UNIVERSITÉ DU QUÉBEC EN OUTAOUAIS

EXPLORATION DE GRAPHES HAMILTONIENS EN PRÉSENCE DE PANNES DE LIENS

MÉMOIRE
PRÉSENTÉ
COMME EXIGENCE PARTIELLE DE LA
MAÎTRISE EN SCIENCES ET TECHNOLOGIES DE L'INFORMATION

PAR

DAVID CAISSY

NOVEMBRE 2015

UNIVERSITÉ DU QUÉBEC EN OUTAOUAIS

Département d'informatique et d'ingénierie

Ce mémoire intitulé :

EXPLORATION DE GRAPHES HAMILTONIENS EN PRÉSENCE DE PANNES DE LIENS

présenté par
David Caissy

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. Mohand Said Allili ..... Président du jury
Dr. Jurek Czyzowicz ..... Membre du jury

Mémoire accepté le : 25 novembre 2015

À mes deux filles, Charlotte et Zoé, qui m'inspirent à me dépasser constamment. Leurs petits encouragements ont eu un énorme impact sur ma détermination à continuer et à ne pas abandonner. Merci les filles !

Remerciements

Je voudrais remercier mon directeur de recherche, le professeur Andrzej Pelc pour avoir été la personne la plus marquante de tout mon cheminement universitaire. En plus d'avoir été un excellent professeur, il a su me motiver et m'encourager quand ce fût nécessaire. Il restera toujours un modèle pour moi et c'est avec énormément de respect que je lui dis merci.

Je profite aussi de l'occasion pour remercier messieurs Mohand Said Allili et Jurek Czyzowicz pour leurs commentaires pertinents sur mon travail. Ils ont sans nul doute contribué à l'amélioration de la qualité de ce mémoire.

Je suis aussi très reconnaissant envers la chaire de recherche en calcul distribué de l'Université du Québec en Outaouais pour l'obtention de la bourse de maîtrise CALDI. Cette bourse m'a été d'une grande utilité pour mener à bien ma recherche dans ce domaine fascinant.

Finalement, un gros merci à Zoé, Charlotte, Andrée, Philippe, mes parents, mon frère et tous ceux qui m'ont encouragés et supportés dans cette aventure.

Table des matières

Liste des figures

Résumé

Nous considérons le problème de l'exploration des réseaux possédant des pannes de liens. Un agent mobile, qui se situe initialement sur le noeud de départ et qui ne connaît pas l'emplacement des pannes, doit explorer la partie sans panne et connexe du réseau en visitant tous ses noeuds. Le *coût* de cette exploration est le nombre de traversées de liens.

Pour un réseau et un noeud de départ donné, le *surplus* d'un algorithme d'exploration est le ratio en pire cas (parmi toutes les configurations de pannes possibles) de son coût sur le coût d'un algorithme optimal qui lui, connaîtrait l'emplacement des pannes. Un algorithme d'exploration est appelé *parfaitement compétitif* si son surplus est le plus petit parmi tous les algorithmes qui ne connaissent pas la configuration des pannes.

Nous avons conçu un algorithme d'exploration parfaitement compétitif pour tous les anneaux et nous avons démontré que, pour les réseaux modélisés par des graphes hamiltoniens, le surplus de n'importe quel algorithme d'exploration de type fouille en profondeur est au plus 10/9 supérieur à celui d'un algorithme parfaitement compétitif. En outre, pour les graphes hamiltoniens de grandeur 24 ou plus, ce surplus est moins de 6% supérieur à celui d'un algorithme parfaitement compétitif.

Mots clés : algorithme, pannes de liens, exploration, agent mobile

Abstract

We consider the problem of exploration of networks, some of whose edges are faulty. A mobile agent, situated at a starting node and unaware of which edges are faulty, has to explore the connected fault-free component of this node by visiting all of its nodes. The *cost* of the exploration is the number of edge traversals.

For a given network and given starting node, the *overhead* of an exploration algorithm is the worst-case ratio (taken over all fault configurations) of its cost to the cost of an optimal algorithm which knows where faults are situated. An exploration algorithm is called *perfectly competitive* if its overhead is the smallest among all exploration algorithms not knowing the location of faults.

We design a perfectly competitive exploration algorithm for any ring, and show that, for networks modeled by hamiltonian graphs, the overhead of any DFS exploration is at most $10/9$ times larger than that of a perfectly competitive algorithm. Moreover, for hamiltonian graphs of size at least 24, this overhead is less than 6% larger than that of a perfectly competitive algorithm.

Keywords : algorithm, faulty edge, exploration, mobile agent

Chapitre 1

Introduction

La visite de tous les noeuds pendant l'exploration de réseaux est l'une des tâches de base effectuées par les agents mobiles. De façon pratique, un agent logiciel peut avoir à récupérer des données situées dans chaque noeud d'un réseau, et un robot mobile peut avoir à prélever des échantillons de terre dans une mine contaminée où les corridors forment un réseau et où les intersections peuvent être représentées par des noeuds. Les graphes nous permettent de modéliser ce genre d'environnements et ils facilitent la définition des problèmes à étudier.

Il est courant que certains liens dans un réseau puissent tomber en panne, empêchant ainsi un agent mobile de les traverser. Dans un réseau informatique, une panne peut s'avérer être un câble défectueux tandis que dans une mine, il peut s'agir d'un corridor obstrué. Dans ces deux situations, il est possible que l'agent ne soit pas en mesure d'atteindre certains noeuds du réseau. La tâche sera donc d'explorer la partie accessible de la manière la plus efficace possible, en traversant le moins de liens possible.

L'agent mobile connaît le réseau ainsi que son point de départ, mais il ne connaît ni le nombre, ni l'emplacement des pannes. L'objectif est de visiter tous les noeuds du graphe en traversant le moins de liens possible. Le coût de l'exploration est exprimé par le nombre de traversées de liens.

Pour analyser la performance d'un algorithme par rapport à un autre, il est nécessaire d'avoir une base commune pour évaluer leurs performances. Ici, nous utilisons un algorithme dit *optimal* comme base pour la comparaison. Cet algorithme clairvoyant connaît

l'emplacement des pannes avant le début de son exploration. Cette connaissance supplémentaire lui permet de bien planifier sa route pour ainsi effectuer un nombre minimal de traversées de liens. Nous appellerons *surplus* le ratio maximum (en pire cas) entre le coût d'exploration d'un algorithme et le coût d'exploration de l'algorithme optimal, le maximum étant pris pour toutes les configurations de pannes possibles pour ce graphe.

Si le surplus d'un algorithme s'avère être le plus petit parmis tous les algorithmes qui ne connaissent pas l'emplacement des pannes, nous dirons que cet algorithme est *parfaitement compétitif*, c'est-à-dire qu'aucun autre algorithme ayant la même information au départ ne peut faire mieux que lui.

Notre but ici est de trouver des algorithmes efficaces pour les graphes hamiltoniens. L'objectif est de minimiser leur surplus et si possible, trouver des algorithmes parfaitement compétitifs. Nous avons conçu un algorithme d'exploration parfaitement compétitif pour tous les anneaux et nous avons démontré que, pour les réseaux modélisés par des graphes hamiltoniens, le surplus de n'importe quel algorithme d'exploration de type fouille en profondeur est au plus 10/9 supérieur à celui d'un algorithme parfaitement compétitif. En outre, pour les graphes hamiltoniens de grandeur 24 ou plus, ce surplus est moins de 6% supérieur à celui d'un algorithme parfaitement compétitif.

Pour ce faire, nous avons structuré ce document de la manière suivante :

- Le chapitre 1 présente une introduction du problème d'exploration de graphes en présence de pannes de liens.
- Le chapitre 2 comporte une revue de la littérature et quelques exemples des travaux qui ont été réalisés sur l'exploration de graphes en général.
- Le chapitre 3 présente le modèle en détail, donne un résumé des résultats de la recherche et présente la méthodologie
- Le chapitre 4 porte sur un algorithme parfaitement compétitif pour tous les anneaux.
- Le chapitre 5 porte sur la performance des algorithmes de la classe $\mathcal{DFS}$ dans les graphes hamiltoniens.
- Ce document se termine par la bibliographie utilisée pour réaliser le chapitre 2.

Chapitre 2

Revue de la littérature

2.1 Introduction

Le problème de l'exploration d'un environnement inconnu par un agent mobile a été grandement étudié dans la littérature. Cet environnement a été modélisé de deux façons distinctes : soit par un terrain géométrique dans un plan, soit comme dans notre cas, par un graphe. De plus, un grand nombre de paramètres peuvent être utilisés pour définir chaque scénario, ce qui permet aux auteurs de bien cibler leur problématique. C'est paramètres, que nous allons présenter tout au long de cette revue de la littérature, permettent de catégoriser la recherche dans ce domaine.

Tout d'abord, nous parlerons de l'exploration et de la navigation dans les terrains géométriques contenant des obstacles de formes diverses inconnus de l'agent. Le but peut être, par exemple, de se déplacer à un endroit précis ou encore, d'établir une carte de cet environnement. Le robot (agent mobile) peut avoir différents types de senseurs comme le toucher ou la vision, et les algorithmes d'exploration proposés peuvent être déterministes ou aléatoires (probabilistes).

En deuxième lieu, nous verrons ce qui a été fait dans la littérature sur l'exploration d'environnement modélisé par un graphe, où l'agent doit suivre les arêtes. Ces graphes peuvent être orientés et fortement connexes ou comme dans notre cas, ils peuvent être non-orientés. Ils peuvent être explorés par un seul ou par plusieurs agents, qui peuvent eux-mêmes avoir une ou plusieurs restrictions, comme une mémoire limitée ou un réservoir d'essence qu'ils doivent remplir à leur base après un certain nombre de traversées

de liens. Si plusieurs agents sont utilisés, le problème peut être la communication entre ces agents pour le partage de l'information.

L'exploration de graphes anonymes où les noeuds ne sont pas marqués a aussi été beaucoup étudiée. Que ce soit en utilisant des jetons ou en marquant les noeuds, l'exploration peut se faire encore une fois avec un seul agent ou en équipe et de façon déterministe ou aléatoire.

L'information que possèdent les agents au départ influence grandement les algorithmes proposés. Dans certains cas, l'agent peut posséder une carte du graphe où le noeud de départ peut être marqué ou non. Si l'agent ne possède pas de carte, alors peut-être connaît-il le nombre de noeud n, le diamètre ou la topologie du graphe ?

Ensuite, nous allons faire un résumé de plusieurs articles portant sur l'exploration de graphes tolérante aux pannes. Les pannes peuvent affecter les liens ou les noeuds et peuvent être permanentes ou intermittentes. De plus, certains noeuds appelés *trous noirs* peuvent être dangereux et détruire chaque agent qui les visite.

Dans la section suivante, nous effecturons un bref survol de quelques articles portant sur l'exploration de graphes dynamiques, c'est-à-dire des graphes dans lequels le nombre de noeuds et de liens changent dans le temps.

Dans cette revue de la littérature, nous nous concentrerons sur l'exploration de graphes non-orientés ayant possiblement des pannes de liens. Nous présenterons un article concernant ce scénario où les auteurs ont découvert un algorithme parfaitement compétitif pour la ligne et ont proposé un algorithme efficace pour les arbres.

2.2 L'exploration de terrains géométriques dans un plan

L'exploration de terrains géométriques dans un plan est très utile pour différentes raisons, comme par exemple pour tracer une carte d'un terrain ou pour localiser des obstacles. Ici, le terrain est une représentation géométrique d'un environnement quelconque, qui pourrait être une pièce, une ville ou même la surface d'une planète où un robot doit se déplacer.

Bar-Eli et al. [3] abordent le *problème de la pièce*, qui consiste à voyager d'un des coins d'une pièce carrée $n \times n$ vers le centre de celle-ci. Cette pièce est parsemée d'obstacles représentés par des rectangles alignés et disjoints. Au départ, le robot connaît n ainsi que la coordonnée de sa cible $(\frac{n}{2}, \frac{n}{2})$, mais il ne connaît pas l'emplacement des obstacles. Si sa cible se trouve à l'intérieur d'un obstacle, il doit en faire le tour complet pour prouver que sa coordonnée se trouve bel et bien à l'intérieur. De plus, le robot acquiert seulement de l'information de façon *tactile*, c'est-à-dire que ce n'est qu'en touchant un obstacle qu'il va pouvoir en déterminer la position.

L'efficacité de leur algorithme est mesurée par un *ratio de compétitivité*, soit le rapport entre le chemin le plus long parcouru par le robot pour toutes les configurations d'obstacles possible et le chemin le plus court à la cible. Les auteurs proposent un algorithme déterministe très efficace permettant de trouver un chemin de longueur $O(n \log n)$. Ils démontrent en outre que ce résultat est en fait la borne inférieure pour tous les algorithmes déterministes, même ceux utilisant l'information visuelle.

Blum et al. [6] ont ensuite proposé une version légèrement différente du même problème. L'élément inconnu ici est encore une fois le nombre et l'emplacement des rectangles alignés et disjoints (les obstacles), mais cette fois, le robot possède le sens de la vision. Dès qu'un obstacle entre dans son champ de vision, le robot connaît immédiatement sa forme, sa taille et son emplacement. De plus, le ratio de compétitivité est maintenant calculé par rapport au chemin le plus court si aucun obstacle n'était présent.

Les auteurs ont d'abord proposé un algorithme pour résoudre le *problème du mur*, où le robot doit se déplacer d'un point s vers un des murs de la pièce en contournant les obstacles. Ensuite, ils ont résolu le problème de la pièce avec les paramètres énumérés ci-haut. En s'appuyant sur les connaissances acquises lors de l'analyse de ces deux problèmes, ils ont ensuite abordé le problème où le robot doit maintenant se déplacer de *point à point*, c'est-à-dire d'un point de départ s donné (et non un coin de la pièce comme précédemment) vers une cible t pouvant elle aussi se trouver n'importe où dans la pièce. Dans ce scénario, pour se déplacer point à point dans la pièce, l'algorithme proposé a le ratio compétitif de $O(\sqrt{n})$.

Outre la navigation, un autre problème intéressant de l'exploration d'environnements géométriques est la création de cartes. Il peut être en effet souhaitable de tracer une carte d'un environnement inconnu pour être ensuite en mesure de le parcourir efficacement. Deng et al. [11] se sont penchés sur ce problème qu'ils ont appelé la *visite de la galerie*. Un agent mobile doit entrer dans une pièce à un endroit donné, être en contact visuel avec toutes la surface de la pièce avant d'en ressortir par un point déterminé au départ. L'environnement est encore une fois une pièce contenant un nombre fini de rectangles alignés avec ses rebords. Le robot connaît les dimensions de la pièce, son point d'entrée et son point de sortie, mais il ne connaît ni le nombre, ni l'emplacement des obstacles.

La performance de l'algorithme est calculée par le ratio en pire cas entre la distance que le robot doit parcourir pour créer la carte et la distance optimale requise pour vérifier cette carte, si le robot en possédait une copie avant de commencer son exploration. Ce ratio est compliqué par le fait que le calcul du trajet optimal pour vérifier la carte est un problème NP-complet, par réduction du problème du *commis-voyageur*. Les auteurs nous présentent un algorithme glouton pouvant tracer la carte d'une pièce de forme polygonale ayant une borne supérieure sur le nombre d'obstacles qu'elle contient.

L'exploration de terrains géométriques est un vaste champ d'étude qui en plus de s'adresser aux situations énumérées ci-dessus, couvre une grande quantité de problèmes, notamment des obstacles de formes différentes aussi que des environnements 3D.

2.3 L'exploration de graphes

Les graphes nous permettent de modéliser toutes sortes de problèmes rencontrés dans la vie de tous les jours. En effet, il est pratique de pouvoir représenter les rues d'une ville ayant des sens uniques par un graphe orienté fortement connexe (comme dans le cas d'applications de navigation par GPS) ou encore, de pouvoir schématiser un réseau d'aqueduc par un graphe non-orienté pour être en mesure de localiser les bris de conduites rapidement dans le futur. Dans ces deux cas, le fait de pouvoir réduire un environnement donné à un graphe nous permet de développer des algorithmes pouvant plus tard s'appliquer à d'autres problèmes. Par exemple, les rues de la ville pourrait aussi bien être les couloirs d'un bâtiment et les algorithmes pour les conduites d'aqueduc pourraient s'appliquer à un réseau électrique. Dans tous les cas, la recherche d'algorithmes d'exploration de graphes trouve un vaste champ d'application dans la vie courante.

Tout au long de cette section, nous dénoterons par $G = (V, E)$ un graphe G contenant un ensemble de noeuds $V = \{v_1, \ldots, v_n\}$ et un ensemble de liens E, où chaque lien peut être représenté par un couple non-orienté de noeuds $\{v_i, v_j\}$ dans le cas de graphes non-orientés et un couple orienté de noeuds $\{v_i, v_j\}$ dans le cas des graphes orientés. En règle générale, un agent mobile va devoir soit traverser tous les liens (et du fait même visiter tous les noeuds) ou visiter chaque noeud sans avoir à passer par tous les liens. Il est à noter que nous utiliserons "graphe" et "réseau" de façon interchangeable dans cette revue de la littérature.

Dans cette section, nous allons nous concentrer sur l'exploration de graphes en présence de contraintes. Dans un premier temps, nous allons aborder l'exploration de graphes effectuée par un seul agent. Par la suite, nous allons nous intéresser aux explorations effectuées par plusieurs agents, où nous verrons comment la communication entre ces agents influence les algorithmes. Finalement, nous nous pencherons sur la recherche de trous noirs avant de terminer avec l'exploration de graphes dangereux et la tolérance aux pannes.

2.3.1 L'exploration de graphes par un seul agent

Il existe une multitude de paramètres entourant l'exploration de graphes par un seul agent [4, 12, 18, 30, 14, 25, 29, 13, 7]. Les chercheurs dans ce domaine tentent de résoudre des problèmes reliés au manque de connaissance des agents mobiles, allant de l'exploration de graphes inconnus à la découverte de pannes. De plus, les noeuds du graphe peuvent être *marqués* (chacun ayant un identificateur unique) ou le graphe peut être *anonyme* (les noeuds n'ont pas d'étiquettes). Voyons d'abord quelques exemples d'algorithme d'exploration de graphes par un agent mobile où les noeuds sont marqués.

Graphes avec noeuds marqués

Deng et al. [12] se sont penchés sur l'exploration de graphes orientés et fortement connexes où le but de l'agent est de parcourir tous les liens. La performance de l'algorithme est mesurée par le ratio entre le nombre de liens traversés et le nombre de liens minimal requis, si l'agent était en possession d'une carte du graphe avant de débuter son exploration. Les auteurs ont découvert que pour les graphes eulériens (graphes où il est possible de parcourir tous les liens exactement une seule fois), ce ratio ne peut être meilleur que 2. De plus, lorsque le graphe n'est pas eulérien et qu'il lui manque d liens pour qu'il en soit un, le ratio de l'algorithme n'est pas borné lorsque d est lui-même sans borne supérieure. Par contre, lorsque d est borné, les auteurs proposent un algorithme pour $d = 1$ où aucun lien n'est visité plus de 4 fois. Le ratio pour $d > 1$ atteint 8 en pire cas. Cet article nous montre bien que l'exploration d'un graphe inconnu, orienté et fortement connexe peut être très coûteux si celui-ci n'est pas un graphe eulérien.

Un autre exemple où le manque de connaissance de l'agent a été mis en perspective est dans un article publié par Dessmark et al. [13]. Ici, un agent doit encore une fois visiter tous les noeuds et parcourir tous les liens d'un graphe inconnu, mais cette fois, il s'agit d'un graphe non-orienté. La performance de l'algorithme est mesurée par son *coût* (nombre de traversées de liens) divisé par le nombre minimal de traversées de liens requis, si l'agent avait une connaissance totale du graphe avant le début de son exploration. Dans le but de mesurer l'impact du manque de connaissance de l'agent, les

auteurs ont considéré trois scénarios : 1) l'agent ne connaît rien du graphe ; 2) l'agent possède une carte, mais le noeud de départ n'est pas marqué et 3) l'agent possède une carte avec le noeud de départ.

Pour les graphes quelconques, il s'avère que la *fouille en profondeur* (DFS) est un algorithme optimal dans tous ces scénarios. Par contre, dans le cas restreint des arbres, la situation est bien différente. Bien que DFS soit optimal dans le scénario 1 où aucune information n'est disponible, la situation est différente lorsqu'une carte est disponible. En effet, si cette carte n'est pas *ancrée* (le noeud de départ n'est pas marqué), le ratio est au moins $\sqrt{3}$ mais strictement moins de 2, ce qui fait que DFS n'est pas optimal. Dans le cas d'une carte où le noeud de départ est marqué, le ratio pour les arbres est de $3/2$.

De façon similaire, dans le cas des graphes de la classe des lignes, DFS est encore une fois optimal dans le scénario 1 et un algorithme ayant un ratio de $\sqrt{3}$ pour le scénario 2 a aussi été proposé. Mais cette fois pour le scénario 3, le ratio de l'algorithme proposé est maintenant de $7/5$. Dans ces deux scénarios, les ratios ci-dessus sont optimaux. On voit bien la pénalité encourue par un algorithme dû à son manque de connaissance dans ces scénarios.

L'impact du manque de connaissance a aussi été le centre de la problématique soulevée par Fraigniaud et al. [25]. Leur étude a porté sur l'impact qu'a l'information disponible sur l'exploration des graphes de la classe des arbres. En particulier, des éléments comme la taille du réseau, son diamètre, la disponibilité d'une carte détaillée ou l'absence total d'information ont été analysés. Le ratio compétitif est calculé de la même façon que pour les deux exemples précédents et tel que démontré plus tôt, dans le cas où aucune information n'est disponible, le ratio de 2 du DFS ne peut être amélioré.

Le problème est donc de déterminer la quantité d'information minimale requise pour permettre un ratio strictement inférieur à 2. Il s'avère que pour un arbre de diamètre D, cette quantité minimale d'information est d'environ $\log \log D$. Cette étude nous montre qu'une petite quantité de données peut avoir une influence significative sur les algorithmes d'exploration.

Un autre problème d'exploration de graphes non-orientés et inconnus consiste à utiliser des agents mobiles (robots) ne possédant pas une liberté totale de mouvement pour parcourir tous les liens du graphe. Duncan et al. [18] ont soulevé ce problème avec des robots soit possédant un réservoir d'essence leur permettant de traverser un certain nombre de liens, auquel cas ils doivent revenir périodiquement à leur point de départ pour refaire le plein, soit attachés à leur point de départ par une corde de longueur l (ce cas étant encore plus contraignant). Ici, l'efficacité de l'algorithme est mesurée par le nombre de traversées de liens pendant l'exploration. Avec leur algorithme *Closest First Exploration* (CFX) qui fonctionne dans les deux scénarios (réservoir d'essence et corde), les auteurs prouvent qu'il est possible d'explorer le graphe en effectuant $\Theta(|E|)$ traversées de liens. Mais comme la borne inférieure est de $\Omega(|E|)$, leur algorithme est optimal à un facteur constant près. Finalement, il est à noter que la *fouille en largueur* (BFS) ainsi que DFS ne peuvent pas résoudre le problème dans ces conditions.

Graphes anonymes

Maintenant, étudions le cas des graphes anonymes, c'est-à-dire des graphes dont les noeuds n'ont pas d'étiquette. Du point de vu de l'agent, tous les noeuds de même degré sont identiques, à l'exception de l'ordre de numérotation des ports, qui elle est arbitraire. Mais comme l'analyse de la performance des algorithmes est généralement faite en pire cas, assumons ici que les ports de tous les noeuds sont numérotés dans le même ordre, les rendant ainsi indissociables pour l'agent. On suppose aussi qu'il est impossible pour un agent de marquer même un seul noeud. Dans ce cas, l'agent ne pourra pas savoir quand arrêter son exploration. Par exemple, dans le cas d'un graphe non-orienté anonyme représenté par un anneau de n noeuds et que l'agent ne connaît pas n, il est facile de voir que l'agent ne saura jamais quand terminer son exploration (figure 2.1).

Il est donc impératif que l'agent puisse soit marquer au moins un noeud, soit qu'il possède assez d'information pour accomplir sa tâche. Panaite et al. [29] ont proposé un algorithme pour explorer les graphes anonymes et non-orientés où il est possible pour l'agent de marquer les noeuds. Le but de l'exploration est encore une fois de parcourir tous les liens et visiter tous les noeuds en effectuant le moins de traversées de liens

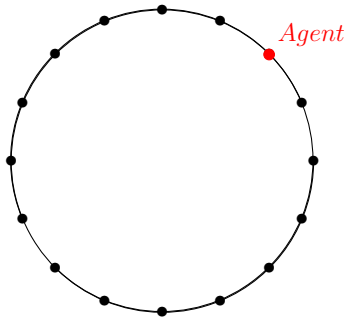

FIGURE 2.1 – Anneau anonyme

possibles. La performance de l'algorithme est alors comparée en pire cas au nombre minimal de traversées de liens requis pour un algorithme possédant une carte détaillée du graphe avant le début de son exploration, la borne inférieure étant de $|E|$ pour les graphes eulériens où les points de départ et d'arrivée sont marqués.

Ces auteurs ont réussi à trouver un algorithme ayant une pénalité linéaire de $O(|V|)$ pour tous les graphes, c'est-à-dire un coût total de $|E| + O(|V|)$. Ils ont aussi démontré que DFS n'est pas optimal car il ne prend pas assez en compte le sous-graphe exploré tandis que les algorithmes gloutons (greedy) ont quant à eux une vision trop locale pour être efficaces. Leur algorithme utilise trois couleurs pour marquer les noeuds lors de l'exploration. En marquant chaque noeud d'une couleur précise (qui peut être amenée à changer plusieurs fois durant l'exploration), l'agent s'assure de garder un *pont* entre sa position courante et le noeud de départ. De plus, ce système de couleurs est important pour marquer les noeuds saturés (où tous ses liens incidents ont déjà été explorés). À l'aide de cette technique de marquage, les auteurs ont réussi à trouver un algorithme ayant une pénalité d'au plus $3|V|$ traversées de liens par rapport à un algorithme clairvoyant qui connaîtrait tous les détails du graphe avant le début de son exploration.

Dans les cas où il est impossible de marquer tous les noeuds, une autre façon d'explorer les graphes anonymes est en utilisant un ou plusieurs *jetons*, qui peuvent être stationnaires ou mobile. Chalopin et al. [7] ont construit un algorithme d'exploration de graphes anonymes quelconques en utilisant un seul *jeton stationnaire*. Ils ont réussi à résoudre ce problème pour les graphes de n noeuds et de degré maximal d avec un coût de $O(n^3 d)$.

Pelc et al. [30] ont développé un algorithme d'exploration de grilles non-orientées et anonymes en utilisant encore une fois un seul jeton stationnaire situé au point de départ. L'agent doit visiter tous les noeuds et traverser tous les liens d'une grille $m \times k$, $1 < m \leq k$ en effectuant le moins de traversées de liens possible. L'agent sait qu'il se trouve dans une grille, mais il n'en connaît pas les dimensions. Les auteurs ont démontré qu'en utilisant les rebords de la grille, il est possible de l'explorer dans un temps quadratique de $O(n^2)$, une amélioration du $O(n^3)$ pour les graphes quelconques.

Pour ce faire, l'algorithme d'exploration commence par classifier la grille pour savoir si elle est de grandeur $2 \times k$ (une échelle) ou de grandeur $m \times k$, $2 < m \leq k$. Si la grille est une échelle, l'agent va ensuite parcourir la grille en suivant un algorithme assez simple pour terminer l'exploration dans un temps de $O(k^2)$. Mais si la grille n'est pas une échelle, alors l'agent va d'abord explorer les alentours du noeud de départ v en suivant un chemin en forme de croix jusqu'à ce qu'il rencontre l'une des bordures de la grille avec un noeud de degré 3 (les quatre noeuds de degré 2 n'étant pas voisin des noeuds de degré 4). De là, l'agent va suivre le bord de la grille et après le troisième coin rencontré (troisième noeud de degré 2), il connaît maintenant les dimensions de la grille. À ce moment, l'agent possède assez d'information pour explorer le reste de la grille dans un temps inférieur à l'algorithme pour les graphes anonymes quelconques [7].

Maintenant, nous allons aborder l'utilisation de *jetons mobiles* dans des graphes orientés, anonymes et fortement connexes. Un agent a pour tâche de cartographier un graphe dans un nombre polynomial (dans la taille du graphe) de traversées de liens. Le but ici est de minimiser le nombre de jetons requis pour accomplir cette exploration. Comparé aux exemples précédent, l'agent peut maintenant déplacer le jeton et le laisser dans un autre noeud. Plus tard, en revenant à un noeud contenant un jeton, il peut soit le laisser sur place, soit le ramasser pour l'utiliser ailleurs.

Bender et al. [4] ont proposé un algorithme ne requérant qu'un seul jeton lorsqu'une borne supérieure sur le nombre n de noeuds est connue. Mais dans le cas où aucune borne supérieure n'est donnée sur le nombre de noeuds, leur algorithme déterministe ne requiert que $\Theta(\log \log n)$ jetons pour cartographier le graphe. De plus, les auteurs montrent dans leur article que les *marches aléatoires* (une succession de déplacements

aléatoires) ne sont pas efficaces dans les graphes orientés car le temps requis pour visiter tous les noeuds peut s'avérer être exponentiel dans n (mais ce temps est polynomial dans les graphes non-orientés). Il n'est donc pas nécessaire d'avoir beaucoup de jetons pour explorer les graphes anonymes.

Mais qu'arrive-t-il si plusieurs jetons identiques et fixes sont situés dans le graphe dans des noeuds différents ? Il peut s'avérer difficile pour un agent mobile de décider si un jeton qu'il vient de rencontrer a déjà été visité ou non. En effet, comme rien ne les différencie, comment peut-il savoir s'il ne visite pas ce noeud pour la première fois ? De plus, si on ajoute que certains de ces jetons sont de type *byzantins* (où ils peuvent être visible ou non à l'agent, la visibilité étant choisi par un adversaire), la tâche semble impossible à réaliser. Il s'avère cependant que sous certaines conditions, il est possible de résoudre ce problème.

Dieudonné et Pelc ont publier un article sur ce sujet [14] où ils ont élaboré un algorithme d'exploration de graphes quelconques non-orientés et anonymes. Pour débuter, les auteurs nous ont montré que si aucune borne supérieure sur le nombre de jetons n'est connue par l'agent, alors l'exploration est impossible par un algorithme déterministe et ce, même si tous les jetons sont fonctionnels (donc pas byzantins). L'exploration est également impossible si tous les jetons sont byzantins, même si leur nombre est connu. Mais dans le cas où une borne supérieure sur le nombre de jetons est connue et qu'au moins un de ces jetons est fonctionnel, alors il est possible d'effectuer l'exploration de manière déterministe dans un temps polynomial dans la taille du graphe (même si celle-ci est inconnue).

2.3.2 L'exploration de graphes par plusieurs agents

Dans cette section, nous allons voir comment plusieurs agents mobiles peuvent collaborer pour explorer un graphe. Nous aborderons la communication entre les agents et l'impact de l'absence de celle-ci. Ensuite, nous verrons comment plusieurs agents peuvent venir à bout de tâches impossibles à accomplir pour un seul agent, comme la recherche de trous noirs et l'exploration de graphes dangereux.

Pour débuter, voyons comment plusieurs agents identiques peuvent explorer un graphe en visitant tous ses noeuds sans aucune communication. Flocchini et al. se sont attaqués à ce problème pour la ligne [20] et pour l'anneau [19], tous deux étant des classes de graphes anonymes et non-orientés. Les agents partent de noeuds différents et ils effectuent leur exploration de façon *asynchrone*, c'est-à-dire que la durée de chaque opération (regarder, calculer et se déplacer) est finie mais non-bornée, et cette durée est déterminée par l'adversaire. Les agents ne se rappellent pas des observations passées, donc chaque cycle de calcul est uniquement basé sur l'opération *regarder* du cycle courant.

Pendant l'opération *regarder*, le robot voit tout le graphe : il voit quels noeuds sont vides, quels noeuds sont occupés et quels noeuds sont des tours (où plus d'un robot se trouvent sur ce noeud). Basé sur cette vue d'ensemble, le robot va décider de façon déterministe pendant l'opération *calcul* s'il reste stationnaire ou s'il se déplace vers un noeud adjacent. Pendant l'opération *déplacer*, le robot applique cette décision. Les déplacements sont instantanés et donc pendant l'opération *regarder*, le robot voit les autres robots sur des noeuds, jamais sur des liens.

Le but est de déterminer dans un premier temps si l'exploration est possible et si c'est le cas, quel est le nombre k de robots nécessaires pour accomplir cette tâche, pour $k < n$. Pour la ligne, il s'avère que l'exploration par k robots est possible si et seulement si $k = 3$, $k = 4$ et n est impair ou $k \geq 5$. Pour ce qui est de l'anneau, il faut que k et n soient relativement premiers. Ensuite, les auteurs nous donnent un algorithme capable de toujours terminer avec $k \geq 17$ robots. Finalement, ils prouvent que $O(\log n)$ robots sont nécessaires pour explorer un anneau de taille suffisament grande.

Maintenant que nous avons vu qu'il est possible sous certaines conditions d'explorer des graphes avec plusieurs agents sans aucune communication, voyons l'effet de celle-ci sur l'exploration des graphes de la classe des arbres. Dans [24], Fraigniaud et al. ont comparé deux scénarios : 1) les agents peuvent communiquer entre eux en écrivant des messages dans les noeuds visités (noeuds de type "whiteboard") et 2) les agents sont inconscients des autres, ils connaissent seulement l'information qu'ils ont acquis pendant leur propre exploration.

Un ensemble de k robots se trouvent initialement sur le même noeud de départ et ils doivent explorer un arbre inconnu avant de revenir à ce point de départ. Le graphe est exploré si chaque lien a été traversé par au moins un agent. La performance de l'algorithme est comparée à celle d'un algorithme optimal qui lui, possèderait une carte de l'arbre avant le début de son exploration. Notons que la planification du trajet optimal parcouru par plusieurs agents est un problème NP-complet.

Dans le premier scénario où la communication est possible entre les agents, l'algorithme proposé effectue l'exploration de n'importe quels arbres avec un surplus de $O(\frac{k}{\log k})$. À l'opposé, dans le scénario 2 où la communication n'est pas permise entre les agents, le surplus de tous les algorithmes doit être d'au moins $\Omega(k)$. Il est donc clair que même lorsque la communication est limitée entre les agents (écrire et lire des messages aux noeuds vs la communication radio par exemple), elle s'avère un avantage indéniable lors de l'exploration.

Regardons maintenant comment deux agents communiquant entre eux par radio peuvent collaborer pour explorer un graphe orienté, fortement connexe et anonyme. Nous avons vu précédemment comment un seul agent peut accomplir cette tâche en employant un ou plusieurs jetons, selon certaines conditions. Mais dans le cas où nous ne pouvons pas utiliser de jetons ou autres moyens de marquage, au moins deux agents sont nécessaires pour compléter l'exploration. Bender et al. [5] ont publié deux algorithmes pouvant résoudre ce problème. Les agents débutent l'exploration au même noeud de départ et ils communiquent entre eux par radio. Le fait qu'ils puissent constamment s'envoyer des messages permet la synchronisation entre ces agents, ils peuvent en effet "s'attendre" et décider ensemble de la prochaine étape de l'exploration.

Le premier algorithme utilise des *séquences de retour en arrière* (homing sequences), qui peuvent être définies par un agent qui reste sur place pendant que l'autre explore les alentours. Lorsque cette sous-section du graphe a été explorée, les deux agents se déplacent vers un endroit non-exploré du graphe pour recommencer cette séquence. Cette façon de faire déterministe utilise dans les faits un des agents pour jouer le rôle de "jeton", marquant ainsi temporairement un noeud pendant que l'autre explore le graphe. En utilisant cet algorithme, il est possible d'explorer n'importe quel graphe dans un temps polynomial dans n.

Le second algorithme présenté par Bender et Slonim utilise les marches aléatoires. En utilisant l'approche probabiliste, ils ont démontré qu'il est possible sous certaines conditions de faire mieux que leur algorithme d'exploration déterministe et ce, avec une forte probabilité de succès. Cette exploration s'effectue encore une fois dans en temps polynomial dans n.

2.3.3 La recherche de trous noirs

Plusieurs chercheurs se sont penchés sur des problèmes portant sur la détection de trous noirs dans les réseaux [10, 9, 23, 2, 17, 15]. Un *trou noir* est un processus stationnaire hautement néfaste qui réside dans un noeud et qui détruit tous les agents qui le visite sans laisser de trace. Comme les agents ne peuvent pas empêcher d'être détruit lorsqu'il visite un trou noir, la seule façon qu'ils ont de se protéger est d'identifier ces noeuds hostiles et d'éviter de les visiter par la suite. Comme le seul moyen de savoir si un noeud est un trou noir est de le visiter, il en découle que la recherche de trous noirs est un problème qui requiert plusieurs agents.

Dans la réalité, les trous noirs peuvent représenter des bugs dans les logiciels ou des problèmes de composantes matérielles (hardware). De plus, ils peuvent servir à modéliser des attaques informatiques sur un réseau, par exemple un virus qui détruirait tout message arrivant sur le serveur infecté. La recherche de trous noirs peut donc être utilisée pour localiser des pannes matérielles ou pour isoler un serveur infecté dans un réseau.

Czyzowicz et al. [10] ont publié un article portant sur la recherche d'un trou noir dans un arbre par deux agents. Les agents ici sont considérés comme "synchronisés" : il existe une borne supérieure (inconnue mais finie) sur le temps requis pour la traverse d'un lien. Le but est de trouver le trou noir (s'il existe) le plus rapidement possible en ayant au moins un agent qui survit à la tâche, et celui-ci doit revenir au point de départ pour terminer sa recherche. Le temps pour localiser le trou noir est le nombre de traversées de liens par les deux agents. Finalement, les agents débutent l'exploration sur le même

noeud de départ, connu pour être sûr, et ils peuvent seulement communiquer entre eux lorsqu'ils se rencontrent.

Comme il existe une borne supérieure sur le temps pour une traverse de lien (normalisée à 1), les agents peuvent utiliser cette information de la manière suivante : Du point de départ v, un agent se déplace vers l'un de ses voisins et revient immédiatement sur ses pas tandis que l'autre attend sur place. Après 2 unités de temps, si l'agent n'est pas revenu au point de départ, l'agent stationnaire survit et connaît maintenant l'endroit où se trouve le trou noir. Mais si par contre l'agent revient à son point de départ, alors les deux agents savent que ce noeud est sûr et ils peuvent tous deux se déplacer vers ce noeud en toute sécurité.

Les auteurs ont construit un algorithme permettant de résoudre ce problème pour des arbres quelconques avec un ratio de 5/3 par rapport au chemin le plus court. Le pire cas étant lorsqu'il n'y a pas de trou noir dans l'arbre ou lorsque le trou noir est le dernier noeud visité, ces deux cas produisant le même résultat. De plus, ils ont trouvé un algorithme optimal pour deux classes d'arbres spécifiques, soit les lignes et les arbres dans lesquels chaque noeud interne possède au moins 2 enfants.

Ces mêmes auteurs ont abordé un problème similaire où deux agents tentent de découvrir l'emplacement d'un trou noir. Mais cette fois-ci, ils se sont attaqué aux graphes quelconques où un sous-ensemble de noeuds, contenant le point de départ, est connu pour être sûr (parce qu'ils auraient par exemple été vérifié auparavant). Le trou noir, s'il existe, se trouverait donc dans l'un des noeuds restant du graphe. Les auteurs ont prouvé que le problème de calculer le plan de recherche optimal pour deux agents est NP-difficile et ils ont proposé une approximation de facteur 9.3 pour ce problème, travaillant en un temps polynomial. La recherche d'un trou noir dans un graphe quelconque par deux agents est donc un problème difficile à résoudre.

À l'opposé de la classe des graphes arbitraires, il est souvent utile d'explorer des problèmes pour des classes de graphes plus restreintes où les leçons apprises vont plus tard nous aider à régler des problèmes plus complexes dans des classes de graphes plus grandes. C'est ce qu'ont fait Dobrev et al. [17] en se penchant sur la recherche d'un trou noir dans un anneau anonyme. Ici, les auteurs ont abordé 2 scénarios : 1) Les agents

partent tous du même point de départ et 2) Les agents sont dispersés dans l'anneau avant le début de l'exploration. Dans ces deux cas, les agents asynchrones communiquent en se laissant des messages aux noeuds (whiteboard).

Dans le premier scénario, ils ont prouvé que 2 agents sont nécessaires et suffisant pour trouver le trou noir dans un nombre de mouvements optimal de $O(n \log n)$. Comme les agents sont asynchrones, il est nécessaire de borner le temps maximal pour qu'un agent se déplace d'un noeud à un autre, appelé *unité de temps*. Ils ont prouvé qu'avec $n-1$ agents, il est possible d'atteindre la borne inférieure de $2n - 4$ unités de temps. Donc lorsque les agents partent du même point de départ, les auteurs ont découvert des algorithmes optimaux sur le nombre de traversées de liens ainsi que sur le nombre d'unités de temps requis pour trouver l'emplacement du trou noir en pire cas. Finalement, ils ont généralisé leur technique pour produire un algorithme offrant un compromis entre les deux.

Pour ce qui est du deuxième scénario où les agents partent de noeuds différents (choisis par l'adversaire), les auteurs ont dans un premier temps trouvé un algorithme optimal sur le nombre de mouvements pour les anneaux orientés, permettant la détection du trou noir en pire cas avec 2 agents. Mais pour les anneaux non-orientés, l'algorithme optimal requiert cette fois 3 agents pour trouver l'emplacement du trou noir, en pire cas.

Flocchini et Santoro se sont joints plus tard à Balamohan et Miri [2] pour pousser cette réflexion sur la recherche d'un trou noir dans les anneaux et proposer des algorithmes optimaux en moyenne et en pire cas. En fait, du point de vu de la complexité reliée au temps, ils ont trouvé qu'en utilisant seulement 2 unités de temps de plus que ce qui est requis en pire cas, ils ont réussi à résoudre le problème en seulement $\frac{7}{4}n - O(1)$ unités de temps en moyenne. De plus, ils ont prouvé que la complexité optimale de $\frac{3}{2}n - O(1)$ en moyenne peut être atteinte sans augmenter le temps en pire cas en augmentant le nombre d'agents à $2(n-1)$. Finalement, ces auteurs ont développé un algorithme asymptotiquement optimal en moyenne et en pire cas, qui utilise seulement 2 agents, améliorant grandement les algorithmes précédents qui utilisaient plutôt $O(n)$ agents.

Dans la recherche d'un trou noir, les deux paramètres les plus importants dans l'élaboration d'une stratégie sont le nombre d'agents requis et le nombre total de mouve-

ments effectués pas ces agents (le *coût*). Dans les réseaux quelconques, lorsque les agents connaissent la topologie, deux agents peuvent trouver un trou noir en $O(n \log n)$. Mais est-il possible de faire mieux dans certains réseaux spécifiques ? Des chercheurs ont développé un algorithme permettant à deux agents de localiser un trou noir avec $O(n)$ mouvements dans des réseaux communs interconnectés [15], et ce résultat reste valide si le graphe est anonyme. Cette classe de réseaux inclus les hypercubes, les étoiles, les tores et plusieurs autres. Ces réseaux sont en outre bi-connectés : si on supprime n'importe quel noeud, le réseau reste connecté. Comme dans plusieurs autres cas décris plus haut, la communication entre les agents asynchrones s'effectue à l'aide de messages écrits et lus aux noeuds visités et ces agents débutent la recherche à partir du même noeud de départ.

Jusqu'à présent, nous avons vu des algorithmes permettant de trouver l'emplacement d'un seul trou noir. Mais que ce passe-t-il s'il y a plus d'un trou noir dans un réseau ? Flocchini et al. [23] ont publié un article portant sur la recherche de plusieurs trous noirs dans un réseau modélisé par un métro dans un environnement urbain. Le problème est d'autant plus complexe que les agents mobiles doivent "prendre le métro", c'est-à-dire qu'ils doivent littéralement attendre qu'un train passe dans la station de métro (noeud) où ils se trouvent et qu'ils peuvent seulement se déplacer dans la direction du train. Les agents, qui débutent leur recherche du même noeud de départ, peuvent communiquer entre eux en écrivant et en lisant des messages aux stations de métro. Notons aussi que les trains se déplacent de manière asynchrone, comme c'est malheureusement trop souvent le cas dans la vie réelle !

Les auteurs ont débuté par présenter les conditions nécessaires pour que la recherche soit possible. Parmi les plus importantes, notons que le nombre d'agents mobile k doit être supérieur au nombre de trous noirs et que ce nombre k doit aussi être supérieur à y, le nombre d'arrêts de train aux stations contenant un trou noir. Ils ont néanmoins réussi à trouver un algorithme permettant de trouver tous les trous noirs dans un temps quadratique dans le nombre de trains et dans la longueur de la plus longue route. De plus, cet algorithme est optimal dans le nombre d'agents utilisés $(y+1)$ et dans le nombre de mouvements des agents.

Jusqu'à présent, nous avons assumé que les agents pouvaient communiquer entre eux en laissant des messages aux noeuds visités. Dobrev et al. [16] ont découvert un algorithme portant sur la recherche d'un trou noir dans les graphes quelconques anonymes où les agents ont chacun un jeton. Ils ne peuvent pas communiquer directement entre eux, alors aucune communication radio, aucun message laissé dans les noeuds et aucune communication entre eux quand ils sont sur le même noeud n'est permis. Ces jetons identiques peuvent être transportés par les agents qui peuvent les laisser soit dans un noeud, soit sur un lien. Plus tard, l'agent qui repasse par ce lien ou revient à ce noeud peut reprendre son jeton. De plus, les agents partent tous du même noeud de départ, ils se déplacent de manière asynchrone (temps borné mais imprévisible) et ils ne connaissent rien du graphe à l'exception du nombre de noeuds n.

Les auteurs ont réussi à construire un algorithme assez complexe pouvant résoudre ce problème dans un temps fini, où au moins un agent survit et connaît l'emplacement du trou noir. Pour ce faire, $\Delta+1$ agents sont requis, où Δ est le degré maximal parmi tous les noeuds du graphe. Leur résultat indique que, contrairement aux attentes, l'utilisation de jetons est aussi performante du point de vue du calcul que l'utilisation de messages laissés dans les noeuds comme moyen de communication entre les agents (temps polynomial dans les deux cas).

En plus des trous noirs résidants dans les noeuds, il peut être possible d'avoir des liens qui détruisent eux aussi les agents sans laisser de trace. Ces liens néfastes pour les agents sont appelés *liens noirs*. Les graphes contenant à la fois des trous noirs et des liens noirs sont appelés *graphes dangereux*.

Flocchini et al. [22] se sont penchés sur le problème de l'exploration de graphes dangereux et inconnus par des agents asynchrones dispersés. Pour que la recherche de trous noirs et l'exploration de graphes dangereux soit possible, trois conditions doivent être rencontrées : 1) la partie *sécuritaire* du graphe (une fois que tous les liens noirs, les trous noirs et leurs liens incidents ont été retirés) doit être connexe ; 2) le nombre k d'agents doit être supérieur à la taille f de la *frontière* (c'est-à-dire l'ensemble de tous les liens et noeuds sécuritaires menant vers la portion non-sécure du graphe) et 3) Le nombre n_s de noeuds sécuritaires ou la taille de la frontière doit être connu par les agents. Les auteurs assument que ces trois conditions ont été rencontrées, ce qui implique

aussi que lorsqu'un lien tombe en panne durant l'exploration du graphe, il ne brise pas la connectivité de la partie sécuritaire du graphe, sinon le problème est clairement insoluble. Ils ont réussi à résoudre ce problème en $O(nm)$ déplacements d'agents, où m représente le nombre de liens dans le graphe.

Flocchini et al. [21] ont plus tard abordé un problème similaire, mais dans le cas extrême où non seulement le graphe contient plusieurs trous noirs et liens noirs, mais cette fois le graphes est anonyme et certains de ses liens tombent en panne de façon arbitraire pendant l'exploration des agents (pannes dynamiques de liens). Une fois un lien en panne, il ne peut pas redevenir fonctionnel par la suite. De plus, un lien ne peut pas tomber en panne pendant qu'un agent le traverse.

La principale contribution de cet article est la preuve qu'il est possible d'explorer un graphe dangereux en présence de pannes de liens dynamiques. Il est intéressant de noter que cette tâche peut être accomplie en utilisant le même nombre d'agents que dans le cas sans panne dynamique, soit $f + 1$ agents. Le coût de cette exploration est de $O(nm^2)$ déplacements d'agents et comparé aux résultats de l'article précédent, nous pouvons observer que le prix de la tolérance aux pannes est un facteur d'au plus $O(m)$ par rapport au coût optimal atteint lorsqu'il n'y a aucune panne dynamique de lien.

La détection de trous noirs et l'exploration de graphes dangereux est un domaine qui a beaucoup été étudié dans la dernière décennie. Un autre domaine où la publication d'articles a été particulièrement proléfique ces derniéres années est l'exploration de graphes dynamiques.

2.3.4 L'exploration de graphes dynamiques

La recherche portant sur les graphes dynamiques [1, 8, 26, 27] a connu un élan de popularité ces dernières années. L'exploration de graphes dynamiques s'apparente en outre très bien aux réseaux en perpétuel changement, tel que les réseaux de téléphonie cellulaire.

Casteigts et al. [8] ont réalisé une revue de la littérature sur le sujet et ils ont défini un cadre de travail commun basé sur les différents concepts, formalismes et résultats qu'ils ont énumérés dans leur recherche. En effet, plusieurs études utilisaient jusqu'alors des termes différents pour définir des concepts similaires. Ils ont donc proposé un cadre de travail unifié qu'ils ont appelé les *graphes à variance temporelle* ou *time-varying graphs* en anglais.

Les graphes à variance temporelle sont des graphes dynamiques dont la topologie évolue dans le temps. Le degré de changement dans ces graphes est généralement trop important pour qu'il soit modélisé en terme de fautes ou de pannes. Dans ces graphes, ces changements ne sont donc pas considérés comme des anomalies, mais plutôt comme faisant partie intégrante du système. Un exemple serait la communication avec un satellite orbitant autour de la terre, où la communication avec une station terrestre donnée serait interrompue lorsque le satellite disparaîtrait à l'horizon. La communication serait à nouveau possible dès que le satellite aurait fait le tour de la terre et qu'il serait à nouveau visible à partir de la station terrestre.

Alboul et al. [1] ont étudié comment une équipe de robots pouvait explorer un environnement inconnu en utilisant une approche basée sur les graphes dynamiques. Dans ce scénario, les robots sont "parachutés" dans un environnement inconnu. Ils doivent dans un premier temps déterminer leur position pour ensuite explorer le graphe. Ils ne possèdent pas de GPS et n'ont pas non plus de carte. Chaque robot obtient de l'information par ses propres sens (lasers, caméras, etc.) en plus d'en recevoir des autres robots. Au début, l'environnement est représenté par une grille triangulaire virtuelle et infinie, où les noeuds représentent des robots aux ressources limitées par leurs senseurs et les liens sont utilisés pour modéliser la portée de ces senseurs. L'équipe est composée d'au moins trois robots et ceux-ci agissent comme des noeuds du graphe, pouvant être soit statiques, soit dynamiques. Une borne inférieure de trois robots est nécessaire pour plusieurs raisons, notamment pour que les robots puissent se localiser adéquatement les uns par rapport aux autres à l'aide de triangulation.

À mesure que l'exploration progresse, les liens et les noeuds non-nécessaires sont retirés de la grille triangulaire virtuelle et les obstacles apparaissent comme des cycles

dans la grille résultante. Dans cet article, la grille virtuelle est dynamique dans le sens où l'on efface des liens et des noeuds à mesure que l'exploration du graphe se précise.

Ilcinkas et al. [27] ont plus tard publié un article portant sur l'exploration de graphes dynamiques connexes à intervalle T (T-Interval-Connected) par un agent mobile. Un graphe dynamique est connexe à intervalle T si pour chaque fenêtre de T unités de temps consécutifs, $T \geq 1$, il existe un sous-graphe connexe couvrant qui est toujours présent (stable) pendant cette période. Dans cet ouvrage, ce graphe stable est un anneau.

L'agent mobile a pour but de visiter tous les noeuds du graphe et le coût de cette exploration est le nombre de traversées de liens pour y arriver. Les auteurs ont considéré deux scénarios : dans le premier, l'agent connaît entièrement et exactement la dynamique du graphe qu'il doit explorer tandis que dans le deuxième, l'agent ne connaît rien de la dynamique du graphe, à savoir quand les liens apparaissent et disparaissent. De plus, pour le second scénario, les auteurs assument qu'il existe une récurrence-δ, $\delta \geq 1$, tel que chaque lien doit apparaître au moins une fois pour chaque δ unités de temps consécutives.

Il s'avère que l'exploration de graphes dynamiques est beaucoup plus complexe que dans le cas statique. En effet, pour le premier scénario, les auteurs ont trouvé un algorithme effectuant l'exploration en $2n-T-1$ unités de temps. Mais dans le cas où l'agent ne connaît pas la dynamique du graphe, la complexité augmente à $n + \frac{n}{max\{1,T-1\}}(\delta - 1) \pm \Theta(\delta)$ unités de temps.

Un peu plus tard, Ilcinkas et al. [26] ont poussé cette réflexion sur les graphes dynamiques basés sur des anneaux à une classe de graphes plus grande, soit les graphes dynamiques basés sur des cactus. Un *cactus* est un graphe dans lequel tous les cycles simples ont au plus un noeud en commun. En assumant que l'agent connaît la dynamique du graphe, il s'avère que $2^{\theta(\sqrt{\log n})}.n$ unités de temps sont nécessaires et suffisantes pour effectuer l'exploration.

Nous allons maintenant aborder la dernière section de cette revue de la littérature, soit l'exploration de graphes en présence de pannes de liens.

2.3.5 L'exploration de graphes en présence de pannes de liens

Dans cette dernuére section, nous considérons l'exploration par un seul agent de graphes non-orientés où certains liens sont en panne. Markou et al. [28] ont publié un article portant sur l'exploration efficace d'arbres en présence de pannes de liens et cet article est celui qui se rapproche le plus à ce mémoire de recherche.

Le scénario est le suivant : un agent possède une carte du graphe où le noeud de départ est marqué, mais il ne connaît pas l'emplacement des pannes. Son but est de visiter tous les noeuds situés dans la partie sans panne et connexe au noeud de départ v. Le *coût* de cette exploration est le nombre de traversées de lien effectué par l'agent. La performance de l'algorithme est mesurée par le ratio de son coût sur celui d'un algorithme dit *optimal*, qui connaîtrait l'emplacement des pannes. Le maximum de ce ratio pour toutes les configurations de pannes est appelé le *surplus*. Un algorithme est dit *parfaitement compétitif* s'il possède le plus petit surplus possible parmi tous les algorithmes qui ne connaissent pas l'emplacement des pannes.

Les auteurs ont réussi à trouver un algorithme parfaitement compétitif pour la ligne et pour ce qui est des graphes de la classe des arbres, ils ont construit un algorithme ayant un surplus qui est au plus 9/8 plus grand que celui d'un algorithme parfaitement compétitif. Les deux algorithmes ont tous deux un temps de calcul local linéaire dans la taille de l'arbre exploré. Leur principale contribution a été de prouver que les stratégies d'exploration naturelles performent bien dans les arbres en présence de pannes de liens.

Chapitre 3

Le modèle, les résultats de la recherche et la méthodologie

3.1 Modèle

3.1.1 Graphes en présence de pannes de liens

Le réseau est modélisé par un graphe connexe simple non-orienté $G = (V, E)$, appelé *graphe* par la suite. Un agent est initialement situé sur le noeud de départ v du graphe et il possède une carte de ce graphe avec sa position de départ marqué. Chaque noeud est identifié de façon distincte et les *ports* de chaque noeud de degré d sont étiquetés par un nombre entier $1, \ldots, d$. Ainsi pour chaque noeud, l'agent sait quel port conduit vers quel noeud voisin. Toutefois, certains liens du graphe sont en panne.

L'ensemble des liens en panne $P \subseteq E$ est appelé la *configuration des pannes*. L'agent au départ ne connait ni l'emplacement des pannes, ni leur nombre. Lorsqu'il visite un noeud pour la première fois, l'agent découvre, s'il y a lieu, lesquels de ses ports correspondent à des liens incidents en panne. Un lien en panne (défectueux) empêche un agent de le traverser. Le sous-graphe sans panne G', résultant du graphe d'origine $G = (V, E)$ après avoir enlevé tous ses liens défectueux, peut être disconnexe. Soit C la composante connexe du sous-graphe G' contenant le noeud de départ v. Nous nommerons C la composante sans panne correspondant au noeud v et à la configuration des pannes P. La tâche de l'agent est *d'explorer* C en visitant tous ses noeuds. L'exploration se termine dès que le dernier noeud de C a été visité. Pour un graphe donné G, un noeud de départ

v et une configuration des pannes P, le *coût* $\mathcal{C}(A, G, v, P)$ d'un algorithme d'exploration A est le nombre de traversées de liens pendant l'exploration de la composante connexe C contenant v et correspondant à P.

Un agent qui connaîtrait P et connaîtrait du fait même la composante C, aurait un algorithme d'exploration avec le coût le plus petit possible. Appelons cet algorithme optimal $opt(G, v, P)$ et notons que cet algorithme d'exploration optimal, ainsi que son coût, peuvent parfois être difficile à définir. Considérons maintenant un algorithme d'exploration A pour un graphe G donné et un noeud de départ v et supposons que cet algorithme ne connaît pas P, tel que supposé dans notre scénario. Une mesure naturelle de la performance d'un tel algorithme [28], est le pire ratio entre son coût et le coût de l'algorithme optimal $opt(G, v, P)$, où le pire cas est pris parmi toutes les configurations possibles de pannes P. Ce nombre $max_{P \subseteq E} \frac{\mathcal{C}(A,G,v,P)}{opt(G,v,P)}$ est appelé le *surplus* de A, et est dénoté $\mathcal{S}_{A,G,v}$. Cette mesure est similaire au ratio compétitif des algorithmes en ligne. Elle mesure la pénalité encourue par un algorithme reliée à son manque de connaissance. Dans le cas des algorithmes en ligne, ils ne connaissent pas les évènements futurs, lesquels sont connus des algorithmes hors ligne (servant de référence). Les algorithmes d'exploration que nous voulons définir ne connaissent pas la configuration des pannes, laquelle est connue de l'algorithme optimal (servant de référence).

Pour un graphe G et un noeud de départ v, un algorithme d'exploration (qui ne connaît pas la configuration des pannes) est appelé *parfaitement compétitif* si son surplus est le plus petit parmi tous les algorithmes d'exploration travaillant sous ces conditions. Notre but est de construire des algorithmes d'exploration avec de petits surplus : soit des algorithmes parfaitement compétitifs, soit des algorithmes ayant un surplus qui excède seulement légèrement le plus petit surplus possible.

Par la suite, lorsque le graphe G et le noeud de départ v seront fixes, nous écrirons $\mathcal{C}(A, P)$ au lieu de $\mathcal{C}(A, G, v, P)$, $opt(P)$ au lieu de $opt(G, v, P)$ et $\mathcal{S}_A$ au lieu de $\mathcal{S}_{A,G,v}$. Si le lien correspondant au port p d'un certain noeud est en panne, nous dirons que pour ce noeud, le port p mène vers un lien *défectueux*, autrement (si le lien est fonctionnel), nous dirons que le port est *libre*.

3.1.2 Énoncé du problème

Étant donné notre algorithme d'exploration parfaitement compétitif pour les anneaux ainsi que les résultats publiés dans [28] pour les lignes et les arbres, nous soulevons les questions suivantes :

1. Est-il possible d'étendre les connaissances acquises dans l'exploration des anneaux à une classe de graphes plus vaste contenant des cycles, soit les *graphes hamiltoniens* ?

2. Quelle serait la performance de la fouille en profondeur (DFS) dans l'exploration de graphes hamiltoniens en présence de pannes de liens ?

3. Quel serait le surplus de DFS dans les *graphes complets* (un type de graphes hamiltoniens) qui contiendraient des pannes de liens ?

3.2 Les résultats de la recherche

Pour tous les anneaux et pour tous les noeuds de départ, nous construisons un algorithme d'exploration parfaitement compétitif. L'algorithme ainsi que son surplus dépendent uniquement de la grandeur de l'anneau. Ceci contraste avec l'algorithme d'exploration parfaitement compétitif pour la ligne définit dans [28], où son comportement dépend plutôt de la distance entre le noeud de départ et le bout de la ligne le plus près, et *non* de la taille de la ligne.

Le temps de calcul total utilisé par l'algorithme d'exploration est linéaire dans le nombre de liens du graphe exploré. Notre contribution principale est l'analyse du surplus de cet algorithme, démontrant que des stratégies naturelles simples performent très bien dans les anneaux en présence de pannes de liens.

Pour un graphe hamiltonien arbitraire de grandeur $n \geq 3$ et pour chaque noeud de départ, nous avons démontré que, pour les réseaux modélisés par des graphes hamiltoniens, le surplus de n'importe quel algorithme d'exploration de type fouille en profondeur est

au plus 10/9 supérieur à celui d'un algorithme parfaitement compétitif. En outre, pour les graphes hamiltoniens de grandeur 24 ou plus, ce surplus est moins de 6% supérieur à celui d'un algorithme parfaitement compétitif.

3.3 Méthodologie

Tous les résultats présentés sont démontrés par des preuves rigoureuses qui utilisent des méthodes combinatoires et des éléments de la théorie des graphes. L'analyse s'effectue en pire cas et couvre toutes les configurations de pannes possibles.

Chapitre 4

Exploration parfaitement compétitive des anneaux

4.1 Introduction et définitions

Dans cette section, nous présentons un algorithme parfaitement compétitif pour tous les anneaux. Un anneau est un graphe $R = (V, E)$, où $V = \{v_1, ..., v_n\}$, $n \geq 3$ et $E = \{\{v_i, v_{i+1}\} : i = 1, ..., n-1\} \cup \{\{v_n, v_1\}\}$. Pour tous les $i > 1$, nous nommerons v_{i-1} le *prédécesseur* de v_i et pour $i < n$, nous nommerons v_{i+1} le *successeur* de v_i. Le prédécesseur de v_1 est v_n et le successeur de v_n est v_1. Pour chaque noeud, nous notons par g le port menant vers le prédécesseur de v et par d le port menant vers le successeur de v.

Soit un anneau R et un noeud de départ v. Pour une configuration des pannes P non-nulle, x représente la distance entre v et la première panne rencontrée en empruntant toujours le port g. De la même façon, y est la distance entre v et la première panne rencontrée en prenant toujours le port d.

Un agent qui connaîtrait P, et du fait même connaîtrait x et y, aurait l'algorithme d'exploration optimal suivant : commencer l'exploration en allant vers la panne la plus proche, pour ensuite revenir en arrière jusqu'à ce qu'une panne soit rencontrée. Donc $opt(P) = 2x + y$ pour $x \leq y$ et $2y + x$ pour $x > y$.

Les pannes empêchent le robot de passer au prochain noeud. Nous pouvons sans perte de généralité nous restreindre à l'exploration d'anneaux contenant au plus 2 pannes. En effet, toutes les configurations de pannes P ayant p pannes, $p > 2$ auront $p - 2$ pannes inaccessibles par le robot, lequel ne pourra atteindre que les pannes à distance x et y de v.

4.2 Énoncé de l'algorithme

La procédure suivante prend deux paramètres : un entier positif s et une direction $t \in \{g, d\}$. Elle permet le déplacement de l'agent d'au plus s pas en prenant toujours le port t et ce, jusqu'à ce qu'une panne soit rencontrée. Cette procédure retourne la valeur booléenne $b = faux$ si une panne est rencontrée et $b = vrai$ dans le cas contraire.

Procédure ALLER-A-DISTANCE (s, t)
    Tant que le port t du noeud courant est libre et que la distance
     parcourue est $< s$ **faire**
        prendre le port t
    Si le port t du noeud courant est libre **alors**
        $b := vrai$
    sinon
        $b := faux$

La prochaine procédure permet à l'agent de prendre de façon successive le port $t \in \{g, d\}$ jusqu'à ce qu'une panne soit rencontrée. Elle permet en outre d'éviter de revenir en arrière au noeud v si aucune panne n'est rencontrée.

Procédure ALLER-AU-BOUT(t)
    Tant que le port t du noeud courant est libre et moins de n noeuds
     ont été visités **faire**
        prendre le port t

Nous pouvons maintenant formuler notre algorithme pour l'exploration des anneaux :

Algorithme `Anneau`

Si le port g mène vers un lien défectueux **alors**
ALLER-AU-BOUT(d)
Sinon
Si le port d mène vers un lien défectueux **alors**
ALLER-AU-BOUT(g)
Sinon
Si $n \leq 5$ **alors**
ALLER-AU-BOUT(g)
ALLER-AU-BOUT(d)
Sinon
Si $6 \leq n \leq 19$ **alors**
ALLER-A-DISTANCE(1, g)
Si b **alors**
ALLER-AU-BOUT(d)
ALLER-AU-BOUT(g)
Sinon
ALLER-AU-BOUT(d)
Sinon
ALLER-A-DISTANCE(2, g)
Si b **alors**
ALLER-AU-BOUT(d)
ALLER-AU-BOUT(g)
Sinon
ALLER-AU-BOUT(d)

Comme le temps de calcul pour chaque visite de noeud est constant, le temps total de calcul de l'algorithme `Anneau` est linéaire dans la taille de l'anneau. La proposition suivante nous donne le surplus de l'algorithme `Anneau`.

Proposition 2.1 *Le surplus de l'algorithme* `Anneau` *est :*

- 1 quand $n = 3$
- $\frac{2n-3}{n}$ quand $4 \leq n \leq 5$
- $\frac{3}{2}$ quand $6 \leq n \leq 7$
- $\frac{2n-2}{n+1}$ quand $8 \leq n \leq 19$
- $\frac{9}{5}$ quand $20 \leq n \leq 23$
- $\frac{2n-1}{n+2}$ quand $n \geq 24$

Preuve. Dans un premier temps, nous pouvons observer que si l'anneau ne contient aucune panne, alors l'algorithme `Anneau` a un coût de $n-1$ pour $n \leq 5$, un coût de n pour $6 \leq n \leq 19$ et un coût de $n+1$ pour $n \geq 20$. Donc pour une configuration des pannes P vide, $\frac{\mathcal{C}(Anneau,P)}{opt(P)} = 1$ pour $n \leq 5$, $\frac{\mathcal{C}(Anneau,P)}{opt(P)} = \frac{n}{n-1}$ pour $6 \leq n \leq 19$ et $\frac{\mathcal{C}(Anneau,P)}{opt(P)} = \frac{n+1}{n-1}$ pour $n \geq 20$. Si P n'est pas vide, alors x et y sont définis. Si $x = 0$ ou $y = 0$, alors $\mathcal{C}(Anneau, P) = opt(P)$. Sinon, considérons les cas suivants :

$n = 3$. Dans ce cas, $\mathcal{C}(Anneau, P) = 2x + y = 2y + x = opt(P)$. Donc dans ce cas, $\mathcal{S}_{Anneau} = 1$.

$4 \leq n \leq 5$. Dans ce cas, $\frac{\mathcal{C}(Anneau,P)}{opt(P)} = \frac{2x+y}{min\{2x+y,2y+x\}}$. Si $x \leq y$, alors $\mathcal{C}(Anneau, P) = 1$. Si $x > y$, alors $\frac{\mathcal{C}(Anneau,P)}{opt(P)} = \frac{2x+y}{2y+x}$ et cette fraction est maximisée avec $x = n-2$, $y = 1$ pour donner $\frac{\mathcal{C}(Anneau,P)}{opt(P)} = \frac{2n-3}{n}$. Comme $1 < \frac{2n-3}{n}$ pour $n \geq 4$, $\mathcal{S}_{Anneau} = \frac{2n-3}{n}$ dans ce cas.

$n = 6$. Dans ce cas, $\frac{\mathcal{C}(Anneau,P)}{opt(P)} = \frac{2+2y+x}{min\{2x+y,2y+x\}}$. Si $x < y$, alors $\frac{\mathcal{C}(Anneau,P)}{opt(P)} = \frac{2+2y+x}{2x+y}$ et cette fraction est maximisée avec $x = 2$, $y = 3$ pour donner $\frac{\mathcal{C}(Anneau,P)}{opt(P)} = \frac{10}{7}$. Si $x \geq y$, alors $\frac{\mathcal{C}(Anneau,P)}{opt(P)} = \frac{2+2y+x}{2y+x}$ et cette fraction est maximisée avec $x = 2$, $y = 1$ pour donner $\frac{\mathcal{C}(Anneau,P)}{opt(P)} = \frac{3}{2}$. Le ratio $\frac{\mathcal{C}(Anneau,P)}{opt(P)}$ pour le cas sans panne est de $\frac{6}{5}$ dans ce cas. Comme $\frac{10}{7} < \frac{3}{2}$ et $\frac{6}{5} < \frac{3}{2}$, nous avons $\mathcal{S}_{Anneau} = \frac{3}{2}$ dans ce cas.

$n = 7$. Dans ce cas, $\frac{\mathcal{C}(Anneau,P)}{opt(P)} = \frac{2+2y+x}{min\{2x+y,2y+x\}}$. Si $x \leq y$, alors $\frac{\mathcal{C}(Anneau,P)}{opt(P)} = \frac{2+2y+x}{2x+y}$ et cette fraction est maximisée avec $x = 2$, $y = 4$ pour donner $\frac{\mathcal{C}(Anneau,P)}{opt(P)} = \frac{3}{2}$. Si $x > y$, alors $\frac{\mathcal{C}(Anneau,P)}{opt(P)} = \frac{2+2y+x}{2y+x}$ et cette fraction est maximisée avec $x = 2$, $y = 1$ pour donner $\frac{\mathcal{C}(Anneau,P)}{opt(P)} = \frac{3}{2}$. Le ratio $\frac{\mathcal{C}(Anneau,P)}{opt(P)}$ pour le cas sans panne est de $\frac{7}{6}$ dans ce cas. Comme $\frac{7}{6} < \frac{3}{2}$, nous avons $\mathcal{S}_{Anneau} = \frac{3}{2}$ dans ce cas.

$8 \leq n \leq 19$. Dans ce cas, $\frac{\mathcal{C}(Anneau,P)}{opt(P)} = \frac{2+2y+x}{min\{2x+y,2y+x\}}$. Si $x \leq y$, alors $\frac{\mathcal{C}(Anneau,P)}{opt(P)} = \frac{2+2y+x}{2x+y}$ et cette fraction est maximisée avec $x = 2$, $y = n-3$ pour donner $\frac{\mathcal{C}(A,P)}{opt(P)} = \frac{2n-2}{n+1}$. Si $x > y$, alors $\frac{\mathcal{C}(A,P)}{opt(P)} = \frac{2+2y+x}{2y+x}$ et cette fraction est maximisée avec $x = 2$, $y = 1$ pour donner $\frac{\mathcal{C}(Anneau,P)}{opt(P)} = \frac{3}{2}$. Le ratio $\frac{\mathcal{C}(Anneau,P)}{opt(P)}$ pour le cas sans panne est de $\frac{n}{n-1}$ dans ce cas. Comme $\frac{2n-2}{n+1} > \frac{3}{2}$ et $\frac{2n-2}{n+1} > \frac{n}{n-1}$ quand $n \geq 8$, nous avons $\mathcal{S}_{Anneau} = \frac{2n-2}{n+1}$ dans ce cas.

$20 \leq n \leq 23$. Dans ce cas, $\frac{\mathcal{C}(Anneau,P)}{opt(P)} = \frac{4+2y+x}{min\{2x+y,2y+x\}}$. Si $x \leq y$, alors $\frac{\mathcal{C}(Anneau,P)}{opt(P)} = \frac{4+2y+x}{2x+y}$ et cette fraction est maximisée avec $x = 3$, $y = n-4$ pour donner $\frac{\mathcal{C}(Anneau,P)}{opt(P)} = \frac{2n-1}{n+2}$. Si $x > y$, alors $\frac{\mathcal{C}(Anneau,P)}{opt(P)} = \frac{4+2y+x}{2y+x}$ et cette fraction est maximisée avec $x = 3$, $y = 1$ pour donner $\frac{\mathcal{C}(Anneau,P)}{opt(P)} = \frac{9}{5}$. Le ratio $\frac{\mathcal{C}(Anneau,P)}{opt(P)}$ pour le cas sans panne est de $\frac{n+1}{n-1}$ dans ce cas. Comme $\frac{2n-1}{n+2} \leq \frac{9}{5}$ et $\frac{9}{5} > \frac{n+1}{n-1}$ quand $20 \leq n \leq 23$, nous avons $\mathcal{S}_{Anneau} = \frac{9}{5}$ dans ce cas.

$n \geq 24$. Dans ce cas, $\frac{\mathcal{C}(A,P)}{opt(P)} = \frac{4+2y+x}{min\{2x+y,2y+x\}}$. Si $x \leq y$, alors $\frac{\mathcal{C}(Anneau,P)}{opt(P)} = \frac{4+2y+x}{2x+y}$ et cette fraction est maximisée avec $x = 3$, $y = n-4$ pour donner $\frac{\mathcal{C}(Anneau,P)}{opt(P)} = \frac{2n-1}{n+2}$. Si $x > y$, alors $\frac{\mathcal{C}(Anneau,P)}{opt(P)} = \frac{4+2y+x}{2y+x}$ et cette fraction est maximisée avec $x = 3$, $y = 1$ pour donner $\frac{\mathcal{C}(Anneau,P)}{opt(P)} = \frac{9}{5}$. Le ratio $\frac{\mathcal{C}(Anneau,P)}{opt(P)}$ pour le cas sans panne est de $\frac{n+1}{n-1}$ dans ce cas. Comme $\frac{2n-1}{n+2} > \frac{9}{5}$ et $\frac{2n-1}{n+2} > \frac{n+1}{n-1}$ quand $n \geq 24$, nous avons $\mathcal{S}_{Anneau} = \frac{2n-1}{n+2}$ dans ce cas.

4.3 Algorithme parfaitement compétitif

Nous montrons maintenant que l'algorithme `Anneau` est parfaitement compétitif, c'est-à-dire qu'il possède le plus petit surplus parmi tous les algorithmes d'exploration de l'anneau qui ne connaissent pas l'emplacement des pannes. Nous adaptons à l'anneau les notions suivantes introduites dans [28], où elles étaient définies de façon similaire pour

la ligne. Nous noterons par $\mathcal{A}_k$ la classe des algorithmes d'exploration de l'anneau qui effectuent initialement k retours (c'est-à-dire, changements de direction), en assumant qu'aucune panne n'a été rencontrée avant les premiers k retours, pour ensuite ALLER-AU-BOUT, et dans le cas où une panne est rencontrée, retourner puis ALLER-AU-BOUT. Chaque algorithme *régulier* d'exploration de l'anneau (défini formellement plus tard) fait partie d'une de ces classes $\mathcal{A}_k$.

Un algorithme de classe $\mathcal{A}_k$ est appelé un algorithme à i-pas, pour $1 \leq i \leq n-2$, s'il effectue i pas avant le premier retour, à moins qu'une panne ne soit rencontrée auparavant. Notons que $\mathcal{A}_0$ est la classe des algorithmes de type fouille en profondeur (DFS) et qu'il existe deux algorithmes de ce type pour l'anneau, chacun débutant leur exploration dans une direction différente. De plus, un algorithme de classe $\mathcal{A}_k$ à i-pas, pour $k > 0$ et $i \geq n-1$ est en fait un algorithme de type DFS, c'est-à-dire un algorithme de classe $\mathcal{A}_0$. L'algorithme `Anneau` est un algorithme de classe $\mathcal{A}_0$ pour $n \leq 5$, un algorithme de classe $\mathcal{A}_1$ à 1-pas pour $6 \leq n \leq 19$ et un algorithme de classe $\mathcal{A}_1$ à 2-pas pour $n \geq 20$.

La preuve que l'algorithme `Anneau` est parfaitement compétitif est divisé en une série de lemmes.

Lemme 2.1 *Tous les algorithmes de classe $\mathcal{A}_0$ ont un surplus d'au moins $\frac{2n-3}{n}$, pour tout $n \geq 3$.*

Preuve. Considérons la configuration des pannes P consistant en une seule panne située à distance 1 du noeud de départ v, dans la direction opposée à celle utilisée par l'algorithme A de classe $\mathcal{A}_0$ au début de son exploration (figure 4.1). Dans ce cas, $\mathcal{C}(A, P) = 2n-3$ et $opt(P) = n$, ce qui conclut la preuve.

Lemme 2.2 *Soit $n \geq 3$. Chaque algorithme de classe $\mathcal{A}_1$ à $(n-2)$-pas a un surplus de $\frac{2n-3}{n-1}$ et chaque algorithme de classe $\mathcal{A}_1$ à i-pas, pour $1 \leq i \leq n-3$, a un surplus de $max\{\frac{n+i-1}{n-1}, \frac{3i+3}{i+3}, \frac{i+2n-3}{i+n}\}$.*

Preuve. Premièrement, considérons un algorithme A de classe $\mathcal{A}_1$ à $(n-2)$-pas et une configuration des pannes P. Si P est vide, alors $\mathcal{C}(A, P) = 2n-3$ et $opt(P) = n-1$,

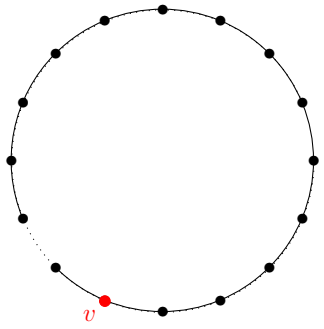

FIGURE 4.1 – Anneau de grandeur 16 ayant une seule panne à distance 1 de v

pour un ratio de $\frac{2n-3}{n-1}$. Sinon, x et y sont définis. Si $x = 0$ ou $y = 0$, alors $\mathcal{C}(A, P) = opt(P)$. Assumons donc que $x > 0$ et $y > 0$. Nous pouvons sans perte de généralité assumer que le premier pas de l'algorithme A se fera par le port l. Nous avons $x \leq n-2$ et donc $\frac{\mathcal{C}(A,P)}{opt(P)} = \frac{2x+y}{min\{2x+y,2y+x\}}$. Cette fraction est maximisée par $x = n-2$ et $y = 1$ pour une valeur de $\frac{2n-3}{n}$. Comme $\frac{2n-3}{n-1} > \frac{2n-3}{n}$, nous avons $\mathcal{S}_A = \frac{2n-3}{n-1}$ dans ce cas.

Pour le restant de la preuve, nous considérons un algorithme A de classe $\mathcal{A}_1$ à i-pas, pour $1 \leq i \leq n-3$, et une configuration des pannes P. Si P est vide, alors $\mathcal{C}(A, P) = n + i - 1$ et $opt(P) = n - 1$ pour un ratio de $\frac{n+i-1}{n-1}$. Sinon, x et y sont définis. Si $x = 0$ ou $y = 0$, alors $\mathcal{C}(A, P) = opt(P)$. Assumons donc que $x > 0$ et $y > 0$. Nous pouvons sans perte de généralité assumer que le premier pas de l'algorithme A se fera par le port l.

Si $x \leq i$ alors $\frac{\mathcal{C}(A,P)}{opt(P)} = \frac{2x+y}{min\{2x+y,2y+x\}}$ et cette fraction est maximisée par $x = i$ et $y = 1$. Donc la valeur maximale de $\frac{\mathcal{C}(A,P)}{opt(P)}$ est de $\frac{2i+1}{i+2}$ dans ce cas.

Si $x > i$ et $x \geq y$ alors $\frac{\mathcal{C}(A,P)}{opt(P)} = \frac{2i+2y+x}{2y+x}$ et cette fraction est maximisée par $x = i+1$ et $y = 1$. Donc la valeur maximale de $\frac{\mathcal{C}(A,P)}{opt(P)}$ est de $\frac{3i+3}{i+3}$ dans ce cas.

Si $x > i$ et $x < y$ alors $\frac{\mathcal{C}(A,P)}{opt(P)} = \frac{2i+2y+x}{2x+y}$ et cette fraction est maximisée par $x = i+1$ et $y = n-i-2$. Donc la valeur maximale de $\frac{\mathcal{C}(A,P)}{opt(P)}$ est de $\frac{i+2n-3}{i+n}$ dans ce cas.

Comme $\frac{3i+3}{i+3} > \frac{2i+1}{i+2}$, il s'ensuit que $\mathcal{S}_A = max\{\frac{n+i-1}{n-1}, \frac{3i+3}{i+3}, \frac{i+2n-3}{i+n}\}$.

Le lemme 2.2 nous donne le corollaire suivant :

Corollaire 2.1 *Chaque algorithme à* 1*-pas de classe* $\mathcal{A}_1$ *a un surplus de :*

- $\frac{3}{2}$ pour $3 \leq n \leq 7$
- $\frac{2n-2}{n+1}$ pour $n > 7$

Chaque algorithme à 2*-pas de classe* $\mathcal{A}_1$ *a un surplus de :*

- $\frac{5}{3}$ quand $n = 4$
- $\frac{9}{5}$ quand $5 \leq n \leq 23$
- $\frac{2n-1}{n+2}$ quand $n > 23$

Nous pouvons observer que pour $3 \leq i \leq n-3$, nous avons $\frac{3i+3}{i+3} \geq 2$. De plus, $\frac{2n-3}{n-1} \geq \frac{3}{2}$ pour $5 \leq n \leq 7$ et $\frac{2n-3}{n-1} \geq \frac{2n-2}{n+1}$ lorsque $n \geq 7$ (i serait moins de 3 pour $n < 5$). Donc par le lemme 2.2 et le corollaire 2.1, il s'en suit que tous les algorithmes d'exploration de classe $\mathcal{A}_1$ à i-pas, $i \geq 3$, ont un surplus supérieur à celui d'un algorithme de classe $\mathcal{A}_1$ à 1-pas. Donc, en cherchant des algorithmes parfaitement compétitifs de classe $\mathcal{A}_1$, nous pouvons nous restreindre aux algorithmes à i-pas, $i \leq 2$.

Il a été observé dans [28] que si un agent, à un moment donné de son exploration, est au noeud w, pour ensuite se déplacer le long d'un lien adjacent à w qu'il a déjà traversé auparavant pour immédiatement revenir à w, alors pour n'importe quelle configuration des pannes, un algorithme qui causerait une telle paire de traversées de liens aurait un surplus strictement plus grand qu'un autre algorithme qui omettrait cette paire de traversées de liens. Donc, dans la recherche d'une algorithme parfaitement compétitif, nous pouvons restreindre notre attention aux algorithmes qui n'effecturaient pas de tels mouvements inutiles. Nous appelons ces algorithmes *réguliers*. Il est facile de voir que chaque algorithme d'exploration régulier dans un anneau tombe dans l'une des classes d'algorithmes $\mathcal{A}_k$.

Lemme 2.3 *Pour chaque algorithme d'exploration A de classe $\mathcal{A}_k$, $k \geq 2$, il existe un algorithme de classe $\mathcal{A}_0$ ou un algorithme de classe $\mathcal{A}_1$ à i-pas, pour $i \leq 2$, avec un surplus plus petit ou égal.*

Preuve. Nous adaptons à l'anneau la preuve présentée pour la ligne dans le lemme 2.5 de [28]. Soit A un algorithme quelconque de classe $\mathcal{A}_k$, $k \geq 2$. Sans perte de généralité, nous pouvons assumer que l'agent débute son exploration en prenant le port l. Le comportement de A peut être décrit comme suit :

parcourir z_1 liens dans une direction ;
se retourner et parcourir $z_1 + z_2$ liens ;
se retourner et parcourir $z_2 + z_3$ liens ;
...
se retourner et parcourir $z_{k-2} + z_{k-1}$ liens ;
se retourner et parcourir $z_{k-1} + z_k$ liens ;
se retourner et ALLER-AU-BOUT ;
se retourner et ALLER-AU-BOUT ;

pour $z_1 < z_3 < z_5 < ...$ et $z_2 < z_4 < z_6 < ...$, en prenant en considération que les pannes les plus près de v (s'il y en a) se trouvent à une distance supérieure à z_{k-1} de v du côté où l'agent a traversé z_{k-1} liens, et à une distance supérieure à z_k de v du côté où l'agent a traversé z_k liens. Ceci implique que $z_{k-1} + z_k \leq n - 2$.

Considérons la configuration de pannes non-vide P pour laquelle la première panne de v dans une direction se trouve à distance $z_{k-1} + 1$ tandis que la première panne de v dans l'autre direction se trouve à distance $z_k + 1$ (il peut s'agir de la même panne). Nous avons
$\mathcal{C}(A, P) = 2(z_1 + ... + z_k) + (2z_{k-1} + 2) + (z_k + 1) = 2(z_1 + ... + z_k) + 2z_{k-1} + z_k + 3.$

Si $z_k \geq z_{k-1}$, nous avons $opt(A) = 2z_{k-1} + z_k + 3$. Le surplus dans ce cas sera donc de $\frac{\mathcal{C}(A,P)}{opt(P)} = \frac{2(z_1+...+z_k)+2z_{k-1}+z_k+3}{2z_{k-1}+z_k+3}$. Et si $2(z_1 + ... + z_k) - 2z_{k-1} - z_k \geq 3$, il s'ensuit que $\frac{\mathcal{C}(A,P)}{opt(P)} \geq 2$ dans ce cas.

Si $z_k < z_{k-1}$ alors nous avons $2(z_1 + ... + z_{k-1}) \geq z_k + 3$ et $opt(A) = 2z_k + z_{k-1} + 3$. Le surplus de l'algorithme sera alors de $\frac{\mathcal{C}(A,P)}{opt(P)} = \frac{2(z_1+...+z_k)+2z_{k-1}+z_k+3}{2z_k+z_{k-1}+3} \geq 2$ dans ce cas.

Comme chaque algorithme A' de classe $\mathcal{A}_0$ possède un surplus $\mathcal{S}(A') < 2$, nous pouvons conclure que dans tous les cas énoncés ci-haut, l'algorithme A' a un surplus inférieur à celui de l'algorithme A.

Si aucun de ces cas ne tient (c'est-à-dire, si nous avons $z_k \geq z_{k-1}$ et $2(z_1 + ... + z_{k-2}) + z_k < 3$), alors la seule possibilité est que $k = 2$ et $z_2 \leq 2$. Dans ce cas, considérons un algorithme A'' de classe $\mathcal{A}_1$ à z_2-pas. Le surplus de l'algorithme A'' est de $\mathcal{S}(A'') = max\{\frac{n+z_2-1}{n-1}, \frac{3z_2+3}{z_2+3}, \frac{z_2+2n-3}{z_2+n}\}$ par le lemme 2.2.

Nous avons donc $\frac{\mathcal{C}(A,P)}{opt(P)} = \frac{4z_1+3z_2+3}{2z_1+z_2+3} \geq \frac{3z_2+3}{z_2+3} \geq \frac{n+z_2-1}{n-1}$, car $z_2 \leq 2$.

Soit P_1 la configuration consistant en une seule panne pour laquelle $y = z_2 + 1$ et $x = n - z_2 - 2$. Nous avons $\mathcal{C}(A, P_1) = 2z_1 + 3z_2 + 2x + 1$ et $opt(P_1) = min\{2z_2 + x + 2, 2x + z_2 + 1\}$. Donc $\mathcal{C}(A, P_1) \geq z_2 + 2n - 3$ et $opt(P_1) \leq z_2 + n$, et donc $\frac{\mathcal{C}(A,P_1)}{opt(P_1)} \geq \frac{z_2+2n-3}{z_2+n}$.

Finalement, considérons la configuration de pannes vide $\emptyset$. Nous avons $\mathcal{C}(A, \emptyset) = 2z_1 + z_2 + n - 1 \geq n + z_2 - 1$ et $opt(\emptyset) = n - 1$. Donc $\frac{\mathcal{C}(A,\emptyset)}{opt(\emptyset)} \geq \frac{n+z_2-1}{n-1}$.

Il s'en suit que dans le cas où $z_k \geq z_{k-1}$ et $2(z_1 + ... + z_{k-2}) + z_k < 3$, nous avons $\mathcal{S}(A'') \leq \mathcal{S}(A)$. Ceci prouve le lemme.

Proposition 2.2 *Le surplus de tous les algorithmes d'exploration de l'anneau est d'au moins :*

- $\frac{2n-3}{n}$ quand $4 \leq n \leq 5$
- $\frac{3}{2}$ quand $6 \leq n \leq 7$
- $\frac{2n-2}{n+1}$ quand $8 \leq n \leq 19$
- $\frac{9}{5}$ quand $20 \leq n \leq 23$
- $\frac{2n-1}{n+2}$ quand $n \geq 24$

Preuve. Par le lemme 2.3, nous pouvons restreindre notre attention aux algorithmes de classe $\mathcal{A}_0$ et à ceux de i-pas de classe $\mathcal{A}_1$, pour $i \leq 2$.

Prenons $4 \leq n \leq 5$. Par le lemme 2.1, chaque algorithme de classe $\mathcal{A}_0$ possède un surplus d'au moins $\frac{2n-3}{n}$. Par le corollaire 2.1, chaque algorithme à 1-pas de classe $\mathcal{A}_1$ possède un surplus d'au moins $\frac{3}{2}$ et chaque algorithme à 2-pas de classe $\mathcal{A}_1$ possède un surplus d'au moins $\frac{9}{5}$. Comme $\frac{2n-3}{n} \leq \frac{3}{2} < \frac{9}{5}$, le surplus de n'importe quel algorithme d'exploration de l'anneau est au moins $\frac{2n-3}{n}$ dans ce cas.

Prenons $6 \leq n \leq 7$. Par le lemme 2.1, chaque algorithme de classe $\mathcal{A}_0$ possède un surplus d'au moins $\frac{2n-3}{n}$. Par le corollaire 2.1, chaque algorithme à 1-pas de classe $\mathcal{A}_1$ possède un surplus d'au moins $\frac{3}{2}$ et chaque algorithme à 2-pas de classe $\mathcal{A}_1$ possède un surplus d'au moins $\frac{9}{5}$. Comme $\frac{3}{2} \leq \frac{2n-3}{n} < \frac{9}{5}$, le surplus de n'importe quel algorithme d'exploration de l'anneau est au moins $\frac{3}{2}$ dans ce cas.

Prenons $8 \leq n \leq 19$. Par le lemme 2.1, chaque algorithme de classe $\mathcal{A}_0$ possède un surplus d'au moins $\frac{2n-3}{n}$. Par le corollaire 2.1, chaque algorithme à 1-pas de classe $\mathcal{A}_1$ possède un surplus d'au moins $\frac{2n-2}{n+1}$ et chaque algorithme à 2-pas de classe $\mathcal{A}_1$ possède un surplus d'au moins $\frac{9}{5}$. Comme $\frac{2n-2}{n+1} < \frac{2n-3}{n} \leq \frac{9}{5}$ pour $8 \leq n \leq 15$ et $\frac{2n-2}{n+1} \leq \frac{9}{5} < \frac{2n-3}{n}$ pour $16 \leq n \leq 19$, le surplus de n'importe quel algorithme d'exploration de l'anneau est au moins $\frac{2n-2}{n+1}$ dans ce cas.

Prenons $20 \leq n \leq 23$. Par le lemme 2.1, chaque algorithme de classe $\mathcal{A}_0$ possède un surplus d'au moins $\frac{2n-3}{n}$. Par le corollaire 2.1, chaque algorithme à 1-pas de classe $\mathcal{A}_1$ possède un surplus d'au moins $\frac{2n-2}{n+1}$ et chaque algorithme à 2-pas de classe $\mathcal{A}_1$ possède un surplus d'au moins $\frac{9}{5}$. Comme $\frac{9}{5} < \frac{2n-2}{n+1} < \frac{2n-3}{n}$, le surplus de n'importe quel algorithme d'exploration de l'anneau est au moins $\frac{9}{5}$ dans ce cas.

Prenons $n \geq 24$. Par le lemme 2.1, chaque algorithme de classe $\mathcal{A}_0$ possède un surplus d'au moins $\frac{2n-3}{n}$. Par le corollaire 2.1, chaque algorithme à 1-pas de classe $\mathcal{A}_1$ possède un surplus d'au moins $\frac{2n-2}{n+1}$ et chaque algorithme à 2-pas de classe $\mathcal{A}_1$ possède un surplus d'au moins $\frac{2n-1}{n+2}$. Comme $\frac{2n-1}{n+2} < \frac{2n-2}{n+1} < \frac{2n-3}{n}$, le surplus de n'importe quel algorithme d'exploration de l'anneau est au moins $\frac{2n-1}{n+2}$ dans ce cas.

Les propositions 2.1 et 2.2 impliquent :

Théorème 2.1 *L'algorithme* `Anneau` *est un algorithme d'exploration parfaitement compétitif pour tous les anneaux.*

Chapitre 5

Exploration de graphes hamiltoniens

5.1 Performance de la fouille en profondeur

L'agent possède une carte précise et pour la plupart des graphes, il existe plusieurs algorithmes de type *fouille en profondeur (DFS)* pour explorer la composante sans panne, dépendamment de l'ordre d'exploration des ports libres à chaque noeud visité. Il est donc approprié de parler de la classe $\mathcal{DFS}$ où chacun de ses membres possède un ordre d'exploration différent, spécifié par une permutation des ports pour chaque noeud. Nous adoptons donc la définition suivante.

Prenons un graphe arbitraire $G = (V, E)$ et n'importe quelle configuration des pannes $P \in G$. Soit v le point de départ de l'agent. Pour tous les noeuds w de G, soit σ_w une permutation des ports à w. Soit $\alpha = \{\ \sigma_w : w \in V\}$. $DFS(\alpha)$ est l'algorithme spécifié en appelant la procédure récursive suivante au noeud v.

Procédure `Explorer`(w)
Soit $(i_1, \ldots, i_k)$ la séquence des ports libres à w dans l'ordre donné par σ_w.
Soit $(v_1, \ldots, v_k)$ la liste des voisins de w et correspondant à ces ports.
Pour chaque $r = 1, \ldots, k$, soit j_r le port à v_r correspondant au lien $\{v_r, w\}$.
marquer(w)
pour $r = 1$ **à** k **faire**
    si v_r n'est pas marqué **alors**
        prendre le port i_r
        `Explorer`(v_r)
        s'il y a encore des noeuds non-marqués **alors** prendre le port j_r

La classe $\mathcal{DFS}$ est définie comme la classe d'algorithmes $DFS(\alpha)$, pour toutes les séquences de permutations $\alpha = \{ \sigma_w : w \in V\}$. Notons que pour chaque algorithme de la classe $\mathcal{DFS}$, la route empruntée par l'agent est en fait une visite d'un arbre couvrant de la composante sans panne C du graphe, contenant le noeud de départ v et correspondant à la configuration des pannes P, dans laquelle l'agent arrête dès que le dernier noeud de la composante a été visité. La branche de l'arbre correspondant à la dernière feuille visitée n'est donc traversée qu'une seule fois. Conséquemment, pour chaque α et chaque P, nous avons $\mathcal{C}(DFS(\alpha), P) \leq 2m-3$, où m est la taille de la composante C. Le temps total de calcul pour tous les algorithmes de la classe $\mathcal{DFS}$ est linéaire dans le nombre de ports, c'est-à-dire linéaire dans le nombre de liens du graphe exploré.

Notre but est de démontrer que si G est un graphe hamiltonien, alors chaque algorithme de classe $\mathcal{DFS}$ a un surplus légèrement supérieur à celui d'un algorithme parfaitement compétitif. Notons qu'un graphe hamiltonien doit avoir au moins trois noeuds, donc nous assumerons dans ce qui suit que la taille n du graphe est d'au moins 3. Débutons par prouver le lemme suivant.

Lemme 3.1 *Soit G un graphe hamiltonien quelconque de grandeur n. Pour tous les algorithmes A de la classe $\mathcal{DFS}$ et pour tous les noeuds de départ, $\mathcal{S}_A \leq \frac{2n-4}{n-1}$.*

Preuve. Considérons une configuration des pannes P et un noeud de départ v quelconque. Soit C la composante sans panne de taille m correspondant à P et contenant v. Considérons maintenant les deux cas suivants.

Cas 1. Il existe un chemin hamiltonien dans C ayant le point de départ v à l'une de ses extrémités.

Soit $(w_1, \ldots, w_m)$ un chemin hamiltonien dans C où $w_1 = v$. C contient tous les liens de ce chemin et possiblement d'autres liens additionnels $\{w_i, w_j\}$. Pour $m = 2$, nous avons $\mathcal{C}(A, P) = opt(P) = 1$. Assumons que $m \geq 3$; nous avons $opt(P) = m - 1$. Soit T un arbre couvrant de C et correspondant à la route d'un agent qui exécute l'algorithme A. T possède $m - 1$ liens. Comme nous l'avons remarqué précédemment, le dernier lien de T n'est traversé qu'une seule fois par l'agent. Nous montrons que le premier lien n'est lui aussi traversé qu'une seule fois. Soit $\{w_1, w_j\}$ ce premier lien traversé. Supposons que ce lien est traversé par l'agent une seconde fois. Cette traversée doit être de w_j vers w_1. Cela implique qu'au temps t de la visite de w_j précédent immédiatement cette traversée, il existe au moins un noeud non-visité. Soit w_s ce noeud non-visité au temps t, ayant le plus grand indice $s < j$ ou le plus petit indice $s > j$. Si $s < j$ alors w_{s+1} a été visité avant le temps t et si $s > j$ alors w_{s-1} a été visité avant le temps t. Dans le premier cas, le port de w_{s+1} correspondant au lien $\{w_{s+1}, w_s\}$ est libre et dans le second cas, le port de w_{s-1} correspondant au lien $\{w_{s-1}, w_s\}$ est libre. Ceci contredit le fait que, selon l'algorithme A, l'agent doit visiter w_s avant le temps t.

Donc le premier lien traversé et le dernier lien traversé ne sont traversés qu'une seule fois. Ceci implique que le coût de l'algorithme A est au plus $2m - 4$, pour $m \geq 3$, dans ce cas.

Cas 2. Il n'existe pas de chemin hamiltonien dans $\boldsymbol{C}$ ayant le point de départ v à l'une de ses extrémités.

Dans ce cas $opt(P) \geq m$ et le coût de l'algorithme A est au plus $2m - 3$ car le dernier lien n'est traversé qu'une seule fois.

Comme $\frac{2m-4}{m-1} \geq \frac{2m-3}{m}$ pour $m \geq 3$, il s'en suit que dans les deux cas $\frac{\mathcal{C}(A,P)}{opt(P)} \leq \frac{2m-4}{m-1}$, chaque fois que la composante C est de taille m. Comme $\frac{2m-4}{m-1}$ est une fonction croissante de m, cette valeur est maximisée pour $m = n$, ce qui conclut la preuve.

Voici le résultat principal de cette section.

Théorème 3.1 *Pour tous les graphes hamiltoniens $G = (V, E)$, pour tous les noeuds de départ v et tous les algorithmes A de la classe $\mathcal{DFS}$, le ratio entre le surplus de A et le surplus d'un algorithme parfaitement compétitif sur G et débutant à v est d'au plus 10/9. En outre, pour $n \geq 24$, ce ratio est moins de 1.06.*

Preuve. Soit B un algorithme parfaitement compétitif sur G et débutant à v. Soit R un cycle hamiltonien quelconque dans G. Par définition, R contient le noeud de départ v. Soit P^* l'ensemble des liens de G qui ne sont pas dans R. Soit $x = \max_{P^* \subseteq P \subseteq E} \frac{\mathcal{C}(B,G,v,P)}{opt(G,v,P)}$. Par définition, $\mathcal{S}_{B,G,v} \geq x$. D'autre part, l'exécution de n'importe quel algorithme pour la configuration des pannes P contenant P^* peut être considéré comme l'exécution d'un algorithme sur un anneau R pour la configuration $P \setminus P^*$. Comme la famille des configurations des pannes dans G contenant P^* est égale à la famille des configurations de pannes $P' \cup P^*$, où P' est une configuration de pannes dans R, nous avons $x \geq \mathcal{S}_{\texttt{Anneau},R,v}$, sinon l'algorithme `Anneau` ne serait pas parfaitement compétitif, ce qui contredirait le théorème 2.1. Il s'en suit que $\mathcal{S}_{B,G,v} \geq \mathcal{S}_{\texttt{Anneau},R,v}$. Donc par la proposition 2.2, $\mathcal{S}_{B,G,v}$ est au moins :

- $\frac{2n-3}{n}$ pour $4 \leq n \leq 5$
- $\frac{3}{2}$ pour $6 \leq n \leq 7$
- $\frac{2n-2}{n+1}$ pour $8 \leq n \leq 19$
- $\frac{9}{5}$ pour $20 \leq n \leq 23$
- $\frac{2n-1}{n+2}$ pour $n \geq 24$

Par le lemme 3.1, le ratio $\frac{\mathcal{S}_{A,G,v}}{\mathcal{S}_{B,G,v}}$ est au plus :

- 1 quand $n \leq 3$
- $\frac{\frac{2n-4}{n-1}}{\frac{2n-3}{n}} \leq \frac{15}{14} \approx 1.0714$ quand $4 \leq n \leq 5$
- $\frac{\frac{2n-4}{n-1}}{\frac{3}{2}} \leq \frac{10}{9} \approx 1.1111$ quand $6 \leq n \leq 7$
- $\frac{\frac{2n-4}{n-1}}{\frac{2n-2}{n+1}} \leq \frac{54}{49} \approx 1.1020$ quand $8 \leq n \leq 19$
- $\frac{\frac{2n-4}{n-1}}{\frac{9}{5}} \leq \frac{35}{33} \approx 1.0606$ quand $20 \leq n \leq 23$
- $\frac{\frac{2n-4}{n-1}}{\frac{2n-1}{n+2}} \leq \frac{1144}{1081} \approx 1.0583$ quand $n \geq 24$

Comme $1 < \frac{1144}{1081} < \frac{35}{33} < \frac{15}{14} < \frac{54}{49} < \frac{10}{9}$, le ratio entre le surplus de A et le surplus d'un algorithme parfaitement compétitif sur G et débutant à v est d'au plus $10/9$. En outre, pour $n \geq 24$, ce ratio n'est jamais plus que 1.06.

5.2 Algorithmes de la classe $\mathcal{DFS}$ dans les graphes complets

Il est naturel de se demander si la limite $\frac{2n-4}{n-1}$ sur le surplus de tous les algorithmes de la classe $\mathcal{DFS}$, obtenue par le lemme 3.1, est stricte. Ce n'est pas le cas pour tous les graphes hamiltoniens, tel qu'en témoigne l'exemple de l'anneau. En effet, par le lemme 2.1, le surplus de chaque algorithme de la classe $\mathcal{DFS}$ sur l'anneau est de $\frac{2n-3}{n} < \frac{2n-4}{n-1}$. Nous pouvons donc nous demander si cette limite est stricte pour certains graphes hamiltoniens. La proposition suivante répond à cette question par l'affirmative.

Proposition 3.1 *Le surplus de chaque algorithme A de la classe $\mathcal{DFS}$ est exactement $\frac{2n-4}{n-1}$ pour le graphe complet de taille $n \geq 3$.*

Preuve. Soit A un algorithme quelconque de la classe $\mathcal{DFS}$ débutant au noeud v pour le graphe complet K_n. Compte tenu du lemme 3.1, il suffit de montrer une seule configuration de pannes P pour laquelle $\frac{\mathcal{C}(A,K_n,v,P)}{opt(K_n,v,P)} = \frac{2n-4}{n-1}$. Cette configuration dépend de l'ensemble des permutations $\alpha = \{\sigma_w : w \in V\}$ pour laquelle $A = DFS(\alpha)$. Soit $\sigma_v = (i_1, \ldots, i_{n-1})$ et soit w le noeud correspondant au port i_1 de v. Soit $\sigma_w = (j_1, \ldots, j_{n-1})$ et soit j_k le port au noeud w qui correspond au lien $\{w, v\}$. Soit $a = \min\{1, \ldots, n-1\} \setminus \{k\}$ et soit $b = \min\{1, \ldots, n-1\} \setminus \{k, a\}$. Soit u' le noeud correspondant au port j_a du noeud w et soit u'' le noeud correspondant au port j_b du noeud w. Nous définissons $v_1 = v$, $v_{n-2} = u'$, $v_{n-1} = w$, $v_n = u''$, et tous les autres noeuds autres que v, u', w, u'' sont nommés arbitrairement $v_2, \ldots, v_{n-3}$. Soit F l'ensemble des liens $\{\{v_1, v_{n-1}\}, \{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{n-1}, v_n\}\}$ (figure 5.1).

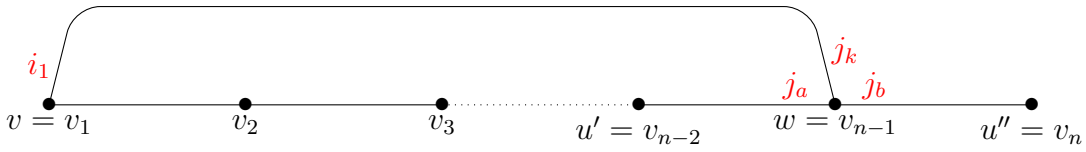

FIGURE 5.1 – L'ensemble des liens F

Considérons la configuration de pannes P telle que tous les liens de K_n sont en pannes exceptés ceux contenus dans l'ensemble des liens F. Pour cette configuration des pannes, le coût de l'algorithme optimal est de $n-1$, correspondant au trajet $v_1, v_2, \ldots, v_n$. L'agent qui exécute l'algorithme A débute son exploration par le port i_1 qui mène au noeud w. Il va ensuite prendre le port j_a, soit le premier port libre du noeud w selon la permutation σ_w et différent du port j_k par lequel l'agent vient d'arriver au noeud w. À ce point, l'agent se trouve au noeud $u' = v_{n-2}$. Il va ensuite se déplacer en suivant le trajet $(v_{n-2}, v_{n-3}, \ldots, v_2)$, pour revenir sur ses pas par le chemin inverse $(v_2, v_3, \ldots, v_{n-2})$ vers $u' = v_{n-2}$. L'agent va par la suite revenir à $w = v_{n-1}$ et terminer son exploration au noeud $u'' = v_n$, pour un coût total de $2n-4$. Donc $\frac{\mathcal{C}(A,K_n,v,P)}{opt(K_n,v,P)} = \frac{2n-4}{n-1}$.

Chapitre 6

Conclusion

Nous avons conçu un algorithme parfaitement compétitif pour les anneaux et nous avons prouvé que pour tous les graphes hamiltoniens, le surplus de *tous* les algorithmes de la classe $\mathcal{DFS}$ excède celui d'un algorithme parfaitement compétitif pour un graphe donné par un facteur de moins de $10/9$. Toutefois, ce résultat ne tient pas pour les graphes arbitraires, comme en témoigne très bien l'exemple d'une ligne dans laquelle le noeud de départ se trouve à distance 1 de l'une de ses extrémités. Dans ce cas, l'algorithme parfaitement compétitif a un surplus de 1 (comme démontré dans [28]) tandis que le "mauvais" DFS (c'est-à-dire celui qui débute son exploration en allant vers l'extrémité la plus lointaine de la ligne) a quant à lui un surplus de $\frac{2n-1}{n+1}$, pour les lignes de longueur n, soit arbitrairement près de 2 pour les longues lignes.

Il reste tout de même l'espoir que le "meilleur" algorithme de la classe $\mathcal{DFS}$, c'est-à-dire l'algorithme DFS ayant le plus petit surplus pour un graphe quelconque donné, puisse avoir un surplus similaire à celui d'un algorithme parfaitement compétitif pour ce même graphe. Même si cela s'avérait vrai, trouver le meilleur DFS (une tâche triviale dans le cas de la ligne) peut s'avérer être très ardu pour des graphes arbitraires. En outre, nous ne savons pas si le "meilleur" DFS pour un graphe arbitraire possède un surplus similaire à celui d'un algorithme parfaitement compétitif pour ce graphe. Même si c'était le cas, nous ne savons pas comment trouver ce DFS ou tout autre algorithme ayant un surplus similaire.

Si nous allons en dehors de la classe $\mathcal{DFS}$, la question la plus importante semble être s'il existe ou non un algorithme d'exploration parfaitement compétitif pour les graphes

arbitraires ayant un temps total de calcul polynômial dans la taille du graphe. Si un tel algorithme n'est pas construit, le principal défi semble être la recherche, pour un graphe arbitraire, d'un algorithme d'exploration ayant un surplus qui dépasse celui d'un algorithme parfaitement compétitif pour ce graphe donné seulement par un petit facteur.

En vue de notre mémoire et de [28], la situation est la suivante. Pour la classe des graphes hamiltoniens et pour la classe des arbres, un tel algorithme existe, et ce facteur est au plus $10/9$ dans le premier cas et $9/8$ dans le second. Pour ces deux classes de graphes, les algorithmes qui performent si bien sont aussi très simples et leur temps de calcul total durant l'exploration est linéaire dans le nombre de liens du graphe exploré. (Comme *chaque* algorithme de la classe $\mathcal{DFS}$ pour les graphes hamiltoniens répond à nos besoins, nous pouvons utiliser celui induit par la permutation croissante des ports à chaque noeud.)

Est-il possible de concevoir, pour un graphe arbitraire, un algorithme d'exploration ayant un surplus qui excède celui d'un algorithme parfaitement compétitif pour ce graph par un facteur d'au plus 1.15 par exemple, et pour lequel le temps total de calcul soit polynômial dans la taille du graphe ?

D'autres variantes intéressantes du problème d'exploration de graphes en présence de pannes de liens serait si l'agent connaîtrait le nombre exact de pannes, ou du moins une borne supérieure sur ce nombre.

Bibliographie

[1] L. Alboul, P. S. Haynes et J. Penders, Dynamic graph-based search in unknown environments. Journal of Discrete Algorithms (JDA) 12, pages 2-13, 2012.

[2] B. Balamohan, P. Flocchini, A. Miri et N. Santoro, Time optimal algorithms for black hole search in rings. Discrete Mathematics, Algorithms and Applications (DMAA) 3(4), pages 457-472, 2011.

[3] E. Bar-Eli, P. Berman, A. Fiat et R. Yan, On-line navigation in a room. Journal of Algorithms (JA) 17, pages 319-341, 1994.

[4] M. A. Bender, A. Fernandez, D. Ron, A. Sahai et S. Vadhan, The power of a pebble : exploring and mapping directed graphs. Information and Computation (IC), pages 1-21, 2002.

[5] M. A. Bender et D. Slonim, The power of team exploration : two robots can learn unlabeled directed graphs. Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS), pages 75-85, 1994.

[6] A. Blum, P. Raghavan et B. Schieber, Navigating in unfamiliar geometric terrain. SIAM Journal on Computing (SICOMP) 26, pages 110-137, 1997.

[7] J. Chalopin, S. Das et A. Kosowski, Constructing a map of an anonymous graph : Applications of universal sequences. Proceedings of the 14th International Conference on Principles of Distributed Systems (OPODIS), pages 119-134, 2010.

[8] A. Casteigts, P. Flocchini, W. Quattrociocchi et N. Santoro, Time-varying graphs and dynamic networks. Proceedings of the 10th International Conference on Ad-hoc, Mobile, and Wireless Networks (ADHOC-NOW), pages 346-359, 2011.

[9] J. Czyzowicz, D. Kowalski, E. Markou et A. Pelc, Complexity of searching for a black hole. Fundamenta Informaticae (FI) 71, pages 229-242, 2006.

[10] J. Czyzowicz, D. Kowalski, E. Markou et A. Pelc, Searching for a black hole in synchronous tree networks. Combinatorics, Probability & Computing (CPC) 16, pages 595-619, 2007.

[11] X. Deng, T. Kameda et C. H. Papadimitriou, How to learn an unknown environment I : the rectilinear case. Journal of the Association for Computing Machinery (JACM) 45, pages 215-245, 1998.

[12] X. Deng et C. H. Papadimitriou, Exploring an unknown graph. Journal of Graph Theory (JGT) 32, pages 265-297, 1999.

[13] A. Dessmark et A. Pelc, Optimal graph exploration without good maps. Theoretical Computer Science (TCS) 326, pages 343-362, 2004.

[14] Y. Dieudonné et A. Pelc, Deterministic network exploration by a single agent with Byzantine tokens. Information Processing Letters (IPL) 112, pages 467-470, 2012.

[15] S. Dobrev, P. Flocchini, R. Kralovic, P. Ruzicka, G. Prencipe et N. Santoro, Black hole search in common interconnection networks. Networks 47, pages 61-71, 2006.

[16] S. Dobrev, P. Flocchini, R. Královic et N. Santoro, Exploring an unknown dangerous graph using tokens. Theoretical Computer Science (TCS) 472, pages 28-45, 2013.

[17] S. Dobrev, P. Flocchini, G. Prencipe et N. Santoro, Mobile search for a black hole in an anonymous ring. Algorithmica 48, pages 67-90, 2007.

[18] C. A. Duncan, S. G. Kobourov et V. S. A. Kumar, Optimal constrained graph exploration. Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 807-814, 2001.

[19] P. Flocchini, D. Ilcinkas, A. Pelc et N. Santoro, Computing without communicating : Ring exploration by asynchronous oblivious robots. Algorithmica 65, pages 562-583, 2013.

[20] P. Flocchini, D. Ilcinkas, A. Pelc et N. Santoro, How many oblivious robots can explore a line. Information Processing Letters (IPL) 111, pages 1027-1031, 2011.

[21] P. Flocchini, M. Kellett, P. C. Mason et N. Santoro, Fault-tolerant exploration of an unknown dangerous graph by scattered agents. Proceedings of the 14th International Conference on Stabilization, Safety, and Security of Distributed Systems (SSS), pages 299-313, 2012.

[22] P. Flocchini, M. Kellett, P. C. Mason et N. Santoro, Map construction and exploration by mobile agents scattered in a dangerous network. Proceedings of 23rd Parallel and Distributed Processing, International Symposium (IPDPS), 2009.

[23] P. Flocchini, M. Kellett, P. C. Mason et N. Santoro, Searching for black holes in subways. Theory of Computing Systems (TCS) 50(1), pages 158-184, 2012.

[24] P. Fraigniaud, L. Gasieniec, D. Kowalski et A. Pelc, Collective tree exploration. Networks 48, pages 166-177, 2006.

[25] P. Fraigniaud, D. Ilcinkas et A. Pelc, Tree exploration with an oracle. Information and Computation (IC) 206, pages 1276-1287, 2008.

[26] D. Ilcinkas, R. Klasing et A. M. Wade, Exploration of constantly connected dynamic graphs based on cactuses. Proceedings of the 21st International Colloquium on Structural Information and Communication Complexity (SIROCCO), pages 250-262, 2014.

[27] D. Ilcinkas et A. M. Wade, Exploration of the T-interval-connected dynamic graphs : The case of the ring. Proceedings of the 20th International Colloquium on Structural Information and Communication Complexity (SIROCCO), pages 13-23, 2013.

[28] E. Markou et A. Pelc, Efficient exploration of faulty trees. Theory of Computing Systems (TCS) 40, pages 225-247, 2007.

[29] P. Panaite et A. Pelc, Exploring unknown undirected graphs. Journal of Algorithms (JA) 33, pages 281-295, 1999.

[30] A. Pelc et A. Tiane, Efficient grid exploration with a stationary token. International Journal of Foundations of Computer Science (IJFCS) 25, pages 247-262, 2014.