dot{(}t_2^i - t_1^i)$.*

En d'autres termes, le temps utilisé par un agent pour le traitement d'un bit pendant une phase est au moins trois plus grand que le temps de traitement d'un bit pendant la phase précédente.

Démonstration. Rappelons que la durée de traitement d'un bit pendant une phase p c'est $2Borne(2^p)$. On a donc $t_2^j - t_1^j = 2Borne(2^j)$ et $t_2^i - t_1^i = 2Borne(2^i)$. On sait aussi que $Borne(2^j) = Borne(2^{i+1}) = 4 \cdot 2^{2i+2} + 2 \cdot 2^{i+1} = 16 \cdot 4^i + 4 \cdot 2^i > 15 \cdot 4^i + 5 \cdot 2^i > 14 \cdot 4^i + 6 \cdot 2^i > 12 \cdot 4^i + 6 \cdot 2^i$. Donc $Borne(2^{i+1}) \geq 12 \cdot 4^i + 6 \cdot 2^i$. Cependant

$12 \cdot 4^i + 6 \cdot 2^i = 3 \cdot (4 \cdot 4^i + 2 \cdot 2^i) = 3Borne(2^i)$. Donc $Borne(2^j) \geq 3Borne(2^i)$. D'où $t_2^j - t_1^j \geq 3(t_2^i - t_1^i)$. □

Lemme 5. *Soient x et y deux étiquettes prises dans l'ensemble $\{1, \ldots, L\}$. Si les suites $TransBin(x)$ et $TransBin(y)$ ont des longueurs différentes alors cette différence est d'au moins 2 bits.*

Démonstration. Soient x et y deux étiquettes prises dans l'ensemble $\{1, \ldots, L\}$. Si les suites $TransBin(x)$ et $TransBin(y)$ ont des longueurs différentes alors les suites $Bin(x)$ et $Bin(y)$ ont aussi des longueurs différentes. Supposons que la suite $Bin(x)$ est plus longue que la suite $Bin(y)$. Si la longueur de $Bin(x)$ est m alors la longueur de $Bin(y)$ est au plus $m - 1$.

Le nombre de bits de $TransBin(x)$ est $2m + 3$ et le nombre de bits de $TransBin(y)$ est au plus $2(m-1)+3 = 2m+1$. Donc la différence entre les longueurs de $TransBin(x)$ et $TransBin(y)$ est d'au moins 2. □

Le théorème suivant prouve l'exactitude de la Procédure *Approche* est estime sa complexité.

Théorème 6. *Supposons que deux agents sont initialement situés à distance au plus D l'un de l'autre et qu'ils ont des étiquettes différentes x et y de l'ensemble $\{1, \ldots, L\}$. Alors la procédure Approche accomplit l'approche de ces agents en $O(D^2 \log L)$ rondes, même si les agents ne connaissent pas l'entier L.*

Démonstration. Définissons tout d'abord deux notions qui nous aiderons dans la preuve.

La première notion, c'est la notion de *phase critique* qui est la phase $c = \lceil \log D \rceil$. La deuxième c'est la notion de la *ronde critique* qui est la première ronde où les deux agents sont déjà dans la phase au moins c. Considérons deux cas.

Cas 1. Les longueurs des suites $Bin(x)$ et $Bin(y)$ sont égales.

Comme $x \neq y$ alors $Bin(x) \neq Bin(y)$ et puisque ces suites ont la même longueur, il existe au moins une position i dans $Bin(x)$ et dans $Bin(y)$ tel que les bits correspondant sont différents, soit $Bin_i(x) \neq Bin_i(y)$. Puisque la différence de bits se trouve à la position i dans $Bin(x)$ et dans $Bin(y)$ alors il y a une différence de bits entre $TransBin(x)$ et $TransBin(y)$ à la position $2i + 2$ car chacun des bits se trouvant avant la position i est remplacé par deux bits et on ajoute un bit 0 au début des suites $TransBin(x)$ et $TransBin(y)$. Considérons la phase critique des deux agents et considérons le bit avec

l'indice b de $TransBin(x)$ et $TransBin(y)$ où ces suites diffèrent. Sans perte de gńéralité, supposons que $TransBin(x)_b = 1$ et $TransBin(y)_b = 0$. Le traitement du b-ème bit de $TransBin(x)$ et $TransBin(y)$ par les agents dans la phase critique correspond à la situation dans le chapitre 4 avec l'agent avec l'étiquette x jouant le rôle de l'agent mobile et l'agent avec l'étiquette y jouant le rôle du trésor. Puisque $2^c \geq D$ il s'ensuit que l'agent avec l'étiquette x s'approche de l'agent avec l'étiquette y pendant ce traitement. Donc l'approche aura lieu au plus tard pendant le c-ième tour de la boucle **repeat forever** de la procédure *Approche*. Il s'ensuit que la procédure *Approche* accomplit l'approche des agents.

Il reste à estimer le temps de l'approche. On sait, d'après le chapitre 4, que les c premiers tours de la boucle **repeat forever** prennent le temps $O(2^{2c} \log L)$. Puisque $c = \lceil \log D \rceil$, ce temps est dans l'ordre $O(D^2 \log L)$.

Cas 2. Les longueurs des suites $Bin(x)$ et $Bin(y)$ sont différentes.

Appelons **agent rapide** celui dont la représentation binaire est plus courte (supposons ici que c'est l'agent avec l'étiquette x) et appelons **agent lent** celui dont la représentation binaire est plus longue (supposons ici que c'est l'agent avec l'étiquette y). Notons que l'agent rapide accomplit ses phases plus rapidement que l'agent lent, car ses phases sont plus courtes.

Soit α la ronde critique et soit β la première phase de l'agent rapide qui commence après la ronde α. Le premier bit de $TransBin(x)$ est 0 et il est traité au début de la phase β. Appelons la période de ce traitement P. Au début de P, l'agent lent exécute encore une phase antérieure à la phase β. Soit z la dernière ronde de P. Notons que $z \in O(2^{2\beta} \log L)$ qui est dans l'ordre de $O(D^2 \log L)$. Il y a deux sous-cas.

Sous-cas 2.1. L'agent lent commence sa première phase débutant après la ronde α avant la ronde z.

Les deux derniers bits traités par l'agent lent dans la phase précédente sont 11. Ces deux derniers bits sont traités après le début de la phase β de l'agent rapide car la différence entre les longueurs de $TransBin(x)$ et $TransBin(y)$ est au moins 2. Il s'ensuit que pendant tout le traitement d'un bit 1 par l'agent lent l'agent rapide traite un bit 0, c'est à dire reste immobile. Ceci correspond à la situation dans le chapitre 4 avec l'agent lent jouant le rôle de l'agent mobile et l'agent rapide jouant le rôle du trésor. Donc l'approche aura lieu, car ce traitement a lieu après la ronde critique α. L'approche aura lieu avant la ronde z, donc le temps de l'approche est $O(D^2 \log L)$.

Sous-cas 2.2. L'agent lent commence sa première phase débutant après la ronde α pendant ou après la ronde z.

Considérons les bits traités par l'agent lent pendant la période P. Pendant toute la période P l'agent lent est encore à une phase antérieure à β, donc son traitement de chaque bit dure au moins 3 fois moins que la longueur de P, en vue du Lemme 4. Nous allons prouver que pendant la période P l'agent lent exécute soit la procédure *Spirale* soit la procédure *SpiraleInverse* au complet, ce qui montrera que l'approche aura lieu avant la ronde z et le temps de l'approche sera $O(D^2 \log L)$, comme dans le sous-cas 2.1.

Si trois bits consécutifs sont complètement traités par l'agent lent au cours de la période P, au moins un de ces bits doit être 1, en vue de la remarque 1, ce qui termine l'argument. Sinon, pendant la période P l'agent lent d'abord finit le traitement d'un bit b', puis traite deux bits consécutifs d_1 et d_2 et ensuite commence le traitement d'un bit b''. Si un des bits d_1 ou d_2 est 1, l'argument est terminé. Sinon, nous avons $d_1 = d_2 = 0$. Il s'ensuit que $b' = b'' = 1$, en vue de la remarque 1. Soit a la longueur de la sous-période de P pendant laquelle l'agent lent traite le bit b' et soit b la longueur de la sous-période de P pendant laquelle l'agent lent traite le bit b''. Cette situation est illustrée dans la figure 6.1.

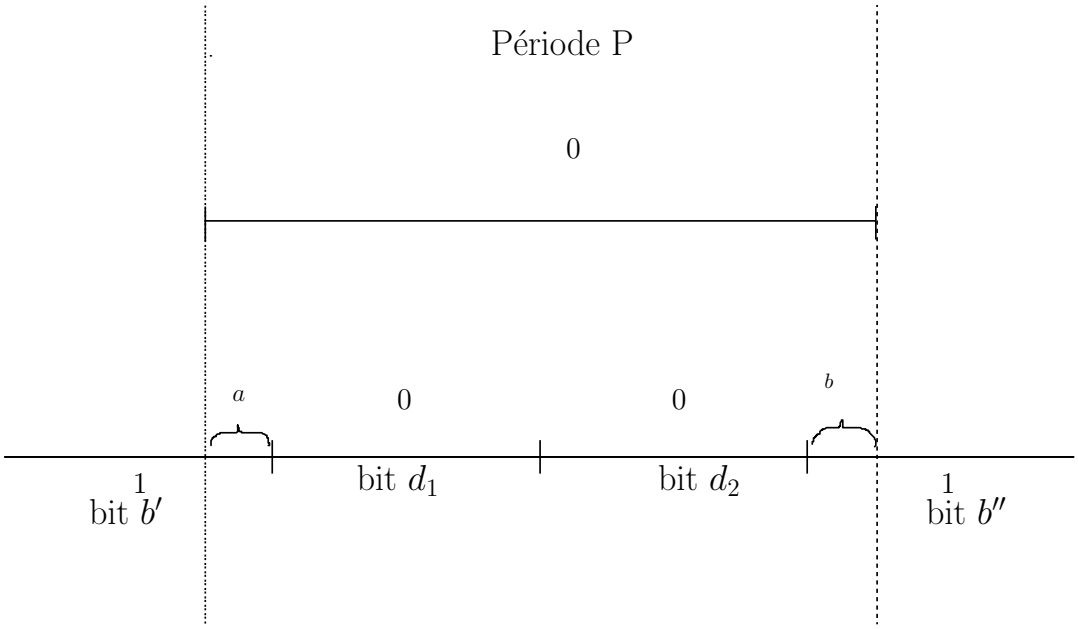

FIGURE 6.1 – Présentation d'un déphasage entre agents

En vue du Lemme 4, au moins une des longueurs a ou b est au moins égale à la moitié du temps de traitement des bits par l'agent lent pendant la période P. Il s'ensuit que soit l'agent lent exécute entièrement la procédure *Spirale* soit il exécute entièrement la procédure $SpiraleInverse$ pendant la période P. Dans les deux cas l'approche a lieu, en vue du chapitre 4. Comme auparavant, l'approche a lieu avant la ronde z, donc le temps de l'approche est $O(D^2 \log L)$. □

6.2 Borne inférieure sur le temps de l'approche

Cette section est consacrée à établir une borne inférieure sur le temps de l'approche en fonction de la distance initiale entre les agents.

Théorème 7. *Le temps d'approche pour deux agents qui débutent simultanément à une distance au plus D est $\Omega(D^2)$, pour n'importe quel algorithme déterministe.*

Démonstration. Considérons un algorithme arbitraire déterministe d'approche $\mathcal{A}$ pour deux agents A et B. Soit $a(t)$ (resp. $b(t)$) le vecteur qui décrit la position de l'agent A (resp. de l'agent B) en temps t, par rapport à sa base (position inititiale). Soit $d(t) = a(t) - b(t)$. Soit T le temps d'approche de l'algorithme $\mathcal{A}$ pour les agents A et B. Pour $i, j \in [-\lfloor \frac{D}{4} \rfloor, \lfloor \frac{D}{4} - 2 \rfloor]$, soit $c_{i,j}$ le carré avec les sommets aux points $(2i, 2j), (2i + 2, 2j), (2i, 2j + 2), (2i + 2, 2j + 2)$.

Un carré $c_{i,j}$ est dit *pair* si les entiers i et j sont pairs. Nous allons considérer seulement les carrés pairs. Nous disons que l'algorithme $\mathcal{A}$ *marque* le carré $c_{i,j}$, s'il existe un temps t de l'exécution de $\mathcal{A}$ tel que le point $d(t)$ est à l'intérieur de $c_{i,j}$. Puisque pour tous t_1, t_2 tels que $|t_1 - t_2| \leq 1$ la longueur du vecteur $d(t_1) - d(t_2)$ est au plus 2, l'algorithme $\mathcal{A}$ peut marquer au plus $T + 1$ carrés pairs.

Supposons que le temps d'approche T est au plus $\frac{D^2}{16}$. L'algorithme $\mathcal{A}$ peut marquer au plus $\frac{D^2}{16} + 1$ carrés pairs, donc il existe un carré pair $c_{i,j}$ qui n'est pas marqué. En fixant la base de l'agent A au point $(0, 0)$ et la base de l'agent B au point $(2i+1, 2j+1)$, on garantit que l'approche n'aura pas lieu. Cette contradiction termine la preuve. □

Puisque le théorème 6 établit le temps de fonctionnement de la procédure *Approche* à $O(D^2 \log L)$, ce temps est éloigné de la borne inférieure du théorème 7 seulement par un facteur $O(\log L)$. En particulier, si la grandeur L de l'espace des étiquettes est constante, la complexité de la procédure *Approche* est optimale.

Chapitre 7

Simulation de l'algorithme

Ce chapitre est consacré à la description du logiciel de simulation de notre algorithme le plus général de l'approche, celui qui travaille sans connaissance de la grandeur de l'espace des étiquettes et qui était formalisé par la Procédure Approche dans le Chapitre 6. Le code du logiciel est disponible à l'adresse `https://github.com/ZacharieMarie/Simulator/tree/master`.

7.1 Informations techniques sur le logiciel

Dans cette section, nous présentons les détails techniques du logiciel utilisé pour simuler notre algorithme, incluant le langage de programmation utilisé, les outils mobilisés, ainsi que les défis rencontrés lors du développement.

7.1.1 Langage et outils

Le logiciel est principalement développé en **JavaScript** pour la logique de simulation, en association avec **HTML** pour la structure de l'interface et **CSS** pour le style et la mise en forme. Le framework **Bootstrap** est utilisé pour la gestion des modales et la mise en page réactive, facilitant ainsi l'interaction avec l'utilisateur. Les animations et déplacements des agents sont orchestrés via le DOM et des fonctions asynchrones, permettant notamment des mouvements fluides (déplacements linéaires et spirales) grâce à l'utilisation de `requestAnimationFrame`.

7.1.2 Défis techniques

Le développement du logiciel a nécessité de relever plusieurs défis techniques, parmi lesquels :

- **Problème d'échelle avec les images :** L'intégration des images représentant les agents a présenté des difficultés liées à leur mise à l'échelle. L'utilisation d'un paramètre d'échelle (la constante `SCALE`) a permis d'ajuster la taille et le positionnement des éléments graphiques, mais a aussi engendré des problèmes de cohérence visuelle sur des écrans de résolutions différentes.
- **Problème de vitesse des agents :** La gestion de la fluidité des mouvements des agents a constitué un autre défi majeur. La fréquence de mise à jour (par exemple, la variable `framsperSecond` fixée à 10) et l'usage des animations basées sur `requestAnimationFrame` nécessitaient un équilibre précis pour éviter que les déplacements ne soient soit trop lents, compromettant la réactivité de la simulation, soit trop rapides, affectant la précision des interactions entre agents.

7.2 Fonctionnalités du logiciel

Le logiciel offre une interface conviviale et interactive pour simuler les déplacements et interactions entre agents. Voici comment s'y prendre :

1. **Configuration initiale :** Au lancement, une modale de configuration s'affiche permettant de définir les paramètres de chaque agent. L'utilisateur doit saisir une étiquette ainsi que les positions initiales en x et y. Ces valeurs sont ensuite transformées en notations binaires grâce à des fonctions dédiées, permettant d'intégrer des informations supplémentaires dans la simulation.
2. **Placement interactif des agents :** En plus de la configuration via la modale, l'utilisateur peut placer les agents directement sur la zone de simulation en cliquant sur l'écran. Cette méthode interactive offre une alternative intuitive pour positionner les agents.
3. **Démarrage de la simulation :** Une fois les agents configurés, il suffit de cliquer sur le bouton « Mettre en marche » pour lancer la simulation. Les agents commencent alors à se déplacer suivant des trajectoires prédéfinies (déplacements linéaires et mouvements en spirale) tout en vérifiant en continu la distance entre eux.

4. **Gestion des collisions et alertes :** Le logiciel surveille en permanence la distance entre les agents. Si celle-ci devient inférieure à un seuil critique, une alerte est déclenchée et une modale d'avertissement s'affiche, interrompant la simulation et confirmant que l'approche est accomplie. Ce mécanisme permet de prévenir les collisions et de garantir la cohérence du processus simulé.

7.3 Capture de l'écran pendant la simulation

Dans cette section nous présentons trois captures de l'écran pendant la simulation de notre logiciel pour la position $x = 0$ et $y = 0$ et l'étiquette de valeur 6 pour le premier agent(agent de couleur verte) et la postion $x = 10$ et $y = 10$ et l'étiquette de valeur 10 pour le deuxième agent. La première capture 7.1 montre le début de la simulation, la deuxième 7.2 montre la situation vers le milieu de la simulation et la troisième 7.3 montre le moment de l'approche.

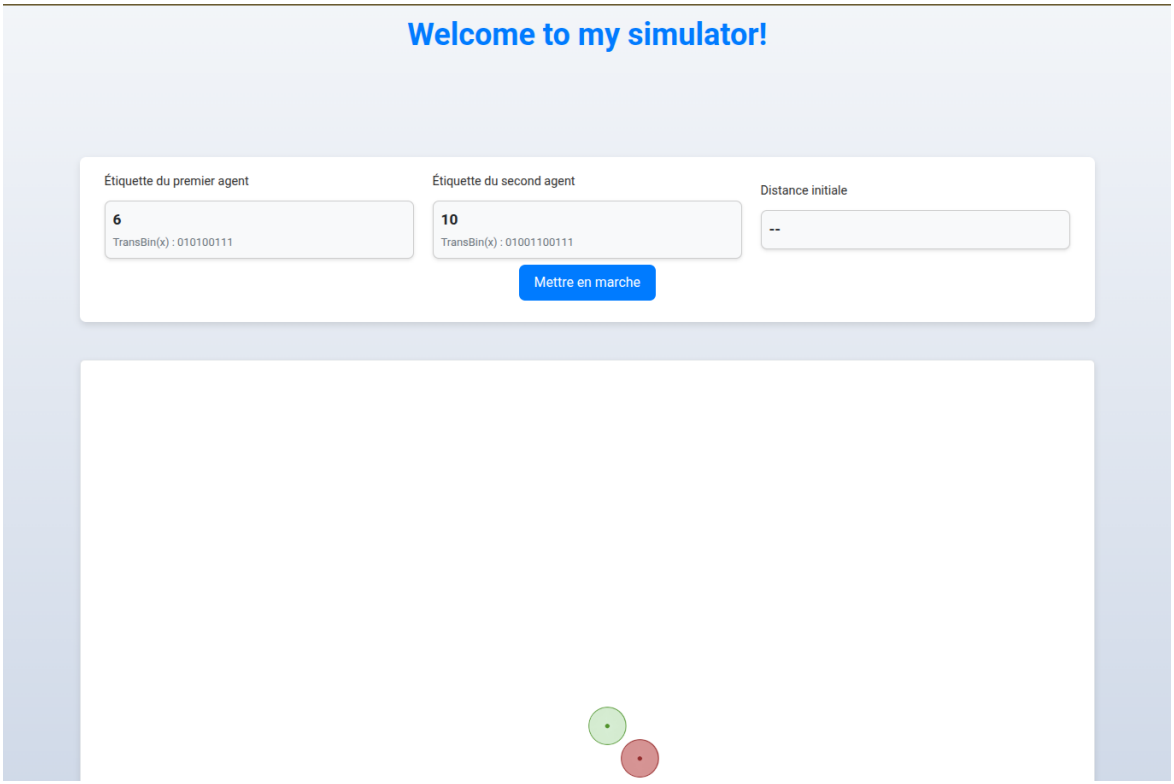

FIGURE 7.1 – Première capture : les agents sont en positions initiale

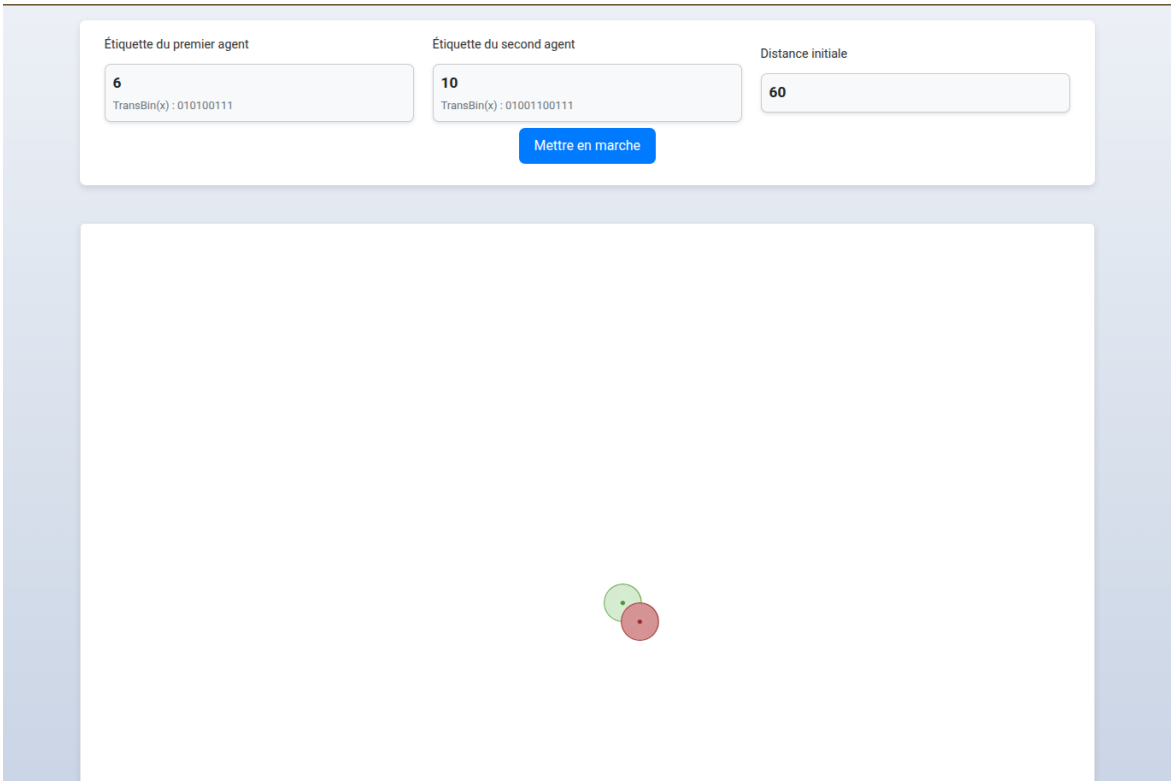

FIGURE 7.2 – Deuxième capture : les agents sont en mouvement

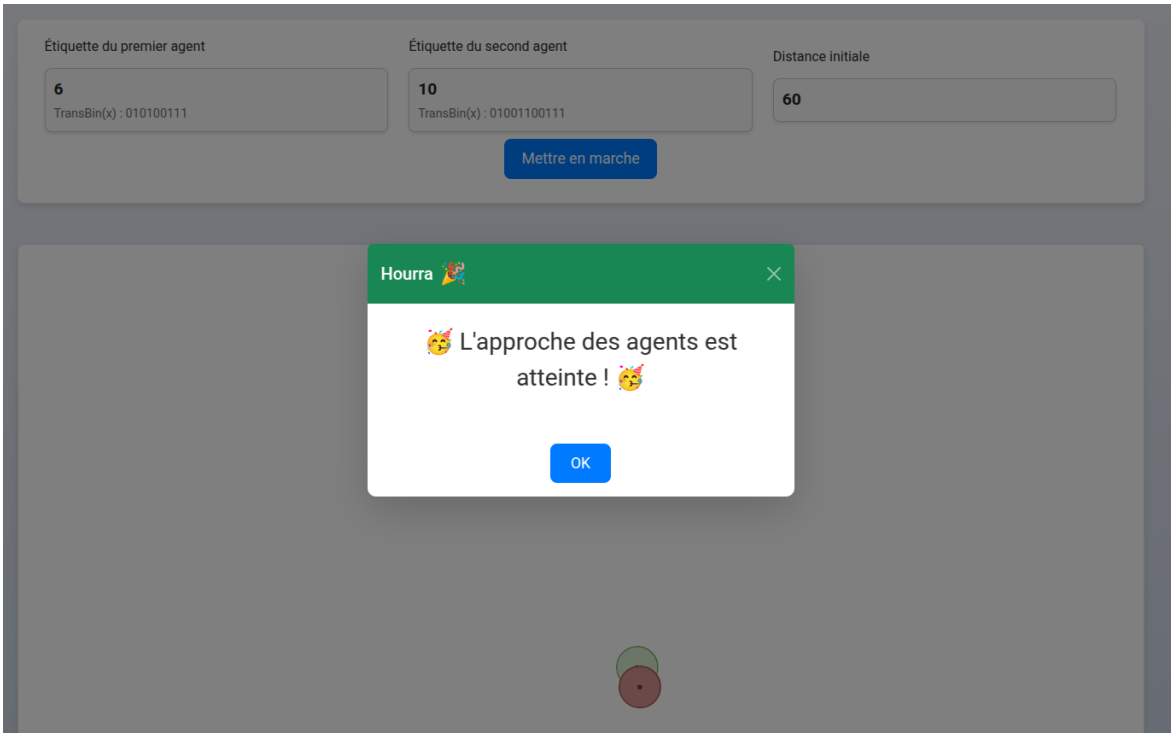

FIGURE 7.3 – Troisième capture : L'approche des agents.

Chapitre 8

Conclusion

Dans ce mémoire, nous avons étudié de manière déterministe deux problèmes fondamentaux liés à la mobilité des agents dans le plan : la chasse au trésor et l'approche des agents. Pour le problème de la chasse au trésor, nous avons proposé un algorithme qui permet à un agent, ne connaissant ni la position du trésor ni la valeur de la distance maximale initiale, de le détecter en effectuant un parcours en spirale. Ce procédé, dont la complexité temporelle est en $O(D^2)$, où D est une borne inconnue sur la distance entre la position initiale de l'agent et le trésor, constitue un outil auxiliaire pour aborder le problème de l'approche.

Dans le problème de l'approche, deux agents mobiles munis d'étiquettes distinctes et opérant de manière synchronisée se déplacent dans le plan afin de se rapprocher l'un de l'autre jusqu'à atteindre une distance au plus de 1 unité. En combinant la méthode de la chasse au trésor et l'exploitation des informations issues des étiquettes, nous avons démontré qu'il est possible d'assurer, de manière déterministe, une rencontre des agents en temps $O(D^2 \log L)$, où D est une borne inconnue sur la distance initiale entre les agents et L est la grandeur de l'espace des étiquettes. Nous avons aussi démontré une borne inférieure $\Omega(D^2)$ sur le temps optimal de l'approche. Ainsi, notre algorithme d'approche est asymptotiquement optimal si la grandeur (inconnue) de l'espace des étiquettes est constante.

Notre analyse repose sur un modèle simplifié dans lequel le plan est exempt d'obstacles, ce qui nous a permis de concentrer notre étude sur les défis intrinsèques aux problèmes de la chasse au trésor et d'approche. Il est à noter que nos résultats peuvent être généralisés facilement au cas où les agents naviguent dans l'espace 3-dimensionnel au lieu du plan. La méthode d'explorer le plan par un parcours en spirale devrait alors

être remplacée par une exploration de l'espace organisée en phases dont la i-ème consiste à parcourir la surface d'un cube centré au point du départ de l'agent et dont le coté est de longueur i. Ainsi, la chasse au trésor pourrait être exécutée dans l'espace en temps $O(D^3)$ et l'algorithme d'approche basé sur elle fonctionnerait en temps $O(D^3 \log L)$, où D est une borne inconnue sur la distance initiale entre les agents et L est la grandeur de l'espace des étiquettes.

Une perspective naturelle pour des travaux futurs consisterait à étendre ces algorithmes à des environnements plus complexes, incluant des terrains arbitraires avec obstacles, afin de refléter des situations réelles et d'enrichir le cadre d'application de nos solutions.

Bibliographie

[1] E. Bampas, J. Czyzowicz, L. Gasieniec, D. Ilcinkas, A. Labourel, Almost Optimal Asynchronous Rendezvous in Infinite Multidimensional Grids, Proc. 24th International Symposium on Distributed Computing (DISC 2010), 297-311.

[2] S. Bouchard, Y. Dieudonné, B. Ducourthial, Byzantine Gathering in Networks, Distributed Computing 29(6) (2016), 435-457.

[3] S. Bouchard, M. Bournat, Y. Dieudonné, S. Dubois, F. Petit, Asynchronous Approach in the Plane : A Deterministic Polynomial Algorithm, Distributed Computing 32(4) (2019), 317-337.

[4] S. Bouchard, Y. Dieudonné, A. Lamani, Byzantine Gathering in Polynomial Time, Distributed Computing 35(3) (2022), 235-263.

[5] M. Cieliebak, P. Flocchini, G. Prencipe, N. Santoro, Distributed Computing by Mobile Robots : Gathering, SIAM J. Comput. 41 (2012), 829-879.

[6] A. Collins, J. Czyzowicz, L. Gasieniec, A. Kosowski, R. A. Martin, Synchronous Rendezvous for Location-Aware Agents, Proc. 25th International Symposium on Distributed Computing (DISC 2011), 447-459.

[7] J. Czyzowicz, A. Kosowski, A. Pelc, How to Meet When You Forget : Log-Space Rendezvous in Arbitrary Graphs, Distributed Computing 25(2) (2012), 165-178.

[8] J. Czyzowicz, A. Kosowski, A. Pelc, Time vs. Space Tradeoffs for Rendezvous in Trees, Distributed Computing 27(2) (2014), 95-109.

[9] Y. Dieudonné, A. Pelc, D. Peleg, Gathering Despite Mischief, ACM Transactions on Algorithms 11(1) (2014), 1-28.

[10] Y. Dieudonné, A. Pelc, V. Villain, How to Meet Asynchronously at Polynomial Cost, SIAM Journal on Computing 44 (2015), 844-867.

[11] G. A. Di Luna, P. Flocchini, L. Pagli, G. Prencipe, N. Santoro, G. Viglietta, Gathering in Dynamic Rings, Theoretical Computer Science 811 (2020), 79-98.

[12] G. A. Di Luna, P. Flocchini, N. Santoro, G. Viglietta, M. Yamashita, Meeting in a Polygon by Anonymous Oblivious Robots, Proc. 34th International Symposium on Distributed Computing (DISC 2020), 445-469.

[13] P. Flocchini, G. Prencipe, N. Santoro, P. Widmayer, Gathering of Asynchronous Oblivious Robots with Limited Visibility, Theoretical Computer Science 337 (2005), 147-168.

[14] P. Flocchini, N. Santoro, G. Viglietta, M. Yamashita, Rendezvous with Constant Memory, Theoretical Computer Science 621 (2016), 57-72.

[15] S. Guilbault, A. Pelc, Asynchronous Rendezvous of Anonymous Agents in Arbitrary Graphs, Proc. 15th International Conference on Principles of Distributed Systems (OPODIS 2011), 421-434.

[16] A. Miller, A. Pelc, Fast Deterministic Rendezvous in Labeled Lines, Proc. 37th International Symposium on Distributed Computing (DISC 2023), 29 :1-29 :22.

[17] A. Ta-Shma, U. Zwick, Deterministic Rendezvous, Treasure Hunts, and Strongly Universal Exploration Sequences, ACM Trans. Algorithms 10 (2014) : 12 :1-12 :15.