UNIVERSITÉ DU QUÉBEC EN OUTAOUAIS

EXPLORATION OPTIMALE D'UN ARBRE PAR UN ESSAIM D'AGENTS MOBILES

MÉMOIRE
PRÉSENTÉ
COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN INFORMATIQUE

PAR

MÉLANIE ROY

MAI 2012

UNIVERSITÉ DU QUÉBEC EN OUTAOUAIS

Département d'informatique et d'ingénierie

Ce mémoire intitulé :

EXPLORATION OPTIMALE D'UN ARBRE PAR UN ESSAIM D'AGENTS MOBILES

présenté par

Mélanie Roy

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

a été évalué par un jury composé des personnes suivantes :

Dr. Andrzej Pelc ..... Directeur de recherche
Dr. Jurek Czyzowicz ..... Codirecteur de recherche
Dr. Larbi Talbi ..... President du jury
Dr. Mohand S. Allili ..... Membre du jury

16 mai 2012

À Hugo, Léa et Julien

Remerciements

Je voudrais remercier mes deux directeurs de recherche Dr. Andrzej Pelc et Dr. Jurek Czyzowicz pour leur soutien financier et d'avoir toujours été là pour répondre à mes questions. Merci à mon conjoint, Hugo Breton pour son soutien, ses encouragements et ses idées géniales. Merci à mes enfants pour avoir mis à l'épreuve le logiciel de simulation. Merci à Ghyslaine Breton et Louise Charron-Roy pour l'aide à la correction du français. Merci à Samuel Guilbault pour ses conseils et suggestions de même qu'à Guillaume Larivière, Pablo Julian Pedrocca et Julian Anaya.

Table des matières

Liste des figures

Liste des tableaux

Résumé

Ce mémoire propose une étude complète de l'exploration optimale d'un arbre enraciné par un essaim d'agents mobiles. L'arbre est connu. L'essaim dispose d'un nombre illimité d'agents qui commencent tous dans le même nœud (la racine) et se déplacent de façon synchrone. Les agents doivent visiter l'arbre le plus rapidement possible. Ils ont une restriction : à chaque ronde, il doit y avoir au maximum une distance de d entre les agents les plus éloignés les uns des autres. Le travail consiste à construire un algorithme optimal d'exploration et à faire trois preuves : la preuve de l'exactitude de l'algorithme, la preuve de son temps d'exécution et la preuve de l'optimalité. D'autre part nous montrons que tous nos algorithmes peuvent être adaptés à la situation où le nombre d'agents est égal au nombre de feuilles de l'arbre. Nous construisons aussi un simulateur qui illustre le fonctionnement de l'algorithme.

Abstract

Optimal tree exploration by a swarm of mobile agents

This text presents a complete study of the optimal exploration of a rooted tree by a swarm of mobile agents. Full knowledge of the tree is assumed. The swarm has an unlimited number of mobile agents starting at the same node (the tree root) and traveling synchronously. These agents must visit the tree as fast as possible. They have a restriction : in each round, there is a maximum distance of d between the agents that are the farthest apart. Our goal is to construct an optimal algorithm, prove its correctness, its time complexity and prove that the algorithm is optimal. We show that all our algorithms can be adapted to the scenario where the number of agents is equal to the number of leaves of the tree. We also built a simulator illustrating how the algorithm works.

Chapitre 1

Introduction

La chute constante du coût du matériel informatique et sa miniaturisation permettent l'utilisation d'une grande quantité de composantes. En même temps que la décroissance des coûts, la complexité des systèmes informatiques et des réseaux augmente. L'exploration des réseaux de plus en plus complexes doit demeurer rapide et efficace. D'où l'importance d'étudier et de comprendre les limites de l'exploration de ces réseaux.

Une façon d'explorer ces réseaux est avec des agents mobiles. Des agents mobiles logiciels sont de petites applications dans un système à architecture distribuée qui servent à effectuer diverses tâches comme la détection de pannes. Dans un contexte différent, en nanotechnologie et en intelligence artificielle, des agents mobiles robots peuvent être utilisés pour explorer des réseaux (physiques) et devoir le faire le plus rapidement possible ou efficacement. Nous pouvons penser par exemple, à l'exploration du système sanguin par des nano-robots.

L'exploration peut concerner soit des réseaux modélisés par des graphes ou des terrains. Dans les graphes, les systèmes informatiques ou les appareils mobiles sont représentés par des nœuds et les connections qui les relient sont représentées par des arêtes. Les agents mobiles sont parfois appelés « robots ».

Les modèles d'exploration des réseaux peuvent varier à différents niveaux. Voici trois axes principaux et des exemples de caractéristiques possibles pour chacune :

- **Les topologies** : des graphes dirigés ou non-dirigés, des arbres, des lignes, des graphes généraux.
- **La façon d'explorer** : le nombre d'agents (un seul, deux, un groupe), les conditions internes (la capacité des robots comme la mémoire et la vision), les conditions externes (le mode de communication, les contraintes sur le positionnement des agents, la synchronie).
- **L'objectif de l'exploration** : la faisabilité (dans les cas avec des contraintes), l'optimalité (par rapport au temps d'exploration, au nombre de traversées d'arêtes, à la mémoire utilisée, au nombre d'agents).

L'exploration des graphes par un seul agent est un sujet de recherche qui a été abordé par plusieurs auteurs [18, 4, 3, 19, 17, 27, 25, 26, 15, 1, 20, 16]. Dans certains travaux, les robots disposent de marqueurs [18, 4] pour explorer un graphe inconnu. Les robots pouvaient alors déposer des marqueurs aux nœuds. Dans un autre cas, le robot devait explorer un graphe par portions [3] et revenir au point de départ avant de poursuivre l'exploration comme s'il devait aller à un point de ravitaillement. Une autre situation qui a été étudiée est un robot qui peut parcourir une certaine distance à partir du nœud de départ mais devait revenir à la source avant de poursuivre l'exploration comme s'il était attaché par une corde [19]. L'optimalité en terme du nombre de traversées d'arêtes a été étudiée pour l'exploration par un seul robot des graphes fortement connexes [15, 1, 20]. Un des travaux sur l'exploration de graphes a été fait pour comprendre la limite de la mémoire [17]. L'exploration avec deux robots des graphes dirigés inconnus [5] et l'exploration des arbres avec un groupe d'agents [22, 24] ont aussi été étudiés. Dans un des travaux avec un groupe d'agents, l'objectif était de trouver le nombre minimal de robots faibles pour explorer un arbre [22]. Un autre avait pour but d'optimiser l'exploration avec différents scénarios de communication [24]. Le temps d'exploration était alors calculé par rapport au nombre d'agents.

1.0.1 Le modèle et le problème

Un essaim d'agents mobiles est un regroupement d'un grand nombre d'agents mobiles qui ne peuvent pas s'éloigner les uns des autres à une distance plus grande qu'une borne prévue d'avance. Aucun travail n'a été fait concernant l'optimisation du temps d'exploration d'un graphe avec un essaim d'agents mobiles. Nous avons choisi d'étudier

la question suivante : Quel est le temps minimal pour l'exploration d'un réseau avec un nombre suffisamment grand d'agents mobiles ? Les agents sont synchrones et ils débutent tous dans la racine de l'arbre. Nous mesurons le temps en nombre de rondes. À chaque ronde, chaque agent peut traverser une arête pour se rendre dans un nœud adjacent ou ne rien faire. Nous ne tenons pas compte du temps du calcul local, seulement du temps de déplacement des agents. Nous restreignons le type de graphes à des arbres qui sont connus d'avance par les agents. Nous imposons une restriction sur la distance entre les agents les plus éloignés les uns des autres dans l'essaim. Cette distance d est mesurée en nombre d'arêtes. Autrement dit, à chaque ronde les agents peuvent se trouver n'importe où dans le graphe, à l'intérieur d'une certaine borne d. Une des raisons pour garder une distance maximale entre les agents peut être la nécessité de maintenir la communication entre eux (par exemple, sans fil) en tout temps.

1.0.2 Les résultats

Ce mémoire présente des algorithmes optimaux pour deux cas : d pair et d impair. Ils sont placés dans ce document en ordre de complexité de compréhension. Pour d pair, nous faisons d'abord $d = 0$, $d = 2$ et ensuite le cas général d pair. Pour d impair, nous faisons d'abord $d = 1$ puis le cas général d impair. L'algorithme pour d pair, ExploreEssaimPair, peut être utilisé pour $d = 2$, à la place de ExploreEssaim2. L'algorithme pour d impair, ExploreEssaimImpair, peut être utilisé pour $d = 1$. Puisque les cas spécifiques $d = 2$ et $d = 1$ sont plus simples à expliquer et à prouver, nous en faisons des sections séparées, préliminaires à la généralisation d pair et d impair.

Lorsque $d = 0$, l'exploration par l'essaim est équivalent à l'exploration par un seul agent. Pour n'importe quel algorithme d'exploration d'un arbre avec un essaim et $d = 0$, nous avons démontré que $2(n-1) - h$ rondes est la borne inférieure où n est le nombre de nœuds de l'arbre et h, la hauteur de l'arbre. L'algorithme trouvé pour $d = 0$, Explore1Agent, fait l'exploration en $2(n-1) - h$ rondes, donc il est optimal.

Pour $d = 2$, nous avons démontré que $2(n-F) - h$ rondes est la borne inférieure pour parcourir un arbre où F est le nombre de feuilles de l'arbre. L'algorithme trouvé pour $d = 2$, ExploreEssaim2, fait l'exploration en $2(n-F) - h$ rondes, donc il est optimal.

Le niveau d'un nœud est calculé ainsi :

$\nu(n_i)$ = le niveau d'un nœud n_i
n_j est le fils de n_i dont le niveau est le plus grand

$$\nu(n_i) = \begin{cases} 0 & \text{si } n_i \text{ est une feuille} \\ \nu(n_j) + 1 & \text{si } n_i \text{ est un nœud interne} \end{cases}$$

Pour d pair, nous avons démontré que $2(n-H-1)-h+d$ rondes est la borne inférieure pour parcourir un arbre où H est le nombre de nœuds de l'arbre dont la hauteur est plus petite ou égale à $d/2$. L'algorithme trouvé pour d pair, ExploreEssaimPair, fait l'exploration en $2(n-H-1)-h+d$ rondes, donc il est optimal.

Un arbre T' est formé à partir d'un arbre T en coupant tous les nœuds de niveau inférieur à $\lfloor d/2 \rfloor$ ainsi que leurs arêtes adjacentes. Pour d impair, nous avons démontré que $\rho + 2n' - m' - h' - 1$ est la borne inférieure pour parcourir un arbre où $\rho = \lfloor d/2 \rfloor$, n' le nombre de nœuds de l'arbre T', m' le nombre de feuilles de T' et h' la hauteur de T'. L'algorithme trouvé pour d impair, ExploreEssaimImpair, fait l'exploration en $\rho + 2n' - m' - h' - 1$ rondes, donc il est optimal.

Le tableau 1.1 résume les résultats obtenus pour chacune des valeurs de d.

Distance maximale (d)	**Algorithme**	**Temps d'exécution**	**Borne inférieure**
0	Explore1Agent	$2(n-1)-h$	$2(n-1)-h$
2	ExploreEssaim2	$2(n-F)-h$	$2(n-F)-h$
pair	ExploreEssaimPair	$2(n-H-1)-h+d$	$2(n-H-1)-h+d$
impair	ExploreEssaimImpair	$\rho+2n'-m'-h'-1$	$\rho+2n'-m'-h'-1$

TABLE 1.1 – Sommaire des résultats pour chacun des algorithmes du projet de mémoire

Chapitre 2

État des connaissances

L'exploration par des agents mobiles ou des robots est un sujet qui touche différents domaines de recherche en informatique. La figure 2.1 est un arbre qui sert à illustrer où se situe ce mémoire par rapport à certains domaines et sujets connexes. La racine de l'arbre, « Exploration par des agents mobiles ou robots », est le sujet général. La feuille la plus à droite représente le sujet de ce mémoire ; l'exploration dans les graphes par un essaim d'agents mobiles synchrones. Les domaines les plus éloignés de notre sujet qui seront abordés sont la sécurité des agents mobiles et les essaims de robots en intelligence artificielle.

L'état des connaissances se fera de gauche à droite selon cet arbre de classification. Il débute par la feuille la plus éloignée, à l'extrémité gauche, pour ensuite descendre l'arbre vers la droite et rejoindre le sujet de ce mémoire. En premier lieu, il sera question de la sécurité des agents mobiles suivi de l'intelligence d'essaim (les essaims de robots en intelligence artificielle), puis, l'exploration dans les terrains et l'exploration dans les graphes. L'exploration des graphes sera subdivisée en plusieurs parties : l'exploration par un seul agent, l'exploration par plusieurs agents (des agents ou des robots asynchrones et synchrones). Pour les graphes, il y aura aussi la description de quelques travaux concernant la tolérance aux pannes, les trous noirs et les nœuds fautifs.

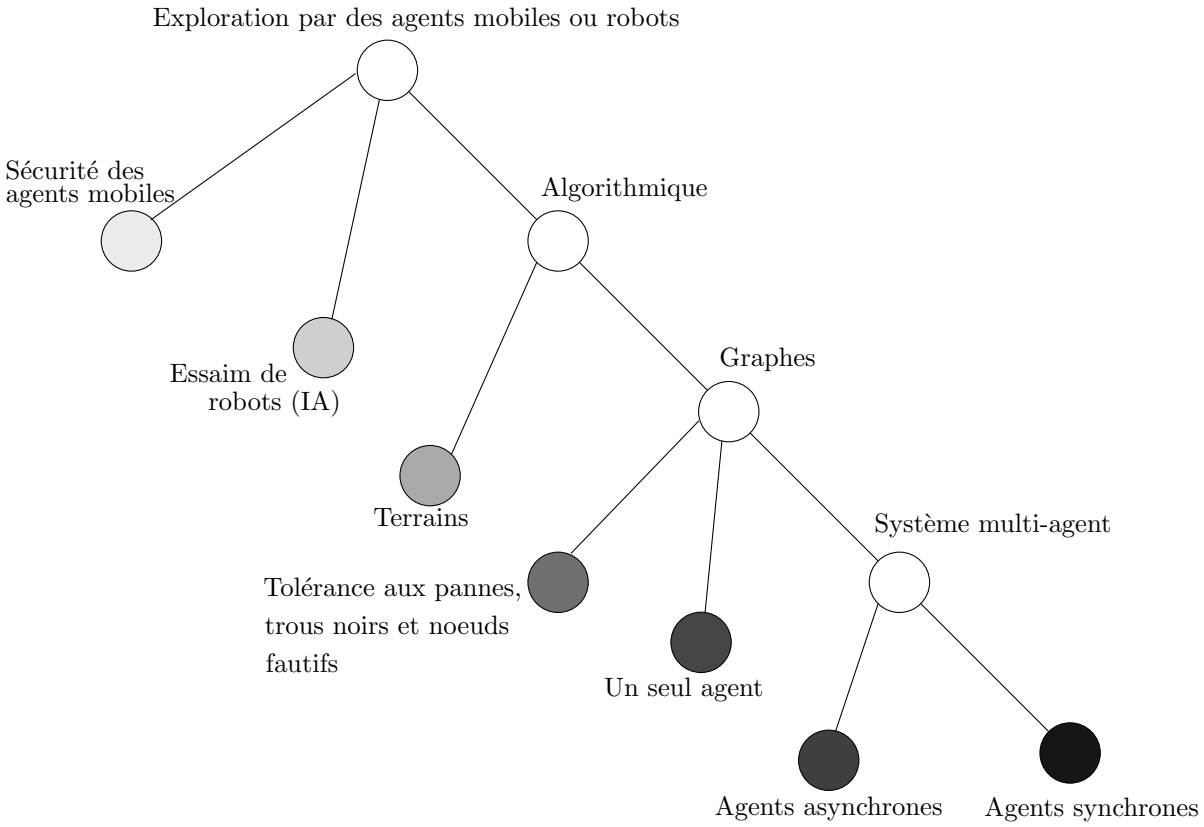

TABLE 2.1 – La classification des domaines et sujets de recherche concernant l'exploration par des robots ou des agents mobiles.

2.1 La sécurité des agents mobiles

Un agent correspond souvent à un petit logiciel qui peut agir de façon autonome. Il peut avoir différentes habiletés et fonctionnalités qu'il utilise pour accomplir des tâches spécifiques. Ce terme a été utilisé originellement en intelligence artificielle et est maintenant répandu dans tous les domaines de l'informatique.

Un agent est dit « intelligent » s'il peut agir de façon autonome sans intervention humaine directe et s'il possède une certaine flexibilité pour réagir à des changements dans son environnement. Un agent est qualifié de « social » lorsqu'il peut interagir avec d'autres agents et il est dit « mobile » lorsqu'il peut se déplacer d'un hôte à l'autre. Une des applications des agents mobiles est de réduire la congestion dans les réseaux.

Cette mobilité les rend vulnérables aux attaques des hôtes et des autres agents. Certains auteurs ce sont intéressés à la sécurité de ces agents mobiles [34, 9].

Voici quelques exemples de problèmes de sécurité décrits dans le travail de Borselius [9] :

- l'observation ou la manipulation du code ou des données d'un agent
- le contrôle du flot de données (incluant la route d'un agent)
- une mauvaise exécution du code (incluant la ré-exécution), entièrement ou en partie
- la modification de l'identité d'un hôte
- l'écoute ou la modification des communications d'un agent
- la falsification des valeurs de retour

Il y a deux types d'attaques possibles impliquant des agents mobiles : une attaque de l'hôte par des agents et une attaque des agents par l'hôte. Les agents mobiles et les systèmes multi-agent présentent plus de risques que les agents non-mobiles et ils sont plus difficiles à protéger. Pour les agents mobiles, la difficulté est accrue à cause des trous noirs. Ils peuvent détruire complètement l'agent sans laisser de traces. Les systèmes mutli-agent sont aussi difficiles à protéger parce que dans de tels systèmes, un agent ne peut résoudre un problème sans aide, en général, il n'y a pas de contrôle centralisé/global, les données sont décentralisées et les calculs sont asynchrones. Pour chacune de ces caractéristiques, il peut y avoir des problèmes de sécurité différents à considérer, ce qui complique la tâche de protection des agents.

La sécurité a un coût. Plus nous voulons de sécurité, plus le coût est élevé. Toutes les solutions demandent des ressources additionnelles et de la communication supplémentaire.

Les mécanismes de sécurité doivent être dynamiques. Ils doivent pouvoir s'adapter si les besoins des systèmes changent. Les besoins de sécurité d'un système à l'autre peuvent varier.

Sander et Tschudin [34] ont démontré qu'il est possible de protéger le travail des agents sans la présence d'équipement résistant au trafique (« tamper resistant hardware »). Leur approche est uniquement logicielle. Ils ont fait la preuve qu'il est possible de faire le chiffrement de fonctions de sorte qu'elles sont protégées de l'hôte sur lequel s'exécute le code. Ils ont suggéré le terme « cryptographie mobile » pour ce domaine de recherche en pleine croissance. Leur solution est un pas de plus vers une solution générale à la sécurité des agents mobiles.

2.2 L'intelligence artificielle et l'intelligence d'essaim

Dans cette section, nous parlons des solutions heuristiques en intelligence artificielle, c'est-à-dire une classe d'algorithmes où on ne s'intéresse pas à l'optimisation de la performance mais plutôt aux résultats pratiques.

2.2.1 La simulation

En robotique, la simulation est utile pour visualiser et mieux comprendre le comportement des robots. En intelligence d'essaim (« swarm intelligence »), il existe cinq comportements populaires que les robots peuvent adopter lorsqu'ils effectuent des tâches collectives : (1) bouger aléatoirement, (2) posséder une carte de l'environnement, (3) posséder une carte de l'environnement mais seulement dans un rayon limité, (4) communiquer les informations sur l'environnement seulement lorsqu'ils se rencontrent et (5) utiliser des phéromones, des interactions locales et la « stigmergie ».

Ces cinq comportements ne sont pas faciles à comprendre lorsqu'ils sont décrits en théorie seulement. Tinham et Menezes [36] ont conçu des simulations avec l'application StarLogo afin d'en faciliter l'explication aux étudiants. Ils ont utilisé le problème d'un essaim de robots qui doivent trouver la sortie d'un labyrinthe inconnu. Leur simulateur pourrait aussi être utilisé, éventuellement, pour comparer les résultats et savoir quel comportement est plus adapté à quelle situation.

2.2.2 La dispersion

Un des buts de la robotique d'essaims est de faire collaborer un grand nombre de minuscules robots afin qu'ils effectuent des tâches complexes qu'un seul robot ne pourrait pas accomplir. Avant d'accomplir une tâche collective, parfois les robots doivent se disperser. Après la dispersion, ils peuvent effectuer en collaboration des tâches comme : le calcul d'une distance, le calcul du chemin le plus court entre deux points, la recherche d'un point d'engorgement, la production d'une carte ou la recherche d'un intrus. Parfois, la dispersion est dans une portion limitée de l'environnement mais il peut aussi s'agir d'occuper tout l'espace.

Hsiang et al. [28] ont trouvé des algorithmes pour que les robots se dispersent le plus rapidement possible dans un environnement. L'environnement qu'ils ont utilisé est polygonal et inconnu. Ils ont expliqué l'importance d'étudier les algorithmes de dispersion avec de tels environnements, irréguliers, complexes pour refléter le mieux possible la réalité. L'environnement avec lequel ils ont travaillé est composé de pixels et modélisé par un sous-ensemble connexe d'une grille à coordonnées entières. Les robots peuvent bouger horizontalement et verticalement. Il peut y avoir un seul robot par pixel. La communication et la détection sont locales et les robots ont une mémoire limitée, de grandeur constante. Lorsque les robots pénètrent dans l'environnement, ils le font par une porte. Cette porte est représentée par un pixel et il peut y avoir plusieurs portes dans le même environnement.

Pour un environnement avec une seule porte, ils ont proposé deux algorithmes : Depth-First Leader-Follower (DFLF) et Breadth-First Leader-Follower (BFLF). Le temps pour remplir toute la région pour ces deux algorithmes est de $2A - 1$ où A est le nombre de pixels. Le temps total parcouru par tous les robots est inférieur avec le BFLF.

Un ratio compétitif (ou le surplus) est :

Le maximum des ratios $\frac{A(D)}{Opt(D)}$ où $A(D)$ est le coût de l'algorithme A pour des données D et $Opt(D)$ est le coût de l'algorithme clairvoyant (qui connaît tout).

Un des objectifs lors de la dispersion des robots avec un environnement à portes multiples est de maintenir un flot de robots qui sortent du plus grand nombre de portes possibles. Un problème rencontré est que les robots ont tendance à bloquer le passage aux autres. Ils ont quand même réussi à trouver une stratégie avec l'algorithme pour portes multiples : Laminar Flow Leader-Follower (LFLF). Il est $O(\log(K+1))-$compétitif et la borne inférieure est de $\Omega(\log(K+1))-$compétitif où K est le nombre de portes. Leurs travaux incluent aussi un simulateur en Java pour l'implémentation de leurs algorithmes.

2.3 L'exploration des terrains

Plusieurs chercheurs ont étudié l'exploration par des agents mobiles ou des robots dans le contexte géométrique. L'environnement est alors appelé terrain. Ce terrain peut être connu d'avance ou inconnu, avoir des obstacles ou non. Lorsque le terrain est connu, les robots possèdent une carte du terrain décrivant les distances.

2.3.1 Les critères et les catégories

Les algorithmes non-heuristiques forment une classe spéciale d'algorithmes qui sont « corrects », c'est-à-dire qui terminent toujours les opérations avec succès et dont il est possible de mesurer la performance. Plusieurs algorithmes non-heuristiques pour l'exploration des terrains inconnus sont décrits dans une étude réalisée par Nageswara et al. [30].

Les problèmes qui retiennent le plus l'attention des chercheurs sont : trouver des méthodes qui garantissent la performance et prouver leur exactitude lorsque le terrain est inconnu. Ils ont divisé les algorithmes en 3 catégories : (1) ceux qui sont exacts mais où on ne s'intéresse pas à la performance, (2) ceux où on s'intéresse à la borne sur la distance parcourue ou le ratio par rapport au chemin le plus court et (3) les automates finis modélisant des agents avec une mémoire limitée. Pour la classification des algorithmes, ils ont utilisé deux critères importants : l'objectif général et le système sensoriel des robots. Chacun de ces critères a des subdivisions :

1. Objectif général

   (a) Classe A : la garantie que l'exploration se fait entièrement, que l'objectif est atteint (par exemple, sortir d'un labyrinthe)
   (b) Classe B : l'optimisation des paramètres (par exemple, l'optimisation de la distance parcourue)
   (c) Classe C : une capacité de calcul limitée (par exemple, exploration par un automate fini)

2. Système sensoriel
   (a) Tactile
   (b) Visibilité (continue ou discrète)

Une vision continue, c'est lorsque le robot peut observer pendant son trajet. Une vision discrète, c'est lorsque le robot effectue une opération spécifique pour observer en un temps donné. Le nombre de fois où le robot peut voir doit être déterminé.

2.3.2 Les obstacles

Les obstacles et le terrain peuvent avoir certaines particularités de la forme et l'orientation. Dans le travail de Blum et al. [7], il est question de l'exploration d'un terrain inconnu avec trois différents types d'obstacles. Un robot débute à un point de départ, explore et traverse le terrain en contournant les obstacles et termine en un point d'arrivée. Ces trois types d'obstacles sont :

- rectangulaires et alignés avec les axes
- rectangulaires avec orientation générale
- convexes en 2D et 3D

L'analyse des résultats se fait par le ratio de la distance parcourue par la distance du chemin le plus court. La plupart de leurs algorithmes sont optimaux à une constante près.

Ils ont décrit deux grandes classes de problèmes :

- Le problème du mur (« The Wall Problem ») : lorsque le point d'arrivée est sur une ligne verticale infinie et que les obstacles sont rectangulaires et orientés avec les axes.
- Le problème de la chambre (« The room Problem ») : le terrain est carré, les obstacles sont rectangulaires et orientés avec les axes, le point de départ se trouve sur un mur et le point d'arrivée, exactement au centre du terrain.

Berman et al.[6] ont résolu le problème ouvert suivant : pour le problème de la chambre, trouver si un algorithme aléatoire est meilleur qu'un algorithme déterministe. Ils ont trouvé un algorithme aléatoire dont la performance est légèrement supérieure à celle du déterministe déjà trouvé. Un robot doit se rendre à une cible en traversant une région inconnue contenant des obstacles rectangulaires alignés avec les axes. L'algorithme a un ratio compétitif de $O(n^{4/9} \log n)$ où n est la distance euclidienne entre la source et le point d'arrivée.

Ils ont utilisé le même algorithme aléatoire pour le problème du mur qui est lui aussi meilleur qu'un algorithme déterministe ($O(n^{4/9} \log n)$-compétitif). En utilisant la même technique, pour la navigation en 3D avec un algorithme aléatoire, cela donne un algorithme $O(n^{2/3-\epsilon})$-compétitif tandis que la borne inférieure pour le ratio compétitif d'un algorithme déterministe est de $\Omega(n^{2/3})$.

Un autre type de problème dans un terrain avec obstacles est celui de la « galerie d'art » (« Gallery Tour »). Le début de l'exploration se fait par une entrée, ensuite, il faut parcourir le terrain dans le but de visualiser tout l'environnement et quitter par une sortie. Tous les points du périmètre des murs et des obstacles doivent pouvoir être observés à partir d'au moins une position du robot pendant le trajet. L'algorithme Greedy-Online est la solution trouvée par Deng et al. [14] à ce problème. Un robot est placé dans une pièce inconnue et seuls deux points sont connus : l'entrée et la sortie. Ce travail est considéré comme la suite de ce qui avait été fait pour les graphes dirigés inconnus mais dans le modèle géométrique.

2.3.3 Le problème du rendez-vous

Un rendez-vous est la rencontre entre deux ou plusieurs robots en un même point dans le but d'échanger de l'information. Il peut se faire dans les terrains ou dans les graphes. Le problème du rendez-vous en robotique a été abordé par Roy et Dudek [33]. Deux robots hétérogènes, asynchrones explorent un environnement inconnu et peuvent communiquer que s'ils se rencontrent. Ils doivent produire une carte de l'environnement. Pour que l'exploration soit plus rapide qu'avec un seul robot, ils doivent se rencontrer et combiner l'information. Le point de rencontre doit être déterminé avant le rendez-vous. La difficulté est de choisir un point de rencontre fiable dans l'environnement inconnu.

Ils ont présenté deux stratégies de rendez-vous. La plus simple, celle qui est utilisée s'il n'y a pas de bruit (les senseurs), se fait en 4 étapes :

1. Explorer l'environnement ;
2. Trouver des bons points pour le rendez-vous ;
3. Choisir un point de rencontre ;
4. Se rendre au point de rencontre.

Ils ont présenté une autre stratégie de rendez-vous avec deux algorithmes : un déterministe et un probabiliste. L'algorithme probabiliste ne détermine pas à l'avance l'ordre des points de rencontre mais génère les probabilités qu'un point de rencontre soit visité pour un certain rendez-vous. L'algorithme déterministe créé la liste de toutes les combinaisons possibles de points de rencontre et spécifie dans quel ordre ils doivent être visités.

Leur travail contient aussi une partie de simulation avec de vrais robots où ils testent différentes stratégies de rendez-vous. Ce qui est intéressant de constater c'est qu'aucune stratégie n'est tout à fait bonne ou mauvaise. Cela dépend des facteurs et des circonstances. Même un algorithme exponentiel peut donner en pratique les meilleurs résultats. C'est le cas par exemple, en absence de synchronie et lorsqu'il n'y a pas de bruit (senseur).

2.3.4 Des robots faibles et le problème du rassemblement

Les robots ou les agents mobiles peuvent avoir différentes habiletés mais ils peuvent aussi être « faibles ». La faiblesse des robots se mesure par l'absence d'habileté. Voici des exemples de faiblesse : l'amnésie (lorsque les robots n'ont aucune mémoire), lorsque les robots sont asynchrones, la décentralisation, l'anonymat et l'absence de communication. En pratique, l'utilité d'avoir des robots faibles c'est qu'ils sont peu coûteux et peuvent être produits en grande quantité.

Cieliebak et al. [10] ont réussi à trouver une façon de rassembler des robots faibles (le problème de « gathering ») en un seul point. Les seules habiletés de leurs robots est la détection de la multiplicité et la vision illimitée. La détection de la multiplicité c'est-à-dire lorsqu'un robot peut savoir s'il y a un seul ou plusieurs robots en un point et la vision illimité c'est lorsqu'un robot observe l'environnement, il le voit au complet.

Grâce à leur solution, à partir de n'importe quelle configuration de départ, des robots mobiles faibles (autonomes, asynchrones, décentralisés, anonymes, sans communication, amnésiques et déterministes) peuvent se rassembler en un point dans le plan lorsque le nombre de robots est supérieur ou égal à 5. D'autres solutions ont déjà été trouvées [2, 11, 23, 31, 35]. Par exemple, lorsque :

- le nombre de robots est 3 ou 4,
- les robots ont une habileté supplémentaire comme la synchronie, un système de coordonnées commun ou des configurations de départ particulières,
- les robots doivent converger vers un même point (sans nécessairement l'atteindre en un temps fini).

L'idée de l'algorithme pour plus de 5 robots est qu'en tout temps, il n'y a qu'un seul point où il y a plus d'un robot. Au départ, tous les robots sont séparés. Le point de rencontre est calculé par la suite.

Flocchini et al. [23] ont repris le même problème et ont présenté une solution pour des robots avec une vision limitée. Les robots sont faibles et asynchrones mais ont tout de même une habileté utile : ils ont un sens de l'orientation commun.

L'idée de base de l'algorithme est de faire bouger les robots pour qu'ils se rencontrent où est situé le robot le plus à droite et le plus en bas possible.

L'idée de l'algorithme :

- Si un robot en voit d'autres à gauche ou au dessus de lui, il ne bouge pas.
- Si un robot voit des robots en bas seulement, il avance à la verticale jusqu'à l'axe horizontale du robot le plus près.
- Si un robot voit des robots à droite seulement, il avance sur l'axe horizontale jusqu'à l'axe verticale du robot le plus près.
- Si un robot voit des robots en dessous et à droite, il calcule la position d'un point (en bas et à droite) et s'y rend.

Les robots doivent demeurer connectés en tout temps par rapport à leur vision, sinon, le rassemblement est impossible.

Un de ces auteurs, Prencipe [32], a prouvé que sans détection de multiplicité, il est impossible de rassembler des robots faibles. À ce moment, les robots sont vraiment faibles et n'ont aucune habileté particulière. Ils sont : amnésiques, anonymes, asynchrones, ne peuvent communiquer. Tout ce que peut faire un robot c'est un « cycle » : analyser (en observant l'environnement), calculer, se déplacer et rester inactif. Les robots peuvent savoir où se situent leurs voisins mais n'ont pas une vision illimitée et ils exécutent le même algorithme. Cette tâche ne peut être accomplie dans ces circonstances.

2.4 L'exploration des graphes

L'exploration des terrains inconnus et des graphes inconnus sont semblables mais présentent certains défis différents. Prenons par exemple la recherche dans un labyrinthe. Dans un graphe, elle est représentée par un graphe planaire. Dans un terrain, il y a une différence majeure qui facilite l'exploration : il est possible de s'orienter avec un compas, de distinguer le nord, le sud, l'est et l'ouest. Des auteurs ont prouvé [8] que trois automates ne peuvent explorer tous les graphes planaires cubiques.

Un des problèmes d'exploration d'un graphe est celui du « voyageur canadien ». La carte du graphe est disponible mais certaines routes (arêtes) sont impraticables (dues à une chute de neige, par exemple) et il n'est pas possible de le savoir à l'avance. Il est possible de savoir si une route (arête) est barrée seulement lorsque le nœud adjacent à cette arête est visité. Ce problème est PSPACE-complet ce qui veut dire très difficile à calculer [30].

2.4.1 Tolérance aux pannes, trous noirs et nœuds fautifs

Un trou noir est un processus situé dans un nœud qui détruit tout, le code de l'agent mobile inclus, sans laisser de traces. Pour la sécurité des agents et des hôtes, il est important de pouvoir les localiser. Localiser les trous noirs et le faire efficacement sont des questions sur lesquels se sont penché Czyzowicz et al. [13, 12]. Leurs travaux concernent la localisation d'un trou noir le plus rapidement possible (ou la conclusion qu'il n'y en a aucun).

Le nombre minimum d'agents pour détecter un trou noir est deux. Les agents débutent dans le même nœud, qui est garantit être sans trou noir. Les agents sont partiellement synchrones c'est-à-dire qu'il y a une borne supérieure sur le temps d'exploration d'un nœud. Ceci est réaliste car en pratique, les réseaux sont rarement totalement asynchrones. Les agents peuvent communiquer seulement quand ils se rencontrent.

La localisation d'un trou noir ne peut se faire en l'absence totale de synchronie parce que si l'agent n'a pas de réponse de l'autre agent, il ne peut savoir si c'est à cause d'une connexion lente ou parce qu'il s'est fait détruire par un trou noir.

L'approximation est une façon de mesurer la complexité des algorithmes. Il s'agit du rapport entre le temps d'un algorithme sur la borne inférieure pour ce problème. Czyzowicz et al. [12] ont démontré que le problème de trouver un trou noir le plus rapidement possible avec deux agents est NP-Difficile et ils ont produit un schéma de recherche au plus 9.3 fois plus lent que le schéma le plus rapide pour des entrées données. Il s'agit d'une approximation de 9.3. Un schéma est une séquence de traversées d'arêtes.

Ce même groupe de chercheurs [13] ont produit un BHS-scheme (Black-Hole-Search-scheme) le plus rapide possible pour toutes les lignes. L'algorithme « Bushy-Tree » produit un BHS-scheme le plus rapide pour n'importe quel arbre dense. Un arbre dense est un arbre dont les nœuds internes ont au moins deux enfants.

L'algorithme qu'ils ont présenté pour un arbre, « Tree », accomplit un rapport de $\frac{5}{3}$. Un rapport de $\frac{5}{3}$ veut dire que cela se fait en au plus $\frac{5}{3}$ du temps le plus court possible.

Deux agents ne sont pas suffisants pour détecter plus d'un trou noir. Il y a aussi une contrainte de connectivité pour que la localisation de tous les trous noirs soit faisable. Dans le cas d'un seul trou noir, la connectivité doit être 2 (« 2-connexe »). La figure 2.1 représente un graphe 2-connexe. Si on enlève un seul nœud (et les arêtes adjacentes), le graphe demeure connexe. Si on enlève 2 nœuds, il est possible de déconnecter le graphe (en enlevant les nœuds 3 et 4, par exemple).

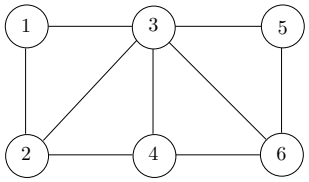

FIGURE 2.1 – Un graphe 2-connexe

Deux de ces auteurs, Markou et Pelc [29], ont travaillé sur un problème semblable mais au lieu de trous noirs, il s'agissait de nœuds fautifs dans une ligne et dans un arbre. Dans ce scénario, les nœuds ne détruisent pas les agents mais ne les laissent pas passer. La contribution majeure de leur recherche est d'avoir prouvé que les stratégies d'exploration naturelles sont performantes avec des arbres où il y a des pannes. Ils ont donné un algorithme parfaitement compétitif pour une ligne et un algorithme avec un surplus de $\frac{9}{8}$ pour un arbre. Un algorithme est parfaitement compétitif si le surplus est le plus petit possible pour un scénario donné.

2.5 L'exploration par un seul agent

L'exploration des graphes par rapport à l'exploration des terrains est parfois plus difficile (voir section 2.4) mais présente tout de même des avantages comme l'élimination de la nécessité de mesurer (géométrie) et la réduction des erreurs de positionnement des robots.

2.5.1 Les marqueurs

L'exploration avec marqueurs est une technique importante de l'exploration des graphes et est utilisée lorsque les nœuds sont anonymes. Un marqueur est un jeton qu'un agent peut laisser dans un nœud pour qu'un agent, possiblement le même, puisse l'observer durant une visite ultérieure de ce nœud. Cette approche permet d'identifier les nœuds pour briser la symétrie et permettre l'exploration ou accroître l'efficacité.

Dudek et al. [18] ont étudié la situation où un robot qui dispose de marqueurs doit traverser un graphe inconnu. Le robot doit aussi produire une carte de l'environnement. Ils étudient seulement les graphes sans cycles ≤ 2. Les arêtes sont numérotées localement dans le sens des aiguilles d'une montre. Sans marqueurs, il est impossible d'explorer les graphes réguliers car les sommets sont identiques et ne peuvent être distingués les uns des autres. Si le robot connaît la borne supérieure sur le nombre de marqueurs, cela est plus facile et la complexité est inférieure. L'algorithme a une meilleure performance s'il peut y avoir plus d'un marqueur. Par contre, les marqueurs peuvent être réutilisés alors il n'y a pas d'avantages à en avoir plus d'une dizaine. Il y a une grande amélioration de la performance entre 1 et 5 marqueurs, une petite amélioration entre 5 et 10 et une amélioration négligeable avec plus de 10 marqueurs.

L'algorithme qu'ils ont trouvé fonctionne en temps polynomial et ils l'ont mesuré de deux façons : (1) en étapes mécaniques, c'est-à-dire traverser le graphe, placer et prendre les marqueurs et (2) en étapes électroniques, les opérations sur les structures de données.

Ce problème a été repris par Bender et al. [4] mais pour un graphe dirigé. Ils ont trouvé un algorithme pour l'exploration d'un graphe dirigé inconnu, non-étiqueté. Ce

graphe est fortement connexe et les arêtes sortantes d'un nœud sont numérotées de 1 à d, où d est le degré du nœud. Un robot débute dans un nœud arbitraire. Après un certain nombre d'étapes, il doit produire une carte. Sans marqueur, il est impossible de le faire. Si le robot connaît la borne supérieure sur le nombre de nœuds, il peut efficacement apprendre le graphe avec un seul marqueur. $\Omega(\log \log n)$ marqueurs sont nécessaires et suffisants si le robot ne connaît pas la borne supérieure sur le nombre de nœuds où n est le nombre de nœuds. Le temps d'exploration de l'algorithme est $O(b \cdot 2n^6d^2)$ si b, la borne supérieure sur le nombre de nœuds, est connue.

Un exemple d'applications de ce type d'algorithme est l'obtention d'une carte d'un réseau ; une portion d'Internet, un réseau local, un système d'égouts, une grotte ou un réseau pour appareils mobiles. Aucune application pratique ne peut utiliser ces algorithmes directement parce qu'ils sont trop lents mais ils peuvent être utiles pour certaines situations particulières. Entre autre, lorsque les liens sont unidirectionnels ou dans un réseau où les nœuds ne peuvent être distingués facilement. Pour être utilisable en pratique, un algorithme avec une complexité inférieure est nécessaire.

2.5.2 L'exploration des graphes par portions

Une autre technique importante et utile de l'exploration des graphes est celle de l'exploration « par portions » (« piecemeal »). L'exploration se fait alors sur une plus petite partie du graphe à la fois. La contrainte d'explorer par portions peut se faire parce qu'il y a une limite sur le nombre d'arêtes à traverser à chaque phase d'une durée limitée, par exemple. Ces limites peuvent être imposées pour une raison d'énergie ; un robot qui doit revenir à la base pour recharger ses piles. Ce genre de robots pourraient être utilisés pour explorer des endroits où il est dangereux pour l'humain de s'aventurer comme à l'intérieur d'un volcan, un site d'enfouissement ou la surface d'une autre planète.

Awerbuch et al. [3] ont présenté le modèle suivant : un robot doit apprendre un nouvel environnement par portions et revenir au point de départ après chaque étape comme à un point de ravitaillement. L'environnement est un graphe non-dirigé arbitraire et inconnu du robot. Le robot peut distinguer les arêtes et les nœuds visités de ceux non-visités.

Dans cet article, ils ont démontré qu'un robot peut explorer n'importe quelle portion de graphe non-dirigé en temps presque linéaire, $O(E+V^{1+O(1)})$. À ce moment, le problème était encore ouvert de prouver si cela pouvait se faire en temps linéaire.

L'algorithme de recherche en profondeur (DFS - Depth First Search) peut faire l'exploration des graphes connexes non-dirigés en temps linéaire lorsque les nœuds sont étiquetés. Avec un graphe dirigé fortement connexe, $G = (V, E)$, et des nœuds étiquetés, un algorithme de recherche vorace (« Greedy Search Algorithm ») peut le faire en temps $O(|V| \cdot |E|)$. Ce temps peut même être amélioré avec certaines techniques. Ce qui pose une plus grande difficulté que le fait que les graphes soient dirigés, c'est l'identité des nœuds. Si les nœuds sont non-étiquetés, c'est plus difficile. [1, 4]

Duncan et al. [19] ont présenté un algorithme optimal à une constante près pour un problème semblable. Il s'agit de l'exploration d'un graphe inconnu avec un robot attaché par une corde. Le robot explore le graphe par portion en revenant périodiquement au point de départ. L'algorithme fonctionne en temps linéaire O(E) si la longueur de la corde est de $(1 + \alpha)r$, où α est une constante positive et r, le rayon du graphe (la distance entre le nœud le plus éloigné et le nœud de départ). S'il s'agit d'un réservoir d'essence au lieu d'une corde, le résultat est semblable, la complexité est la même et la grosseur du réservoir est de $2(1 + \alpha)r$. Il est possible de le faire sans connaître r, mais à ce moment, à mesure que le graphe est découvert, si le rayon est plus grand que celui estimé, il faut pouvoir permettre à la corde de s'allonger ou au réservoir de grossir.

2.5.3 L'optimisation de la mémoire

L'exploration des graphes par des agents mobiles avec le moins de mémoire possible est pratique car la plupart des agents sont petits, peu sophistiqués. Diks et al. [17] se sont intéressés au problème de l'exploration d'un arbre inconnu avec le moins de mémoire possible pour ces 3 situations :

- l'exploration perpétuelle : où le robot explore le graphe mais n'a pas besoin de s'arrêter ;

– l'exploration avec arrêt (sans retour) : le robot doit s'arrêter à un certain moment après l'exploration ;

– l'exploration avec arrêt dans le nœud de départ (avec retour) : le robot doit s'arrêter dans le nœud de départ après l'exploration.

Ils ont trouvé un algorithme pour la situation la plus difficile, où le robot doit s'arrêter au nœud de départ après l'exploration, et qui le fait avec $O(\log^2 n)$ bits de mémoire. Ce résultat a été par la suite amélioré dans [27]. Ils ont trouvé les bornes supérieures et inférieures pour chacune de ces tâches :

Exploration	**Borne supérieure**	**Borne inférieure**
Perpétuelle	$O(\log \Delta)$ bits	$\lceil \log \Delta \rceil$ bits
Sans Retour	$O(\log \Delta + \log m)$ bits	$\Omega(\log \log \log n)$ bits
Avec Retour	$O(\log^2 n)$ bits	$\Omega(\log n)$ bits

TABLE 2.2 – Sommaire des résultats des travaux de Diks et al. [17]

Δ = le degré maximum des nœuds de l'arbre
m = borne supérieure sur le nombre de nœuds
n = nombre de nœuds de l'arbre

Gasieniec et al. [27] ont affirmé que si on peut identifier les nœuds, DFS est la façon la plus rapide d'explorer un arbre. Ils ont expliqué aussi que si les nœuds n'ont pas d'identificateurs, il doit y avoir une possibilité de distinguer les ports localement, sinon, un agent ne peut faire la différence entre celui qu'il a visité précédemment de celui qu'il doit visiter. Dans leur travail, les ports peuvent être distingués mais ils sont allés encore plus loin en ne les identifiant pas explicitement. Ils sont ordonnés de façon circulaire mais les numéros ne sont pas connus. De cette façon, la présence d'un cycle peut empêcher l'exploration. Ils restreignent donc la sorte de graphes à des arbres.

L'algorithme d'exploration avec retour trouvé (LogExploration) pour tout arbre d'au plus n nœuds utilise $O(\log n)$ bits de mémoire. Ce qui correspond à la borne inférieure prouvée antérieurement. L'arbre est non-orienté, l'agent ne connaît pas le graphe, ni sa grandeur. Ce qui prend le plus de mémoire, la difficulté majeure, c'est de savoir si l'arbre a été entièrement exploré sans connaître sa grandeur.

Deux chercheurs, Fraigniaud et Ilcinkas [25], ont travaillé sur un problème semblable : l'exploration d'un graphe dirigé avec peu de mémoire. Le graphe est inconnu et les ports sont étiquetés localement. Ils ont présenté deux modèles différents : avec un robot et avec un agent. Le robot dispose de marqueurs tandis que l'agent dispose d'un tableau blanc (« whiteboard ») à chaque nœud où il peut écrire de l'information. L'identificateur du port par lequel l'agent quitte le nœud et de l'information au sujet du nœud est le type d'information que l'agent écrit sur un tableau blanc. Un marqueur est l'équivalent d'un tableau blanc de 1 bit.

Pour le modèle du robot : ils ont fait la preuve que $\Omega(n \log d)$ bits de mémoire sont nécessaires pour l'exploration perpétuelle (sans arrêt) d'un graphe dirigé fortement connexe à n nœuds de degré maximal d. Ils ont présenté un algorithme qui utilise $O(n \cdot d \log n)$ bits de mémoire et utilise un seul marqueur. Celui-ci est pour une exploration avec arrêt et il a un temps exponentiel. La preuve avait déjà été faite que pour qu'un algorithme fonctionne en temps polynomial dans ces conditions, il faut au minimum $\Omega(\log \log n)$ marqueurs. Ils ont présenté un autre algorithme avec un temps polynomial, qui utilise $O(n^2 d \log n)$ bits de mémoire et $O(\log \log n)$ marqueurs. Cet algorithme est donc optimal pour le nombre de marqueurs.

Pour le modèle de l'agent : ils ont fait l'observation qu'un agent amnésique ne peut explorer un graphe avec arrêt et font la preuve que l'exploration avec arrêt ne peut se faire avec une mémoire de zéro bits. Ils ont montré aussi qu'un agent peut explorer avec arrêt en utilisant un tableau blanc de $O(\log d)$ bits à chaque nœud. L'algorithme DFS accompli l'exploration avec retour et arrêt au nœud de départ avec $O(1)$ bits de mémoire et des tableaux blancs à chaque nœud de $O(\log d)$ bits de mémoire.

Un rapport compétitif c'est le ratio compétitif maximisé pour tout graphe et tout nœud. Pour battre le rapport compétitif de 2 du DFS, il faut beaucoup d'information sur le graphe, ce qui n'est pas réaliste en pratique. Fraigniaud et al. [26] ont réussi à le faire avec un montant minimal d'information en utilisant un oracle. Un oracle fournit à l'agent un nombre limité de bits d'information sur l'arbre. Ils ont trouvé un algorithme qui bat le rapport compétitif de 2 et qui utilise un oracle de grandeur $O(\log \log D)$, où D est le diamètre de l'arbre.

2.5.4 L'optimisation de l'exploration

Deng et Papadimitriou [15] ont travaillé sur le problème d'optimiser l'exploration en mesurant le nombre de traversées d'arêtes. Le problème est le suivant : un robot doit explorer un graphe inconnu fortement connexe et produire une carte. Les nœuds sont reconnus s'ils sont visités pour la deuxième fois. Le but de leur travail est de minimiser le ratio, c'est-à-dire, le nombre d'arêtes traversées par l'algorithme comparé au nombre optimal d'arêtes traversées (lorsque le graphe est connu).

Un graphe eulérien est un graphe qui contient un cycle eulérien. Un cycle eulérien est un chemin qui parcourt chaque arête une seule fois. La déficience d'un graphe est le nombre d'arêtes qu'il faut ajouter à un graphe pour qu'il devienne eulérien.

Le résultat principal de leur travail est un algorithme qui accomplit un ratio borné lorsque la déficience est bornée mais ce ratio est exponentiel, $3(d+1)^{2d+1}$ où d est la déficience du graphe. Ils ont démontré que lorsque l'information partielle à propos du graphe est disponible, minimiser le pire ratio est PSPACE-complet.

Pour les graphes non-dirigés, DFS est la meilleure solution avec un ratio de compétitivité de 2, et il est optimal. Le problème est plus difficile pour les graphes dirigés fortement connexes. Des chercheurs ont travaillé sur ce type de graphes [1, 20] où un robot doit construire une carte d'un environnement inconnu. À tout moment, le robot connaît tous les nœuds visités et il peut les reconnaître s'il sont visités pour une deuxième fois. Albers et Henzinger [1] ont trouvé un algorithme qui améliore la borne supérieure prouvés par Deng et Papadimitiou [15] $d^{O(d)}m$, où d est la déficience du graphe et m, le nombre d'arêtes. La nouvelle borne supérieure est donc $d^{O(\log d)}m$. Un autre des résultats de leur recherche est la preuve que pour les algorithmes Greedy, DFS et BFS, il existe un graphe de déficience d qui nécessite $2^{\Omega(d)}$ traversées d'arêtes.

Fleischer et Trippen [20] sont les premiers à avoir trouvé un algorithme avec un ratio polynomial par rapport à la déficience du graphe. L'algorithme utilise au plus $O(d^8m)$ traversées d'arêtes.

Dessmark et Pelc [16] ont repris le même problème (l'optimalité de l'exploration d'un graphe inconnu) sous un angle différent. Un agent doit traverser tous les nœuds et toutes les arêtes. Le graphe est connexe et non-dirigé. L'agent doit utiliser le moins de traversées d'arêtes possible. Ils s'intéressent à 3 scénarios :

- n'avoir aucune information sur le graphe ;
- avoir une carte isomorphique du graphe (« unanchored map »), c'est-à-dire une carte qui décrit pour chacun des nœuds quel port mène à quel voisin ;
- avoir une carte avec le nœud du départ marqué (« anchored map »).

Les nœuds sont distincts et les ports numérotés (de 1 à $deg(v)$, v étant le nœud). Une des contributions est l'établissement des bornes inférieures pour démontrer que les algorithmes trouvés sont optimaux. Voici les résultats des algorithmes pour les 3 classes de graphes (lignes, arbres et graphes généraux) :

	Sans carte	**Carte isomorphique**	**Carte avec un nœud marqué**
Lignes	(DFS) surplus = 2	surplus $=\sqrt{3}$	surplus = 7/5
Arbres	(DFS) surplus = 2	surplus < 2 (borne inférieure $=\sqrt{3}$)	surplus = 3/2
Graphes généraux	(DFS) surplus = 2	(DFS) surplus = 2	(DFS) surplus = 2

TABLE 2.3 – Sommaire des résultats de Dessmark et Pelc [16]

2.6 Les systèmes multi-agent

2.6.1 Les agents asynchrones

Une autre façon d'optimiser l'exploration des graphes est de le faire avec plusieurs agents. Deux auteurs, Bender et Slonim [5], ont étudié l'exploration avec deux robots coopérants. En même temps que l'exploration, les robots tracent une carte du graphe. Ils ont prouvé que deux robots coopérants peuvent explorer n'importe quel graphe dirigé

fortement connexe de nœuds non-étiquetés en temps polynomial. Chaque nœud a des arêtes sortantes numérotées de 0 à $d-1$ (où d est le nombre d'arêtes sortantes à ce nœud). Les robots peuvent reconnaître s'ils sont dans le même nœud. Ils peuvent communiquer par radio en tout temps. Cette communication radio est utilisée pour synchroniser les actions des robots même si les robots sont considérés comme étant asynchrones. Pour des robots synchrones, cette communication radio peut être remplacée par une chaîne de caractères aléatoire de longueur polynomiale qu'ils partagent.

Le fait que les robots soient asynchrones est une difficulté. Flocchini et al. [21] ont étudié une situation où en plus d'être asynchrones, les robots sont amnésiques et ne peuvent communiquer. Leur seule habileté est une vision illimitée. Ils doivent explorer un anneau non-orienté où les nœuds sont non-étiquetés. Pour effectuer l'exploration, les robots fonctionnent par cycle. Un cycle comprend 3 actions : l'observation de l'environnement, le calcul et le déplacement. En un temps fini, tout le graphe doit être exploré et les robots doivent tous s'arrêter. On a prouvé que $O(\log n)$ robots sont requis pour explorer un anneau de grandeur n. Lorsque n est arbitrairement grand, $\Omega(\log n)$ sont nécessaires. n et k doivent être co-premiers ; aucune solution n'existe si k divise n, k étant le nombre de robots.

Les même chercheurs ont repris le problème [22] mais pour l'exploration des arbres. Ils ont voulu trouver la grandeur minimale d'un groupe de robots permettant de résoudre ce problème. Comme précédemment, les robots sont amnésiques, asynchrones, ont une vision illimitée et ne peuvent communiquer. L'arbre est inconnu, non-orienté, non-étiqueté. Au départ, il ne doit pas y avoir plus d'un robot par nœud. L'idée de la solution est de créer une mémoire artificielle en codant la configuration des robots pour qu'ils puissent savoir quels nœuds ont été visités.

Ils ont fait la preuve que pour certains arbres à n nœuds, il est parfois nécessaire d'avoir $\Omega(n)$ robots, même si le degré maximal des nœuds est 4. Pour un degré maximal de 3, $\Theta(\frac{\log n}{\log \log n})$ robots sont requis et suffisants pour l'exploration. Une grande difficulté est la symétrie d'un arbre. Ils ont expliqué aussi que s'il n'y a aucun automorphisme non-trivial, alors 4 robots sont toujours suffisants et parfois nécessaires. Un automorphisme est un isomorphisme de l'ensemble de nœuds en lui-même. De façon moins formelle : un arbre possédant un automorphisme non-identique a une certaine symétrie.

2.6.2 Les agents synchrones

La synchronie des agents lors de l'exploration est un avantage. Cela facilite la coordination des déplacements des agents. C'est le type d'agents qui sont utilisés dans ce mémoire. Fraigniaud et al. ont utilisé des agents synchrones dans [24]. Le problème est semblable à celui de ce mémoire : les agents font l'exploration d'arbres. Dans [24], il y a trois scénarios différents pour la communication :

1. Lecture-écriture : les robots communiquent en lisant et écrivant des messages aux nœuds ;
2. Globale : tous les robots communiquent à chaque ronde ;
3. Aucune : les robots ne peuvent pas communiquer.

Les robots sont initialement situés dans la racine de l'arbre. Ils doivent le parcourir le plus rapidement possible en minimisant le temps. Le temps est calculé ainsi : lorsqu'un robot traverse une arête, cela prend une unité de temps. Le temps total est celui du robot qui a pris le nombre d'unités de temps maximales parmi tous les robots. Les robots ne sont pas anonymes, ils ont des identificateurs distincts et ils exécutent le même algorithme. Les nœuds ne sont pas étiquetés et les ports sont étiquetés localement seulement. Un robot qui traverse une arête connaît l'étiquette du port aux deux extrémités de l'arête.

Pour trouver une façon optimale de parcourir un arbre, même si l'arbre et le nœud de départ sont connus, est un problème NP-difficile.

Si les robots ne peuvent pas communiquer, il n'y a aucun avantages à avoir plusieurs robots pour l'exploration et le surplus est de $\Omega(k)$ (où k est le nombre de robots). C'est le même résultat que pour un seul robot. Avec la communication omniprésente, le surplus est de $O(\frac{k}{\log k})$. L'algorithme qu'ils ont trouvé travaille en temps $O(D + \frac{n}{\log k})$ pour tout arbre à n nœuds et de diamètre D. Ils ont fait la preuve que n'importe quel algorithme d'exploration avec plusieurs agents a un surplus d'au moins $2 - \frac{1}{k}$. Avec une communication de type lecture-écriture, l'algorithme qu'ils ont présenté travaille en temps $O(D + \frac{n}{\log k})$ et le surplus est de $O(\frac{k}{\log k})$.

Ce mémoire aborde un sujet qui se rapproche de deux des problèmes ouverts de l'article précédent en présentant des algorithmes optimaux en terme de temps d'exécution pour l'exploration d'arbres avec un essaim d'agents mobiles. Le nombre d'agents mobiles disponibles est suffisamment grand, ils ont une communication omniprésente mais la borne de l'essaim, c'est-à-dire la distance entre les agents les plus éloignés, est limitée et définie (d). La borne supérieure et la borne inférieure sur la grandeur de l'essaim (le nombre d'agents) sont des questions ouvertes.

Chapitre 3

Exploration d'un arbre

3.1 Terminologie et notation

Dans cette section nous présentons quelques définitions qui seront utilisées dans les sections suivantes (de 3.2 à 3.6).

Arbre : Un arbre est un graphe connexe sans cycle et non-orienté.

Arbre enraciné : Un arbre enraciné est un arbre où un unique nœud est désigné comme la racine r.

Feuille : Une feuille est un nœud de degré 1 de l'arbre enraciné, différent de la racine. (Si la racine est de degré 1, on ne la considère pas comme une feuille.)

La profondeur d'une feuille : La profondeur d'une feuille c'est sa distance de la racine.

Branche : Une branche est le chemin simple entre la racine et une feuille.

Diamètre de l'arbre : Le diamètre de l'arbre (dénoté D) est la distance entre les deux nœuds les plus éloignés.

Hauteur de l'arbre : La hauteur de l'arbre est dénotée h et c'est la distance la plus longue entre la racine et une feuille.

$\rho = \lfloor d/2 \rfloor$ (où d est la borne de l'essaim.)

$dist(v, w)$ est la distance entre les nœuds v et w.

$\Gamma_\rho(v) = \{w \in V : dist(v, w) \leq \rho\}$

$\Gamma(c) = \Gamma_1(c)$

Essaim avec noyau : Il existe un nœud c tel que l'ensemble des nœuds occupés par les agents est $\Gamma_\rho(c)$. (voir figure 3.1)

La figure 3.1 représente un essaim avec noyau lorsque $d = 4$. La distance qui sépare les agents les plus éloignés est égale à 2.

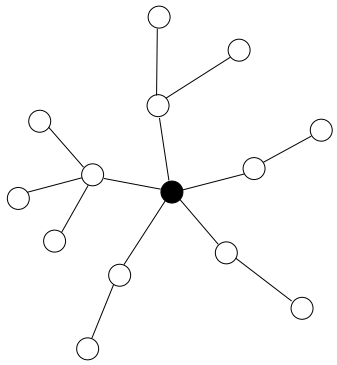

FIGURE 3.1 – Un essaim avec noyau lorsque $d = 4$

Algorithme avec noyau : Un algorithme avec noyau est un algorithme qui fonctionne de la façon suivante :
Pour chaque ronde $i \leq \rho$, l'ensemble des nœuds occupés par les agents est $\Gamma_i(r)$ où r est la racine.
Pour chaque ronde $i > \rho$, il existe un nœud c tel que l'ensemble des nœuds occupés par les agents est $\Gamma_\rho(c)$.

Un ver : Un ver est un ensemble de deux nœuds adjacents occupés par des agents $\{t, q\}$.

Un **noyau double** est un ver.

Essaim avec un noyau double : Il existe deux nœuds adjacents t et q tel que l'ensemble des nœuds occupés par les agents est : $\Gamma_\rho(t) \cup \Gamma_\rho(q)$.

La figure 3.2 représente un essaim avec noyau double lorsque $d = 5$.

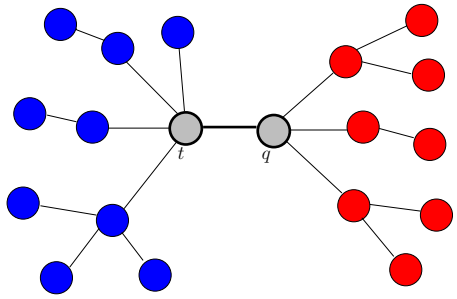

FIGURE 3.2 – Un essaim avec noyau double lorsque $d = 5$

Le niveau d'un nœud :
$\nu(n_i)$ = le niveau d'un nœud n_i
n_j est le fils de n_i dont le niveau est le plus grand

$$\nu(n_i) = \begin{cases} 0 & \text{si } n_i \text{ est une feuille} \\ \nu(n_j) + 1 & \text{si } n_i \text{ est un nœud interne} \end{cases}$$

Un arbre T' :
Un arbre T' est un sous-arbre formé à partir d'un arbre T en coupant les nœuds (ainsi que leurs arêtes adjacentes) dont le niveau est inférieur à ρ. (voir figure 3.3)

La figure 3.3 représente un arbre T' lorsque $d = 4$.

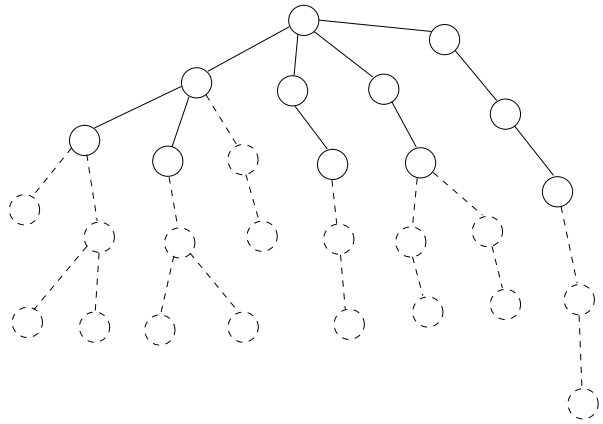

FIGURE 3.3 – Un arbre T' lorsque $d = 4$

3.2 Exploration optimale d'un arbre par un agent mobile

Dans cette section, nous présentons un algorithme qui fait l'exploration d'un arbre avec un agent mobile. Le temps d'exécution de l'algorithme est optimal. Nous décrivons l'algorithme, faisons la preuve de l'exactitude, calculons le temps d'exécution et faisons la preuve de l'optimalité.

3.2.1 L'algorithme

Pour un arbre enraciné en r, de hauteur h, l'algorithme Explore1Agent effectue un parcours en profondeur (DFS) à partir de r qui visite tous les nœuds et termine le parcours dans une feuille de profondeur h, sans retour à la racine.

3.2.2 La preuve de l'exactitude

L'agent traverse toutes les arêtes de l'arbre dans les deux sens en faisant le DFS sauf pour la dernière branche de hauteur h. L'agent traverse les arêtes de la dernière branche une seule fois, dans un sens. Puisque toutes les arêtes sont traversées au moins une fois, tous les nœuds de l'arbre sont visités.

3.2.3 Le temps d'exécution

Un arbre à n nœuds a $n-1$ arêtes. Toutes les arêtes sont parcourues deux fois sauf une branche de hauteur h, qui est parcourue une seule fois. L'agent termine l'exploration dans une feuille de profondeur h. Le temps d'exploration est donc :

$$2(n-1-h)+h =$$

$$2(n-1)-2h+h =$$

$$2(n-1)-h$$

3.2.4 La preuve de l'optimalité

Lemme 3.2.1. *Chaque algorithme d'exploration d'un arbre qui termine dans une feuille de profondeur h travaille en temps supérieur ou égal à $2(n-1)-h$.*

Démonstration. Chaque arête sauf la dernière branche de hauteur h doit être traversée au moins deux fois (puisqu'il y a un chemin unique de la racine à chaque feuille). Chaque arête de la dernière branche de hauteur h doit être traversée au moins une fois. Ce qui donne au total la borne inférieure de :

$$2(n-1)-h$$

□

Puisque le coût de l'algorithme Explore1Agent est égal au coût minimal de l'exploration d'un arbre avec un seul agent, nous avons le corollaire suivant :

Corollaire 3.2.2. *L'algorithme Explore1Agent est optimal.*

3.3 Exploration optimale d'un arbre par un essaim d'agents mobiles pour une borne $d = 2$

Dans cette section, nous présentons un algorithme qui explore un arbre avec un essaim d'agents mobiles. Cet algorithme est optimal pour le temps d'exécution. La contrainte est qu'à chaque ronde, il ne peut y avoir une distance supérieure à 2 arêtes entre les agents les plus éloignés les uns des autres. Nous décrivons l'algorithme, faisons la preuve de l'exactitude, calculons le temps d'exécution et faisons la preuve de l'optimalité.

Algorithme avec noyau (pour $d = 2$) : algorithme d'exploration tel qu'à chaque ronde $k > 1$, il existe un essaim avec noyau.

c_k est l'ensemble d'agents qui forment le noyau de l'essaim à la ronde k.

3.3.1 L'algorithme

L'algorithme ExploreEssaim2 fonctionne de la façon suivante : un arbre T' est formé à partir de l'arbre T en coupant toutes les feuilles et leur arête adjacente (voir figure 3.4). À la première ronde, un groupe d'agents reste dans la racine et d'autres agents sont déplacés dans tous les nœuds adjacents à la racine. Le nœud central (le noyau, c_k), effectue l'algorithme Explore1Agent (voir section 3.2) dans T'. Les autres agents se déplacent simultanément au noyau de la façon suivante :

v étant le nœud où se trouve c_k et w, le nœud où se trouve c_{k+1}. À chaque ronde :
- Tous les agents situés dans un nœud adjacent à v mais qui ne sont pas dans w se déplacent dans v ;
- Des groupes d'agents situés dans w se déplacent dans tous les nœuds adjacents à w, sauf v.

La figure 3.5 représente le déplacement du noyau de v à w et le déplacement des autres agents de la ronde k à la ronde $k + 1$.

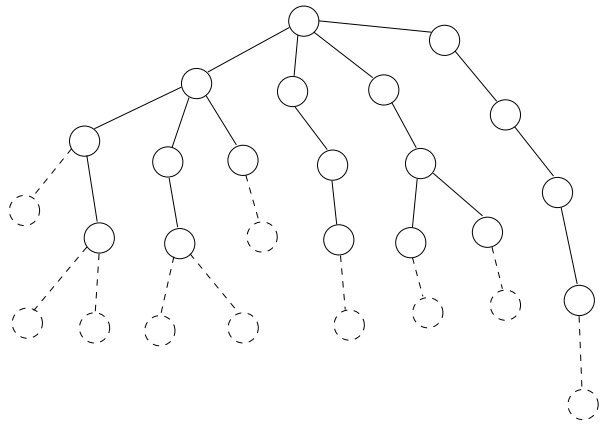

FIGURE 3.4 – Un arbre T' lorsque $d = 2$

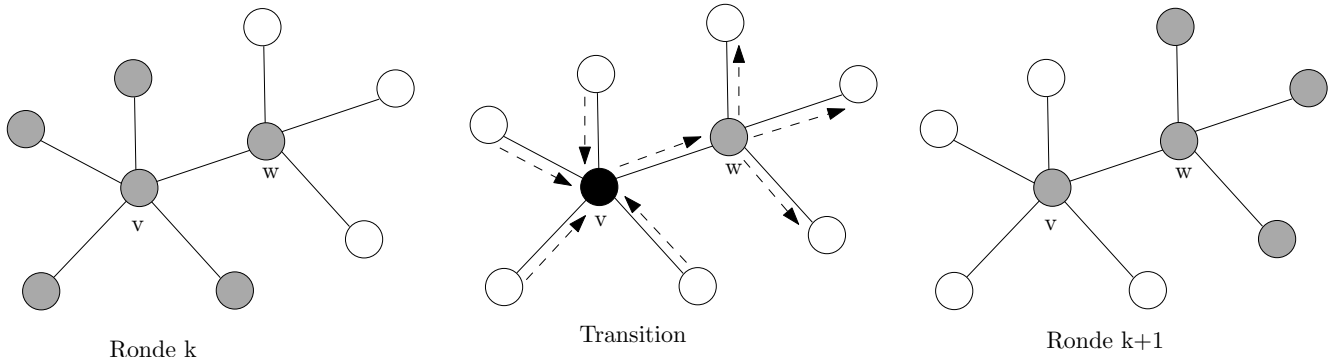

FIGURE 3.5 – Le déplacement des agents de la ronde k à la ronde $k + 1$

3.3.2 La preuve de l'exactitude

Un algorithme « correct » signifie qu'il accomplit avec succès sa tâche pour toutes les valeurs d'entrées possibles.

Théorème 3.3.1. *L'algorithme ExploreEssaim2 est correct.*

Démonstration. L'algorithme ExploreEssaim2 appelle la procédure Explore1Agent pour T' avec le noyau. La procédure Explore1Agent visite tous les nœuds de T' avec les agents du noyau de l'essaim. Toutes les feuilles de T' sont donc visitées par les agents du noyau. Puisque toutes les feuilles de T sont adjacentes aux feuilles de T', toutes les feuilles de

T sont visitées par les agents de l'essaim qui se trouvent dans les nœuds adjacents au noyau. Donc, tous les nœuds de T sont visités. □

3.3.3 Le temps d'exécution

Le temps d'exécution de l'algorithme ExploreEssaim2 est égal à $\tau + 1$ où τ est égal au temps d'exploration du noyau dans l'arbre T' et une ronde supplémentaire pour la formation de l'essaim avec noyau. Parcourir T' avec le noyau est égal au temps de Explore1Agent dans T' :
$2(n' - 1) - h'$ où n' est le nombre de nœuds de T' et h', la hauteur de T'.
$n' = (n - F)$ où F est le nombre de feuilles de T et n, le nombre de nœuds de T.
$h' = (h - 1)$

$$Coût(ExploreEssaim2) = 1 + 2(n' - 1) - h' =$$

$$1 + 2(n - F - 1) - (h - 1) =$$

$$2(n - F - 1) - h + 2 =$$

$$2(n - F) - h$$

3.3.4 La preuve de l'optimalité

Nous avons prouvé à la section 3.2 que l'exploration de l'arbre T' par un agent, utilisé comme procédure dans ExploreEssaim2, est optimale. Il reste à prouver que l'algorithme avec noyau est une façon optimale d'explorer T.

Soit A un algorithme quelconque d'exploration qui respecte la borne $d \leq 2$. Nous allons construire un algorithme A' avec noyau tel qu'à chaque ronde, l'ensemble des nœuds occupés par les agents de A' inclue l'ensemble des nœuds occupés par les agents de A.

Pour $k \geq 1$, l'ensemble des nœuds occupés par les agents de A (respectivement A') à la ronde k est dénoté par N_k (respectivement N'_k).

Nous prouvons par induction sur k que $N_k \subseteq N'_k$.

L'algorithme A' est le suivant :

1. Dans la première ronde, les agents de la racine r se répartissent dans tout $\Gamma(r)$; r est le noyau de l'essaim ; $c_1 := r$.
2. **Tant que** A n'a pas terminé l'exploration **Faire** :
   (a) Soit c_k le noyau de A' à la fin de la ronde k,
   (b) **Si** $N_{k+1} \subseteq N'_k$ **Alors**
   $c_{k+1} := c_k$
   Sinon
   Soit $Z = N_{k+1} \backslash N'_k$,
   Soit y l'unique voisin commun de Z et c_k,
   $c_{k+1} := y$.

Les autres agents bougent de la façon suivante : les agents des nœuds adjacents à c_k mais pas à c_{k+1} vont à c_k. Les agents du nœud c_{k+1} se répartissent dans tous les nœuds adjacents à c_{k+1}.

La figure 3.6 représente un ensemble Z et le nœud y (en rouge).

Lemme 3.3.2. *y est unique.*

Démonstration. (Par contradiction) Supposons qu'y n'est pas unique. Alors il y a deux nœuds de N_k, y et y' dont les nœuds adjacents, z et z', peuvent être visités à la ronde $k+1$. Puisque $dist(y, y') \geq 1$ alors $dist(z, z') \geq 3$. Par définition, à la ronde $k+1$, des agents peuvent visiter z et z', ce qui est une contradiction puisque $dist(z, z') > 2$. □

Lemme 3.3.3. *Pour chaque k, $N_k \subseteq N'_k$.*

Démonstration. Par induction sur k :

- Lorsque $k = 1$, il est évident que $N_k \subseteq N'_k$.
- Supposons que l'inclusion est vraie pour k. Nous la prouvons pour $k+1$:

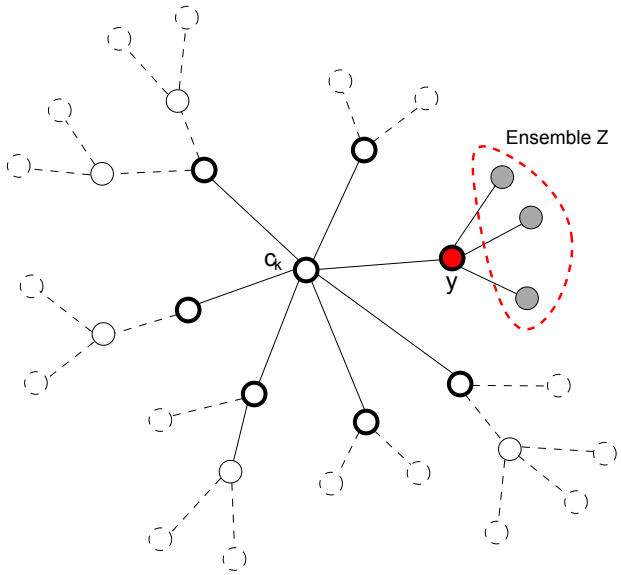

FIGURE 3.6 – Un ensemble Z et le nœud y lorsque $d = 2$

Cas 1 $N_{k+1} \subseteq N'_k$

Alors $N'_{k+1} = N'_k$, donc $N_{k+1} \subseteq N'_{k+1}$

Cas 2 $Z = N_{k+1} \backslash N'_k \neq \emptyset$

Alors chaque élément de N_{k+1} doit appartenir à $\Gamma(y)$, car autrement il serait à distance au moins 3 d'un élément de Z. Donc $N_{k+1} \subseteq N'_{k+1}$.

□

Le Lemme 3.3.3 implique le théorème suivant :

Théorème 3.3.4. *Pour chaque algorithme A d'exploration d'un arbre par l'essaim avec une borne $d = 2$, il existe un algorithme A' avec un noyau tel que son coût est inférieur ou égal à celui de A, c'est-à-dire :*

$$Coût(A') \leq Coût(A)$$

Lemme 3.3.5. *Tout algorithme avec noyau, lorsque $d = 2$, a un coût d'au moins $2(n - F) - h$.*

Démonstration. L'essaim avec noyau doit être créé avec un coût d'au moins une unité de temps (une unité de temps est égal à une ronde). Pour que tous les nœuds de T soient visités, toutes les feuilles de T doivent être visitées. Pour que toutes les feuilles de T soient visitées, le noyau doit parcourir au moins tout l'arbre T' et visiter toutes les feuilles de T'. Pour que le noyau explore tout l'arbre T', toutes les arêtes de T' doivent être traversées au moins 2 fois (une fois dans chaque sens) sauf la dernière branche de T', de hauteur h', dont les arêtes doivent être traversées au moins une fois. Le coût est donc au moins :

$$1 + 2(n' - 1) - h'$$

où n' est le nombre de nœud de T'
h' est la hauteur de T'
$n' = n - F$ où F est le nombre de feuilles de T
$h' = h - 1$ où h est la hauteur de T
Donc ce coût est au moins (voir section 3.3.3 pour les détails du calcul) :

$$2(n - F) - h$$

□

Puisque le coût de l'algorithme ExploreEssaim2 est égal au coût minimal de l'exploration d'un arbre avec essaim pour $d = 2$, nous avons le corollaire suivant :

Corollaire 3.3.6. *L'algorithme ExploreEssaim2 est optimal.*

3.4 Exploration optimale d'un arbre par un essaim d'agents mobiles pour une borne paire $d = 2\rho$ où $\rho > 1$

Dans cette section, nous présentons un algorithme qui explore un arbre avec un essaim d'agents mobiles. Cet algorithme est optimal pour le temps d'exécution. La contrainte est qu'à chaque ronde, il ne peut y avoir une distance supérieure à 2ρ où $\rho > 1$ entre les agents les plus éloignés les uns des autres. Nous décrivons l'algorithme, faisons la preuve de l'exactitude, calculons le temps d'exécution et faisons la preuve de l'optimalité.

3.4.1 L'algorithme

L'algorithme ExploreEssaimPair se fait en 2 étapes :
- La formation de l'essaim avec noyau
- L'exploration de l'arbre

La formation de l'essaim avec noyau (les ρ premières rondes) :
Un groupe d'agents reste dans la racine, formant le noyau. Les autres agents se déplacent dans les nœuds à une distance ρ de la racine en ρ étapes. À l'étape $i \leq \rho$, des groupes d'agents mobiles se déplacent dans les nœuds à une distance i de la racine. Si $h \leq \rho$, l'exploration est terminée après h rondes. Sinon, après ρ rondes, l'exploration de l'arbre commence.

L'exploration de l'arbre :
Le noyau, c_k, effectue l'algorithme Explore1Agent (voir section 3.2) dans T'. Les autres agents se déplacent simultanément au noyau de la façon suivante :

v étant le nœud où se trouve c_k et w, le nœud où se trouve c_{k+1}. Trois types d'agents composent l'essaim N'_k :
- Les agents B : sont dans un nœud b où $dist(b, v) < dist(b, w) \wedge dist(b, v) \neq 0 \wedge dist(b, w) \neq 0$.
- Les agents C : sont dans un nœud c où $dist(c, w) < dist(c, v) \wedge dist(c, v) \neq 0$.

– Les agents D : ceux qui se trouvent dans le nœud v.

À chaque ronde :

– Les agents B se déplacent de b dont la $dist(b, v) = x$ aux nœuds adjacents b' dont $dist(v, b') = x - 1$;
– Les agents C se déplacent de c dont la $dist(c, w) = x$ aux nœuds adjacents c' dont $dist(w, c') = x + 1$;
– Les agents D se déplacent de v à w.

La figure 3.7 représente le déplacement des agents de la ronde k à la ronde $k + 1$.

3.4.2 La preuve de l'exactitude

Théorème 3.4.1. *L'algorithme ExploreEssaimPair est correct.*

Démonstration. L'algorithme ExploreEssaimPair appelle la procédure Explore1Agent pour T' avec le noyau. La procédure Explore1Agent visite tous les nœuds de T' avec les agents du noyau de l'essaim. Toutes les feuilles de T' sont donc visitées par les agents du noyau. Puisque toutes les feuilles de T sont à une distance $\leq \rho$ de T', toutes les feuilles de T sont visitées par les agents de l'essaim qui se trouvent dans les nœuds à une distance $\leq \rho$ du noyau. Donc, tous les nœuds de T sont visités. □

3.4.3 Le temps d'exécution

Si $h \leq \rho$, le temps d'exécution de l'algorithme ExploreEssaimPair est h. Sinon, le temps d'exécution est égal au temps de l'exploration de l'arbre T' par un agent (le noyau) augmenté d'au plus ρ rondes pour la formation de l'essaim avec noyau. Le temps d'exploration de T' avec le noyau est égal au temps de Explore1Agent dans T' :
$2(n' - 1) - h'$ où n' est le nombre de nœuds de T' et h', la hauteur de T'.
H est le nombre de nœuds n_i dont $h(n_i) < \rho$
$n' = n - H$, le nombre de nœuds de T'

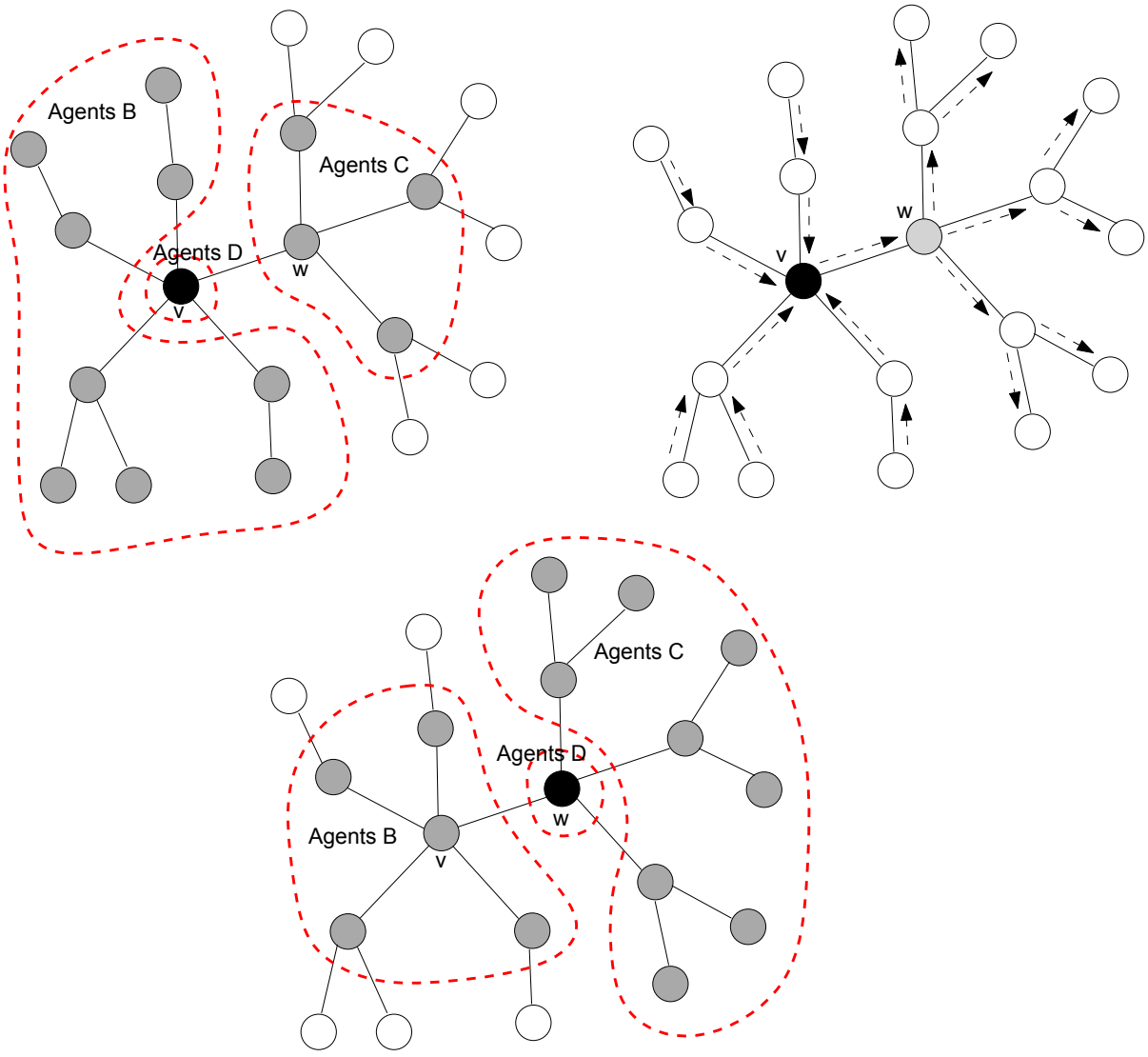

FIGURE 3.7 – Le déplacement des agents de la ronde k à la ronde $k+1$

$h' = h - \rho$, la hauteur de T'

La création de l'essaim avec noyau prend un temps $\leq \rho$.

$$Coût(ExploreEssaimPair) \leq \rho + 2(n - H - 1) - (h - \rho) =$$

$$2(n - H - 1) - h + d$$

3.4.4 La preuve de l'optimalité

Nous avons prouvé à la section 3.2 que l'exploration de l'arbre T' avec un agent, utilisé comme procédure dans ExploreEssaimPair, est optimale. Il reste à prouver qu'un algorithme avec noyau est une façon optimale de parcourir T.

Soit A un algorithme quelconque d'exploration qui respecte la borne maximale $d = 2\rho$ (où $p > 1$). Nous allons construire un algorithme A' avec noyau tel qu'à chaque ronde, l'ensemble des nœuds occupés par les agents de A' inclue l'ensemble des nœuds occupés par les agents de A.

Pour $k \geq 1$, l'ensemble des nœuds occupés par les agents de A (respectivement A') à la ronde k est dénoté par N_k (respectivement N'_k).

Nous prouvons par induction sur k que $N_k \subseteq N'_k$.

L'algorithme A' est le suivant :

1. À la ronde 0, tous les agents sont dans la racine.
2. À chaque ronde $i \leq \rho$, les agents se déplacent dans un nœud adjacent afin d'occuper les nœuds dans $\Gamma_\rho(r)$ à la ronde ρ ; r est le noyau de l'essaim.
3. **Tant que** A n'a pas terminé l'exploration **Faire** :
   (a) Soit c_k le noyau de A' à la fin de la ronde k,
   (b) **Si** $N_{k+1} \subseteq N'_k$ **Alors**
   $c_{k+1} := c_k$
   Sinon
   Soit $Z = N_{k+1} \backslash N'_k$,
   Soit $Y = \{y \in N_k : \exists z \in Z,\ y \text{ adjacent à } z\}$,
   Soit w le voisin de c_k tel qu'il existe $y \in Y$ et $dist(c_k, y) = dist(w, y) + 1$
   $c_{k+1} := w$.

Les autres agents se déplacent simultanément avec le déplacement du noyau de c_k à c_{k+1} de la même façon que dans l'algorithme ExploreEssaimPair.

Lemme 3.4.2. *w est unique.*

Démonstration. (Par contradiction) Supposons qu'w n'est pas unique. Alors il y a deux nœuds de N_k adjacents à c_k, w et w' dont les nœuds à une distance m , u et u', peuvent être visités à la ronde $k+1$. Puisque $dist(w, w') = 2$ alors $dist(u, u') = d + 2$. Par définition, à la ronde $k+1$, des agents peuvent visiter u et u', ce qui est une contradiction puisque $dist(u, u') > d$. □

La figure 3.8 illustre un ensemble $Y = \{y_1, y_2, y_3, y_4\}$ lorsque $d = 4$ et des nœuds w, w', u et u'.

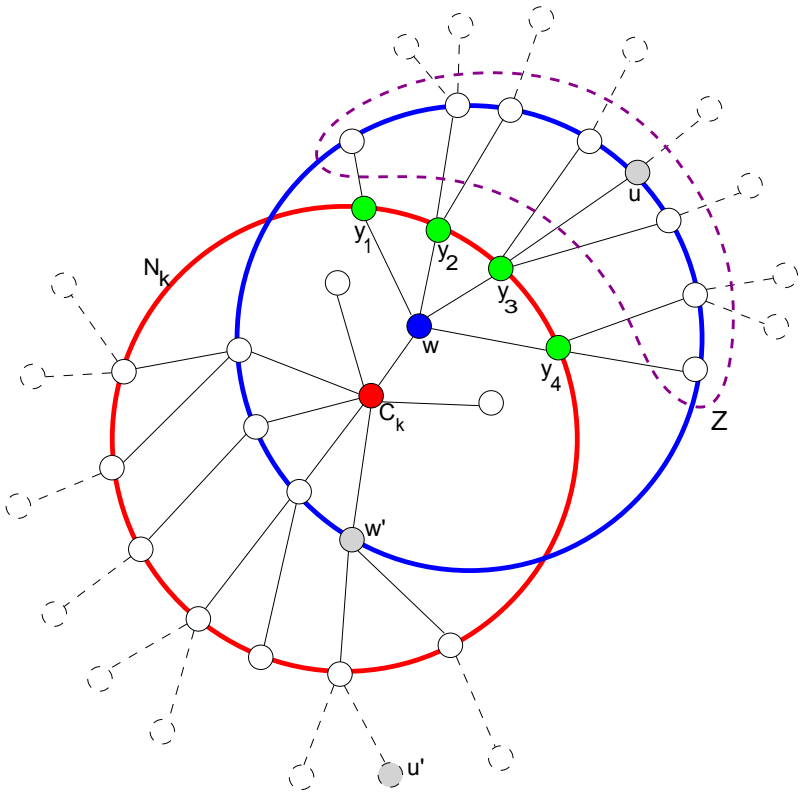

FIGURE 3.8 – Un ensemble Y lorsque $d = 4$

Lemme 3.4.3. *Pour chaque k, $N_k \subseteq N'_k$.*

Démonstration. Par induction sur k :

- Lorsque $k \leq \rho$, il est évident que $N_k \subseteq N'_k$.
- Supposons que l'inclusion est vraie pour k. Nous la prouvons pour $k+1$:

  Cas 1 $N_{k+1} \subseteq N'_k$

  Alors $N'_{k+1} = N'_k$, donc $N_{k+1} \subseteq N'_{k+1}$

  Cas 2 $Z = N_{k+1} \backslash N'_k \neq \emptyset$

  Alors chaque élément de N_{k+1} doit appartenir à $\Gamma_\rho(w)$, car autrement il serait à distance au moins $\rho + 1$ d'un élément de Z. Donc $N_{k+1} \subseteq N'_{k+1}$.

□

Le Lemme 3.4.3 implique le théorème suivant :

Théorème 3.4.4. *Pour chaque algorithme A d'exploration d'un arbre par un essaim pour $d = 2\rho$ où $\rho > 1$, il existe un algorithme A' avec un noyau tel que son coût est inférieur ou égal à celui de A, c'est-à-dire :*

$$Coût(A') \leq Coût(A)$$

Lemme 3.4.5. *Cas 1 : Lorsque $h \leq \rho$, tout algorithme avec noyau a un coût d'au moins h.*
Cas 2 : Lorsque $h > \rho$, tout algorithme avec noyau a un coût d'au moins $2(n - H - 1) - h + d$.

Démonstration. Cas 1 ($h \leq \rho$) :
Puisque les agents débutent dans la racine et que la distance entre la feuille la plus profonde et la racine est h, alors le coût est au moins h.

Cas 2 ($h > \rho$) :
L'essaim avec noyau doit être créé avec un coût de ρ unités de temps. Pour que toutes les feuilles de T soient visitées, le noyau doit parcourir au moins tout l'arbre T' et visiter toutes les feuilles de T'. Pour que le noyau explore tout l'arbre T', toutes les arêtes de

T' doivent être traversées au moins 2 fois (une fois dans chaque sens) sauf la dernière branche de T', de hauteur h', dont les arêtes doivent être traversées au moins une fois. Le coût est donc au moins :

$$\rho + 2(n' - 1) - h'$$

où n' est le nombre de nœud de T'
h' est la hauteur de T'
H est le nombre de nœuds n_i dont $\nu(n_i) < \rho$
$n' = n - H$, le nombre de nœuds de T'
$h' = h - \rho$, la hauteur de T'
Donc le coût est au moins (voir section 3.4.3 pour les détails du calcul) :

$$2(n - H - 1) - h + d$$

□

Puisque le coût de l'algorithme ExploreEssaimPair est égal au coût minimal de l'exploration d'un arbre avec essaim pour $d = 2\rho$ où $\rho > 1$, nous avons le corollaire suivant :

Corollaire 3.4.6. *L'algorithme ExploreEssaimPair est optimal.*

3.5 Exploration optimale d'un arbre par un essaim d'agents mobiles pour une borne d=1

Dans cette section, nous présentons un algorithme qui explore un arbre avec un essaim d'agents mobiles. Cet algorithme est optimal pour le temps d'exécution. La contrainte est qu'à chaque ronde, il ne peut y avoir une distance supérieure à un entre les agents les plus éloignés les uns des autres c'est-à-dire tous les agents doivent être soit dans un nœud, soit dans deux nœuds adjacents. Nous présentons quelques définitions, décrivons l'algorithme, faisons la preuve de l'exactitude, calculons le temps d'exécution et faisons la preuve de l'optimalité.

3.5.1 Définitions

Fixons un algorithme A pour un agent qui effectue un DFS et qui termine dans une des branches les plus profondes. Une « suite régulière » de nœuds de l'arbre, V, induit par cet algorithme est une suite de nœuds $(v_0, v_1, v_2, v_3, ..., v_k)$ tel que v_i est le nœud visité par l'algorithme à la ronde i. Étant donné que chaque feuille est visitée une seule fois, cette suite induit l'ordre de visite des feuilles où chaque feuille a un numéro de visite. Soit $(f_1, f_2, ..., f_m)$ une sous-suite de V contenant les feuilles en ordre de visite. On appelle une feuille « paire » si cette feuille a un numéro de visite pair et « impaire » une feuille dont le numéro de visite est impair.

Soit $(v_0, v_1, v_2, ...)$ la suite régulière où v_0 est la racine de T. Soit $[i_1, i_2, i_3, ...]$ tel que $f_i = v_{i_j}$ pour $j = 1, 2, ..., m$. Intuitivement, i_j est l'indice dans la suite régulière de la feuille f_j. On subdivise la suite des indices $[0, ..., k]$ de la suite régulière en segments $[0, ..., i_1]$, $[i_1 + 1, ..., i_2]$, ..., $[i_{m-1} + 1, ..., i_m]$. Le segment est pair s'il termine par l'indice d'une feuille paire et le segment est impair s'il termine par l'indice d'une feuille impaire.

3.5.2 L'algorithme

L'algorithme ExploreEssaim1 fonctionne de la façon suivante : à la ronde 0, tous les agents sont dans la racine. À la ronde 1, les agents sont partagés en deux sous-ensembles t et q. À la ronde 1, t est dans le nœud v_1 et q dans le nœud v_0. t est appelé la tête et

q, la queue. L'ensemble $\{t, q\}$ est appelé un ver. À chaque ronde $i > 1$, la tête est dans le nœud $v_{Tete(i)}$ et la queue dans le nœud $v_{Queue(i)}$ où $Tete(i)$ et $Queue(i)$ sont données par les formules suivantes. $Tete(i+1)$ et $Queue(i+1)$ ne sont pas définis si soit $Tete(i)$ ou $Queue(i)$ est l'indice de la dernière feuille.

$$Tete(i+1) = \begin{cases} 1 & \text{si } i = 0 \\ Tete(i)+1 & \text{si } v_{Tete(i)+1} \text{ n'est pas une feuille paire} \\ Tete(i)+3 & \text{si } v_{Tete(i)+1} \text{ est une feuille paire} \end{cases}$$

$$Queue(i+1) = \begin{cases} 0 & \text{si } i = 0 \\ Queue(i)+1 & \text{si } v_{Queue(i)+1} \text{ n'est pas une feuille impaire} \\ Queue(i)+3 & \text{si } v_{Queue(i)+1} \text{ est une feuille impaire} \end{cases}$$

La figure 3.9 illustre où se trouve la tête (bleue) et la queue (rouge) à la ronde 7 lors de l'exploration d'un arbre par l'algorithme ExploreEssaim1. Nous pouvons voir que puisque $Tete(7)+1$ n'est pas l'indice d'une feuille paire, la tête sera dans v_8 à la ronde 8. La queue sera dans v_9 à la ronde 8, $Queue(7)+1$ n'est pas l'indice d'une feuille impaire.

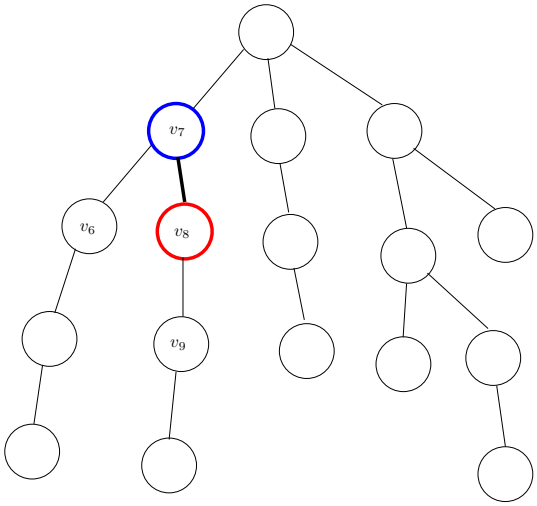

FIGURE 3.9 – La position de la tête (nœud bleu) et de la queue (nœud rouge) dans une arbre à la ronde 7.

Le lemme suivant décrit certaines propriétés importantes de l'algorithme.

Lemme 3.5.1. *Pour $i \geq 1$*

1. *Si $Tete(i)$ est dans un segment impair, alors $Queue(i) = Tete(i) - 1$.*
2. *Si $Tete(i)$ est dans un segment pair, alors $Queue(i) = Tete(i) + 1$.*
3. *$Tete(i)$ n'est pas l'indice d'une feuille paire et $Queue(i)$ n'est pas l'indice d'une feuille impaire.*
4. *$Tete(i)$ et $Queue(i)$ appartiennent au même segment.*

Démonstration. Preuve par induction selon la ronde i :

Si $i = 1$,
$Tete(1) = 1$
$Queue(1) = 0$
$Tete(1)$ est dans un segment impair, $Queue(1) = Tete(1) - 1 = 0$ donc la partie 1 du lemme est vérifiée. $Tete(1) = 1$ n'est pas l'indice d'une feuille paire, $Queue(1) = 0$ n'est pas l'indice d'une feuille impaire donc la troisième partie du lemme est vérifiée. Ils sont tous les deux dans le premier segment donc la partie 4 du lemme est vérifiée.

Supposons que le lemme est vrai pour i, prouvons-le pour $i+1$. Nous avons 3 cas pour lesquels les 4 conditions du lemme sont vérifiées.

Cas 1 Ni $Tete(i)$ ni $Queue(i)$ n'est l'indice d'une feuille.

Par l'algorithme,
$Tete(i + 1) = Tete(i) + 1$
$Queue(i + 1) = Queue(i) + 1$
Puisque $Tete(i)$ et $Tete(i + 1)$ sont dans le même segment, les parties 1 et 2 du lemme sont vraies.
Partie 3.
Par contradiction : supposons que $Tete(i+1)$ est l'indice d'une feuille paire, donc $Tete(i)$ est dans un segment pair. Par le point 2 du lemme, $Tete(i) = Queue(i) - 1$ donc $Tete(i + 1) = Tete(i) + 1 = Queue(i) - 1 + 1 = Queue(i)$. Étant donné que $Queue(i)$

n'est pas l'indice d'une feuille, alors $Tete(i+1)$ n'est pas l'indice d'une feuille, ce qui est une contradiction.
Le raisonnement pour $Queue(i+1)$ est semblable :
Par contradiction, supposons que $Queue(i+1)$ est l'indice d'une feuille impaire, donc $Queue(i)$ est dans un segment impair. Par le point 4 de l'hypothèse d'induction, $Tete(i)$ est dans un segment impair. Par le point 2 du lemme, $Queue(i) = Tete(i) - 1$. Donc $Queue(i+1) = Queue(i)+1 = Tete(i)$. Étant donné que $Tete(i)$ n'est pas l'indice d'une feuille, alors $Queue(i+1)$ n'est pas l'indice d'une feuille, ce qui est une contradiction.
La partie 4 est vérifiée puisque $Tete(i+1)$ est dans le même segment que $Tete(i)$ et $Queue(i+1)$ est dans le même segment que $Queue(i)$.

Cas 2 $Tete(i)$ **est l'indice d'une feuille impaire.**

Par l'algorithme,
$Tete(i+1) = Tete(i) + 1$
$Queue(i+1) = Queue(i) + 3$
Par hypothèse d'induction, on a $Queue(i) = Tete(i)-1$ donc $Tete(i+1) = Tete(i)+1 = Queue(i)+2 = Queue(i+1)-1$ donc $Tete(i+1) = Queue(i+1)-1$, $Tete(i)$ est dans un segment pair donc la partie 2 est prouvée. Les conditions de la partie 1 ne sont pas satisfaites donc il n'y a rien à prouver. $Tete(i+1)$ n'est pas l'indice d'une feuille paire car deux feuilles sont à distance au moins deux. $Queue(i+1)$ ne peut pas être l'indice d'une feuille impaire car deux feuilles impaires doivent être à distance plus grande que deux l'une de l'autre, ce qui prouve la partie 3.
$Tete(i+1)$ sera dans le segment suivant et $Queue(i+1)$ sera dans le segment suivant. Ainsi, $Tete(i+1)$ est le premier indice du segment suivant et $Queue(i+1)$ est le deuxième indice du segment suivant. Puisque chaque segment est de longueur supérieure à 1, $Tete(i+1)$ et $Queue(i+1)$ appartiennent au même segment ce qui prouve la partie 4.

Cas 3 $Queue(i)$ **est l'indice d'une feuille paire.**

Par l'algorithme,
$Tete(i+1) = Tete(i) + 3$,
$Queue(i+1) = Queue(i) + 1 = Tete(i) + 1 + 1 = Tete(i+1) - 1$.

Puisque $Queue(i+1)$ et $Tete(i+1)$ sont dans un segment impair, la propriété 1 est vérifiée. Les conditions de la partie 2 ne sont pas satisfaites donc il n'y a rien à prouver. Pour prouver la partie 3, observons que $v_{Queue(i)}$ et $v_{Tete(i+1)}$ sont à distance 2 et donc $Tete(i+1)$ ne peut pas être l'indice d'une feuille paire. $v_{Queue(i+1)}$ étant adjacent à $v_{Queue(i)}$, $v_{Queue(i+1)}$ ne peut pas être l'indice d'une feuille impaire.
Pour prouver la partie 4, observons que $Queue(i+1)$ est le premier indice du segment suivant et $Tete(i+1)$ est le deuxième indice du segment suivant. Puisque chaque segment est de longueur supérieure à 1, $Tete(i+1)$ et $Queue(i+1)$ appartiennent au même segment. □

3.5.3 La preuve de l'exactitude

Théorème 3.5.2. *L'algorithme ExploreEssaim1 est correct.*

La preuve du théorème est divisée en quatre lemmes.

Lemme 3.5.3. $v_{Queue(i+1)}$ *est adjacent à* $v_{Queue(i)}$.

Démonstration. Selon le lemme 3.5.1, $Queue(i)$ n'est pas l'indice d'une feuille impaire. Nous avons 2 cas :

(1) $Queue(i)$ **n'est pas l'indice d'une feuille paire.**

(1.1) $Queue(i)+1$ est l'indice d'une feuille impaire.
Par définition de la suite régulière :
$v_{Queue(i)} = v_{Queue(i)+2}$
$v_{Queue(i)+3}$ est adjacent à $v_{Queue(i)+2}$
Donc, $v_{Queue(i)+3}$ est adjacent à $v_{Queue(i)}$
Par l'algorithme, $Queue(i)+3 = Queue(i+1)$
Alors $v_{Queue(i+1)}$ et $v_{Queue(i)}$ sont adjacents.

(1.2) $Queue(i)+1$ n'est pas l'indice d'une feuille impaire.
Selon l'algorithme :
$Queue(i+1) = Queue(i)+1$
Alors $v_{Queue(i+1)}$ et $v_{Queue(i)}$ sont adjacents.

(2) $Queue(i)$ est l'indice d'une feuille paire.

Si $Queue(i)$ est l'indice d'une feuille, alors $Queue(i)+1$ n'est pas l'indice d'une feuille puisqu'un segment est de longueur supérieure à 1. Selon l'algorithme :
$Queue(i+1) = Queue(i)+1$
Alors $v_{Queue(i+1)}$ et $v_{Queue(i)}$ sont adjacents.

□

Lemme 3.5.4. *$v_{Tete(i+1)}$ est adjacent à $v_{Tete(i)}$.*

Démonstration. Selon le lemme 3.5.1, $Tete(i)$ n'est pas l'indice d'une feuille paire. Nous avons 2 cas :

(1) $Tete(i)$ n'est pas l'indice d'une feuille impaire.

(1.1) $Tete(i)+1$ est l'indice d'une feuille paire.
Par définition de la suite régulière :
$v_{Tete(i)} = v_{Tete(i)+2}$
$v_{Tete(i)+3}$ est adjacent à $v_{Tete(i)+2}$
Donc, $v_{Tete(i)+3}$ est adjacent à $v_{Tete(i)}$
Par l'algorithme, $Tete(i)+3 = Tete(i+1)$
Alors $v_{Tete(i+1)}$ et $v_{Tete(i)}$ sont adjacents.

(1.2) $Tete(i)+1$ n'est pas l'indice d'une feuille paire.
Selon l'algorithme :
$Tete(i+1) = Tete(i)+1$
Alors $v_{Tete(i+1)}$ et $v_{Tete(i)}$ sont adjacents.

(2) Si $Tete(i)$ est l'indice d'une feuille impaire.

Si $Tete(i)$ est l'indice d'une feuille, alors $Tete(i)+1$ n'est pas l'indice d'une feuille puisqu'un segment est de longueur supérieure à 1. Selon l'algorithme :
$Tete(i+1) = Tete(i)+1$
Alors $v_{Tete(i+1)}$ et $v_{Tete(i)}$ sont adjacents.

□

Lemme 3.5.5. *$v_{Queue(i)}$ et $v_{Tete(i)}$ sont adjacents, pour $i \geq 1$.*

Démonstration. À la ronde 1, on a $Tete(i) = 1$ et $Queue(i) = 0$ donc le lemme est satisfait.
Supposons $i > 1$. Selon le lemme 3.5.1, si $Tete(i)$ est dans un segment impair, alors $Queue(i) = Tete(i)-1$ et si $Tete(i)$ est dans un segment pair, alors $Queue(i) = Tete(i)+1$. Dans les deux cas, $v_{Queue(i)}$ et $v_{Queue(i)}$ sont adjacents. □

Lemme 3.5.6. *Tous les nœuds de l'arbre T sont visités.*

Démonstration. Selon l'algorithme, l'ensemble d'agents de t visite toutes les feuilles impaires de T et l'ensemble d'agents de q visite toutes les feuilles paires donc toutes les feuilles sont visitées. L'exploration de toutes les feuilles implique l'exploration de tous les nœuds de T. □

3.5.4 Le temps d'exécution

Nous définissons maintenant l'arbre $\overline{T}$:
L'arbre $\overline{T}$ est un arbre qui résulte de l'arbre T en coupant les feuilles paires $\{f_2, f_4, ...\}$ ainsi que les arêtes adjacentes à ces feuilles (voir Figure 3.10).

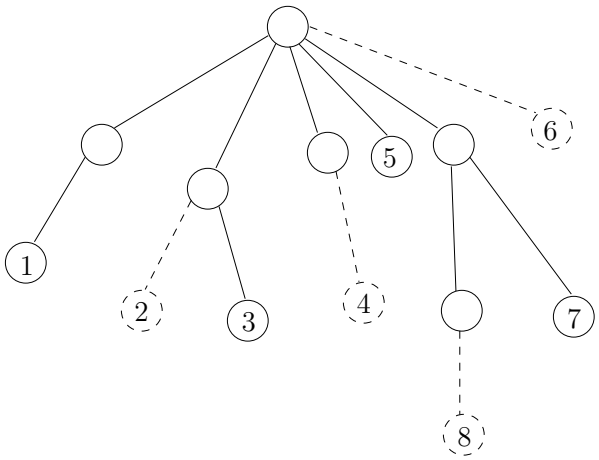

FIGURE 3.10 – L'arbre T et $\overline{T}$ lorsque $d = 1$

La figure 3.10 représente l'arbre T et $\overline{T}$.

L'exploration optimale d'un arbre par un agent est $2(n-1) - h$ (voir section 3.2.3) où n est le nombre de nœuds de l'arbre et h, la hauteur de l'arbre. Le temps d'exécution de l'algorithme ExploreEssaim1 est égal au temps du parcours de la tête du ver dans l'arbre $\overline{T}$.

Cas 1 - Le nombre m de feuilles de l'arbre T est pair. La dernière feuille visitée par l'algorithme ExploreEssaim1 est paire. Toutes les feuilles paires sont visitées par la queue du ver. Le temps d'exploration est :

$$2(n' - 1) - (h - 1)$$

où n' est le nombre de nœuds de $\overline{T}$ et h, la hauteur de T.

$$m = 2b$$

$$n' = n - b$$

$$2(n' - 1) - (h - 1) = 2(n - b - 1) - (h - 1)$$

$$= 2n - 2b - 2 - h + 1$$

$$= 2n - m - h - 1$$

Cas 2 - Le nombre m de feuilles de l'arbre T est impair, $m = 2b + 1$. La dernière feuille visitée par l'algorithme ExploreEssaim1 est impaire. Toutes les feuilles impaires sont visitées par la tête du ver. Le temps d'exploration est :

$$2(n' - 1) - h'$$

où n' est le nombre de nœuds de $\overline{T}$ et h', la hauteur de $\overline{T}$.

$$m = 2b + 1$$

$$n' = n - b$$

$$h' = h$$

$$\begin{aligned} 2(n' - 1) - h' &= 2(n - b - 1) - h \\ &= 2n - 2b - 2 - h \\ &= 2n - m - h - 1 \end{aligned}$$

*Remarque : le coût du déploiement du ver n'est pas compté puisqu'il se forme en même temps que le déplacement des agents à la ronde 1, lorsque le premier nœud est visité et c'est la tête qui le visite.

3.5.5 La preuve de l'optimalité

Prenons un algorithme optimal $\mathcal{A}$ qui doit respecter la borne maximale $d \leq 1$. Il visite au plus deux nœuds adjacents simultanément pour respecter la condition $d \leq 1$.

Lemme 3.5.7. *Un algorithme optimal ne visite pas deux feuilles aux numéros de visite consécutifs (par rapport à cet algorithme) par le même ensemble d'agents.*

Démonstration. (Par contradiction) Prenons n'importe quel algorithme $\mathcal{A}$ optimal. Cet algorithme doit visiter toutes les feuilles. Chaque feuille doit être visitée pour la première fois dans une autre ronde car deux feuilles sont à distance au moins deux. Ordonnons les feuilles dans l'ordre de leur première visite. Considérons deux feuilles consécutives z et z'. Supposons que z et z' sont visités par le même (groupe d') agent. Soit $y \geq 2$ la distance entre les feuilles. Le nombre de rondes entre les deux visites doit être au moins

y. Considérons le groupe d'agents A qui visite la feuille z. À ce moment, sans perte de généralité, le groupe B est dans l'unique nœud adjacent à A. Modifions la partie de l'algorithme $\mathcal{A}$ entre les premières visites de z et z' en faisant visiter la feuille z' par le groupe B, $(y-1)$ rondes après la première visite de z par A. À ce moment, tous les agents de A sont dans son unique voisin. Le reste de l'algorithme $\mathcal{A}$ ne change pas, sauf que les rôles des groupes A et B sont échangés, l'algorithme modifié prend un temps strictement plus petit que l'algorithme $\mathcal{A}$, ce qui est une contradiction. □

Lemme 3.5.8. *Pour visiter T, un algorithme optimal doit avoir le coût d'au moins $2n-m-h-1$, où n est le nombre de nœuds, m le nombre de feuilles et h est la hauteur de T.*

Démonstration. Un algorithme optimal $\mathcal{A}$ ne peut visiter deux feuilles aux numéros de visite consécutifs avec le même groupe d'agent (Lemme 3.5.7). Pour que tous les nœuds de T soient visités, toutes les feuilles de T doivent être visitées. L'algorithme $\mathcal{A}$ doit donc visiter les feuilles de T de façon alternée avec deux groupes d'agents. Appelons tête le groupe d'agents qui visite toutes les feuilles impaires et queue le groupe qui visite les feuilles paires (dans l'ordre de visite de l'algorithme $\mathcal{A}$). Soit T^* l'arbre résultant de couper les feuilles paires et leurs arêtes adjacentes. Pour que toutes les feuilles impaires de T soient visitées par la tête, toutes les arêtes de T^* doivent être traversées au moins deux fois par la tête (une fois dans chaque sens) sauf la dernière branche dont les arêtes doivent être traversées au moins une fois.

Cas 1 -
$m = 2b$
La dernière feuille sera visitée par la queue (car le nombre de feuille est pair). Le coût du parcours de la tête dans T^* sera au moins $2n - m - h - 1$.

Cas 2 -
$m = 2b + 1$
La dernière feuille visitée est impaire et elle sera visitée par le tête. Le coût du parcours de la tête dans T^* sera au moins $2n - m - h - 1$.

□

Puisque le coût de ExploreEssaim1 est égal au coût minimal d'exploration d'un arbre avec un essaim d'agents mobiles lorsque $d = 1$ (Lemme 3.5.8), nous avons le corollaire suivant :

Corollaire 3.5.9. *L'algorithme ExploreEssaim1 est optimal.*

3.6 Exploration optimale d'un arbre par un essaim d'agents mobiles pour une borne impaire $d = 2\rho+1$ où $\rho > 1$

Dans cette section, nous présentons un algorithme qui explore un arbre enraciné avec un essaim d'agents mobiles. Cet algorithme est optimal pour le temps d'exécution. La contrainte est qu'à chaque ronde, il ne peut y avoir une distance supérieure à $2\rho+1$ (où $\rho > 1$) entre les agents les plus éloignés les uns des autres. Nous présentons quelques définitions, décrivons l'algorithme, faisons la preuve de l'exactitude, calculons le temps d'exécution et faisons la preuve de l'optimalité.

3.7 Définitions

Algorithme avec noyau double : Un algorithme avec noyau double est un algorithme qui fonctionne de la façon suivante :
Pour chaque ronde $i \leq \rho + 1$, l'ensemble des nœuds occupés par les agents est $\Gamma_i(r) \cup \Gamma_i(v_1)$ où r est la racine et v_1 le deuxième nœud de la suite régulière (voir section 3.5.1). Pour chaque ronde $i > \rho + 1$, il existe un ensemble de deux nœuds adjacents appelé le noyau double $\{t, q\}$ tel que l'ensemble des nœuds occupés par les agents est $\Gamma_\rho(t) \cup \Gamma_\rho(q)$.

3.7.1 L'algorithme

Si $D \leq d$ alors pendant les ronde $i = 1, ..., h$, les agents se déplacent pour occuper les nœuds dans $\Gamma_i(r)$. Après h rondes, tous les nœuds sont explorés. Si $D > d$, l'algorithme consiste de deux phases : la formation de l'essaim et l'exploration. La première phase dure $\rho+1$ rondes. Soit $s = v_1$ le second nœud de la suite régulière dans l'exécution de la procédure ExplorerEssaim1 dans l'arbre T'. Pendant la première ronde de la première phase, un groupe d'agents reste dans r et l'autre se déplace à s. Pendant les rondes $1+i$, pour $i = 1, ..., \rho$, les agents se déplacent pour occuper tous les nœuds de $\Gamma_i(r) \cup \Gamma_i(s)$. Après la ronde $\rho + 1$, tous les nœuds de $\Gamma_\rho(r) \cup \Gamma_\rho(s)$ sont occupés, l'essaim est formé et la phase d'exploration débute.

Soit $\{u_{\rho+1}, v_{\rho+1}\}$, où $u_{\rho+1} = r$ et $v_{\rho+1} = s$ le noyau double à la ronde $\rho+1$. Pendant la phase d'exploration, le noyau double exécute l'algorithme ExploreEssaim1 dans l'arbre T'. Plus précisément, pendant la ronde $\rho+i$, le noyau double est composé de deux nœuds adjacents ; les mêmes qu'occupe le ver dans l'algorithme ExploreEssaim1 pendant la ronde i lors de l'exploration de l'arbre T'. Il reste à définir les mouvements des autres agents. Supposons que pendant la ronde k le noyau double est $\{u, v\}$ et pendant la ronde $k+1$ il est $\{v, w\}$. (Notons que dans l'algorithme ExploreEssaim1 les ensembles des nœuds occupés par les agents pendant les rondes consécutives ont l'intersection 1.)

Soit $R = \{z : dist(z, u) = \rho \wedge dist(z, v) = \rho + 1\}$ et $F = \{z : dist(z, v) = \rho \wedge dist(z, w) = \rho - 1\}$.

Les agents qui étaient dans $z \in R$ à la ronde k se déplacent à la ronde $k+1$ dans l'unique voisin de z, z', tel que $dist(z', v) = \rho$. Les agents qui étaient dans $z \in F$ à la ronde k occupent $\Gamma(z)$ à la ronde $k+1$. Tous les autres agents ne bougent pas. L'algorithme procède jusqu'au moment où l'algorithme ExploreEssaim1 dans T' soit complété.

3.7.2 La preuve de l'exactitude

Théorème 3.7.1. *L'algorithme ExploreEssaimImpair est correct.*

Démonstration. L'algorithme ExploreEssaimImpair appelle la procédure ExploreEssaim1 pour T'. Tous les nœuds de l'arbre T' sont visités par le noyau double de l'essaim. Toutes les feuilles de T' sont donc visitées par les agents du noyau double. Puisque toutes les feuilles de T sont à une distance plus petite ou égale à ρ de T', toutes les feuilles de T sont visitées par les agents de l'essaim qui se trouvent dans les nœuds à une distance plus petite ou égale à ρ du noyau double. Donc, tous les nœuds de T sont visités. □

3.7.3 Le temps d'exécution

Soit D, le diamètre de l'arbre T. Si $D \leq d$, le temps d'exécution est h. Supposons que $D > d$, alors le temps d'exécution de l'algorithme ExploreEssaimImpair peut être

décomposé en deux parties : Le temps pour la formation de l'essaim et le temps d'exploration. Le premier est $\rho + 1$. Le deuxième est égal au temps d'exploration de l'arbre T' par un ver.

Notons que l'intersection de l'intervalle du temps de la formation de l'essaim et du temps de l'exploration de T est exactement une ronde (à la fin de la formation de l'essaim, le ver est déjà formé). Puisque le temps de l'exploration de T' par un ver est $2n' - m' - h' - 1$ où n' est le nombre de nœud de T', m' est le nombre de feuilles de T' et h', la hauteur de T', le temps total est :

$$\rho + 2n' - m' - h' - 1$$

3.7.4 L'optimalité

Lemme 3.7.2. *L'algorithme ExploreEssaimImpair est optimal.*

Démonstration. Si $D \leq d$, l'optimalité est évidente car aucun algorithme ne peut explorer l'arbre T en temps plus petit que h et le temps de l'algorithme ExploreEssaimImpair est h dans ce cas. Si $D > d$, comme pour le cas d pair, on prouve d'abord que pour chaque algorithme d'exploration A par un essaim avec d impair, il existe un algorithme A' avec noyau double pour le même d qui a le même temps d'exécution que A. La construction de A' à partir de A est semblable à celle pour d pair. En utilisant le lemme 3.5.8, on prouve que A' doit avoir un temps d'au moins $\rho + 2n' - m' - h' - 1$. □

3.8 Le nombre d'agents suffisant pour explorer un arbre de façon optimale

Dans cette section, nous allons prouver que pour chaque algorithme optimal d'exploration d'un arbre par un essaim avec une borne d, il existe un algorithme avec la même borne et le même temps qui utilise λ agents, où λ est le nombre de feuilles de l'arbre.

Soit $\{f_1, ..., f_\lambda\}$ l'ensemble des feuilles et soit $\{a_1, ..., a_\lambda\}$ l'ensemble des agents. Soit A un algorithme optimal et S_i l'ensemble des nœuds occupés par les agents exécutants A à la i-ème ronde. On construit un algorithme A' qui utilisent les agents $\{a_1, ..., a_\lambda\}$ comme suit : À la ronde i, l'agent a_j est dans l'unique nœud $v(i, j)$ de S_i le plus proche de f_j. C'est possible car $v(i, j)$ et $v(i+1, j)$ sont soit égaux soit adjacents. Donc tous les nœuds seront visités car la feuille f_j est visitée par l'agent a_j. Puisque, par définition, l'ensemble S_i, pour chaque i, a le même diamètre au plus d, l'algorithme A' respecte la borne d. Le temps de A et de A' est le même. Donc l'algorithme A' utilisant λ agents travaille aussi en temps optimal.

3.9 Documentation du logiciel de simulation

Dans cette section nous décrivons comment fonctionne le logiciel de simulation. Ce logiciel est une application Windows qui a été conçu avec Microsoft Visual Studio et écrit en $C\sharp$. La simulation est une unique fenêtre où se déroulent toutes les activités incluant la création d'un arbre, la sélection des paramètres et le parcours de l'arbre par un essaim. La fenêtre comprend 10 sections (illustrées par la figure 3.11) :

(1) Une barre de message qui donne des instructions au sujet de la création manuelle d'un arbre ou du choix de la grandeur de l'essaim.

(2) L'état de l'exploration qui donne de l'information au sujet de l'étape du parcours de l'arbre (la création de l'arbre, la création de l'essaim et le parcours par l'essaim).

(3) L'arbre au milieu de la fenêtre où se déroule l'animation.

(4) Trois boutons qui créent un arbre instantanément si on ne souhaite pas en créer un manuellement. Il y a trois choix de hauteur d'arbre (3, 4 et 5). Il y a la possibilité d'ajouter des nœuds à ces arbres déjà faits. En cliquant sur un de ses boutons, un arbre apparaîtra dans la section 3.

(5) La vitesse de parcours : il est possible de choisir la vitesse de parcours en sélectionnant une des options (Lente, Moyenne ou Rapide)

(6) Borne de l'essaim : c'est dans cette boîte de texte que nous entrons la borne (d) de l'essaim (par défaut la borne est à 0 ce qui équivaut à un seul agent).

(7) Pause : ce bouton sert à arrêter la simulation pour un temps indéterminé. Lorsque l'animation est en arrêt, le bouton indique « Continuer ». Il faut cliquer à nouveau sur ce bouton pour poursuivre l'animation.

(8) Effacer : ce bouton sert à effacer l'arbre (ne laissant qu'un nœud, la racine).

(9) Explorer : ce bouton sert à démarrer l'exploration de l'arbre par l'essaim.

(10) Étape par étape : ce bouton sert à démarrer l'exploration de l'arbre par l'essaim mais de façon étape-par-étape. À chaque ronde, il faut cliquer sur le bouton pour faire progresser le parcours de l'essaim.

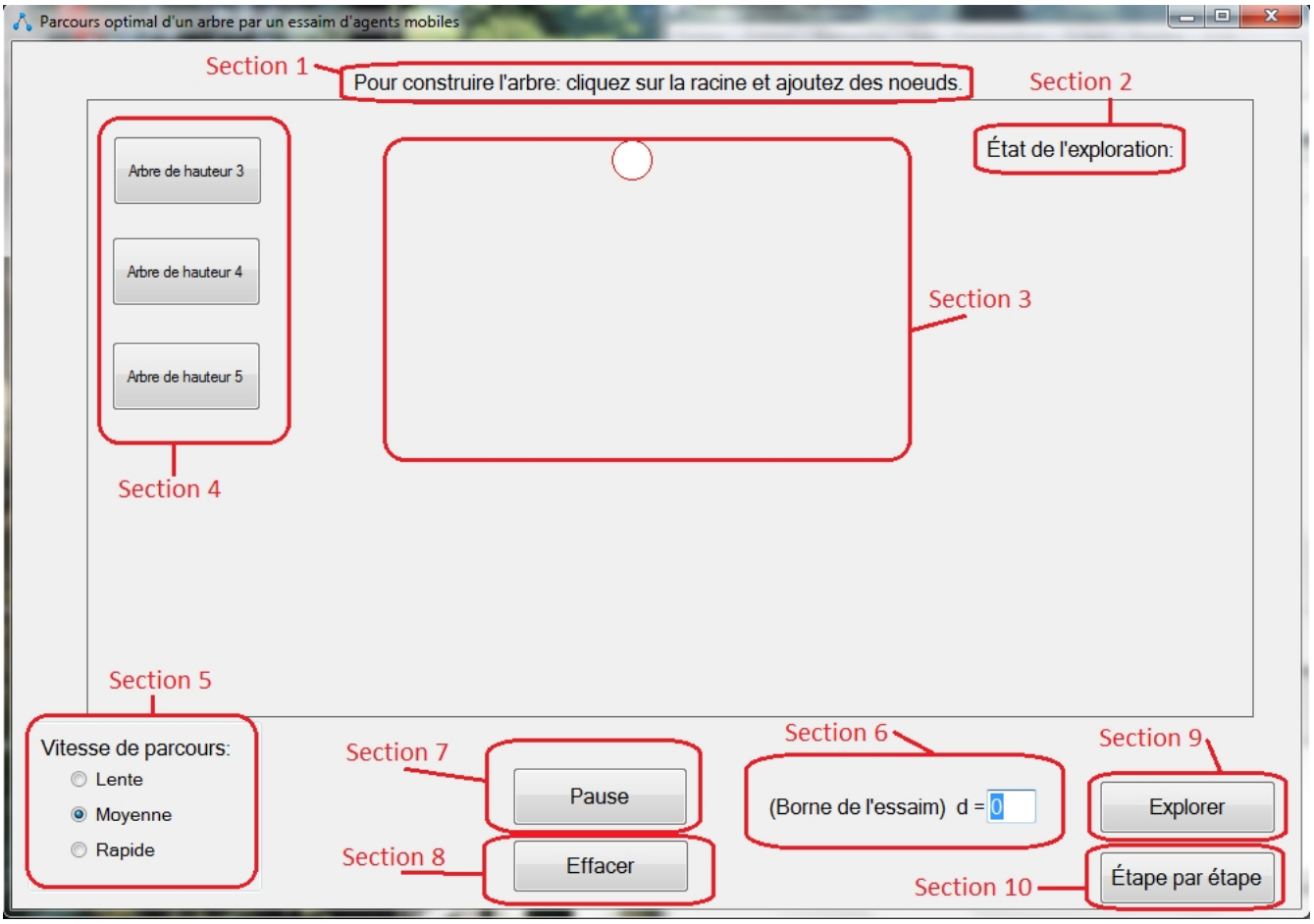

FIGURE 3.11 – Les différentes sections du logiciel de simulation

La figure 3.12 illustre le logiciel à une certaine étape de l'exploration d'un arbre de hauteur 5 par un essaim ayant une borne d égale à 5.

Les étapes de la simulation :

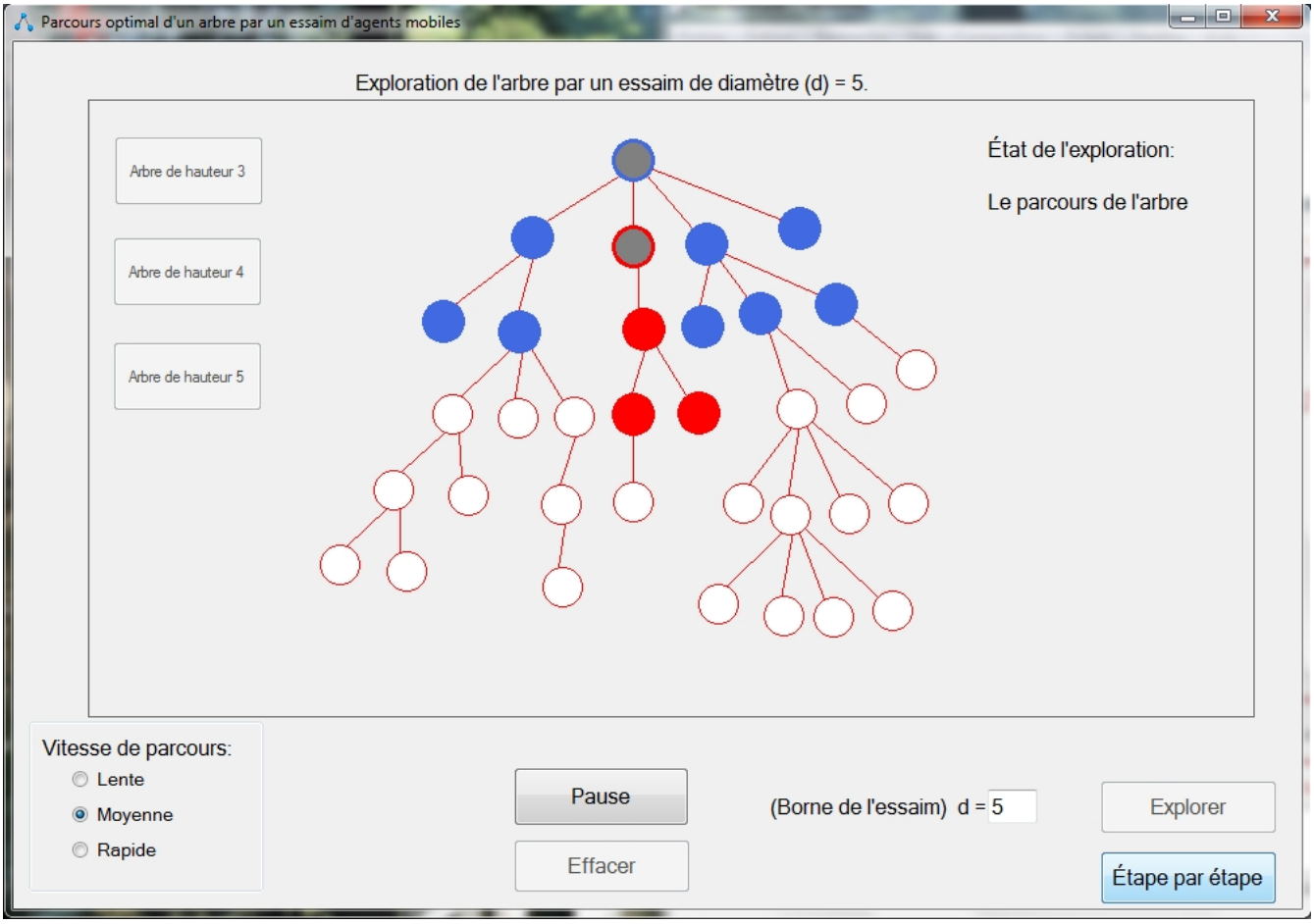

FIGURE 3.12 – Le logiciel de simulation

(1) Créer l'arbre soit en cliquant sur les nœuds pour en ajouter d'autres ou en cliquant sur un des boutons de la section 4 pour ajouter un arbre tout fait.

(2) Choisir la borne maximale (d) de la grandeur de l'essaim.

(3) Choisir la vitesse de parcours (section 5).

(4) Démarrer l'exploration en cliquant sur « Explorer » ou sur « Étape par étape ».

Chapitre 4

Conclusion

Nous avons trouvé des algorithmes d'exploration d'arbres enracinés avec un essaim d'agents mobiles pour les bornes d pair et d impair et nous avons prouvé qu'ils sont optimaux. Nous avons trouvé une borne supérieure pour le nombre d'agents requis pour parcourir un arbre de façon optimale. Il s'avère qu'il suffit d'utiliser autant d'agents qu'il y a de feuilles dans l'arbre. Nous avons aussi conçu un logiciel de simulation pour visualiser l'exploration d'un arbre par un essaim d'agents.

Dans un premier temps, nous nous sommes limité aux arbres connus d'avance par les agents. Il reste plusieurs questions ouvertes dont celles-ci : (1) Trouver le nombre minimal d'agents dans un essaim de borne d pour parcourir un arbre donné de façon optimal. (2) Trouver des algorithmes optimaux du parcours des graphes par des essaims pour d'autres topologies de réseaux que les arbres. (3) Le même scénario mais pour des graphes inconnus.

Bibliographie

[1] Albers, S., and Henzinger, M. R. Exploring unknown environments. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing* (1997), STOC '97.

[2] Ando, H., Oasa, Y., Suzuki, I., and Yamashita, M. Distributed memoryless point convergence algorithm for mobile robots with limited visibility. *Robotics and Automation, IEEE Transactions on 15*, 5 (1999), 818–828.

[3] Awerbuch, B., Betke, M., Rivest, R. L., and Singh, M. Piecemeal graph exploration by a mobile robot. In *Proceedings of the Eighth Annual Conference on Computational Learning Theory* (1995), COLT '95, ACM, pp. 321–328.

[4] Bender, M. A., Fernández, A., Ron, D., Sahai, A., and Vadhan, S. The power of a pebble : Exploring and mapping directed graphs. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing* (1998), ACM, pp. 269–278.

[5] Bender, M. A., and Slonim, D. K. The power of team exploration : Two robots can learn unlabeled directed graphs. In *In Proceedings of the Thirty Fifth Annual Symposium on Foundations of Computer Science* (1994), pp. 75–85.

[6] Berman, P., Blum, A., Fiat, A., Karloff, H., Rosen, A., and Saks, M. Randomized robot navigation algorithms. In *Proceedings of the seventh annual ACM-SIAM symposium on Discrete algorithms* (1996), SODA '96, Society for Industrial and Applied Mathematics, pp. 75–84.

[7] Blum, A., Raghavan, P., and Schieber, B. Navigating in unfamiliar geometric terrain. In *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing* (1991), STOC '91, ACM, pp. 494–504.

[8] Blum, M., and Kozen, D. On the power of the compass (or, why mazes are easier to search than graphs). In *Proceedings of the 19th Annual Symposium on Foundations of Computer Science* (1978), IEEE Computer Society, pp. 132–142.

[9] Borselius, N. Mobile agent security. *Electronics and Communication Engineering Journal 14*, 5 (2002), 211–218.

[10] Cieliebak, M., Flocchini, P., Prencipe, G., and Santoro, N. Solving the robots gathering problem. In *Proceedings of the 30th international conference on Automata, languages and programming* (2003), ICALP'03, Springer-Verlag, pp. 1181–1196.

[11] Cieliebak, M., and Prencipe, G. Gathering autonomous mobile robots. In *SIROCCO* (2002), pp. 57–72.

[12] Czyzowicz, J., Kowalski, D., Markou, E., and Pelc, A. Complexity of searching for a black hole. *Fundamenta Informaticae 71* (2006), 229–242.

[13] Czyzowicz, J., Kowalski, D., Markou, E., and Pelc, A. Searching for a black hole in synchronous tree networks. *Comb. Probab. Comput. 16* (2007), 595–619.

[14] Deng, X., Kameda, T., and Papadimitriou, C. How to learn an unknown environment i : The rectilinear case. *Journal of the ACM 45* (1997), 215–245.

[15] Deng, X., and Papadimitriou, C. H. Exploring an unknown graph. *Journal of Graph Theory 32* (1990), 265–297.

[16] Dessmark, A., and Pelc, A. Optimal graph exploration without good maps. *Theor. Comput. Sci. 326* (2004), 343–362.

[17] Diks, K., Fraigniaud, P., Kranakis, E., and Pelc, A. Tree exploration with little memory. In *Journal of Algorithms* (2004), vol. 51, Academic Press, Inc., pp. 38–63.

[18] Dudek, G., Jenkin, M., Milios, E., and Wilkes, D. Robotic exploration as graph construction. *Robotics and Automation, IEEE Transactions on 7*, 6 (1991), 859 –865.

[19] Duncan, C. A., Kobourov, S. G., and Kumar, V. S. A. Optimal constrained graph exploration. In *Proceedings 12th Annual ACM-SIAM Symposium on Discrete Algorithms* (2001), pp. 807–814.

[20] Fleischer, R., and Trippen, G. Exploring an unknown graph efficiently. In *In Proceedings 13th Annual European Symposium on Algorithms* (2005), Springer-Verlag, pp. 11–22.

[21] Flocchini, P., Ilcinkas, D., Pelc, A., and Santoro, N. Computing without communicating : Ring exploration by asynchronous oblivious robots. In *Proceedings of the 11th international conference on Principles of distributed systems* (2007), OPODIS'07, Springer-Verlag, pp. 105–118.

[22] Flocchini, P., Ilcinkas, D., Pelc, A., and Santoro, N. Remembering without memory : Tree exploration by asynchronous oblivious robots. In *Proceedings of the 15th international colloquium on Structural Information and Communication Complexity* (2008), SIROCCO '08, Springer-Verlag, pp. 33–47.

[23] Flocchini, P., Prencipe, G., Santoro, N., and Widmayer, P. Gathering of asynchronous robots with limited visibility. *Theoritical Computer Science 337* (2005), 147–168.

[24] Fraigniaud, P., Gasieniec, L., Kowalski, D. R., and Pelc, A. Collective tree exploration. *Networks 48* (2006), 166–177.

[25] Fraigniaud, P., and Ilcinkas, D. Digraphs exploration with little memory. In *STACS* (2004), pp. 246–257.

[26] Fraigniaud, P., Ilcinkas, D., and Pelc, A. Tree exploration with an oracle. 24–37.

[27] Gasieniec, L., Pelc, A., Radzik, T., and Zhang, X. Tree exploration with logarithmic memory. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (2007), SODA '07, Society for Industrial and Applied Mathematics, pp. 585–594.

[28] Hsiang, T.-R., Arkin, E., Bender, M., Fekete, S., and Mitchell, J. Algorithms for rapidly dispersing robot swarms in unknown environments. In *Algorithmic Foundations of Robotics V*, vol. 7 of *Springer Tracts in Advanced Robotics*. Springer Berlin / Heidelberg, 2004, pp. 77–94.

[29] Markou, E., and Pelc, A. Efficient exploration of faulty trees. *Theory of Computing Systems 40* (2007), 225–247.

[30] Nageswara, R. S. V., Srikumar, K., Weimin, S., and Sitharama, I. S. Robot navigation in unknown terrains : Introductory survey of non-heuristic algorithms, 1993.

[31] Prencipe, G. Instantaneous actions vs. full asynchronicity : Controlling and coordinating a set of autonomous mobile robots. In *Proceedings of the 7th Italian Conference on Theoretical Computer Science* (2001), ICTCS '01, Springer-Verlag, pp. 154–171.

[32] Prencipe, G. Impossibility of gathering by a set of autonomous mobile robots. *Theoretical Computer Science 384* (2007), 222–231.

[33] Roy, N., and Dudek, G. Collaborative robot exploration and rendezvous : Algorithms, performance bounds and observations. *Autonomous Robots 11* (2001), 117–136.

[34] Sander, T., and Tschudin, C. F. Protecting mobile agents against malicious hosts. In *Mobile Agents and Security* (1998), Springer-Verlag, pp. 44–60.

[35] Suzuki, I., and Yamashita, M. Distributed anonymous mobile robots : Formation of geometric patterns. *SIAM J. Comput. 28* (1999), 1347–1363.

[36] Tinkham, A., and Menezes, R. Simulating robot collective behavior using starlogo. In *Proceedings of the 42nd Annual Southeast Regional Conference* (2004), ACM-SE 42, ACM, pp. 396–401.