Orsay
N° d'ordre : 8693

THÈSE

présentée pour obtenir le grade de
Docteur en Sciences de l'Université Paris XI, Orsay
Spécialité : Informatique

—

Jeux des gendarmes et du voleur dans les graphes

Mineurs de graphes, stratégies connexes, et approche distribuée.

Nicolas NISSE

—

Soutenue le 2 juillet 2007 devant le jury composé de :

Directeur de thèse :	Pierre FRAIGNIAUD	Directeur de Recherche
Rapporteurs :	Dieter KRATSCH	Professeur
	Stéphan THOMASSÉ	Professeur
Examinateurs :	Jean-Paul ALLOUCHE	Directeur de Recherche
	Dimitrios M. THILIKOS	Professeur

Remerciements

Je tiens tout d'abord à remercier Pierre Fraigniaud sans qui cette thèse n'aurait jamais vu le jour. Le plaisir que j'ai pris à effectuer ces trois années de thèse est en grande partie dù au fait qu'il a toujours parfaitement su me guider dans cet apprentissage du métier de chercheur. J'ai beaucoup appris à ses côtés et travailler avec lui a toujours été très agréable. Bref, merci Pierre !

Je remercie chaleureusement Dieter Kratsch et Stéphan Thomassé d'avoir accepté la relecture de ce manuscript. Leurs remarques et commentaires m'ont été d'une aide précieuse. Merci également à Jean-Paul Allouche d'avoir bien voulu présider mon jury de soutenance, ainsi qu'à Dimitrios Thilikos d'avoir fait le long voyage depuis Athènes.

Un des facteurs qui contribuent à la réussite d'une thèse est l'environnement de travail dans lequel celle-ci a été effectuée. Je ne sais pas si on peut parler d'environnement de travail, mais mes comparses de bureau ont indubitablement participé à l'élaboration de cette thèse. Un grand merci à David et Philippe pour les deux ans d'amusements, de musique, et aussi, un peu, de travail. Merci beaucoup à tous les thésards qui ont contribué à la bonne ambiance dans l'équipe GrafCom : Anh, Lynda, Rafael, Taoufik et Waddie. En particulier, je voudrais remercier David qui a souvent subi les foudres de mon humeur au cours de cette troisième année un peu stressante. J'ai beaucoup apprécié les discussions que nous avons tenues à propos de nos thèses respectives.

Je voudrais également remercier certains de mes "ainés" qui m'ont beaucoup appris pendant ces trois années, que ce soit scientifiquement ou non. Merci donc pour leurs précieux conseils à Pascal Berthomé, Lélia Blin, David Coudert, Frédéric Mazoit, Sébastien Tixeuil et Ioan Todinca. Merci bien sûr à tous les membres de l'équipe GrafCom et plus généralement du LRI qui m'ont accueilli au cours de ces trois ans.

Avec ma thèse s'achève également la période de gestation du workshop IMAGINE que nous mijotons depuis que, il y a deux ans autour d'un verre de vodka polonaise, Nathalie Mitton, David Ilcinkas et moi-même avons décidé de créer ce workshop. Le plus dur a finalement été d'en trouver le nom. J'ai pris beaucoup de plaisir à préparer cet événement avec eux et je les en remercie. Longue vie à IMAGINE !

Mes amis n'étant pas tous des fervents admirateurs de la théorie des graphes, il m'a fallu beaucoup d'efforts pour les convaincre qu'étudier les jeux des gendarmes et du voleur dans les graphes ("qu'est-ce que c'est un graphe, déjà ?") était un vrai travail. Je ne sais pas si je les ai vraiment convaincu, mais je les remercie infiniment pour leur soutien. Merci donc aux Supéléciens du *cd0d* (et associés) : Ben, Doodoo, Guitou, Jib, Lilou, Loeiz, Thomas, Thibault. Merci bien sûr aux *Clownys* que je suis très heureux d'avoir pour amis depuis si longtemps : Merci Alex, Bennou, Coco, Florent, Greg, Marie, Nico et Olivier. Et une pensée particulière pour Karim.

Pour finir, je remercie de tout cœur toute ma famille qui m'a soutenu depuis toujours. Merci particulièrement à ma mère, ma tante, ma grand-mère et mes sœurs pour le pot qu'elles ont cuisiné et organisé, et qui a conclu en beauté ces trois années de thèse.

Table des matières

Introduction

Dans cette thèse, nous considérons le problème des gendarmes et du voleur dans les graphes. Ce problème traite de la capture, par une équipe d'entités mobiles (les gendarmes ou les chercheurs), d'une autre entité mobile (le voleur ou le fugitif) qui se déplace dans un graphe. Etant donné un graphe, le problème consiste à déterminer le nombre minimum de gendarmes nécessaires pour capturer tout fugitif dans le graphe, et à calculer une *stratégie de capture* correspondante. A chaque étape d'une stratégie de capture, le graphe peut être divisé en deux parties : la partie *contaminée* est celle accessible par le fugitif, et la partie *propre* désigne la partie dans laquelle le fugitif ne peut pas se trouver. Nous étudions, en particulier, certaines contraintes qui peuvent être imposées aux stratégies. Par exemple, dans le cadre de certaines applications pratiques (comme la sécurisation de réseaux), il est raisonnable d'imposer que la partie propre soit connexe à chaque étape, afin d'assurer des communications sûres entre les gendarmes. Une autre contrainte à laquelle nous nous intéressons consiste à imposer, pour des raisons liées à la complexité algorithmique du problème, que la partie propre ne décroisse jamais. Dans cette thèse, nous nous intéressons principalement au problème des gendarmes et du voleur d'un point de vue algorithmique, et pour sa relation avec des paramètres importants liés aux décompositions des graphes.

Les fondements du problème

Comment secourir une personne perdue dans un réseau de souterrains, et ce de façon à ce que le moins de secouristes ne risquent leur vie ? Breish [Bre67], puis Parson [Par78a], ont traduit ce problème en termes mathématiques. La personne égarée circule dans un graphe, en suivant les arêtes du graphe, arbitrairement vite. Cette personne est *invisible*, c'est-à-dire que les secouristes ne peuvent la voir que lorsqu'ils la croisent sur une arête du graphe, ou lorsqu'ils occupent le même sommet qu'elle. Le problème qui se pose alors est de déterminer une stratégie de recherche de la personne égarée qui utilise le moins de secouristes possibles. Les jeux des gendarmes et du voleur dans les graphes étaient formalisés. Nous employons le pluriel car, comme nous le verrons, de nombreuses variantes de jeux impliquant une équipe de gendarmes et un voleur ont été étudiées. Ces variantes diffèrent par la vitesse de déplacement des gendarmes et du voleur, la visibilité que les gendarmes ont du voleur, les contraintes imposées à la partie propre du graphe, etc. Outre le sauvetage de spéléologues égarés et la capture de toutes sortes de fugitifs, les jeux des gendarmes et du voleur dans les graphes s'appliquent au "nettoyage" de réseaux contaminés, comme par exemple un réseau de pipeline envahit par un gaz toxique, ou bien un réseau de type Internet contaminé par un virus. Dans ce contexte, il n'y a plus de fugitif à proprement parler. Les gendarmes se déploient dans le réseau, en nettoyant les nœuds et les liens sur leur passage, et en évitant la recontamination qui provient des nœuds ou des liens encore contaminés. Nous verrons que cette vision des jeux des gendarmes et du voleur est équivalente à la variante définie par Breish et Parson.

Une des raisons pour laquelle les jeux des gendarmes et du voleur ont été étudiées dans les graphes vient du fait qu'elles fournissent un excellent cadre pour formaliser la manière dont des agents mobiles coopèrent pour venir à bout d'un objectif commun. Ces jeux ont fait l'objet de nombreuses études dans le domaine de l'intelligence artificielle et ont donné lieu à des applications pratiques dans des domaines aussi variés que la sécurité dans les réseaux informatiques ou la localisation de cibles mobiles par des robots.

D'un point de vue plus fondamental, les stratégies de capture dans les graphes trouvent une application dans le cadre de l'étude de la complexité des machines de Turing. Informellement, il existe un lien étroit entre les stratégies de capture et les jeux de jetons (*pebble games*). Ces derniers ont été utilisés pour modéliser des problèmes d'allocation de registres lors de calculs effectués par un processeur. Les jeux de jetons ont permis l'étude des compromis entre le temps nécessaire à une machine de Turing pour effectuer un calcul, et la place nécessaire pour réaliser ce même calcul.

La principale application de l'étude des jeux des gendarmes et du voleur dans les graphes est certainement la relation qu'elle entretient avec les décompositions de graphes définies dans le cadre de la théorie des mineurs de graphes développée par Robertson et Seymour dans les années 1980. Nous verrons que la relation entre stratégies de capture et décompositions de graphe a permis de prouver de nombreux résultats très importants dans le cadre de la théorie des graphes, tantôt en utilisant l'aspect stratégie de capture, tantôt en utilisant l'aspect décompositions.

Stratégies de capture

Comme nous l'avons évoqué, l'importance des stratégies de capture vient en majeure partie des liens étroits qu'elles entretiennent avec les décompositions de graphes. Cette relation se révèle au travers d'une propriété cruciale des stratégies de capture : la monotonie. Nous dirons qu'une stratégie est *monotone* si elle n'autorise aucune *recontamination*, i.e., la partie propre du graphe ne décroît jamais. A la fin des années 1980, un premier résultat majeur relatif aux stratégies de capture monotones d'un fugitif invisible fut obtenu par LaPaugh d'une part, et Bienstock et Seymour d'autre part, [BS91, LaP93], prouvant que “la recontamination n'aide pas”. En d'autres termes, il existe toujours une stratégie monotone qui capture un fugitif invisible dans un graphe en utilisant le nombre minimum de chercheurs. Ce résultat est véritablement crucial puisque les décompositions linéaires d'un graphe sont en correspondance univoque avec les stratégies monotones dans ce graphe [Kin92, EST94]. Dans le même temps, une autre variante des jeux des gendarmes et du voleur dans les graphes fut introduite par Seymour et Thomas [ST93] qui considèrent la poursuite d'un fugitif *visible*. Dans cette variante, les gendarmes voient la position du voleur à chaque étape de la stratégie. Seymour et Thomas ont prouvé la monotonie de cette variante, et sa correspondance avec les décompositions arborescentes des graphes.

Au début des années 2000, une nouvelle variante voit le jour. Celle-ci est motivée par certaines applications pratiques des stratégies de capture. Dans cette variante, le fugitif est invisible. Cependant, on impose que la partie nettoyée soit constamment connexe. Cette contrainte est motivée, entre autre, par le besoin d'assurer des communications sûres entre les chercheurs. Barrière *et al* [BFFS02] définissent ainsi les *stratégies de capture connexes*. La question de la monotonie de cette variante s'est bien sûr alors posée. Cependant, s'il est possible de réaliser des stratégies monotones et connexes utilisant le nombre minimum de chercheurs dans les arbres [BFFS02], Dyer *et al.* [YDA04] ont prouvé que ce n'est plus vrai dans le cas général : la monotonie a donc un coût dans le cadre connexe. Une seconde question relative aux stratégies connexes s'est posée : quelle est la proportion de chercheurs supplémentaires nécessaires pour capturer un fugitif dans un graphe, lorsque l'on impose à la stratégie d'être connexe ? Dans les arbres, Barrière *et al.* [BFST03] ont prouvé que deux fois plus de chercheurs suffisent toujours. Dans le cas général, Fomin *et al.* [FFT04] ont prouvé que le nombre de chercheurs suffisant pour capturer un fugitif dans un graphe de m arêtes est multiplié par au plus $O(\log m)$ lorsque l'on impose à la stratégie d'être connexe. Cependant, Barrière *et al.* [BFFS02] conjecturent que le résultat qu'ils ont obtenu dans le cas des arbres est en fait valide dans le cas général.

Contributions de la thèse

Cette thèse est consacrée à poursuivre l'étude des jeux des gendarmes et du voleur dans les graphes. Le document est divisé en trois parties dans lesquelles nous étudions différentes variantes des jeux des gendarmes et du voleur. Dans la première partie, nous définissons puis étudions une nouvelle variante qui établit un lien entre les stratégies de capture d'un fugitif invisible et les stratégies de capture d'un fugitif visible, et établit donc un pont entre décompositions linéaires et décompositions arborescentes. La seconde partie du document est consacrée à l'étude des stratégies de capture connexes, tout d'abord en considérant un fugitif invisible, puis un fugitif visible. La dernière partie du document présente l'étude des stratégies de capture connexes dans un contexte décentralisé. Ces trois parties sont précédées d'un chapitre préliminaire (chapitre 1) dans lequel nous développons trois des motivations principales du problème : l'étude de la coopération entre des agents mobiles pour mener à bien un but commun, l'étude, par l'intermédiaire de jeux de jetons, du compromis entre complexité temporelle et complexité spatiale des machines de Turing, et enfin la théorie des mineurs de graphes. La seconde section de ce même chapitre définit formellement les jeux des gendarmes et du voleur dans les graphes. La dernière section du chapitre consiste en un état de l'art des jeux des gendarmes et du voleur dans les graphes.

Résumé des principales contributions. La première partie de la thèse est constituée d'un unique chapitre (chapitre 2). Nous y définissons *les stratégies de capture non-déterministes* dans les graphes. Dans cette variante, le fugitif est invisible et arbitrairement rapide. Pour aider les chercheurs, nous autorisons cependant qu'ils puissent voir le fugitif un nombre limité de fois. Un paramètre $q \geq 0$ étant fixé, les chercheurs peuvent demander à connaître la position du fugitif au plus q fois lors de l'exécution de la stratégie de capture. Une telle stratégie est appelée *stratégie (non-déterministe) q-limitée*. Si $q = 0$, cette variante est strictement équivalente à celle définie par Parson et Breish, dans laquelle le fugitif est invisible. De même, si le paramètre q n'est pas borné, nous obtenons une variante équivalente à celle définie par Seymour et Thomas, dans laquelle le fugitif est visible constamment.

Nous définissons ensuite une version paramétrée des décompositions arborescentes des graphes. Informellement, une *décomposition arborescente q-branchée*, $q \geq 0$, d'un graphe est une décomposition arborescente de ce graphe telle que l'arbre de décomposition à une "hauteur de branchement" au plus q. Dans un premier temps, nous prouvons l'équivalence entre décomposition arborescente q-branchée et stratégie q-limitée monotone, pour tout $q \geq 0$. Cette relation nous permet de prouver que le problème de décision associé à celui qui consiste à déterminer une stratégie q-limitée monotone qui utilise le nombre minimum de chercheurs est un problème NP-complet. Dans un second temps, nous prouvons que, pour tout $q \geq 0$, les stratégies q-limitées vérifient la propriété de monotonie. C'est-à-dire que, pour tout graphe G et pour tout $q \geq 0$, il existe une stratégie de capture q-limitée monotone pour G qui utilise le nombre minimum de chercheurs.

Enfin, nous proposons un algorithme exponentiel exact qui, étant donné un graphe G et $q \geq 0$, calcule une stratégie q-limitée monotone qui utilise le nombre minimum de chercheurs pour G. Nous établissons également une relation entre le nombre minimum de chercheurs nécessaires à la capture d'un fugitif en posant au plus q questions, et le nombre minimum de chercheurs nécessaires à la capture d'un fugitif lorsqu'une question supplémentaire est autorisée. Plus précisément, nous prouvons qu'autoriser une question supplémentaire permet, au mieux, de diviser le nombre de chercheurs par deux.

La deuxième partie de cette thèse est consacrée à l'étude des stratégies de capture connexes. Elle est composée de deux chapitres. Le premier chapitre de cette partie (chapitre 3) s'attache à l'étude des stratégies de capture connexes d'un fugitif invisible. Le second chapitre de cette partie (chapitre 4) considère les stratégies de capture connexes d'un fugitif visible.

Plus précisément, dans le chapitre 3, nous étudions le rapport ρ entre le nombre minimum de chercheurs nécessaires à la capture connexe d'un fugitif invisible dans un graphe G, et le nombre minimum de chercheurs nécessaires à la capture d'un fugitif invisible dans G, sans imposer la contrainte de connexité. Nous définissons tout d'abord les décompositions arborescentes connexes d'un graphe. Puis, nous proposons un algorithme en temps polynomial qui transforme toute décomposition arborescente d'un graphe en une décomposition arborescente connexe, sans en augmenter la largeur. En utilisant la notion de décomposition arborescente connexe d'un graphe, nous prouvons que le rapport ρ est borné supérieurement par $\log n$ pour tout graphe (connexe) de n sommets. Nous utilisons également la notion de décomposition arborescente connexe pour borner supérieurement ρ dans la classe des graphes de cordalité et de largeur arborescente bornées. Ce résultat nous permet de borner ρ par $2\,(k+1)$ pour tout graphe chordal G, de largeur arborescente au plus k.

Le chapitre 4 est consacré à l'étude des stratégies de capture connexes d'un fugitif visible. Dans ce contexte, ρ désigne le rapport entre le nombre minimum de chercheurs nécessaires à la capture connexe d'un fugitif visible dans un graphe G, et le nombre minimum de chercheurs nécessaires à la capture d'un fugitif visible dans G, sans imposer la contrainte de connexité. Nous prouvons que ce rapport est borné supérieurement par $\log n$ dans la classe des graphes connexes de n sommets. Nous prouvons, de plus, que, pour tout $n_0 \geq 1$, il existe $n \geq n_0$ et un graphe G de n sommets tel que $\rho \geq C.\log n$ pour ce graphe, où C est une constante indépendante de G, de n_0 et de n. Enfin, nous prouvons que les stratégies de capture connexes d'un fugitif visible ne satisfont pas la propriété de monotonie. Plus exactement, nous prouvons que, pour tout $k_0 \geq 1$, il existe $k \geq k_0$ et un graphe G tels qu'il existe une stratégie de capture connexe d'un fugitif visible utilisant au plus k chercheurs, mais que toute stratégie de capture monotone et connexe d'un fugitif visible utilise au moins $k+1$ chercheurs. Nous proposons notamment un graphe de 56 sommets pour lequel $k = 4$.

La dernière partie de cette thèse est consacrée à la présentation de deux algorithmes répartis destinés au calcul de stratégies de capture monotones et connexes d'un fugitif invisible. Dans ce contexte, les chercheurs sont modélisés par des robots de mémoire $O(\log n)$ bits, qui sont lancés dans un graphe de n sommets qu'ils doivent nettoyer. La stratégie est déterminée localement par les chercheurs.

Plus précisément, dans le chapitre 5, nous considérons que les chercheurs n'ont aucune connaissance du graphe dans lequel ils sont lancés. Pour permettre aux chercheurs de se déplacer, le graphe est muni d'une orientation locale. Soient m le nombre d'arêtes du graphe, et n son nombre de sommets. Les sommets du graphe ne portent aucun identifiant, et sont munis d'une zone de mémoire locale de taille $O(m \log n)$ bits accessible par les chercheurs en lecture et en écriture. Nous proposons un algorithme réparti permettant à k chercheurs de réaliser une stratégie de capture connexe, dans tout graphe G et en partant d'un sommet v de G, où k est le nombre minimum de chercheurs nécessaires pour capturer un fugitif invisible de manière connexe et monotone dans G en partant de v. De plus, après l'exécution de l'algorithme, la description d'une stratégie de capture monotone et connexe utilisant k chercheurs est distribuée sur les zones de mémoire locale des sommets de G. Si les chercheurs ne sont pas synchronisés, au plus un chercheur supplémentaire est nécessaire et suffisant. Notre stratégie de capture s'exécute en temps exponentiel, mais le problème de décision associé est NP-difficile.

Dans le chapitre 6, nous nous proposons d'aider les chercheurs en leur fournissant de l'information à propos du graphe dans lequel ils sont lancés. Pour cela, nous utilisons la notion de conseil définie par Fraigniaud *et al.* [FIP06]. Informellement, il s'agit d'une mesure de la quantité d'information minimale qu'il est nécessaire d'apporter à des chercheurs pour leur permettre d'accomplir, de manière distribuée et "efficacement", une tâche dans un graphe. Cette information est définie sous forme d'une chaîne de bits qu'un oracle distribue sur les sommets du graphe. Sur chaque sommet, les chercheurs peuvent accéder à ces bits d'information et s'en servir pour déterminer et exécuter la stratégie de capture. Nous proposons un algorithme réparti utilisant un nombre de bits d'information au plus $O(n \log n)$, permettant à k chercheurs de réaliser une stratégie de capture monotone et connexe, dans tout graphe G et en partant d'un sommet v de G, où k est le nombre minimum de chercheurs nécessaires pour capturer un fugitif invisible de manière connexe et monotone dans G en partant de v. Enfin, nous prouvons qu'il n'existe pas d'algorithme réparti utilisant $o(n \log n)$ bits d'information et permettant à k chercheurs de réaliser une stratégie de capture monotone et connexe, dans tout graphe G.

Nous concluons cette thèse en proposant quelques perspectives (chapitre 7).

Chapitre 1

Préliminaires

Dans ce chapitre, nous présentons quelques unes des motivations de l'étude des jeux des gendarmes et du voleur dans les graphes (cf., section 1.1). Puis, nous formalisons les notions qui seront utilisées dans cette thèse (cf., section 1.2). Enfin, nous faisons un tour d'horizon de l'étude des jeux des gendarmes et du voleur dans les graphes (cf., section 1.3).

1.1 Motivations

Dans cette section, nous passons en revue quelques domaines dans lesquels les jeux des gendarmes et du voleur ont fait l'objet de nombreuses études. Dans la section 1.1.1, nous décrivons l'application de ces jeux dans le cadre de l'intelligence artificielle, et, plus précisément, de l'auto-coordination d'équipes d'agents mobiles coopérant pour atteindre un but commun. La section 1.1.2 s'attache à décrire deux jeux de jetons très étudiés pour établir des compromis de complexité temps/espace pour les machines de Turing. Nous verrons ainsi le lien qui existe entre jeux des gendarmes et du voleur et jeux de jetons. Enfin, la section 1.1.3 décrit les grandes lignes de la théorie des mineurs de graphes de Robertson et Seymour, et donne une première idée de la dualité forte qui lie les jeux des gendarmes et du voleur et les décompositions de graphe.

1.1.1 Auto-coordination d'agents mobiles

Trouver une personne se déplaçant dans un environnement, connu ou non, est une tâche à laquelle la plupart d'entre nous ont été confrontés. Dans un supermarché, par exemple, il n'est pas rare de devoir chercher une personne qui, après avoir dit qu'elle allait voir tel ou tel rayon, ne reparaît plus. Dans un tout autre contexte, les opérations militaires consistent souvent à traquer une cible mouvante dans un territoire hostile, ou bien à sécuriser une zone. Dans de tels cas, la tâche pourrait être dévolue à une équipe de robots coopérant dans le but de mener à bien leur mission. C'est donc très naturellement, que LaValle et Hutchinson [LH93] ont fait du problème de *poursuite-évasion*, également appelé problème de *proie-prédateurs*, un cadre de travail très général pour étudier la coordination d'une équipe d'agents mobiles.

De manière générale, nous désignons par *agent mobile*, toute entité mobile et autonome vis à vis d'un opérateur humain, c'est-à-dire une entité physique mobile dont les décisions sont issues d'un programme embarqué, par exemple un robot explorant un territoire inconnu et inaccessible par des humains, ou des agents logiciels évoluant dans un réseau de type internet. Dans cette section, nous appellerons un *environnement continu polygonal*, un environnement-2D dont les bords sont délimités par des polygones (voir, par exemple, la figure 1.1). Formellement, le but du problème de poursuite-évasion est de localiser un intrus, dont le point de départ est inconnu et qui peut se déplacer arbitrairement vite et de manière imprévisible dans un environnement continu polygonal [LLG+97, GLL+97]. Les agents mobiles peuvent disposer d'une perception totale (perception circulaire sur 360 degrés) ou partielle (perception circulaire restreinte sur $\alpha > 0$ degrés). Dans ce contexte, les agents se déplacent dans l'environnement pour parvenir à détecter l'intrus. Un problème fondamental consiste à minimiser le nombre d'agents nécessaires pour détecter l'intrus à coup sûr.

Ce problème peut être mis en relation avec d'autres problèmes similaires et intensivement étudiés dans le cas de la coopération entre agents. Citons comme exemples le problème de la galerie d'art (*art gallery problem*) [O'R87] et celui de la ronde du gardien (*watchman tour problem*) [CN88]. Dans le premier cas, le problème consiste à déterminer les positions (fixes) d'un nombre minimum de gardiens de musée, de façon à ce que tout point du musée soit visible d'au moins un gardien. Le second problème vise à déterminer le circuit que doit suivre un unique gardien de manière à ce qu'il puisse voir tous les points du musée au moins une fois.

LaValle et Hutchinson [LH93] motivent leur choix du problème de poursuite-évasion, comme cadre général pour étudier la coordination d'une équipe d'agents mobiles, par les raisons suivantes. Ce problème fournit un formalisme approprié pour les diverses tâches que doivent réaliser les agents : rassembler des informations, prendre une décision qui vise un but commun, agir en fonction de la décision prise, etc. Le problème de poursuite-évasion permet de formaliser un compromis entre la planification et le contrôle. En effet, la stratégie d'un joueur peut être décidée à l'avance ou bien en cours de partie. Chaque joueur peut ainsi déterminer (i.e., planifier) sa propre stratégie à l'avance, celle-ci se déroulant alors sans tenir compte des informations apprises au cours du jeu. Les agents peuvent aussi structurer les informations acquises au fur et à mesure du jeu, de manière à orienter leur stratégie (i.e., la contrôler). Considérant un problème auquel sont confrontés des agents mobiles qui doivent coopérer pour atteindre un objectif commun, LaValle et Hutchinson proposent un formalisme qui prend en compte l'*espace des configurations* qui représente la position des agents, un *ensemble d'actions* que peut effectuer chaque agent à une étape donnée, une *fonction de transition* dans l'espace d'états des agents, un *espace de mesure* qui représente ce que perçoivent les agents, une *fonction de mesure* allant de l'espace des configurations dans l'espace de mesure, une *structure d'information* qui est un ensemble rassemblant toutes les informations acquises par un agent entre la première phase et la phase courante, un *espace des stratégies* qui est une fonction allant de la structure d'information vers l'ensemble des actions possibles, et enfin une fonction de coût qui pondère les actions possibles de l'ensemble des agents à la phase courante.

Ce formalisme établi par le biais du jeu de poursuite-évasion forme ainsi une sorte de paradigme fédérateur qui permet d'étudier les problèmes de coopération entre agents dont on peut lister de nombreuses applications comme le contrôle du trafic aérien [BO95], les stratégies militaires [Isa65], la localisation d'une cible mobile, la surveillance d'un immeuble par des caméras mobiles, etc. Pour un tour d'horizon des applications des agents logiciels mobiles, voir [PK98]. Citons, en particulier, une application naturelle du problème de poursuite-évasion : la sécurité dans les réseaux informatiques. L'intrus peut consister en un virus informatique qui contamine les liens et les nœuds d'un réseau, en se propageant à travers les liens. Des agents logiciels peuvent être utilisés pour nettoyer le réseau, tout en le sécurisant, c'est-à-dire en évitant que le virus ne se propage d'avantage.

Pour conclure ce petit tour d'horizon lié aux agents mobiles, remarquons que l'étude des stratégies de capture dans les graphes a permis de démontrer plusieurs résultats dans le cadre de la poursuite-évasion dans un environnement continu. Guibas *et al.* [GLL+97] prouvent notamment que pour tout graphe planaire G, il existe un environnement continu polygonal F tel que le problème de poursuite-évasion avec perception totale peut être résolu par k agents dans F si et seulement si il existe une stratégie de capture d'un fugitif invisible utilisant k agents dans G. Ceci permet d'établir des bornes sur le nombre minimum d'agents nécessaire pour résoudre le problème de poursuite-évasion. Par exemple, $\Theta(\sqrt{h}+\log n)$ agents sont nécessaires et suffisants pour résoudre le problème de poursuite-évasion dans un environnement polynomial avec n bords et h trous [GLL+97]. Dans l'introduction de cette thèse, nous avons déjà noté l'importance des stratégies de capture monotones dans les graphes. Il est surprenant de remarquer qu'il existe un environnement polygonal F très simple, pour lequel un agent suffit à résoudre le problème de poursuite-évasion, mais qui nécessite un nombre de recontaminations linéaire en le nombre de bords de F [GLL+97]. Dans cet environnement représenté sur la figure 1.1, le problème de poursuite-évasion peut être résolu par un agent et quatre étapes de recontamination.

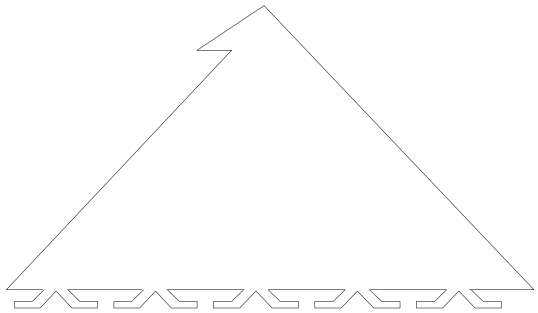

FIG. 1.1 – Environnement polygonal

1.1.2 Jeux de jetons et compromis temps/espace

Nous présentons dans cette section deux *jeux de jetons* (*pebble games*) qui ont principalement été introduits pour modéliser l'allocation de registres dans un processeur. Le premier de ces jeux n'implique que des jetons monochromes, disons noirs. Le second est une variante du premier qui implique des jetons de deux couleurs, disons blancs et noirs. Nous expliquerons le lien entre ces jeux et les jeux des gendarmes et du voleur.

Etant donné un graphe orienté et acyclique, le but du premier jeu est d'arriver à une situation dans laquelle tous les sommets du graphe ont été occupés par un jeton, tout en respectant les trois règles suivantes :

- un jeton peut être retiré d'un sommet à tout moment ;
- un jeton peut être placé sur un sommet si tous ses prédécesseurs sont occupés par des jetons ;
- si chaque prédécesseur d'un sommet est occupé par un jeton, un de ces jetons peut être déplacé sur ce sommet.

L'espace requis par le jeu est le nombre maximum de jetons utilisés. Sethi [Set73] a montré que le problème de déterminer le nombre minimum de jetons pour résoudre ce jeu est NP-difficile, et Gilbert *et al.* [GLT79] l'ont prouvé PSPACE-complet, en modifiant un résultat de Lingas [Lin78]. Pour tous graphes de n sommets, $O(\frac{n}{\log n})$ jetons sont suffisants pour résoudre le jeu, et cette borne est atteinte [PTC76].

La principale application de ce jeu vient de ce qu'il modélise l'allocation des registres lors des calculs réalisés par un processeur [Set73]. Intuitivement, chaque sommet du graphe représente une valeur qui est calculée à partir de la valeur des prédécesseurs du sommet. Les sources (i.e., les sommets sans prédécesseurs) représentent les valeurs d'entrée. Chaque jeton représente un registre. Ces jeux trouvent de nombreuses autres applications. Par exemple, ils ont également été appliqués à l'étude des schémas récursifs [PH70].

Une autre mesure peut être prise en compte dans l'étude de ce jeu de jetons. Le *temps* requis est le nombre d'applications des règles précédentes. Ainsi, il est possible d'étudier le compromis entre le nombre de jetons et le temps requis pour gagner le jeu. Ceci permet à ce jeu d'être utilisé pour analyser les compromis temps-espace des machines de Turing [Coo73]. Ainsi, des compromis temps-espace concernant plusieurs importantes notions algorithmiques en ont été déduits : récursion linéaire [Cha72], transformation de Fourier rapide [SS79, Tom78], multiplication de matrices [Tom78], etc.

Lengauer et Tarjan [LT82] ont prouvé plusieures bornes concernant le compromis temps-espace. Notamment, pour une classe spéciale de graphes de permutation qui implémente l'inversion de bits, ils montrent que $S * T = \Theta(n^2)$, avec n la taille du graphe, S le nombre de jetons utilisés et T le temps de calcul.

Cook et Sethi [CS74] définissent une variante de ce jeu, en ajoutant des jetons blancs et les trois règles suivantes :

- un jeton blanc peut être placé sur un sommet libre à tout moment ;
- un jeton blanc peut être supprimé d'un sommet si tous ses prédécesseurs sont occupés ;
- si un jeton blanc occupe un sommet dont tous les prédécesseurs sont occupés, sauf un, alors le jeton peut être déplacé sur le prédécesseur inoccupé.

Initialement, le graphe est vide. Le but est de terminer avec un graphe vide après que tous les sommets ont été occupés. Les jetons blancs représentent des suppositions non-déterministes réalisées en cours de calcul. Lengauer et Tarjan [LT82] prouvent, entre autre, que, dans la classe des graphes de permutation, le compromis temps-espace devient :

$T = \Theta(n^2/S^2) + \Theta(n)$.

Une version monotone des jeux de jetons a été définie. Dans ce cas, chaque sommet du graphe ne peut être occupé par un jeton qu'une seule fois. Il est connu [LT79] que cette contrainte augmente sensiblement le nombre de jetons nécessaires pour gagner le jeu.

Kirousis et Papadimitriou [KP86] prouvent qu'il existe une étroite relation entre les jeux de jetons et les stratégies de capture dans les graphes. Ainsi, ils définissent l'indice d'échappement-sommet d'un graphe G comme le nombre minimum de chercheurs pour capturer un fugitif invisible dans G (cf., section 1.2.4), et prouvent que l'indice d'échappement-sommet d'un graphe G est égal au nombre de jetons minimum, parmi toutes les orientations acycliques D de G, qui sont nécessaires pour gagner dans D de manière monotone. Notons que ce résultat est valide à la fois pour le jeu de jetons monochromes et pour le jeu de jetons bichromatiques. Dans le cadre du jeu de jetons noirs et blancs, ils prouvent également que si on associe à un graphe G une orientation D de degré entrant au plus k, le nombre de jetons minimum qui sont nécessaires pour gagner dans D de manière monotone est au plus $k+1$ fois l'indice d'échappement-sommet de G.

1.1.3 Théorie des mineurs de graphes

Au début des années 1980, Neil Robertson et Paul Seymour initient la publication d'une série d'articles, intitulés "*Graph minors*" [RS83, RS04], dont les derniers ont été publiés ces dernières années (voir [Lov06, RS85] pour un tour d'horizon). L'impact du désormais célèbre *théorème structurel de Robertson et Seymour* est considérable dans de nombreux domaines, dont la théorie des graphes et l'algorithmique [MiK07].

Nous introduisons tout d'abord quelques éléments de terminologie. Un graphe H est un *mineur* d'un graphe G, si H est un sous-graphe d'un graphe obtenu à partir de G par une succession de contractions d'arêtes. Une famille $\mathcal{F}$ de graphes est dite *invariante par mineurs*, ou *close par mineurs*, si tout mineur d'un graphe appartenant à $\mathcal{F}$ appartient également à $\mathcal{F}$. Par exemple, la famille des graphes planaires ou celle des graphes k-coloriables sont des familles invariantes par mineurs. Un graphe G fait partie de l'ensemble des *obstructions* d'une famille $\mathcal{F}$ de graphes si $G \notin \mathcal{F}$ et si G est minimal pour la relation de minoration, c'est-à-dire que pour tout mineur H de G, $H \in \mathcal{F}$.

Le but de Robertson et Seymour était de répondre aux deux questions fondamentales suivantes [RS85] :

- **Conjecture de Wagner** (1937) : Soit $G_i, i \geq 1$, une séquence infinie de graphes indexés par les entiers positifs. Il existe $i < j$ tels que G_i est isomorphe à un mineur de G_j.
- **Problème des chemins disjoints** : Soit $k > 0$. Existe-t-il un algorithme en temps polynomial qui décide si, étant donné un graphe G, et des sommets $s_1, \cdots, s_k$ et $t_1, \cdots, t_k$, il existe k chemins sommet-disjoints $P_1, \cdots, P_k$, tel que P_i relie s_i à t_i pour tout i, $1 \leq i \leq k$?

[RS95] et [RS04] répondent par l'affirmative aux deux questions. Un corollaire important de la conjecture de Wagner implique que l'ensemble des obstructions d'une famille close par mineurs est fini. Ce résultat peut être illustré par le fameux théorème de Kuratowski [Kur30], selon lequel un graphe est planaire si et seulement si il n'admet ni K_5, ni $K_{3,3}$ comme mineur.

Ce corollaire nous permet d'expliquer un aspect important des résultats obtenus par Robertson et Seymour. En effet, tester si un graphe G appartient à une famille $\mathcal{F}$ close par mineurs revient à vérifier que G n'admet aucune obstruction de $\mathcal{F}$ comme mineur. Or, la réponse au problème des chemins disjoints implique justement que, étant donné un graphe H, il existe un algorithme en temps $O(n^3)$ qui décide si H est un mineur d'un graphe G de n sommets [RS95]. Le théorème suivant en découle :

Théorème 1 (Robertson et Seymour) *Soit $\mathcal{F}$ une famille close par mineurs. Il existe un algorithme en temps polynomial qui décide si un graphe G appartient à $\mathcal{F}$.*

Apportons quelques bémols à ce théorème. En effet, bien que fini, le nombre d'obstructions d'une famille close par mineurs peut être très grand. Par exemple, Thilikos [Thi00] a exhibé les 57 obstructions de la famille des graphes de *linear width* au plus deux. Glover *et al* [GHW79] ont déterminé un ensemble de 103 obstructions à la famille des graphes plongeables sur le plan projectif, Archdeacon [Arc80] ayant prouvé que cette liste était complète. De plus trouver toutes ces obstructions se révèle être un challenge dans la plupart des cas. Par exemple, trouver toutes les obstructions aux familles des graphes plongeables sur un tore (plusieurs milliers d'obstructions sont déjà connues dans ce cas), ou sur la bouteille de Klein, restent des problèmes ouverts. D'autre part, étant donné un graphe H, le "O" de la complexité $O(n^3)$ de l'algorithme vérifiant si un graphe admet H comme mineur cache une constante dépendant de H qui peut être très grande. Ceci relativise la portée pratique du théorème 1.

Pour comprendre le lien entre les stratégies de capture dans les graphes et la théorie des mineurs, expliquons informellement le théorème structurel dont découle la validité de la conjecture de Wagner. L'idée initiale est que les graphes les plus "simples" sont les arbres (graphes acycliques). Ainsi, dès 1960, Kruskal a prouvé que la conjecture de Wagner était valide dans le cas des arbres [Kru60]. Pour tirer partie de ce fait, Robertson et Seymour prouvent que tout graphe qui exclue un graphe H comme mineur se décompose sous forme d'un "puzzle" dont les "pièces" s'organisent en une structure arborescente. Sans entrer dans les détails, disons seulement que chaque "pièce" est un graphe qui, en supprimant un nombre fini de sommets, est "presque" plongeable sur une surface dans laquelle H ne se plonge pas [RS03a]. L'idée de la preuve du théorème de Wagner consiste donc à généraliser le théorème de Kuratowski [RS90b] pour chaque "pièce", puis celui de Kruskal à la structure arborescente [RS90a, RS03b]. Cette structure particulière des graphes excluant un mineur a permis le développement de nombreux algorithmes pour divers problèmes NP-complet : problème du voyageur de commerce, couverture de cardinalité minimum, coloration des sommets [ALS91], etc. Lorsque les instances de ces problèmes sont restreintes à des classes de graphes excluant un mineur fixé, ces algorithmes ont une complexité polynomiale [DHT05, DFHT05].

Dans le cadre de leur étude, Robertson et Seymour ont défini la *largeur arborescente* (pour *treewidth* en anglais) d'un graphe, ainsi que la décomposition associée : la *décomposition arborescente* [Bod98, RS86, RS91]. Toutes les notions et résultats énoncés ci-dessous seront définis formellement aux sections 1.2.5 et 1.3.2. Informellement, la largeur arborescente mesure la proximité d'un graphe avec un arbre. Nous verrons dans les prochaines sections qu'une décomposition arborescente d'un graphe correspond à une stratégie de capture monotone d'un fugitif visible dans ce graphe. Ainsi, la largeur arborescente d'un graphe est égale (à un près) au nombre minimum de chercheurs nécessaires pour capturer un fugitif visible dans le graphe. Cette dualité a notamment permis de définir la notion de *buissons* (*brambles*, nommé d'après Reed [Ree97]). Utilisant le fait qu'un buisson constitue une stratégie de fuite pour un fugitif, Seymour et Thomas [ST93] prouvent que posséder un grand buisson interdit à un graphe d'être de petite largeur arborescente. Par exemple, si un graphe G admet une grille comme mineur, sa largeur arborescente est au moins la largeur (au sens usuel) de la grille. En effet, si un fugitif peut s'enfuir dans la grille face à une équipe de chercheurs, il lui suffit d'appliquer la même stratégie dans G face au même nombre de chercheurs.

Au delà de la théorie des mineurs, la décomposition arborescente a eu un impact très important en algorithmique [Bod93, Bod97, Bod06]. En effet, de nombreux problèmes NP-complets deviennent polynomiaux, voire même linéaires, lorsque leurs instances sont restreintes à des classes de graphes de largeur arborescente bornée. Citons le théorème de Courcelle [CM93], selon lequel tout problème exprimable en logique monadique du second ordre peut être résolu en temps linéaire dans des classes de graphes de largeur arborescente bornée. Bodlaender et Kloks [Bod96, BK96] proposent un algorithme linéaire qui calcule une décomposition arborescente de tout graphe de largeur arborescente bornée. Robertson et Seymour [RS91] ont également défini la notion de décomposition en branche d'un graphe. Comme nous le verrons, cette décomposition est très proche de la décomposition arborescente. Hicks [Hic04] propose un algorithme en temps linéaire pour déterminer si un graphe G admet comme mineur un graphe H (fixé), pour tout graphe G de largeur de décomposition en branche bornée. La plupart de ces algorithmes utilisent la programmation dynamique [Bod88].

1.2 Problématique

Nous présentons dans cette section la terminologie que nous utiliserons dans la thèse. La section 1.2.1 est consacrée à la terminologie classique des graphes. Dans la section 1.2.2, nous définissons les principales notions relatives aux jeux des gendarmes et du voleur dans les graphes, et nous précisons le but que poursuivent ces jeux. Dans la section 1.2.3, nous établissons plusieurs classifications de ces jeux en fonctions des principaux critères qui les caractérisent. Ces classifications se concentrent progressivement vers les seuls jeux auxquels nous nous attacherons dans cette thèse. Dans la section 1.2.4, nous formalisons plus précisément le jeu originellement défini par Parson ainsi que les jeux qui en ont été dérivés, ceux que nous allons étudier au cours de cette thèse. Enfin, la section 1.2.5 s'attache à la définition de certaines décompositions de graphe dont nous allons beaucoup nous servir par la suite.

1.2.1 Terminologie

L'essentiel de la terminologie, concernant les graphes, utilisée dans cette thèse est celle employée dans le livre de Claude Berge [Ber73].

Un *graphe* G est constitué d'un ensemble de *sommets*, noté $V(G)$ (ou V s'il n'y a pas d'ambiguïté) et d'un ensemble d'*arêtes*, noté $E(G) \subseteq V \times V$ (ou E s'il n'y a pas d'ambiguïté). Nous ne considèrerons que des graphes *simples*, c'est-à-dire sans boucle ni arête multiple, et *finis*. Dans la suite de cette thèse, nous noterons $n = |V(G)|$, le nombre de sommets du graphe G, et $m = |E(G)|$, son nombre d'arêtes. Etant donné un graphe $G = (V, E)$, le graphe H est un *sous-graphe* de G si $V(H) \subseteq V$ et $E(H) \subseteq E$. Un graphe G est un *sur-graphe* d'un graphe H si H est un sous-graphe de G. Etant donné un ensemble $S \subseteq V$ de sommets de G, nous notons $G[S]$ le *sous-graphe de G induit par S*, c'est-à-dire le graphe H tel que $V(H) = S$ et $E(H) = \{\{x, y\} \in E \mid x \in S \textit{ et } y \in S\}$. Etant donné un ensemble d'arêtes $F \subseteq E$, nous notons $G[F]$ le sous-graphe de G induit par F, c'est-à-dire le graphe H tel que $V(H) = \{x \in V \mid \exists e \in F\ ,\ x \in e\}$ et $E(H) = F$. Un graphe H est un *mineur* d'un graphe G, si H est un sous-graphe d'un graphe obtenu à partir de G par une succession de contractions d'arêtes. La distance entre deux sommets x et $y \in V(G)$, dans un sous-graphe H de G est notée $\mathrm{dist}_H(x, y)$. Nous noterons $\deg(v)$ le degré d'un sommet v. La cordalité d'un graphe G est égale à la longueur du plus grand cycle induit par G, c'est-à-dire la longueur du plus grand cycle sans corde de G. Un graphe *cordal* est un graphe de cordalité au plus 3. Etant donnés deux ensembles $A, B \subseteq E$, la frontière $\delta(A, B)$ entre ces deux ensembles est l'ensemble des sommets incidents à une arête de A et à une arête de B. Soit $F \subseteq E$, nous posons $\delta(F) = \delta(F, E \setminus F)$. Etant donné un ensemble S de sommets de G, nous définissons de même la frontière $\delta(S)$ de cet ensemble comme l'ensemble des arêtes dont l'une des extrémités appartient à S et l'autre non. Etant donné $v \in V$, $N(v)$ est l'ensemble des voisins de v, et $N[v] = N(v) \cup \{v\}$.

Certaines classes de graphes vont être souvent mentionnés au cours de cette thèse. Soit $k > 0$. Nous noterons $P_k = \{v_0, \cdots, v_{k-1}\}$, la *chemin* de k sommets, et C_k, le *cycle* de k sommets. K_k dénotera la *clique* de k sommets, c'est-à-dire le graphe complet de k sommets. Soit $a, b > 0$. $K_{a,b}$ dénotera le graphe biparti complet constitué d'une partition de a sommets et d'une seconde partition de b sommets. Un graphe d'intervalle est un graphe qui peut être représenté de la façon suivante. Chaque sommet du graphe est associé à un intervalle de $\mathbb{R}$, et il existe une arête entre deux sommets si les intervalles correspondants sont d'intersection non vide.

1.2.2 Jeux des gendarmes et du voleur

Dans cette section, nous présentons les principes des jeux des gendarmes et du voleur dans les graphes, ainsi que les principales notions associées. Nous en déduisons une classification de ces jeux en fonction des capacités relatives des différents protagonistes : les gendarmes (ou chercheurs) et le voleur (ou fugitif). Les *jeux des gendarmes et du voleur*, également appelé *jeux de capture*, traitent tous du même problème : la localisation et la capture d'un fugitif dans un graphe, par une équipe de chercheurs. Le fugitif est contraint de se déplacer en suivant les arêtes du graphe. Il peut se déplacer d'un sommet à un autre s'il existe un chemin entre ces deux sommets et qu'aucun sommet ni arête de ce chemin n'est occupé par un chercheur.

Une première discrimination qui peut être faite entre les différents jeux des gendarmes et du voleur découle de la manière dont les chercheurs se déplacent dans le graphe. Selon les cas, le fugitif n'est pas capturé de la même façon. Deux types de mouvements sont possibles pour les chercheurs : le long des arêtes ou en sautant d'un sommet arbitraire à un autre. Dans tous les cas, le fugitif est capturé lorsqu'il occupe le même sommet ou la même arête qu'un chercheur. Dans certains cas que nous expliciterons par la suite, le fugitif est également capturé s'il occupe une arête dont les deux extrémités sont occupées par des chercheurs.

D'autres moyens de capturer le fugitif ont été étudiés dans la littérature, mais nous ne les considérerons pas dans cette thèse. Citons, par exemple, un modèle introduit par Fomin, Kratsch et Müller [FKM03] selon lequel le fugitif est capturé s'il occupe un sommet dominé par un sommet occupé par un chercheur.

Une *stratégie* pour les chercheurs (resp., pour le fugitif) est une séquence de mouvements des chercheurs (resp., du fugitif). Chaque mouvement d'une stratégie est appelé une *étape* de cette stratégie. Une *stratégie de capture* est une stratégie pour les chercheurs qui aboutit à la capture du fugitif quelle que soit la séquence de mouvements effectuée par ce dernier, c'est-à-dire une stratégie telle que, pour toute stratégie du fugitif, il existe une étape à laquelle le fugitif est capturé. En d'autres termes, une stratégie de capture est une stratégie gagnante pour les chercheurs. Une *stratégie d'échappement* est une stratégie pour le fugitif qui lui assure de n'être jamais capturé, quelle que soit la stratégie des chercheurs. En d'autres termes, une stratégie d'échappement est une stratégie gagnante pour le fugitif.

Soit G un graphe et soit S une stratégie pour les chercheurs dans ce graphe. A une étape donnée de S, un sommet (ou une arête) de G est *propre*, si le déroulement de la stratégie jusqu'à cette étape assure que le fugitif n'occupe pas ce sommet (cette arête) à cette étape, sans qu'il n'ait été capturé. La *partie propre* du graphe est le sous-graphe composé des sommets et des arêtes propres. Respectivement, un sommet (ou une arête) de G est *contaminée* s'il (elle) n'est pas propre. La *partie contaminée* du graphe est donc le complémentaire de la partie propre. Si il existe une étape de la stratégie telle qu'un sommet (ou une arête) du graphe qui était propre redevient contaminé(e), nous dirons qu'il y a eu *recontamination*.

Comme le lecteur l'a certainement remarqué, le champ lexical du nettoyage est omniprésent dans la terminologie précédente. Cela s'explique par la remarque suivante. Capturer un fugitif invisible et arbitrairement rapide est équivalent au nettoyage d'un graphe dont les arêtes et les sommets auraient été contaminés (par exemple, un réseau de pipeline envahi par un gaz toxique [Par78a]). C'est pourquoi, dans la suite, nous emploierons indifféremment la terminologie relative à la capture d'un fugitif ou au nettoyage d'un graphe contaminé. Si, à une étape d'une stratégie, le mouvement d'un chercheur entraîne qu'un sommet ou une arête qui était contaminé devient propre, nous dirons que le chercheur a *nettoyé* ce sommet ou cette arête. Nous dirons également qu'un chercheur qui occupe un sommet sur le chemin entre un sommet v (resp., une arête e) propre et un sommet ou une arête contaminé, *garde* ce chemin, ou *préserve* v (resp., e) de la recontamination.

Dans le cadre de notre étude, nous ne considérons que les jeux qui permettent au fugitif de voir les chercheurs à tout moment. Le cas où le fugitif ne connaît pas la position des chercheurs, ni ne voit leur déplacement peut en effet être traité par un chercheur effectuant une marche aléatoire [AKL+79, ARS+03]. Dans le cas d'un chercheur invisible, le problème qui est posé est de minimiser le temps au bout duquel le fugitif sera capturé. Dans [ARS+03], Adler *et al.* propose une stratégie de capture utilisant un chercheur et dont le nombre d'étapes est $O(n \log D)$ en moyenne, pour tout graphe G de n sommets et de diamètre D. Dans la suite de cette thèse, nous considérerons que le fugitif connaît la position de tous les chercheurs à tout moment et qu'il voit les mouvements des chercheurs au moment où ils sont effectués.

Remarquons que pour tout graphe G de n sommets, il existe un entier $k > 0$ tel qu'il existe une stratégie de capture dans G impliquant au plus k chercheurs. Si les chercheurs ne peuvent pas se déplacer le long des arêtes, la stratégie consiste simplement à placer un chercheur sur chaque sommet de G. Sinon, il suffit de placer $n-1$ chercheurs sur $n-1$ sommets distincts, le n^{ieme} chercheur parcourant l'ensemble des arêtes G entre ces $n-1$ sommets, et, pour finir, les chercheurs situés sur les sommets adjacents au dernier sommet contaminé se déplacent vers celui-ci. Le problème qui se pose consiste donc à déterminer quel est le plus petit entier $k > 0$ tel qu'il existe une stratégie impliquant k chercheurs.

Définition 1 *Etant donné un graphe G,* l'indice d'échappement *du graphe G est le plus petit entier k tel qu'il existe une stratégie de capture dans G impliquant au plus k chercheurs.*

Intuitivement, plus l'indice d'échappement d'un graphe est grand, plus le fugitif a de libertés pour s'échapper. Bien entendu, l'indice d'échappement d'un graphe dépend de la variante du jeu considérée. Etant donné un graphe G, une stratégie de capture qui implique un nombre de chercheurs égal à l'indice d'échappement de G (pour la variante considérée) est dite *optimale*.

Mentionnons que d'autres critères que le nombre de chercheurs ont été étudiés dans la littérature. Citons par exemple la minimisation du *coût* d'une stratégie [FG00], défini comme la somme, sur toutes les étapes, du nombre de chercheurs présent dans le graphe à chaque étape. Un autre exemple consiste en la minimisation du temps durant lequel un chercheur occupe un sommet au cours d'une stratégie [FHT04]. Dans cette thèse, nous ne considérerons que le problème qui consiste à minimiser l'indice d'échappement d'un graphe.

1.2.3 Classifications

Dans cette section, nous établissons plusieurs classifications des jeux des gendarmes et du voleur. Pour ce faire, nous commençons par définir plusieurs critères qui vont nous permettre de différencier les multiples variantes de ces jeux : la visibilité du fugitif, la vitesse de déplacement du fugitif, et la simultanéité ou l'alternance des mouvements des chercheurs et du fugitif.

Un premier critère auquel il sera beaucoup fait référence dans cette thèse, concerne la visibilité du fugitif. Le fugitif est dit *visible* si, à chaque étape d'une stratégie, les chercheurs connaissent la position que le fugitif occupe à cette étape. Sinon, il est dit *invisible*. Bien entendu, les jeux qui visent à la capture d'un fugitif visible sont plus favorables aux chercheurs que ceux qui considèrent un fugitif invisible. Il est facile de se convaincre que, pour tout graphe G, l'indice d'échappement pour capturer un fugitif visible dans G est au plus l'indice d'échappement pour capturer un fugitif invisible dans G. Par exemple, dans la classe des arbres, il existe une stratégie de capture d'un fugitif visible qui implique au plus un nombre constant de chercheurs : il suffit aux chercheurs de suivre le fugitif et de l'acculer dans une feuille. Au contraire, si le fugitif est invisible, l'indice d'échappement peut devenir logarithmique en la taille de l'arbre [MHG$^+$88].

La vitesse de déplacement du fugitif permet également de distinguer des variantes des jeux des gendarmes et du voleur. Ainsi, à chaque étape d'une stratégie, soit le fugitif peut se déplacer vers n'importe quel sommet ou arête du graphe (dans la mesure où il ne croise pas de chercheur sur son chemin), soit ses déplacements sont restreints à une distance bornée de sa position courante. Si la vitesse du fugitif est bornée, nous dirons que le fugitif est *lent*, sinon il sera dit *rapide*.

Un dernier critère qui permet d'établir une classification des différentes variantes du jeu des gendarmes et du voleur concerne la simultanéité, ou l'alternance, des mouvements des chercheurs et du fugitif : soit les chercheurs et le fugitif se déplacent simultanément, soit ils se déplacent chacun à leur tour. Il est facile de se convaincre que le cas tour-à-tour est plus favorable aux chercheurs que le cas où des déplacements simultanés sont autorisés. Par exemple, dans le cas d'une clique, un chercheur est suffisant pour capturer un fugitif dans le cadre des jeux tour-à-tour, alors qu'il faut autant de chercheurs que la clique a de sommets dans le cas des déplacements simultanés (cf., théorème 2 de la section 1.2.4).

Aux vues des trois critères que nous venons d'énoncer, il apparaît que certaines variantes de ces jeux n'ont pas vraiment d'intérêt. Par exemple, un chercheur peut capturer tout fugitif visible en une étape dans un jeu tour-à-tour s'il peut sauter d'un sommet à tout autre.

La classification qui est représentée sur le tableau 1.1, donne une vision globale des différentes variantes des jeux des gendarmes et du voleur. Bien que les différentes variantes dont il est fait mention dans ce tableau ne soient pas toujours comparables en terme d'indice d'échappement, la tendance générale est que les variantes présentées dans les colonnes les plus à droite et les lignes les plus basses du tableau sont plus favorables au fugitif que les variantes en haut à gauche.

A notre connaissance, les principales variantes du jeu des gendarmes et du voleur qui ont été étudiées jusqu'à présent peuvent être divisées en trois catégories.

Fugitif visible de vitesse bornée, mouvements tour-à-tour. Ces jeux ont été étudiés sous le nom de *cops and robber games*. Ils ont été introduits indépendamment par Nowakowski et Winkler [NW83], et par Quilliot [Qui83]. Notons que la variante des jeux tour-à-tour dans le cas d'un fugitif invisible a été étudiée par Clarke et

déplacements	fugitif lent		fugitif rapide	
	visible	invisible	visible	invisible
tour-à-tour	Cops and robber games [Qui83, NW83]	[CN00, CN01]	?	?
simultanés	?	Helicopter search game [Fom98, Fom99]	[ST93]	Graph searching [Bre67, Par78b]

TAB. 1.1 – Classification générale des jeux des gendarmes et du voleur

Nowakowski. Dans cette variante, le fugitif est invisible, mais les chercheurs sont aidés : ils peuvent disposer d'informations partielles obtenues grâce à des radars placés sur les arêtes [CN00], ou bien ils peuvent placer des trappes qui limitent les déplacements du fugitif [CN01].

Fugitif de vitesse non bornée, mouvements simultanés. Nous ferons référence aux jeux appartenant à cette catégorie comme étant les jeux de *type Parson*. Ceux-ci ont été étudiés sous le nom de *graph searching problems*. Cette catégorie peut être divisée en deux sous-catégories. Le cas d'un fugitif invisible, défini initialement par Breish [Bre67] et Parson [Par78a, Par78b], joue un rôle particulier puisqu'il s'agit de la première variante des jeux des gendarmes et du voleur qui ait été définie. Le cas d'un fugitif visible a été introduit par la suite par Seymour et Thomas [ST93].

Fugitif invisible de vitesse bornée, mouvements simultanés. Dans cette variante, moins étudiée que les précédentes, et introduite sous le nom de *helicopter search game* par Fedor Fomin [Fom98], la question qui se pose est de déterminer la vitesse maximale que peuvent atteindre les mouvements du fugitif pour qu'il existe une stratégie de capture utilisant un chercheur qui saute d'un sommet à un autre. Fomin [Fom98, Fom99] établit un lien entre l'inverse de cette vitesse et divers paramètres de graphes dont la largeur linéaire (cf., section 1.2.5).

A notre connaissance, aucune des variantes correspondant aux trois cases du tableau 1.1 qui sont marquées d'un point d'interrogation n'ont encore été étudiées*.

Dans cette thèse, nous nous intéressons principalement aux jeux qui correspondent aux deux cases en bas à droite du tableau 1.1. C'est-à-dire la catégorie des jeux de type Parson, qui vise à la capture d'un fugitif arbitrairement rapide, et tels que les mouvements des chercheurs et du fugitif ont lieu simultanément. Dans le reste de cette section, nous présentons une classification plus précise des jeux des gendarmes et du voleur appartenant à cette catégorie. Pour cela, nous définissons deux contraintes qui peuvent être imposées aux stratégies des chercheurs : la monotonie et la connexité.

Définition 2 *Une stratégie de capture est dite* monotone *si elle n'admet aucune étape de recontamination. Un jeu de capture est dit* monotone *si, pour tout graphe G, le fait d'imposer aux stratégies de capture d'être monotones n'augmente pas l'indice d'échappement.*

*Notons que l'étude de la variante tour-à-tour dans le cas d'un fugitif visible mais arbitrairement rapide a été initiée très récemment par François Laviolette [Lav07]

Il y a plusieurs autres façons équivalentes d'exprimer la monotonie d'une stratégie. Ainsi, une stratégie de capture est monotone si, lorsqu'un sommet ou une arête est propre, il le reste jusqu'à la fin. Ou encore, une stratégie est monotone si chaque sommet (resp., arête) n'est occupé (resp., traversée) qu'une seule fois par un chercheur. L'intérêt des stratégies monotones est qu'elles sont beaucoup plus simples à concevoir et à analyser. Un second intérêt qui vaut la peine d'être mentionné est que les stratégies de capture monotones assurent que la capture du fugitif est effectuée en un nombre d'étapes polynomial en la taille du graphe. Il sera beaucoup fait référence à cette propriété de monotonie dans la suite (voir sections 2.3 et 4.3). *A priori*, il semble que l'indice d'échappement d'un graphe puisse augmenter lorsque l'on impose aux stratégies de capture d'être monotones. Nous verrons que ce n'est pas toujours le cas. Une des questions fondamentales de la théorie des jeux des gendarmes et du voleur est de savoir si le fait de satisfaire la monotonie impose aux stratégies de capture l'utilisation de plus de chercheurs. Dans le cas d'une réponse affirmative, une nouvelle question se pose naturellement. Quelle proportion de chercheurs supplémentaires est-elle nécessaire ? Nous verrons que la réponse à cette question dépend du type de stratégie envisagée.

La contrainte de connexité découle naturellement des applications pratiques relevant des stratégies de capture dans les graphes. Reprenons ici l'exemple, évoqué en introduction, d'une équipe de secouristes essayant de retrouver un spéléologue égaré dans une grotte. Le modèle de Parson possède de sérieux inconvénients s'il devait être employé par les secouristes. En effet, ces derniers ne peuvent pas être "placés" sur n'importe quel nœud du réseau de souterrains. Ils doivent bien entendu passer par l'entrée de la grotte. D'autre part, ils ne peuvent pas non plus se téléporter d'un nœud du réseau à un autre. Enfin, il est plus sûr pour les secouristes eux-mêmes de rester groupés afin d'éviter qu'ils ne s'égarent à leur tour. Le modèle de Parson comporte également de sérieuses lacunes dans un autre contexte. Dans un contexte militaire, ou bien dans le cadre de la sécurité dans les réseaux informatiques, les communications entre les chercheurs sont capitales pour la coordination des chercheurs. Ainsi, il parait crucial d'imposer que les communications soient effectuées à travers des zones sûres. Forts de tous ces arguments, Barrière *et al.* ont introduit la propriété de *connexité* pour les stratégies de capture [BFFS02, BFST03].

Définition 3 *Une stratégie de capture est dite* connexe *si, à chaque étape, la zone propre (i.e., l'ensemble des sommets et des arêtes propres) induit un sous-graphe connexe.*

Une question fondamentale qui se pose dans ce contexte est de savoir si le fait de satisfaire la contrainte de connexité impose aux stratégies de capture l'utilisation de plus de chercheurs. De plus, dans le cas d'une réponse affirmative, quelle proportion de chercheurs supplémentaires est-elle nécessaire ? Nous verrons que la réponse à cette question dépend notamment des classes de graphes considérées.

La classification qui est représentée sur le tableau 1.2, représente les quatre jeux des gendarmes et du voleur que nous considérons dans cette thèse. Pour chacun d'eux, nous indiquons s'il satisfait la propriété de monotonie.

	stratégie de capture	
	non connexe	connexe classe des arbres / cas général
invisible	monotone [LaP93, BS91]	monotone [BFFS02] / non monotone [YDA04]
visible	monotone [ST93]	monotone [trivial] / non monotone [FN06b]

TAB. 1.2 – Classification des jeux étudiés dans la thèse

1.2.4 Modèle de Parson et ses dérivés

Dans cette section, nous définissons formellement le jeu des gendarmes et du voleur originellement introduit par Parson [Par78a, Par78b], puis, nous présentons les modèles qui en ont dérivé.

a) Stratégies-arête. Le modèle défini par Parson vise à la capture d'un fugitif invisible, les mouvements étant réalisé simultanément par les chercheurs et le fugitif. La définition initiale de Parson considérait le graphe comme un environnement continu, il n'en a déduit une modélisation discrète qu'ensuite.

Soit G un graphe connexe plongé dans un espace à 3 dimensions. C'est-à-dire que les sommets de G sont représentés par des points de l'espace et les arêtes par les courbes dont les extrémités sont les points correspondant aux sommets, tels que les intérieurs des courbes représentant les arêtes soient deux à deux disjoints. Les chercheurs et le fugitif se déplacent de manière continue d'un point de G à un autre. Nous dirons que le fugitif est *capturé* si, à un moment donné, il occupe le même point du graphe qu'un chercheur.

Définition 4 (Parson [Par78b]) *Pour tout $k > 0$, soit $\mathcal{C}_k(G)$ l'ensemble des familles $F = \{f_1, \cdots, f_k\}$ avec, pour tout i, $1 \leq i \leq k$, $f_i : [0, \infty[\rightarrow G$ une fonction continue. Une* stratégie de capture continue *(general sweep strategy) pour G est une famille $F \in \mathcal{C}_k(G)$ telle que, pour toute fonction continue $h : [0, \infty[\rightarrow G$, il existe $t_h \in [0, \infty[$ et il existe $i \in \{1, \ldots, k\}$, tels que $f_i(t_h) = h(t_h)$.*

Intuitivement, k représente le nombre de chercheurs. Une stratégie de capture $F \in \mathcal{C}_k(G)$ est donc un ensemble de trajectoires déterminées pour chaque chercheur (f_i représente la trajectoire du chercheur numéro i), qui assure que, quelle que soit la trajectoire h du fugitif dans le graphe, il existe un chercheur qui occupera le même point que le fugitif à un certain instant t_h. En d'autres termes, une stratégie de capture assure que quelle que soit la stratégie adoptée par le fugitif, il sera capturé.

Nous avons cité la définition précédente car elle a été la première définition formelle des jeux des gendarmes et du voleur. Cependant, nous ne nous en servirons pas. Dans cette thèse, nous considérons les variantes discrètes qui sont issues de cette définition initiale. La première variante que nous définissons est la plus intuitive.

Définition 5 (Parson [Par78b]) *Etant donné un graphe G, une* stratégie-arête *dans G est une séquence d'opérations, ou étapes, choisies parmi les trois opérations suivantes :*

– *Placer un chercheur sur un sommet du graphe ;*

- *Retirer (ou supprimer) un chercheur d'un sommet du graphe ;*
- *Déplacer un chercheur le long d'une arête du graphe.*

Dans ce modèle, nous dirons qu'une arête est nettoyée lorsqu'elle est traversée par un chercheur. Cette arête reste propre si tout chemin entre cette arête et une arête contaminée est gardé par un chercheur. De la même manière que dans le cas continu, un fugitif invisible et arbitrairement rapide est capturé si, à une étape donnée, il occupe la même position qu'un chercheur. Etant donné un graphe G, une *stratégie de capture-arête* est une stratégie-arête qui garantit la capture du fugitif quelle que soit sa stratégie. En d'autres termes, une stratégie de capture-arête est une stratégie-arête gagnante pour les chercheurs.

Définition 6 *L'*indice d'échappement-arête $es(G)$ *(pour edge-search number en anglais) d'un graphe G, est le plus petit entier k tel qu'il existe une stratégie de capture-arête pour G utilisant au plus k chercheurs.*

Il est aisé de montrer que ce modèle de stratégie de capture est équivalent aux stratégies de capture continues définies précédemment. C'est-à-dire que, pour tout graphe G, et pour tout $k > 0$, il existe une stratégie de capture continue dans $\mathcal{C}_k(G)$ si et seulement si $es(G) \leq k$.

Donnons quelques exemples que nous allons retrouver souvent par la suite. Dans le modèle de stratégie-arête, un chercheur suffit à nettoyer un chemin : il suffit de le placer sur une des extrémités du chemin et de le déplacer le long des arêtes jusqu'à atteindre l'autre extrémité, i.e., $es(P_n) = 1$. Il est facile de se convaincre que dès que le graphe contient un cycle, un seul chercheur n'est plus suffisant. Pour tout $n > 2$, $es(C_n) = 2$: placer les deux chercheurs sur un sommet du cycle, puis déplacer l'un d'entre eux autour du cycle. La preuve du théorème suivant est très représentative de l'importance du résultat de monotonie de LaPaugh [LaP93] et Bienstock et Seymour [BS91] que nous présenterons en détail à la section 1.3.2. Indiquons simplement que ce résultat stipule qu'il existe toujours une stratégie de capture-arête optimale monotone. D'autre part, le principe de la preuve du théorème ci-dessous est récurrent dans le cadre de l'étude des stratégies de capture.

Théorème 2 *Pour tout $n > 3$, $es(K_n) = n$*

Preuve. Prouvons tout d'abord que n chercheurs sont suffisants. La stratégie des chercheurs consiste à placer $n-1$ chercheurs sur $n-1$ sommets distincts de la clique. Le n^{ieme} chercheur se déplace le long de toutes les arêtes entre ces $n-1$ sommets. Enfin, chacun des $n-1$ chercheurs quitte le sommet sur lequel on l'avait placé par la dernière arête encore contaminée incidente à ce sommet. La stratégie que nous venons de définir est une stratégie de capture-arête pour K_n, qui implique n chercheurs.

Considérons une stratégie de capture-arête monotone optimale S pour K_n. Montrons que S implique au moins n chercheurs. Supposons que S implique au plus $n-1$ chercheurs dans le but d'arriver à une contradiction. Soit u le premier sommet dont toutes les arêtes incidentes sont nettoyées par S. Considérons l'étape s qui consiste à nettoyer une arête $e = \{u, v\}$ incidente à u, alors que exactement $n-3$ arêtes incidentes à u, et différentes de e, sont déjà propres. Soient $\{v_1, \cdots, v_{n-3}\}$ les $n-3$ sommets distincts de v, tels que l'arête $\{u, v_i\}$ est propre, pour tout i, $1 \leq i \leq n-3$. Finalement, soit w le sommet de $V(K_n) \setminus \{v_1, \cdots, v_{n-3}, u, v\}$. Puisque la stratégie est monotone, au cours de l'étape s considérée, chaque sommet de $\{u, v_1, \cdots, v_{n-3}\}$ doit être occupé par un chercheur, soit un total de $n-2$ chercheurs qui ne peuvent pas bouger durant cette étape. Au cours de l'étape s, l'unique chercheur restant, que nous dirons *libre*, va se déplacer le long de e. Ainsi, au cours de l'étape s, le sommet w n'est occupé par aucun chercheur. Par conséquent, au cours de l'étape s, les arêtes incidentes à w sont soit toutes propres, soit toutes contaminées. Comme u est le premier sommet dont toutes les arêtes incidentes sont nettoyées par S, toutes les arêtes incidentes à w sont contaminées. Ainsi, si le chercheur libre qui nettoie e se déplace de v vers u, l'arête e est immédiatement recontaminée par $\{v, w\}$, ce qui contredit le fait que S est monotone. Donc, le mouvement qui a lieu au cours de l'étape s consiste à déplacer le chercheur libre le long de e, de u vers v. On en déduit que juste avant l'étape s, toutes les arêtes incidentes à v étaient contaminées.

La situation juste après l'étape s est la suivante. Un chercheur occupe u dont l'unique arête incidente contaminée est $\{u, w\}$. Chaque sommet dans $\{v, v_1, \cdots, v_{n-3}\}$ est occupé par un chercheur. Pour tout $1 \leq i \leq n-3$, les arêtes $\{v, v_i\}$ et $\{w, v_i\}$ sont contaminées. Le seul mouvement possible à la phase $s+1$ est le déplacement du chercheur occupant u le long de $\{u, w\}$ vers w. En effet, n'importe quel autre mouvement induirait la recontamination d'au moins une arête ou un sommet.

La situation juste après la phase $s+1$ est la suivante. Chacun des sommets de $\{w, v, v_1, \cdots, v_{n-3}\}$ est occupé par un chercheur et incident à au moins deux arêtes contaminées. Aucun mouvement évitant la recontamination n'est donc possible, ce qui contredit le fait que S est monotone. Pour conclure, il suffit d'énoncer le résultat selon lequel si il existe une stratégie de capture-arête utilisant au plus k chercheurs, alors il existe une stratégie de capture-arête monotone utilisant au plus k chercheurs [BS91, LaP93]. □

Une question naturelle qui se pose est de connaître le comportement de l'indice d'échappement-arête d'un graphe vis-à-vis des mineurs de graphes. Il est très simple de prouver le résultat suivant en notant que la contraction d'une arête ne peut pas augmenter le nombre de chercheurs nécessaires pour nettoyer un graphe. Plus précisément, étant donné un graphe G, un mineur H de G et une stratégie de capture-arête pour G, nous pouvons en déduire une stratégie de capture-arête pour H induite par celle de G et n'utilisant pas plus de chercheurs.

Lemme 1 *Soit $k > 0$. La famille des graphes d'indice d'échappement-arête au plus k est close par mineur.*

b) Stratégies-sommet. Dans le cadre de leur étude des stratégies de capture en relation avec les jeux de jetons, Kirousis et Papadimitriou [KP86] définissent l'*indice d'échappement-sommet*. Nous verrons dans la section 1.3.2 que ce paramètre est fortement lié à des paramètres importants des graphes, comme la largeur linéaire et la séparation-sommet. De plus, ce paramètre est, le plus souvent, bien plus facile à manipuler que l'indice d'échappement-arête.

Définition 7 (Kirousis et Papadimitriou [KP86]) *Etant donné un graphe G, une* stratégie sommet *pour G est une séquence d'opérations, ou étapes, parmi les deux suivantes :*
- *Placer un chercheur sur un sommet du graphe ;*
- *Retirer un chercheur d'un sommet du graphe.*

Dans le modèle des stratégies-sommet, les chercheurs ne sont plus autorisés à se déplacer le long des arêtes du graphe. La stratégie revient à une succession de sauts (d'un sommet à un autre) des chercheurs. Dans la littérature, l'image communément employée est celle de chercheurs se déplaçant en hélicoptère d'un sommet à l'autre. Dans ce modèle, une arête est nettoyée lorsque ses deux extrémités sont occupées par un chercheur. Le fugitif (invisible et arbitrairement rapide) est capturé si, à une étape donnée, il occupe le même sommet qu'un chercheur, ou s'il occupe une arête dont les deux extrémités sont occupées. Etant donné un graphe G, une *stratégie de capture-sommet* est une stratégie-sommet qui garantit la capture du fugitif quelle que soit sa stratégie. En d'autres termes, une stratégie de capture-sommet est une stratégie-sommet gagnante pour les chercheurs.

Définition 8 *L'*indice d'échappement-sommet $ns(G)$ *(pour node-search number en anglais) d'un graphe G, est le plus petit entier k tel qu'il existe une stratégie de capture-sommet pour G utilisant au plus k chercheurs.*

On vérifie facilement que, pour tout $n > 0$, $ns(P_n) = 2$, $ns(C_n) = 3$, et $ns(K_n) = n$.

Discutons du lien qui existe entre les deux paramètres que nous venons de définir. Premièrement, si nous disposons d'une stratégie de capture-sommet, il est facile de la transformer en une stratégie de capture-arête en utilisant un chercheur de plus. En effet, à chaque fois que les deux extrémités d'une arête sont occupées, il suffit d'utiliser le chercheur supplémentaire pour traverser l'arête. Réciproquement, si S est une stratégie de capture-arête, il est facile de la modifier pour obtenir une stratégie de capture-sommet utilisant un chercheur de plus. A chaque étape au cours de laquelle un chercheur traverse une arête $\{x, y\}$ de x à y, dans S, il suffit de laisser ce chercheur sur x et de placer le chercheur supplémentaire sur y. Nous obtenons donc les égalités suivantes. Pour tout graphe G :

$$ns(G) - 1 \leq es(G) \leq ns(G) + 1 \tag{1.1}$$

De plus, toutes les valeurs possibles de ces inégalités sont atteintes. Si G est restreint à une unique arête, $es(G) = 1 < ns(G) = 2$; si G est une étoile, $ns(G) = es(G) = 2$; et $es(K_{3,3}) = 5 > ns(K_{3,3}) = 4$ [KP86].

Pour terminer cette discussion du lien entre ns et es, posons $G^{///}$ le graphe obtenu à partir de G en remplaçant chaque arête par trois arêtes en parallèle, et G^{++} le graphe obtenu à partir de G en remplaçant chaque arête par trois arêtes en série. Kirousis et Papadimitriou [KP86] ont montré que, pour tout graphe G :

$$ns(G) = es(G^{///}) - 1 \quad \text{et} \quad es(G) = ns(G^{++}) - 1 \tag{1.2}$$

Ainsi, les deux variantes de stratégies que nous avons définies ont des comportements très similaires, et il est aisé de transposer tout résultat concernant l'une de ces variantes à l'autre.

c) Stratégies mixtes Nous définissons à présent un modèle que nous n'utiliserons pas par la suite, mais qu'il convient de mentionner ici puisqu'il a permis d'obtenir des résultats très importants sur les stratégies de capture dans les graphes. Bienstock et Seymour [BS91] définissent en effet cette variante afin de prouver que les stratégies de capture satisfont la propriété de monotonie (cf., section 1.3.2). Cette variante a également été définie indépendament par Takahashi *et al.* [TUK91, TUK95].

Définition 9 (Bienstock et Seymour) *Etant donné un graphe G, une* stratégie mixte *pour G est une séquence d'opérations élémentaires définies de la même façon que dans le cas d'une stratégie de capture-arête. Une arête est nettoyée lorsqu'elle est traversée par un chercheur, ou lorsque ses deux extrémités sont occupées par un chercheur.*

Etant donné un graphe G, une *stratégie de capture-mixte* est une stratégie-mixte qui garantit la capture du fugitif quelle que soit sa stratégie. En d'autres termes, une stratégie de capture-mixte est une stratégie-mixte gagnante pour les chercheurs.

Définition 10 *L'*indice d'échappement-mixte *$mixs(G)$ (pour mixed-search number en anglais) d'un graphe G est le plus petit entier k tel qu'il existe une stratégie de capture-mixte pour G utilisant au plus k chercheurs.*

Posons $G^{//}$ le graphe obtenu à partir de G en remplaçant chaque arête par deux arêtes en parallèle, et G^{+} le graphe obtenu à partir de G en remplaçant chaque arête par deux arêtes en série. Il est facile de montrer [BS91] que, pour tout graphe G :

$$mixs(G^{+}) = es(G) \quad \text{et} \quad mixs(G^{//}) = ns(G) \tag{1.3}$$

Grâce aux relations 1.3, un bon nombre de propriétés des stratégies de capture-mixte peuvent être transposées aux modèles précédents. Notons que récemment, Yang a défini une variante de ce modèle [Yan07].

d) Contraintes additionelles. Dans la suite de cette thèse, nous considérerons alternativement les stratégies de capture-arête et les stratégies de capture-sommet. Nous préciserons toujours en début de section quelle variante nous utiliserons, et la désignerons alors simplement par *stratégie de capture.* En particulier, dans les parties I et II, nous utiliserons les stratégies de capture-sommet, et dans la partie III, nous utiliserons les stratégies de capture-arête.

Nous avons défini les deux types de stratégies de capture-sommet (resp., capture-arête) relativement à la capture d'un fugitif invisible. De manière similaire, il est possible de définir les stratégies de capture-sommet relativement à la capture d'un fugitif visible.

Définition 11 *L'*indice d'échappement-sommet visible *d'un graphe G, noté $vns(G)$, est le plus petit entier $k > 0$ tels qu'il existe une stratégie de capture-sommet dans G utilisant au plus k chercheurs pour capturer un fugitif visible.*

Il est enfin possible d'imposer aux stratégies de capture d'être monotone et/ou connexe. Les indices d'échappement correspondants seront désignés de la manière suivante. De manière générale, un "m" (resp., un "c") dans le symbole représentant un indice d'échappement signifiera que la stratégie correspondante est monotone (resp., connexe). Par exemple, $mcvns(G)$ désigne l'indice d'échappement-sommet visible monotone et connexe d'un graphe G, c'est-à-dire le plus petit nombre de chercheurs nécessaires pour qu'il existe une stratégie de capture-sommet visible connexe et monotone utilisant ce nombre de chercheurs.

1.2.5 Décompositions de graphe

Nous nous consacrons maintenant à la définition d'un certain nombre de décompositions de graphe dont nous verrons qu'elles ont une grande importance dans l'étude des jeux des gendarmes et du voleur dans les graphes.

Définition 12 (Robertson et Seymour [RS86]) *Une décomposition arborescente d'un graphe G est une paire $(T, \mathcal{X})$, telle que $T = (V(T), E(T))$ est un arbre, et $\mathcal{X} = (X_t)_{t \in V(T)}$ est une famille de sous-ensembles de sommets de G, appelés sacs, qui satisfait :*

C1. $\bigcup_{t \in V(T)} X_t = V(G)$;

C2. *pour toute arête $\{x, y\} \in E(G)$, il existe $t \in V(T)$ tel que $\{x, y\} \subseteq X_t$;*

C3. *pour tout a, b et $c \in V(T)$ tels que c est sur le chemin entre a et b, $X_a \cap X_b \subseteq X_c$.*

Un ensemble de sommets $S \subset V$ est un *séparateur* d'un graphe connexe G s'il existe deux sommets distincts a et $b \in V$ qui appartiennent à deux composantes connexes distinctes de $G \setminus S$, alors tout chemin entre a et b contient un sommet de S. La propriété **C3** entraîne que, si G est connexe, pour toute arête $\{x, y\} \in E(T)$ telle que X_x n'est pas inclus dans X_y, $X_x \cap X_y$ est un séparateur du graphe. Cette caractéristique des décompositions arborescentes en terme de séparateurs du graphe va nous être très utile pour établir un lien entre ces décompositions et les stratégies de capture.

Remarquons qu'un graphe G admet toujours une décomposition arborescente triviale réduite à un seul sac. La largeur d'une décomposition arborescente $(T, \mathcal{X})$, notée $w(T, \mathcal{X})$, est égale à $\max_{t \in V(T)} |X_t| - 1$. La largeur arborescente d'un graphe G, notée $tw(G)$ (pour *treewidth* en anglais), est la largeur minimum d'une décomposition arborescente de G parmi toute les décompositions arborescentes du graphe G. Notons que, si la dénomination "*treewidth*" vient de Robertson et Seymour [RS86], ce type de décomposition avait déjà été intensivement étudiée pour ses applications algorithmiques au préalable [Gav74, RT75, Gol80]. Ainsi, une *décomposition arborescente optimale* d'un graphe G (i.e., une décomposition arborescente de largeur $tw(G)$) peut être obtenue à partir de l'arbre de cliques d'un sur-graphe cordal de G dont la clique maximale est de taille minimum [Gol80, Bod98].

Définition 13 (Robertson et Seymour [RS83]) *Une décomposition linéaire d'un graphe G est une décomposition arborescente $(P, \mathcal{X})$, telle que P est un chemin.*

Nous noterons une telle décomposition par une séquence $\{X_1, \cdots, X_r\}$ de sous-ensembles de sommets de G. La largeur linéaire, notée $pw(G)$ (pour *pathwidth* en anglais), d'un graphe G est la largeur minimum d'une décomposition linéaire de G parmi toute les décompositions linéaires du graphe G. Une *décomposition linéaire optimale* d'un graphe G (i.e., une décomposition linéaire de largeur $pw(G)$) peut être obtenue à partir de l'arbre de cliques d'un sur-graphe d'intervalle de G dont la clique maximale est de taille minimum [KP85].

Dans le cadre de la théorie des mineurs de graphes, Robertson et Seymour ont noté qu'il était parfois plus facile de manipuler des ensembles d'arêtes plutôt que des ensembles de sommets. C'est pourquoi, ils ont introduit la décomposition en branche des graphes [RS91, ST94, Hic05].

Définition 14 (Robertson et Seymour [RS91]) *Une décomposition en branche d'un graphe G est définie par une paire (T,τ) où $T=(V(T),E(T))$ est un arbre dont les sommets internes sont de degré trois, et τ est une bijection entre l'ensemble des feuilles de T et l'ensemble des arêtes de G.*

Etant donnée une arête e de T, il résulte de la suppression de e dans T deux arbres $T_1^{(e)}$ et $T_2^{(e)}$. Soit $F_i^{(e)}$ l'ensemble des feuilles de T qui appartiennent à T_i, $i \in \{1,2\}$. On définit alors une *e-coupe* comme la paire $(E_1^{(e)}, E_2^{(e)})$ où $E_i^{(e)} \subset E(G)$ est l'ensemble des arêtes de G induit par les feuilles de $T_i^{(e)}$, i.e., $E_i^{(e)} = \bigcup_{v \in F_i^{(e)}} \tau(v)$, $i \in \{1,2\}$. Notons que $E_1^{(e)} = E(G) \setminus E_2^{(e)}$. La largeur d'une décomposition en branche, notée $w(T,\tau)$ est égale à $\max_e |\delta(E_1^{(e)})|$ (où δ est la fonction "frontière" définie dans la section 1.2.1). La largeur en branche d'un graphe G, notée $bw(G)$ (pour *branchwidth* en anglais), est la largeur minimale parmi toutes les largeurs des décompositions en branches de G.

Notons qu'il existe un lien très étroit entre largeur arborescente et largeur en branche [RS91]. En particulier, pour tout graphe G, on a :

$$bw(G) \leq tw(G) + 1 \leq \max\{\frac{3}{2}\ bw(G), 2\} \tag{1.4}$$

La notion de e-coupe peut également être définie dans le cas d'une décomposition arborescente $(T,\mathcal{X})$ d'un graphe G. Etant donnée une arête e de T, il résulte de la suppression de e dans T deux arbres $T_1^{(e)}$ et $T_2^{(e)}$. On définit alors une *e-coupe* comme la paire $(G_1^{(e)}, G_2^{(e)})$ où $G_i^{(e)}$ est le sous-graphe de G induit par les sommets contenus dans les sacs de $T_i^{(e)}$, i.e., $G_i^{(e)} = G[\cup_{v \in V(T_i^{(e)})} X_v]$, $i \in \{1,2\}$.

Nous définissons une propriété des différentes décompositions de graphes qu'il convient de mettre en relation avec la propriété de connexité des stratégies de capture. Une décomposition arborescente, linéaire ou en branche, d'un graphe G est dite *connexe* lorsque, pour toute arête e de l'arbre, la e-coupe induit deux sous-graphes connexes. C'est-à-dire $G_1^{(e)}$ et $G_2^{(e)}$ sont connexes, ou, dans le cas de la décomposition en branche, $G[E_1^{(e)}]$ et $G[E_2^{(e)}]$ sont connexes. On note $ctw(G)$ (resp., $cpw(G)$ et $cbw(G)$) la largeur minimale de toutes les décompositions arborescentes (resp., linéaires et en branches) connexes de G.

Un résultat surprenant est que la contrainte de connexité n'augmente pas la largeur dans le cas des décompositions arborescentes et en branche. Ainsi, pour tout graphe connexe G, $tw(G) = ctw(G)$ [Gav74, Gol80, PS97]. Seymour et Thomas [ST94] ont prouvé que pour tout graphe 2-connexe G, $bw(G) = cbw(G)$. Une preuve constructive de ce résultat est donnée par Fomin *et al.* [FFT04]. Malheureusement, ce résultat n'est plus valable dans le cas des décompositions linéaires. Pour tout $n > 0$, il existe en effet des graphes G de $O(n)$ sommets tels que $cpw(G)/pw(G) = \Omega(n)$ [Nis04]. L'exemple le plus simple est une étoile à trois branches chacune de longueur n.

1.3 Etat de l'art

Ce chapitre est consacré à un tour d'horizon des principaux résultats en relation avec les stratégies de capture. La section 1.3.1 présente un bref état de l'art des jeux tour à tour. Dans la section 1.3.2, nous présentons les principaux résultats relatifs aux stratégies de capture où le fugitif est arbitrairement rapide et les mouvements des chercheurs et du fugitif sont simultanés. Nous présentons, en particulier, les liens qui existent entre ces stratégies de capture et les décompositions de graphe. Nous y verrons un certain nombre de résultats concernant les stratégies monotones, et l'importance de ces derniers. Dans la section 1.3.3 sont énumérées un certains nombre d'études des stratégies de capture d'un fugitif invisible dans des classes de graphes particulières. Enfin, la section 1.3.4 s'attache à un état de l'art des stratégies de capture connexe. En particulier, nous verrons que plusieurs algorithmes répartis ont été proposés pour calculer ce type de stratégies.

1.3.1 Un jeu tour-à-tour

Nous avons vu dans la section 1.2.3 que les jeux tour-à-tour et les jeux de type Parson appartiennent à deux classes de variantes du jeu des gendarmes et du voleur. Bien que cette thèse soit essentiellement consacrée aux stratégies de capture où les mouvements des chercheurs et du fugitif ont lieu simultanément, nous pensons que les principaux travaux concernant les jeux tour-à-tour valent la peine d'être présentés ici car la manière d'envisager les jeux appartenant à ces deux classes comporte de nombreuses similitudes. Dans cette section, nous nous intéressons à la variante des jeux tour-à-tour définie indépendamment par Quilliot [Qui83] d'une part, et Nowakowski et Winkler [NW83] d'autre part (voir [Als04] pour un tour d'horizon).

Ce jeu tour-à-tour implique deux joueurs. L'un, $\mathcal{C}$, joue les chercheurs (*the cops*), l'autre, $\mathcal{R}$, le fugitif (*the robber*). Tous d'abord, $\mathcal{C}$ place tous les chercheurs sur des sommets du graphe G. Puis, $\mathcal{R}$ place le fugitif sur un sommet de G. Les deux joueurs jouent ensuite alternativement. Lorsque c'est son tour, $\mathcal{C}$ choisit un sous-ensemble de chercheurs et déplace chacun de ces chercheurs sur un sommet adjacent de celui qu'il occupe. Ensuite, $\mathcal{R}$ peut ne rien faire, ou bien déplacer le fugitif sur un sommet voisin de sa position courante. Chaque joueur a une information complète de la situation et peut s'en servir pour décider de son prochain mouvement. $\mathcal{C}$ gagne si un chercheur occupe le même sommet que le fugitif. Si $\mathcal{R}$ peut éviter ce cas de figure infiniment, c'est lui qui gagne. Dans ce contexte, l'indice d'échappement (*copnumber*), noté $cp(G)$, d'un graphe G est le plus petit nombre de chercheurs qui assure la victoire de $\mathcal{C}$ dans G.

En 1983, Quilliot [Qui83] d'une part, et Nowakowski et Winkler [NW83] d'autre part, proposent la même caractérisation des graphes d'indice d'échappement 1 (voir également [HLSW02]). Cette caractérisation est intéressante puisqu'elle établit d'ores et déjà un lien entre les stratégies de capture et un ordre sur les sommets du graphe. Nous verrons qu'une telle caractérisation existe également dans le cas des stratégies de type Parson.

Théorème 3 (Quilliot [Qui83] ; Nowakowski et Winkler [NW83]) *L'indice d'échappement d'un graphe vaut un si et seulement si il existe un ordre $\{v_1, \cdots, v_n\}$ sur ses sommets tels que, pour tout $i < n$, il existe $i < j \leq n$ tel que $N[v_i] \cap \{v_i, \cdots, v_n\} \subseteq N[v_j] \cap \{v_i, \cdots, v_n\}$.*

Ce théorème implique que les arbres, les graphes cordaux (l'ordre considéré est l'ordre d'élimination simplicial [Gol80]) et les graphes de cordalité au plus 4 [Far87, AF88, Che97] ont un indice d'échappement égal à un.

Etant donné un graphe G et un entier $k > 0$, une *k-configuration* est un ensemble d'au plus k sommets de G. Intuitivement, une k-configuration représente la situation atteinte à un tour du jeu, c'est-à-dire que les sommets représentent la position des chercheurs. Soit G un graphe et $k > 0$ un entier. Le *graphe des k-configurations* est un graphe orienté, dont l'ensemble des sommets est l'ensemble des k-configurations, et tel qu'il y a une arête d'une k-configuration C vers une autre C', si une stratégie des chercheurs peut passer de la configuration C à la configuration C' en un tour. Dans [HM03], Hahn et MacGillivray proposent un algorithme en temps $O(n^5)$ qui détermine si un graphe possède un indice d'échappement égal à un. Ils en déduisent un algorithme qui détermine en temps polynomial si un graphe possède un indice d'échappement égal à k, k étant fixé. Ce dernier consiste à appliquer l'algorithme pour $k = 1$ au graphe des configurations possibles avec k chercheurs. Si k fait partie de l'entrée du problème, Goldstein et Reingold [GR95] ont prouvé que déterminer si $cp(G) \leq k$, avec des positions initiales fixées, est NP-difficile. Dans le cas d'un graphe orienté, fortement connexe, les mêmes auteurs prouvent que le problème est NP-difficile sans que les positions initiales ne soient fixées.

Un certain nombre de bornes inférieures et supérieures du nombre de chercheurs d'un graphe ont également été proposées. Aigner et Fromme [AF84] prouvent que le nombre de chercheurs d'un graphe de maille (longueur du plus petit cycle) au moins 5 est borné inférieurement par son degré minimum. Frankl [Fra87] améliore ce résultat en prouvant que, pour tout graphe G de maille au moins $8t-3$ et de degré minimum $d+1$, $cp(G) > d^t$. Andreae [And84] prouve que pour tout $k \geq 3$ et pour tout $n \geq 1$, il existe un graphe k-régulier G tel que $cp(G) \geq n$. Une borne supérieure triviale pour l'indice d'échappement d'un graphe G est le cardinal d'un ensemble dominant de taille minimum de G. Neufeld et Nowakowski [NN98] proposent plusieurs bornes relatives à l'indice d'échappement dans le cas de graphes produits. Andreae [And86] prouve que, pour tout graphe H, avec $u \in V(H)$ et $H \setminus u$ sans sommet isolé, si un graphe G n'admet pas H comme mineur, alors $cp(G) \leq |E(H \setminus u)|$. Ce résultat repose sur une décomposition du graphe et sur un résultat à la base de nombreuses stratégies de capture dans le modèle tour-à-tour, et que nous détaillons dans le paragraphe suivant.

Soit P un plus court chemin dans un graphe G. Après un nombre fini d'étapes, un unique chercheur peut protéger ce chemin, c'est-à-dire suffit à assurer que si le fugitif se place sur un sommet de P, alors il sera capturé à l'étape suivante. (En effet, il suffit à ce chercheur de suivre "l'ombre" du fugitif sur P. L'*ombre* d'un sommet de G sur le sous-graphe P de G est une application f de $V(G)$ dans $V(P)$ telle que, pour tout $\{x, y\} \in E(G)$, $\{f(x), f(y)\} \in E(P)$. Il est facile de vérifier qu'au bout d'un nombre fini d'étapes, la distance entre le chercheur poursuivant l'ombre du fugitif et l'ombre de ce fugitif est inférieure ou égale à la distance entre le fugitif et son ombre sur P). Ce résultat permet de donner de bonnes bornes supérieures pour le nombre de chercheurs pour des graphes plongeables sur des surfaces de genre borné. Ainsi, Aigner et Fromme [AF84] prouvent que 3 chercheurs suffisent à capturer un fugitif dans tout graphe planaire. L'idée générale est de réduire récursivement la zone contaminée de G, c'est-à-dire la zone accessible par le fugitif. Intuitivement, il s'agit d'utiliser deux chercheurs pour "entourer" le fugitif dans un cycle composé de deux plus courts chemins. Le troisième chercheur divise ensuite la zone où se tient le fugitif en protégeant un troisième plus court chemin. Quilliot [Qui85] prouve que $3 + 2k$ chercheurs suffisent à attraper un fugitif dans un graphe de genre k. Pour cela, des équipes de 2 chercheurs protègent chacune un cycle non contractile, réduisant ainsi récursivement le genre de la zone contaminée. Lorsque la zone contaminée est plane, il suffit alors d'appliquer la stratégie proposée par Aigner et Fromme avec 3 chercheurs. Schröder [Sch01] améliore ce résultat en prouvant que $3 + \lfloor \frac{3}{2}k \rfloor$ chercheurs suffisent à attraper un fugitif dans un graphe de genre k.

Pour en terminer avec les jeux tour-par-tour, mentionnons simplement une variante étudiée par Clarke et Nowakowski [CN05]. Dans cette variante, les chercheurs sont obligés de se déplacer par paire, de telle façon qu'à chaque tour, deux chercheurs d'une même paire doivent occuper le même sommet ou deux sommets voisins.

1.3.2 Généralités sur les stratégies de capture de type Parson

A partir de cette section et jusqu'à la fin de cette thèse, nous considérons uniquement les stratégies de capture de type Parson, c'est-à-dire relatives à un fugitif arbitrairement rapide, et où les mouvements des chercheurs et du fugitif sont simultanés (voir [Bie91] pour un tour d'horizon).

Dans cette section, nous présentons divers résultats de monotonie que satisfont les stratégies de capture. Nous expliquons ensuite comment ces résultats permettent d'établir un lien entre stratégie de capture et décomposition de graphes. Rappelons, que sauf indication contraire, *stratégie de capture* fait référence à une stratégie de capture invisible, et que les résultats obtenus pour l'un des trois modèles capture-arête, capture-sommet ou capture-mixte peuvent facilement être transposés aux deux autres.

L'un des résultats les plus importants de l'étude des stratégies de capture est sans aucun doute celui de LaPaugh [LaP93], selon lequel *"la recontamination n'aide pas"* et dont une preuve très élégante a été proposée par Bienstock et Seymour [BS91].

Théorème 4 (LaPaugh [LaP93]) *Pour tout graphe G, $mes(G) = es(G)$.*
(Bienstock et Seymour [BS91]) *Pour tout graphe G, $mmixs(G) = mixs(G)$.*

En d'autre terme, pour tout graphe G, il existe toujours une stratégie de capture-mixte optimale qui est monotone. Nous dirons que *le jeu de capture-mixte est monotone.* Il est très facile d'étendre ce résultat aux stratégies de capture-sommet, en utilisant la relation 1.3. En effet, à partir d'une stratégie de capture-sommet d'un graphe G, utilisant k chercheurs, il est facile de déduire une stratégie de capture-mixte du graphe $G^{//}$ utilisant au plus k chercheurs. D'après le théorème 4, il existe une stratégie de capture-mixte monotone pour $G^{//}$ utilisant au plus k chercheurs. On en déduit une stratégie de capture-sommet monotone de G utilisant au plus k chercheurs.

Le théorème 4 est très important car, comme nous l'avons déjà dit, les stratégies de capture monotones ont un nombre d'étapes polynomial en la taille du graphe. Ainsi, une stratégie monotone peut servir de certificat polynomial pour le problème d'optimisation qui consiste à déterminer l'indice d'échappement d'un graphe G. Du même coup, ce problème est dans NP.

Megiddo *et al.* [MHG+88] prouvent que le problème de déterminer l'indice d'échappement d'un graphe G est NP-difficile, en le réduisant au problème qui consiste à trouver une coupe minimale équilibrée. Ainsi, on en déduit le théorème suivant :

Théorème 5 (Megiddo *et al.* [MHG+88]) *Le problème de décision suivant est NP-complet :*
Entrée : un graphe G et un entier $k > 0$,
Question : $es(G) \leq k$?

Une seconde conséquence importante de la monotonie des stratégies de capture est que les stratégies monotones d'un graphe sont exactement les décompositions linéaires de ce graphe. Pour expliquer cette propriété, définissons la notion de séparation-sommet d'un graphe. Soit G un graphe, et f un ordre sur ses sommets, i.e., une bijection entre $V(G)$ et $\{1, \cdots, n\}$. Soit $\mathcal{L}_f(i) = \{x \in V \mid \exists y, \{x,y\} \in E, f(x) \leq i \text{ et } f(y) > i\}$, et soit $\lambda(f) = \max_{1 \leq i \leq n} |\mathcal{L}_f(i)|$. La sommet-séparation du graphe G, $\lambda(G)$, est le minimum de $\lambda(f)$ pris sur tout les ordres f. Kinnersley [Kin92] a prouvé que $\lambda(G) = pw(G)$ pour tout graphe G.

Soit S une stratégie de capture-sommet monotone pour un graphe G. Soit $\{v_1, \cdots, v_n\}$ l'ordre dans lequel les sommets sont occupés pour la première fois par un chercheur. On en déduit un ordre f, où $f(v_i) = i$. Pour tout $i \in \{1, \cdots, n\}$, $\mathcal{L}_f(i)$ est l'ensemble des sommets qui préservent $\{v_1, \cdots, v_i\}$ de la recontamination issue de $\{v_{i+1}, \cdots, v_n\}$. Puisque la stratégie est monotone, pour tout $i \in \{1, \cdots, n\}$, il existe une étape de la stratégie au cours de laquelle tous les sommets de $\mathcal{L}_f(i)$ sont occupés (pour préserver la zone propre de la recontamination), et un chercheur est placé sur un nouveau sommet (placer un chercheur sur un sommet est alors la seule opération autorisée, puisque supprimer un chercheur impliquerait une recontamination). Donc $ns(G) \geq \lambda(G) + 1$.

Réciproquement, soit $\{X_1, \cdots, X_r\}$ une décomposition linéaire de G. Considérons la stratégie suivante :

1. placer les chercheurs sur les sommets de X_1,
2. pour i allant de 2 à r faire

(a) supprimer les chercheurs de $X_i \backslash X_{i-1}$ (il reste des chercheurs sur chaque sommet de $X_i \cap X_{i-1}$),

(b) placer des chercheurs sur les sommets de X_i.

Les propriétés de la décomposition linéaire, dont en particulier que $X_i \cap X_{i-1}$ est un séparateur, assurent que cette stratégie est une stratégie de capture-sommet monotone. Donc $ns(G) \leq pw(G) + 1$. On en déduit le théorème suivant.

Théorème 6 (Ellis, Sudborough et Turner [EST94])
Pour tout graphe G, $ns(G) = pw(G) + 1$.

Notons que l'indice d'échappement-arête a également une interprétation en terme d'un paramètre de graphe : la *linear width*. Informellement, ce paramètre est le pendant de la séparation sommet en terme d'ordre sur les arêtes. Thilikos [Thi00] a prouvé que pour tout graph G 2-connexe, $es(G) = lw(G)$ avec $lw(G)$ la linear width de G. Utilisant ce fait, Bodlaender et Thilikos [BT04] prouvent qu'à k fixé, le problème de déterminer si $es(G) \leq k$ peut être résolu en temps linéaire.

Revenons sur le résultat de monotonie de LaPaugh. La preuve de LaPaugh est constructive. En effet, LaPaugh transforme une stratégie de capture-arête en une stratégie monotone sans augmenter le nombre de chercheurs nécessaires. En utilisant la notion de capture-mixte, Bienstock et Seymour [BS91] simplifient considérablement la preuve du théorème de LaPaugh (cf., section 2.3). Intuitivement, la preuve proposée par Bienstock et Seymour consiste à définir une fonction de poids sur les stratégies, et à prouver qu'une stratégie de poids minimum est monotone. Plus exactement, par un procédé d'optimisation locale, Bienstock et Seymour prouvent que toute étape non monotone d'une stratégie peut être modifiée en une étape monotone en diminuant strictement le poids total de la stratégie.

Une question qui se pose naturellement est de savoir si les stratégies de capture visant à capturer un fugitif visible satisfont aussi la propriété de monotonie et s'il existe également un lien entre les stratégies de capture visible et décomposition de graphe. La réponse est positive dans les deux cas. A l'instar de Bienstock et Seymour, Seymour et Thomas [ST93] prouvent que *la recontamination n'aide pas* pour la capture d'un fugitif visible. Cependant, la preuve qu'ils proposent est radicalement différente de celles existantes dans le cas d'un fugitif invisible. Ainsi, leur preuve n'est pas constructive et utilise intrinsèquement le lien entre stratégie de capture-sommet monotone visible et décomposition arborescente.

Plus précisément, Seymour et Thomas définissent un buisson (pour *bramble* en anglais) d'un graphe G comme un ensemble $\mathcal{B}$ de sous-ensembles de sommets qui se touchent deux à deux (c'est-à-dire d'intersection non nulle ou liés par une arête). L'ordre d'un buisson est la taille d'un transversal minimum de $\mathcal{B}$ (un ensemble de sommets qui intersecte tout élément de $\mathcal{B}$). Un buisson d'ordre k correspond exactement à une stratégie d'échappement d'un fugitif face à $k-1$ chercheurs. En effet, à chaque configuration employant $k-1$ chercheurs est associé un ensemble de $\mathcal{B}$ qui n'est occupé par aucun chercheur (puisqu'il n'existe pas de transversal de taille $k-1$). Cet ensemble sert de refuge (*haven*) au fugitif.

Posons $buisson(G)$ l'entier maximum k tel qu'il existe un buisson d'ordre k pour G. On a donc :

$$buisson(G) \geq k \Rightarrow vns(G) > k - 1$$

Théorème 7 (Seymour et Thomas [ST93]) *Pour tout graphe G et pour tout $k \geq 1$, $buisson(G) = k \Leftrightarrow tw(G) = k - 1$.*

De plus, disposant d'une décomposition arborescente $(T, \mathcal{X})$, une stratégie de capture monotone visible peut être aisément déduite de $(T, \mathcal{X})$ de la manière suivante. Soit $v \in V(T)$, placer des chercheurs sur les sommets de X_v. Ensuite, répéter le processus suivant jusqu'à ce que le fugitif soit capturé. Soit u le voisin de v tel que le fugitif se trouve sur un sommet de $G[V(T_u)]$, avec T_u le sous-arbre de $T \setminus \{u, v\}$ qui contient u. Retirer les chercheurs sur $X_v \setminus X_u$, et placer des chercheurs sur les sommets de X_u.

On vérifie facilement par induction sur n que, réciproquement, étant donné une stratégie de capture monotone visible utilisant k chercheurs, dont les premières étapes consistent à placer des chercheurs sur les sommets de $U \subseteq V$, G possède une décomposition arborescente de largeur $k - 1$ dont un sac est U. On obtient donc :

$$tw(G) = k - 1 \Leftrightarrow mvns(G) = k$$

Pour résumer, s'il existe un buisson d'ordre k, il n'existe pas de stratégie utilisant k chercheurs, donc il n'existe pas de stratégie monotone utilisant k chercheurs, d'où $tw(G) > k - 1$. Réciproquement, s'il existe une stratégie utilisant k chercheurs, il n'existe pas de buisson d'ordre k, d'où $tw(G) \leq k - 1$ et il existe une stratégie monotone utilisant k chercheurs. Une lacune de cette preuve est qu'elle n'est pas constructive.

Théorème 8 (Seymour et Thomas [ST93]) *Pour tout graphe G, $mvns(G) = vns(G) = tw(G) + 1$.*

Le lien entre stratégies de capture invisible ou visible avec les largeurs linéaire ou arborescente permet de profiter des résultats obtenus dans le cadre de la théorie des décompositions de graphe. Ainsi, Arnborg *et al.* [ACP87] prouvent que déterminer la largeur arborescente d'un graphe est un problème NP-complet. Donc,

Théorème 9 (Arnborg, Corneil et Prokurowski [ACP87] ; Seymour et Thomas [ST93])
Le problème de décision suivant est NP-complet :
Entrée : un graphe G et un entier $k > 0$,
Question : $vns(G) \leq k$?

Bodlaender et Kloks [Bod96, BK96] prouvent que si k est un paramètre fixé, il existe un algorithme linéaire en la taille du graphe qui détermine si $tw(G) \leq k$ et, si c'est le cas, calcule une décomposition arborescente optimale. Par ailleurs, Bouchitté *et al.* [BKMT04] proposent un algorithme approché qui calcule une décomposition arborescente de largeur au plus $\log n$ fois l'optimale. Ce résultat a été amélioré grâce à un algorithme de Feige *et* *al.* [FHL05a], qui permet d'obtenir des décompositions arborescentes de largeur approchée à un facteur $\sqrt{\log tw}$. Ces algorithmes peuvent bien entendu être utilisés pour calculer des stratégies de capture visible.

Mentionnons, que l'indice d'échappement-sommet visible n'est pas le seul paramètre qui peut être mis en relation avec la largeur arborescente d'un graphe. Dendris *et al.* [DKT97] étudient une autre variante de stratégie de capture dans laquelle le fugitif est invisible et arbitrairement rapide, mais est contraint de rester immobile, sauf si un chercheur va se placer sur le sommet qu'il occupe. Dendris *et al.* prouvent que cette variante est également monotone et que le nombre minimum de chercheurs pour attraper un tel fugitif est égal à la largeur arborescente du graphe, plus un. Les auteurs donnent aussi des résultats pour cette variante lorsque la vitesse du fugitif est limitée. Par exemple, si sa vitesse est limitée à un (il ne peut se déplacer que d'une arête à la fois), le jeu reste monotone.

Dans la partie I de cette thèse, nous introduisons une nouvelle variante des stratégies de capture, dans laquelle la visibilité du fugitif est paramétrée. Cette variante fait le lien entre les stratégies de capture visible et les stratégies de capture invisible.

1.3.3 Stratégies de capture dans certaines classes de graphes

Dans cette partie, nous présentons quelques résultats obtenus sur les stratégies de capture d'un fugitif invisible dans des classes de graphes particulières. Notons que certains des résultats de la liste, non exhaustive, suivante ont été obtenus dans le cadre de l'étude de la compléxité des largeurs arborescentes et linéaires dans certaines classes de graphes, et cela indépendamment des jeux de capture.

Comme souvent, la classe des arbres a été intensivement étudiée. Le problème de calculer la largeur linéaire d'un arbre est connu comme étant polynomial depuis la fin des années 1980 [MHG$^+$88]. Ellis, Sudborough et Turner [EST94] prouvent que ce problème est linéaire, mais l'algorithme qu'ils proposent n'effectue le calcul d'une décomposition linéaire optimale (i.e., d'une stratégie de capture) qu'en un temps $O(n \log n)$. Il faut attendre les années 2000, et Skodinis [Sko03], pour obtenir un algorithme qui calcule une stratégie optimale en temps linéaire. Megiddo *et al.* [MHG$^+$88] ont également prouvé que, pour tout arbre T, $es(T) \leq 1 + \log_3(n-1)$, et que cette borne est atteinte dans le cas des arbres ternaires complets. Tous ces résultats reposent principalement sur la caractérisation des arbres d'indice d'échappement k, donnée par Parson. Ainsi, pour tout arbre T, $es(T) \geq k+1$ si et seulement si il existe un sommet v tel que $T \setminus v$ est composé d'au moins trois arbres d'indice d'échappement-arête au moins k.

Le cas des grilles est intéressant parce qu'il montre comment utiliser les buissons pour déterminer l'indice d'échappement d'un graphe. Soit une grille carré de côté l. Il est facile de vérifier que $l+1$ chercheurs sont suffisants pour nettoyer la grille en avançant les chercheurs de front, colonne après colonne, d'un bord à l'autre de la grille. La preuve du fait que $l+1$ chercheurs sont nécessaires est beaucoup plus technique mais devient très simple lorsque l'on exhibe un buisson d'ordre $l+1$ de la grille. Pour ce faire, considérons une grille carré de côté l. Soit L la dernière ligne et C la dernière colonne moins le sommet appartenant à L. Soit $G' = G \setminus (L \cup C)$. G' est une grille carré de côté $l-1$. Soit X l'ensemble de tous les sous-ensembles de sommets constitués d'une ligne et d'une colonne de G'. Alors, $X \cup \{L\} \cup \{C\}$ est un buisson d'ordre $l+1$. En effet, il faut un sommet de $V(L)$ pour couvrir L, un sommet de $V(C)$ pour couvrir C, et au moins $l-1$ sommets de $V(G')$ pour couvrir tous les ensembles de X.

La largeur linéaire est un paramètre qui a beaucoup été étudiée dans un certain nombre de classes de graphes fréquemment rencontrées dans la littérature. Chandran et Kavitha [CK06] ont prouvé que l'indice d'échappement d'un hypercube de n sommets est $\Theta(n/\sqrt{\log n})$. Gustedt [Gus93] a prouvé que le problème de déterminer la largeur linéaire est NP-complet même si l'entrée est restreinte à la classe des graphes cordaux. Bodlaender et Möhring [BM93] ont proposé un algorithme en temps linéaire pour calculer la largeur linéaire lorsque l'entrée est restreinte à la classe des cographes. Bodlaender, Kloks et Kratsch [BKK95] ont proposé un algorithme en temps polynomial pour ce problème dans la classe des graphes de permutation. Notons que dans ces deux dernières classes de graphes, la largeur linéaire est égale à la largeur arborescente [BM93, BKK95]. En d'autres termes, dans un graphe de permutation ou un cographe, le fait de voir le fugitif ne permet pas d'économiser des chercheurs.

Un graphe est *planaire extérieur* s'il est planaire et s'il peut être plongé dans le plan de telle sorte que tout sommet soit sur la face extérieure. La classe des graphes planaires extérieurs a des liens étroits avec la classe des arbres. En particulier, le dual simplifié (dual moins le sommet représentant la face extérieure) d'un graphe planaire extérieur est un arbre. C'est donc naturellement que plusieurs algorithmes d'approximation ont été proposés pour calculer la largeur linéaire dans cette classe de graphes. Ainsi, Govindan *et al.* [GLY98] proposent un algorithme avec un rapport d'approximation 3. Récemment, Coudert *et al.* [CHS06] proposent un algorithme plus simple, dont le rapport d'approximation est 4, mais qui permet d'établir que la largeur linéaire d'un graphe planaire extérieur est au plus deux fois celle de son dual, améliorant ainsi le résultat de Fomin *et al* [FTT05].

Pour finir, mentionnons les graphes de largeur arborescente 2 (graphes *série-parallèle*) et 3. Ces classes de graphes ont également été intensivement étudiées. Plusieurs algorithmes ont été proposés pour calculer les décompositions arborescentes [AP86, AP92] et les décompositions linéaires [EM04] de tels graphes.

1.3.4 Stratégies de capture connexes

En général, nettoyer un graphe de manière connexe demande plus de chercheurs que si la contrainte de connexité n'est pas imposée. Ainsi, pour tout arbre ternaire D_k de hauteur $k \geq 3$, $cns(D_k) > ns(D_k)$. Non seulement les stratégies de capture connexes peuvent nécessiter plus de chercheurs, mais encore ces stratégies s'avèrent plus difficiles à étudier. Une caractéristique des stratégies de capture connexes qui les rend si difficiles à étudier est que la famille des graphes d'indice d'échappement connexe au plus $k > 3$ n'est pas close par mineur. Par exemple, le graphe H décrit sur la figure 1.2 est un mineur d'une grille G de largeur $k > 3$, et $cns(H) = 3k/2 + 1 > cns(G) = k + 1$ [BFST03].

Il est facile de démontrer que le problème de déterminer les indices d'échappement

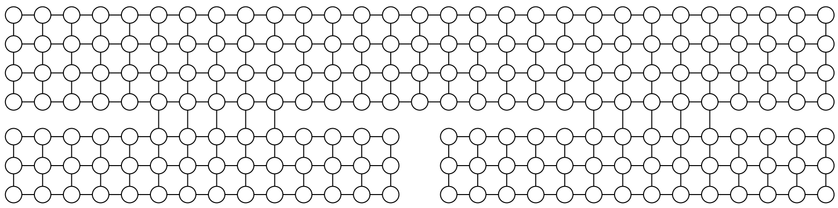

FIG. 1.2 – Graphe H [BFST03] mineur d'une grille de largeur 7 et tel que $cns(H) = 10$

connexe visible et invisible d'un graphe G est NP-difficile. En effet, soit G un graphe connexe, et soit G' le graphe obtenu en ajoutant à G un sommet adjacent à tous les sommets de G. Il est facile de vérifier que $cns(G') = ns(G) + 1$ et $cvns(G') = vns(G) + 1$. Le théorème suivant découle alors des théorèmes 5 et 9.

Théorème 10 *Les problèmes de décision suivant sont NP-difficile :*

- *Entrée : un graphe G et un entier $k > 0$,*
  Question : $cns(G) \leq k$?
- *Entrée : un graphe G et un entier $k > 0$,*
  Question : $cvns(G) \leq k$?

Deux questions relatives aux stratégies de capture connexe ont été principalement étudiées.

Question 1 *Les problèmes de décision relatifs aux stratégies de capture connexes appartiennent-ils à la classe NP ?*

Notons que le fait que les jeux correspondant aux stratégies de capture connexes soient monotones impliquerait une réponse affirmative à la question précédente.

Question 2 *Existe-t-il une constante $\alpha \geq 1$ telle que $ces(G) \leq \alpha\ es(G)$ pour tout graphe G ?*

Barrière *et al.* [BFFS02] introduisent les stratégies de capture connexes pour remédier au fait que les stratégies de capture-sommet et les stratégies de capture-arête ne peuvent être appliquées en pratique dans de nombreux contextes (du moins tant que la téléportation et les communications sûres n'existeront pas). Utilisant le même principe d'optimisation locale que Bienstock et Seymour, Barrière *et al.* prouvent que, pour tout arbre T, il existe une stratégie de capture monotone connexe qui utilise $ces(T)$ chercheurs. En d'autres termes, le jeu de capture connexe restreint à la classe des arbres est monotone. Barrière *et al.* proposent également un algorithme linéaire qui calcule $ces(T)$ pour tout arbre T. Cet algorithme consiste en un étiquetage dynamique (en partant des feuilles jusqu'à la racine) des sommets d'un arbre.

Dans le cadre des stratégies de capture connexes, la classe des arbres joue un rôle particulier puisque Barrière *et al.* [BFST03] répondent également à la seconde question ci-dessus dans le cas d'instances restreintes à cette classe de graphe. Ainsi, pour tout arbre T,

$$cs(T) \leq 2\ s(T) - 2. \tag{1.5}$$

L'égalité est obtenue pour un arbre ternaire de hauteur 4. Notons ici que, à notre connaissance, il n'existe pas de graphe G tel que l'indice d'échappement connexe de ce graphe soit plus de deux fois supérieur à son indice d'échappement. Dans le cas général, Fomin *et al.* [FFT04] prouvent que, pour tout graphe de m arêtes, $cs(G) \leq (1 + \log m)s(G)$. Plus précisément, à partir d'une décomposition en branche connexe de largeur $bw(G)$, Fomin *et al.* calculent une stratégie de capture connexe utilisant $bw(G)(1 + \log m)$ chercheurs.

Le jeu de capture connexe diffère du jeu "classique" par le fait que, dans le cas général, le jeu de capture connexe n'est pas monotone. Ce résultat négatif est dû à Alspach *et al.* [YDA04]. Le graphe G représenté figure 1.3 est tel que $mcns(G) = 290 > cns(G) = 281$. Les conventions de représentation utilisées dans cette figure sont les suivantes.

- Les cercles représentent des graphes complets dont le nombre de sommets est le nombre indiqué dans le cercle.
- Les doubles lignes entre deux cliques représentent des couplages parfaits entre ces deux cliques si elles sont de même taille, entre la clique la plus petite et un sous-graphe de la plus grosse sinon,
- les doubles lignes en pointillés représentent des chemins de cliques liées par des couplages parfaits.

Le principe de la "non-monotonie" de la stratégie de capture connexe dans un graphe, comme celui représenté sur la figure 1.3, vient du fait que les chercheurs peuvent nettoyer un passage A pour atteindre, et nettoyer, une certaine partie du graphe, puis rejoindre cette zone par un chemin B, et enfin laisser recontaminer la zone A dans le but de libérer des chercheurs pour pouvoir progresser par la suite.

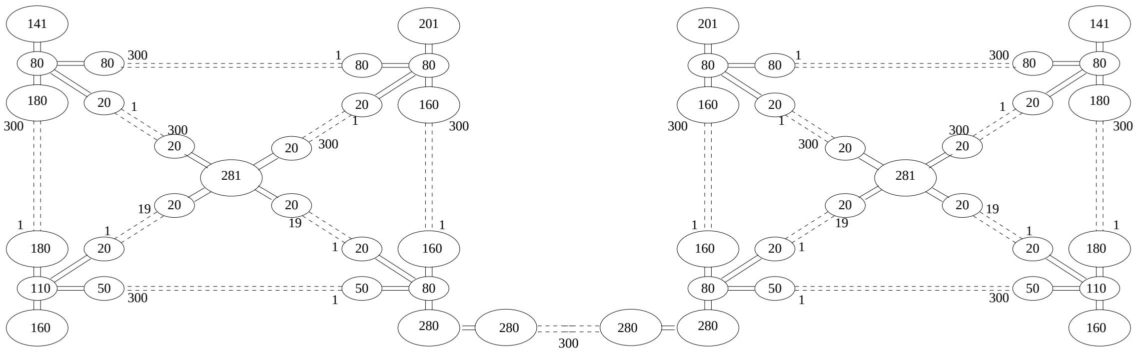

FIG. 1.3 – Graphe G tel que $mces(G) = 290 > ces(G) = 281$.

Une conséquence importante du résultat de Alspach *et al.* est que le problème de savoir si le premier problème de décision énoncé dans le théorème 10 appartient à NP est ouvert.

Dans la partie II de cette thèse, nous commençons par donner de nouveaux éléments de réponses à la question relative au rapport $ces(G)/es(G)$. Dans le chapitre 4, nous répondons à cette même question dans le cas des stratégies de capture connexe visible.

1.3.5 Stratégies de capture dans un contexte réparti

Une des principales applications des stratégies de capture connexes a été la conception d'algorithmes de capture dans un contexte décentralisé. Dans cette section, nous présentons le jeu des gendarmes et du voleur dans ce contexte.

Tout d'abord, définissons l'environnement dans lequel évoluent les chercheurs dans ce contexte (le modèle que nous présentons dans cette section est le modèle qui sera étudié dans la partie III). Un réseau est *a priori* un graphe *anonyme*, c'est-à-dire que ses sommets ne portent aucun identifiant ni étiquette. Notons que dans un graphe anonyme, deux sommets de même degré ne peuvent pas être distingués l'un de l'autre (dans le chapitre 6, nous étiquetterons cependant les sommets du graphe pour aider les chercheurs). Pour permettre aux chercheurs d'évoluer dans un graphe, ils doivent être capable de distinguer les différentes arêtes incidentes aux sommets qu'ils occupent. Dans ce but, les $\deg(v)$ arêtes incidentes à un sommet v sont étiquetées de 1 à $\deg(u)$. Ces étiquettes sont appelées *numéro de port*. Une fonction qui attribue à chaque incidence du graphe (chaque couple constitué d'un sommet et d'une arête incidente à ce sommet) un numéro de port est appelée une *orientation locale* de ce graphe. Notons que les deux extrémités d'une arête ont un numéro de port, et que ces numéros de port peuvent être distincts. Nous supposerons de plus que chaque sommet contient une zone locale de mémoire, appelée *tableau blanc*. Les tableaux blancs sont supposés être accessibles par les chercheurs, en lecture comme en écriture, grâce à un processus d'exclusion mutuelle équitable. Les chercheurs ne sont plus des jetons déplacés par un "joueur" central, mais sont représentés par des *automates de Mealy*. La représentation des chercheurs par des automates de Mealy est en effet traditionnelle dans un cadre réparti, comme, par exemple, pour l'exploration d'un graphe [FIRT06, Ilc06] ou la recherche de "trous noirs" [DFPS06].

Définition 15 *Un* automate de Mealy *est un quadruplet* $(\mathcal{S}, s_{init}, \mathcal{F}, \delta)$.
- $\mathcal{S}$ *est l'ensemble des états ;*
- $s_{init} \in \mathcal{S}$ *est l'état initial ;*
- $\mathcal{F} \subseteq \mathcal{S}$ *est l'ensemble éventuellement vide des états finaux ;*
- $\delta \ : \ \mathcal{S} \times (\mathbb{N} \cup \perp) \times \mathbb{N} \times \mathbb{N} \rightarrow \mathcal{S} \times (\mathbb{N} \cup \perp) \times (\mathbb{N} \cup \perp)$ *est la fonction de transition.*

Initialement, l'automate est dans l'état s_{init} sur un sommet donné appelé la *base*. Supposons qu'un automate dans un état non final s occupe un sommet v dont le tableau blanc contient ℓ. Si, à l'étape précédente, l'automate s'est déplacé, soit p le port par lequel il est arrivé sur v. Sinon, $p = \perp$. L'automate calcule alors $\delta(s, p, \ell, \deg(v)) = (s', t, p')$. Il passe dans l'état s', écrit t sur le tableau blanc de v si $t \neq \perp$, et quitte v par le port p', si $p' \neq \perp$, ou reste sur v si $p' = \perp$. En d'autres termes, la décision d'un automate sur un sommet v (se déplacer par un certain numéro de port, changer d'état, écrire sur le tableau blanc) est prise localement, et dépend uniquement de :

1. l'état courant de l'automate ;
2. les informations présentes sur le sommet (c'est-à-dire sur l'étiquette du sommet courant, ou sur le tableau blanc du sommet) ;

3. les états des chercheurs présents sur le sommet (cette information peut être transmise par l'intermédiaire des tableaux blancs), et
4. potentiellement le numéro de port d'arrivée si l'automate vient d'arriver sur le sommet.

Dans un tel contexte, le problème de stratégies de capture connexe peut être reformulé de la manière suivante. L'instance du problème consiste en un graphe connexe G qui vérifie les caractéristiques que nous venons d'énoncées, et d'un sommet particulier désigné comme étant la *base*.

Définition 16 *L'indice d'échappement monotone et connexe d'un graphe G, en partant de $v_0 \in V(G)$, noté $mces(G, v_0)$ désigne le plus petit entier $k \geq 1$ tel qu'il existe une stratégie de capture monotone et connexe de G utilisant k chercheurs et telle que le premier sommet propre est v_0.*

Initialement tous les chercheurs, modélisés par des automates tels que nous les avons décrits ci-dessus, se trouvent sur la base et ignorent tout du graphe dans lequel ils sont lancés. Le problème consiste à déterminer une stratégie de capture connexe pour G, commençant sur la base, et de telle sorte que la stratégie est calculée en temps réel par les chercheurs. C'est-à-dire que la stratégie est calculée localement par les chercheurs. Nous imposons de plus que la base reste propre jusqu'à la fin de la stratégie. Dans la suite de cette thèse, nous ferons référence à ce problème comme étant le problème du *nettoyage réparti.*

Définition 17 *Le* problème du nettoyage réparti *consiste à déterminer un protocole réparti $\mathcal{P}$ tel que, pour tout graphe G et toute base $v_0 \in V(G)$, $\mathcal{P}$ permettent à $mces(G, v_0)$ chercheurs de réaliser une stratégie de capture connexe de G en partant de v_0, et de telle sorte que cette stratégie soit calculée localement par les chercheurs et que la base reste toujours propre.*

Si de plus la stratégie de capture connexe est contrainte d'être monotone, nous parlerons du problème de *nettoyage monotone réparti.*

Une question qui se pose traditionnellement dans un environnement réparti consiste à étudier l'impact de l'asynchronisme de l'environnement. Ainsi, supposons que les actions effectuées par les chercheurs prennent un temps fini mais non borné. Flocchini *et al.* prouvent qu'il n'existe pas de protocole pour résoudre le problème de nettoyage monotone réparti qui utilise $mces(G)$ chercheurs dans un environnement asynchrone. Plus précisément, Flocchini *et al.* [FHL05b, FHL06] prouvent le théorème suivant :

Théorème 11 *Tout protocole réparti requiert $mces(G)+1$ chercheurs pour nettoyer toute grille asynchrone G.*

Ainsi, tout protocole de nettoyage réparti nécessite au moins un chercheur de plus que le nombre de chercheurs optimal dans le cas centralisé. Donnons une idée de la preuve de ce résultat grâce aux graphes représentés sur la figure 1.4. Le graphe G consiste en un cycle de 6 sommets, dont trois sommets distincts sont désignés par v_0, g et d. De plus, il y a un sommet pendant, adjacent au sommet d. Le graphe H est une copie de G à laquelle un sommet pendant adjacent au sommet g a été ajouté. Il est trivial de montrer que $mces(G, v_0) = 2 < mces(H, v_0) = 3$.

Soit $\mathcal{P}$ un protocole de nettoyage réparti qui permette à des chercheurs de nettoyer tout réseau à partir de n'importe quelle base. Considérons les exécutions de $\mathcal{P}$ dans les deux réseaux G et H, en partant dans les deux cas de v_0. Les chercheurs ignorent dans lequel des deux réseaux G ou H ils sont lancés. Nous montrons que deux chercheurs ne peuvent découvrir dans quel réseau ils évoluent. En effet, pour qu'une stratégie préserve la base de la recontamination, le premier chercheur doit se déplacer dans une direction, et le second dans l'autre. Considérons les deux faits suivants : (1) un chercheur ne peut dépasser un embranchement tout seul sans que la propriété de connexité ne soit violée, et (2) puisque le réseau est asynchrone, les deux chercheurs ne peuvent pas se mettre d'accord pour s'attendre l'un l'autre s'ils arrivent à un embranchement, de manière à vérifier si il y a un seul embranchement ou plusieurs. En effet, dans G cela pourrait fonctionner, mais dans H, ils attendraient indéfiniment. Donc, deux chercheurs ne peuvent distinguer les deux réseaux G et H. Par conséquent, $\mathcal{P}$ ne peut pas nettoyer G avec 2 chercheurs, et doit utiliser au moins un troisième chercheur, i.e., $mces(G, v_0) + 1$ chercheurs.

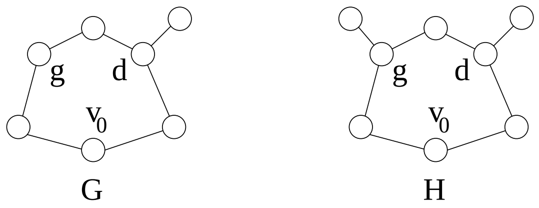

FIG. 1.4 – Illustration de la nécessité d'un chercheur supplémentaire en asynchrone.

Des protocoles répartis ont été conçus pour résoudre le problème du nettoyage réparti monotone dans des topologies particulières, comme les grilles [FLS05], les hypercubes [FHL05b], les cycles avec cordes [FHL06] et les graphes de Sierpinski [Luc07]. Bien entendu, tous ces protocoles tirent avantage de la connaissance de la topologie donnée aux chercheurs. Ces protocoles impliquent des chercheurs dont la mémoire est logarithmique en la taille des graphes. Les articles mentionnés ci-dessus s'intéressent également au temps d'exécution de ces protocoles, ainsi qu'au nombre de mouvements effectués par les chercheurs. Les auteurs proposent aussi des protocoles traitant le cas d'un fugitif visible.

Dans la partie III de cette thèse, nous proposons un algorithme plus général qui résout le problème du nettoyage réparti dans tous les graphes. Nous déterminons également de quelle quantité d'information minimum concernant la structure des graphes, les chercheurs doivent disposer pour pouvoir résoudre le problème du nettoyage réparti monotone dans tous les graphes.

Première partie

Stratégies de capture non-déterministes

Contenu de la partie

Cette partie est consacrée à la définition et à l'étude des stratégies de capture non-déterministes. Nous avons introduit cette variante dans le but d'établir une unique approche pour les stratégies de capture visibles et les stratégies de capture invisibles. Informellement, dans cette variante, le fugitif est invisible mais les chercheurs peuvent recevoir de l'aide d'un oracle qui connaît constamment la position du fugitif. Le nombre de fois que les chercheurs peuvent faire appel à l'oracle est cependant limité.

Le résultat principal de cette partie est l'interprétation des stratégies de capture non-déterministes en terme de décompositions arborescentes de graphes. Cette correspondance entre décompositions et stratégies nous permet d'établir un certain nombre de résultats relatifs aux stratégies de capture non-déterministes. Nous proposons notamment un algorithme exponentiel exact qui calcule des stratégies de capture non-déterministes optimales dans les graphes.

La correspondance entre décompositions et stratégies découle principalement de la propriété de monotonie que satisfont les stratégies de capture non-déterministes. La preuve que nous donnons de cette propriété est la généralisation des preuves existantes pour la monotonie des stratégies de capture visibles et la monotonie des stratégies de capture invisibles. Cette généralisation est à notre avis assez importante, et laisse envisager des perspectives intéressantes que nous détaillons dans la conclusion de la thèse.

Chapitre 2

Stratégies non-déterministes

Ce chapitre décrit une approche unifiée des stratégies de capture visible et invisible. Pour cela, nous définissons les stratégies de capture non-déterministes.

2.1 Introduction

Les stratégies de capture invisibles et les stratégies de capture visibles ont été intensivement étudiées au cours des années 1990. En particulier, cela est dû à leur interprétation en termes de décompositions de graphes. Une question naturelle qui se pose consiste à savoir ce qui se passe “entre” ces deux variantes. Plus précisément, que se passe-t-il lorsque le fugitif est *a priori* invisible, mais qu'à un certain nombre d'étapes de la stratégie, les chercheurs ont la possibilité de demander sa position ? C'est dans le but de répondre à cette question que nous avons introduit les stratégies de capture non-déterministes.

Dans ce chapitre, nous considérons uniquement les stratégies de capture-sommet, c'est à dire que les chercheurs peuvent “sauter” de sommet en sommet. Ainsi, nous définissons une variante non-déterministe des stratégies de capture-sommet. Bien entendu, des variantes non-déterministes des stratégies de capture-arête et des stratégies de capture-mixte peuvent être définies de la même façon. Dans les stratégies de capture-sommet non-déterministes (nous dirons plus simplement stratégie de capture non-déterministe), le fugitif est invisible. Cependant, un oracle peut voir le fugitif en permanence. Une opération supplémentaire est permise aux chercheurs : ceux-ci peuvent poser des questions à l'oracle pour qu'il leur donne la position du fugitif. Les stratégies considérées sont ainsi des séquences d'opérations choisies parmi les trois opérations élémentaires suivantes :

- Placer un chercheur sur un sommet du graphe ;
- Supprimer un chercheur d'un sommet du graphe ;
- Demander la position du fugitif à l'oracle.

A chaque question, une composante connexe de la partie contaminée est renvoyée par l'oracle, les autres composantes qui étaient contaminées deviennent propres. Informellement, l'oracle renvoie la composante connexe, isolée par les chercheurs, dans laquelle se trouve le fugitif au moment de la question.

Une autre façon de présenter les stratégies de capture non-déterministes nous permet d'expliquer pourquoi nous avons choisi le qualificatif de *non-déterministe*. Un programme de capture non-déterministe est un programme qui, étant donné un graphe G et un entier $k \geq 1$, calcule une stratégie de capture non-déterministe utilisant au plus k agents. Le choix des composantes contaminées après chaque question est effectué de façon non-déterministe. Un programme gagne si, pour tout fugitif, au moins une exécution du programme capture le fugitif.

Une stratégie de capture non-déterministe est paramétrée en fonction du nombre de fois que les chercheurs ont le droit de faire appel à l'oracle, ou, en d'autres termes, par le nombre d'étapes non-déterministes. Si aucune question n'est permise, nous retrouvons la définition des stratégies de capture-sommet invisibles. A l'opposé, dans le cas où le nombre de questions n'est pas limité, le modèle est équivalent à celui des stratégies de capture-sommet visibles (il suffit de poser une question à chaque étape). Ainsi, les stratégies de capture non-déterministes constituent bien une approche unifiée des variantes visible et invisible connues jusqu'alors. De la même manière que dans ces deux cas extrêmes, un enjeu majeur est celui de la monotonie de cette variante. Nous prouvons que le jeu associé aux stratégies de capture non-déterministes est monotone. Cela nous permet d'établir un lien entre ces stratégies et des décompositions de graphes, les décompositions arborescentes branchées, dont nous verrons qu'elles établissent un lien entre décompositions arborescentes et décomposition linéaires.

Dans la section 2.2, nous définissons formellement la notion de stratégie de capture non-déterministe. Parallèlement, nous définissons la notion de décomposition arborescente q-branchée, $q \geq 0$. Nous prouvons que ces décompositions arborescentes paramétrées ont une interprétation en terme de stratégies de capture non-déterministes monotones.

La section 2.3 est consacrée à l'un des résultats les plus intéressants de cette thèse. Nous y prouvons que le jeu de capture non-déterministe est monotone. Ce résultat permet d'unifier les preuves de monotonie dans les cas des jeux de capture visible [ST93] et invisible [BS91].

Dans la section 2.4, nous proposons un algorithme exponentiel exact pour le calcul de stratégie de capture non-déterministe. Nous déterminons ensuite le nombre de chercheurs qu'il est possible d'économiser lorsqu'une question supplémentaire est accordée. Nous prouvons que s'il est possible de poser une question supplémentaire à l'oracle, le nombre de chercheurs est dans le meilleur des cas divisé par deux, et qu'il existe des graphes pour lesquels c'est effectivement le cas.

Les résultats des sections 2.2 et 2.4 ont été réalisés en collaboration avec Fedor V. Fomin et Pierre Fraigniaud. Ils ont donné lieu à une publication dans la conférence MFCS 2005 [FFN05]. Les résultats de la section 2.3 ont été réalisés en collaboration avec Frédéric Mazoit. Ils ont donné lieu à une publication dans la conférence WG 2007 [MN07].

2.2 Unification des décompositions linéaires et arborescentes

Dans cette section, nous proposons une définition formelle des stratégies de capture non-déterministes. Nous définissons ensuite une version paramétrée des décompositions arborescentes. Enfin, nous montrons l'équivalence entre ces décompositions et les stratégies de capture non-déterministes monotones.

2.2.1 D'un fugitif invisible à un fugitif visible

Intuitivement, étant donné un graphe G, une stratégie de capture non-déterministe pour G est une séquence de paires, telle que chaque paire est composée d'un sous ensemble de V, représentant la position des chercheurs, et d'un sous-ensemble de E, représentant la partie propre de G. Plus précisément, une *stratégie non-déterministe* est une séquence ordonnée de paires $(Z_i, A_i)_{i\in[0,l]}$ telle que

- pour tout $0 \leq i \leq l$, $Z_i \subseteq V$ et $A_i \subseteq E$;
- $Z_0 = \emptyset$ et $A_0 = \emptyset$;
- pour tout $0 \leq i < l$, une des propriétés suivantes est vérifiée :

  1. (placer un chercheur) il existe $x_{i+1} \in V$, tel que $Z_{i+1} = Z_i \cup \{x_{i+1}\}$ et $A_{i+1} = A_i \cup B_{i+1}$ avec B_{i+1} l'ensemble des arêtes dont une extrémité est x_{i+1}, et l'autre extrémité se trouve dans Z_i ;
  2. (supprimer un chercheur) il existe $x_{i+1} \in V$, tel que $Z_{i+1} = Z_i \setminus \{x_{i+1}\}$ et $A_{i+1} \subseteq A_i$ est l'ensemble des arêtes qui restent propres après avoir supprimé l'agent du sommet x_{i+1} (les arêtes de $A_i \setminus A_{i+1}$ sont recontaminées) ;
  3. (poser une question) $Z_{i+1} = Z_i$ et A_{i+1} est défini de la manière suivante. Une composante connexe C de $G \setminus Z_{i+1}$ telle que $E(C) \cap A_i = \emptyset$ est choisie de manière non-déterministe. Soit C' l'ensemble des arêtes incidentes à au moins un sommet de $V(C)$. Alors, $A_{i+1} = E \setminus C'$.

Un *programme non-déterministe* est un programme qui prend un graphe G et un entier $k \geq 1$ en entrées, et renvoie une stratégie non déterministe pour G utilisant au plus k chercheurs. Un programme non déterministe est *gagnant* pour G si pour tout fugitif (c'est-à-dire, pour toute stratégie possible d'un fugitif), au moins une stratégie calculée par ce programme capture ce fugitif. Un programme non-déterministe est *monotone* si toutes les stratégies qu'il calcule sont monotones. Le nombre de chercheurs utilisés par un programme non-déterministe est le nombre maximum de chercheurs utilisés par toutes les stratégies que ce programme calcule.

Un *programme non-déterministe q-limité*, ou *q-programme*, est un programme non-déterministe dont les stratégies qu'il renvoie posent au plus q questions. *L'indice d'échappement q-limité* d'un graphe G, noté $s_q(G)$, est le nombre minimum de chercheurs utilisés par un programme q-limité gagnant dans un graphe G. De même, nous définissons *l'indice d'échappement monotone q-limité* d'un graphe G, noté $ms_q(G)$, comme étant le nombre minimum de chercheurs utilisés par un programme monotone q-limité gagnant dans un graphe G.

Notons que, pour $q = 0$, cette définition coïncide avec celle de l'indice d'échappement-sommet de Kirousis et Papadimitriou [KP85], et que, pour $q = \infty$, cette définition coïncide avec celle de l'indice d'échappement-sommet visible de Seymour et Thomas [ST93].

Dans la suite de cette section, nous considérons uniquement des stratégies monotones telles que chaque sommet du graphe n'est occupé qu'une seule fois. La section 2.3 est consacrée à prouver que cette propriété n'est, en fait, pas pénalisante en termes de nombre de chercheurs (cf., théorème 13).

2.2.2 Décomposition arborescente branchée

Cette courte section est destinée à introduire la notion de décomposition arborescente branchée. Informellement, il s'agit de décompositions arborescentes "classiques" paramétrées par un entier $q \geq 0$ représentant le nombre maximum de branchements le long d'un chemin de l'arbre à partir de sa racine. Nous verrons dans la section suivante que ces décompositions ont une interprétation intéressante en termes de stratégies de capture non-déterministes.

Une décomposition arborescente *enracinée* $(T, \mathcal{X}, r)$ d'un graphe G est une décomposition arborescente $(T, \mathcal{X})$ de G telle que T est enraciné en $r \in V(T)$. Un *sommet de branchement* d'un arbre enraciné est un sommet avec au moins deux fils. Pour tout $q \geq 0$, un *arbre q-branché* est un arbre T enraciné en $r \in V(T)$ et tel que tout chemin dans T de r à une feuille contient au plus q sommets de branchements.

Définition 18 *Une* décomposition arborescente q-branchée *d'un graphe G est une décomposition arborescente $(T, \mathcal{X}, r)$ de G telle que T est q-branché avec r pour racine.*

En particulier, une décomposition linéaire enracinée en l'une de ses extrémités est une décomposition arborescente 0-branchée, et une décomposition arborescente "classique" est une décomposition arborescente arbitrairement branchée. Pour tout graphe G, la *largeur arborescente q-branchée* de G, notée $tw_q(G)$, est la largeur minimum d'une décomposition q-branchée de G parmi toutes les décompositions arborescentes q-branchées de G.

La figure 2.1 représente les variations de la largeur q-branchée d'un graphe G, c'est-à-dire, le graphique de la fonction $f_G : \mathbb{N} \to \mathbb{N}$ tel que $f_G(q) = tw_q(G)$. Sur cette figure, $\tau(G) = \min\{q \geq 0 \mid tw_q(G) = tw(G)\}$ et $\pi(G) = \max\{q \leq \tau(G) \mid tw_q(G) = pw(G)\}$.

2.2.3 Interprétation en terme de stratégies de capture

Dans cette section, nous montrons que la largeur arborescente q-branchée et l'indice d'échappement monotone q-limité sont égaux à 1 près. Cette égalité nous sera utile pour la conception d'algorithmes et le calcul de bornes concernant l'indice d'échappement monotone q-limité.

Théorème 12 *Pour tout $q \geq 0$, pour tout graphe G, $tw_q(G) = ms_q(G) - 1$.*

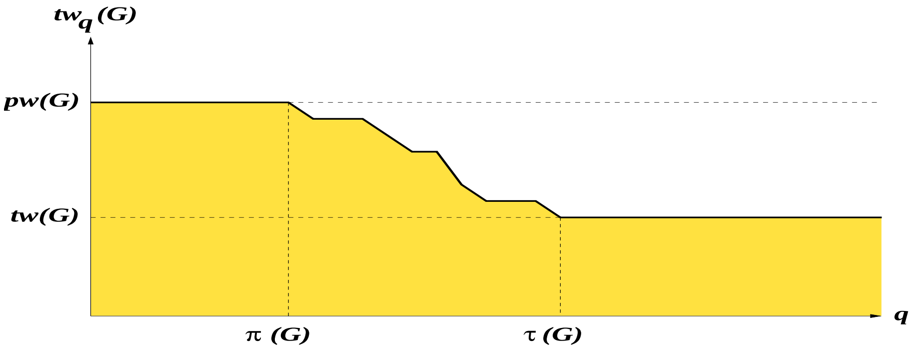

FIG. 2.1 – Largeur q-branchée d'un graphe G

Preuve. Soit $(T, \mathcal{X}, r)$ une décomposition arborescente q-branchée de largeur k. Pour tout sommet i de T, nous posons $d(i)$ l'ensemble des descendants de i dans T. Nous définissons une stratégie de capture q-limitée qui utilise k chercheurs. Initialement, les chercheurs sont placés sur les sommets contenus dans X_r. Supposons qu'à une certaine étape de la stratégie, il existe un sommet i de T, tel que les chercheurs sont situés sur les sommets contenus dans X_i, et l'ensemble des sommets contaminés est $\bigcup_{j \in d(i)} X_j \setminus X_i$. Si i est une feuille, la stratégie est terminée car le graphe est entièrement nettoyé. Supposons que i est un sommet interne de T. La stratégie dépend alors du nombre de fils de i.

- *Cas A : i possède un unique fils f.* Tout d'abord, nous supprimons les chercheurs des sommets de $X_i \setminus X_f$. Puis, nous plaçons des chercheurs sur les sommets de X_f. Comme X_i et X_f contiennent au plus $k+1$ sommets, au plus $k+1$ chercheurs sont utilisés. D'après les propriétés $C2$ et **C3** des décompositions arborescentes, pour tout sommet contaminé $v \in \bigcup_{j \in d(i)} X_j \setminus X_i$ et tout sommet propre u, tout chemin entre v et u contient un sommet de $X_i \cap X_f$. Donc, lorsque l'on supprime les chercheurs de $X_i \setminus X_f$, aucun sommet n'est recontaminé. La situation atteinte est telle que les chercheurs sont situés sur les sommets contenus dans X_f, et l'ensemble des sommets contaminés est $\bigcup_{j \in d(f)} X_j \setminus X_f$.
- *Cas B : i possède plus d'un fils.* Dans ce cas, une question est posée à l'oracle. Soit C la composante connexe de $G[\bigcup_{j \in d(i)} X_j \setminus X_i]$ que donne l'oracle. Il existe un unique fils f de i tel que $C \cap X_f \neq \emptyset$. Nous supprimons les chercheurs des sommets de $X_i \setminus X_f$. Puis, nous plaçons des chercheurs sur les sommets de X_f. Là encore, la situation obtenue est telle que les chercheurs sont situés sur les sommets contenus dans X_f, et l'ensemble des sommets contaminés est $\bigcup_{j \in d(f)} X_j \setminus X_f$.

Le graphe est nettoyé lorsque les chercheurs atteignent une feuille de la décomposition. Le nombre de chercheurs utilisés est au plus $\max_{j \in V(T)} |X_j| \leq k+1$. Puisque pour toute feuille i de T, le chemin de r à i contient au plus q sommets de branchement, le cas B est exécuté au plus q fois. Donc, le nombre de questions posée à l'oracle est au plus q. D'où $ms_q(G) \leq tw_q(G) + 1$.

Pour prouver l'inégalité inverse, nous prouvons une assertion plus forte.

Assertion 1 *Supposons qu'il existe un q-programme monotone et gagnant pour G, utili-*

sant $k+1$ chercheurs, et telle que, initialement, les chercheurs sont placés sur les sommets $X \subseteq V(G)$. Alors, il existe une décomposition arborescente q-branchée $(T, \mathcal{X}, r)$ de G, de largeur au plus k, et telle que $X_r = X$.

Nous prouvons cette assertion par induction sur q. Pour $q = 0$, nous obtenons la décomposition linéaire voulue $P = \{X_0, X_1, \cdots, X_m\}$ de la façon suivante. Nous posons $X_0 = X$, et, pour tout $i \geq 1$, X_i est l'ensemble des sommets occupés par les chercheurs après la i^{eme} étape de la stratégie. Pour vérifier que P est bien une décomposition linéaire, il suffit de remarquer que chaque sommet doit être occupé à une étape par un chercheur, et par conséquent sera contenu dans l'un des sacs de P. Chaque paire de sommets adjacents $\{u, v\}$ est contenue dans un même sac de P, car il existe une étape de la stratégie au cours de laquelle les deux sommets u et v sont occupés par des chercheurs. La troisième propriété des décompositions provient de la monotonie de la stratégie et du fait que chaque sommet n'est occupé qu'une fois.

Soit $q \geq 1$ et supposons que, pour tout $q' < q$, l'assertion est satisfaite. Considérons une stratégie q-branchée monotone pour G, utilisant au plus $k+1$ chercheurs. Supposons que la première question est posée à l'étape $t \geq 0$. Soit X l'ensemble des sommets occupés par des chercheurs et S l'ensemble des sommets nettoyés à cette étape. Soient $G_1, \cdots, G_p$ les sous-graphes de G obtenus à partir des composantes connexes de $G \setminus S$ et des sommets de X. Chacun de ces sous-graphes peut être nettoyé par $k+1$ chercheurs, en posant au plus $q-1$ questions et en commençant avec les chercheurs sur les sommets de X. D'après l'hypothèse d'induction, pour tout $1 \leq i \leq p$, il existe une décomposition arborescente $q-1$-branchée $(T^{(i)}, \mathcal{X}^{(i)}, r^{(i)})$ de G_i de largeur au plus k, et telle que $X^{(i)}_{r^{(i)}} = X$.

Nous pouvons maintenant construire une décomposition arborescente $(T, \mathcal{X}, r)$ de G. Soient $X_1, \cdots, X_t$ les ensembles de sommets occupés par les chercheurs durant les t premières étapes de la stratégie. En particulier, $X_t = X$. Nous construisons une décomposition linéaire $(X_1, \cdots, X_t)$ enracinée en X_1. Puis, nous ajoutons les décompositions arborescentes $(T^{(i)}, \mathcal{X}^{(i)}, r^{(i)})$, $1 \leq i \leq p$, et identifions chaque sommet r_i avec le sommet t de la décomposition linéaire. La décomposition obtenue est une décomposition arborescente q-branchée, de largeur au plus k, de G. □

2.3 Monotonie du jeu de capture non-déterministe

Comme nous l'avons vu à la section 1.3.2, la monotonie est vérifiée dans le cas du jeu de capture visible [ST93] et dans celui du jeu de capture invisible [BS91]. Cependant, les démonstrations de ces deux résultats sont radicalement différentes. En effet, Bienstock et Seymour [BS91] donnent une preuve constructive de la monotonie du jeu de capture invisible, en transformant une stratégie quelconque en une stratégie monotone, sans augmenter le nombre de chercheurs impliqués. Au contraire, la preuve de Seymour et Thomas [ST93] n'est pas constructive. Un des intérêts de cette section est qu'elle fournit une preuve constructive générale qui transforme toute stratégie de capture q-limitée, en une stratégie de capture q-limitée monotone, y compris pour $q = \infty$ (c'est-à-dire, dans le cas du jeu de capture visible).

Avant de continuer, rappelons le principe de la preuve de Bienstock et Seymour [BS91]. Ils considèrent une stratégie de capture invisible (non nécessairement monotone) sous la forme d'une séquence de sous-ensembles d'arêtes (c'est-à-dire, les sous-ensembles successifs d'arêtes propres). Une séquence $\{E_0, \cdots, E_l\}$ de sous-ensembles de E, telle que $E_0 = \emptyset$, $E_l = E$, et, pour tout $0 \leq i < l$, $|E_{i+1} \setminus E_i| \leq 1$, est appelée une *croisade*. Une croisade $\{E_0, \cdots, E_l\}$ est de taille $k = \max_{0 \leq i \leq l} |\delta(E_i)|$. Bienstock et Seymour prouvent l'équivalence entre les croisades de taille k et les stratégies de capture-mixte utilisant k chercheurs. L'idée principale de Bienstock et Seymour est de choisir une croisade vérifiant :

1. $\sum_{0 \leq i \leq l} (|\delta(E_i)| + 1)$ est minimum ;
2. $\sum_{0 \leq i \leq l} |E_i|$ est minimum parmi les croisades vérifiant (1).

Ils prouvent qu'une telle croisade est monotone. De leur preuve, il est possible de déterminer des optimisations locales successives telles que, en partant d'une croisade non-monotone et en procédant à ces optimisations locales, il est possible de construire une croisade monotone.

La preuve que nous proposons est une généralisation de la preuve résumée ci-dessus. C'est dans l'optique de cette preuve que, dans la définition des stratégies de capture non-déterministes, la séquence des ensembles d'arêtes propres (c'est-à-dire, la séquence A_i) est explicitement définie : cette séquence est en fait redondante puisqu'elle peut être obtenue à partir de la séquence des positions des chercheurs.

Dans cette section, nous avons besoin d'étendre la définition de frontière d'un ensemble d'arêtes d'un graphe à une partition de l'ensemble d'arêtes. Soit $\mathcal{E} = \{E_1, \cdots, E_p\}$ une partition de E. La *frontière de cette partition*, notée $\delta(\mathcal{E})$, est $\bigcup_{1 \leq i \leq p} \delta(E_i)$. La frontière d'une partition est donc l'ensemble des sommets incidents à une arête dans une partie et à une arête dans une autre. Dans cette section, nous dirons qu'une partition est *dégénérée* si au moins un de ses éléments est l'ensemble vide.

2.3.1 Arbre de capture

Pour prouver la propriété de monotonie du jeu de capture non-déterministe, nous définissons une structure auxiliaire, appelée arbre de capture, inspirée des *tree-labelling* définis par Robertson and Seymour [RS91]. Les *tree-labelling* sont des décompositions en branche relachées. Dans une décomposition en branche, à chaque sommet est associée une partition en trois des arêtes du graphes. Les *tree-labelling*, eux, associent à chacun de leur sommet, un recouvrement en trois ensembles de l'ensemble des arêtes du graphe. Cette relaxation permet de réaliser des optimisations locales sur les *tree-labelling*. Grâce à cette structure, Robertson and Seymour [RS91] prouvent un théorème de dualité entre les décompositions en branche et les *enchevêtrements* (*tangles*). Cette dualité est analogue à celle qui existe entre largeur arborescente et buissons, présentée à la section 1.3.2.

Définition 19 *Un arbre de capture d'un graphe G est un triplet (T, α, β) avec T un arbre, α une application des incidences de T (c'est-à-dire, les couples $(v, e) \in V \times E$ avec e incidente à v) dans les parties de E, et β une application des sommets de T dans les parties de E tels que :*

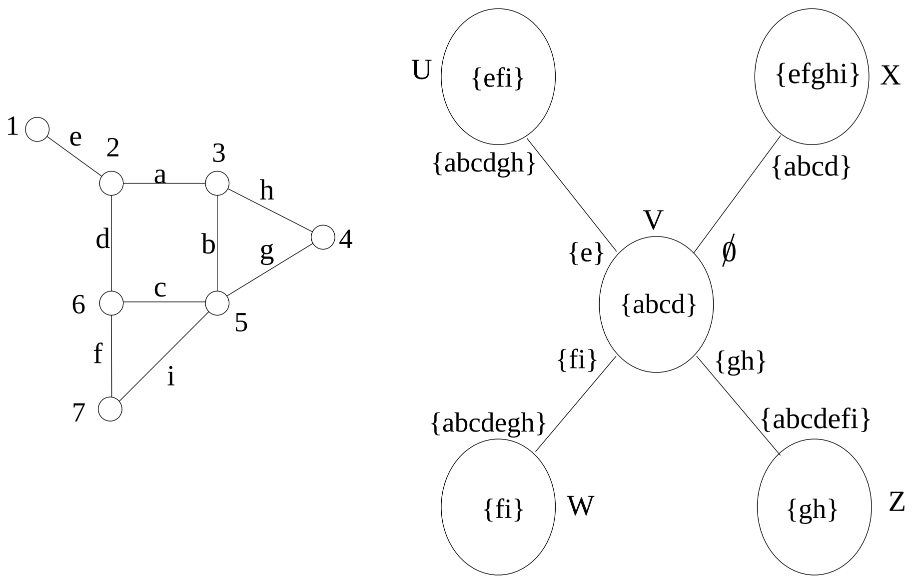

FIG. 2.2 – Exemple d'un graphe G et d'un arbre de capture de largeur 7 pour G

- *pour tout arête $e = \{u, v\}$ de T, $\alpha(u, e) \cap \alpha(v, e) = \emptyset$;*
- *pour toute feuille v de T incidente à une arête e de T, $\alpha(v, e) \neq E$;*
- *pour tout sommet v de T incident à $e_1, \ldots, e_p$, $\big\{\beta(v)\big\} \cup \mu(v)$ est une partition (éventuellement dégénérée) de E, où $\mu(v) = \{\alpha(v, e_1,), \ldots, \alpha(v, e_p)\}$.*

Notons qu'un graphe G admet toujours un arbre de capture trivial (T, α, β), tel que T est réduit à un sommet v, α n'est pas spécifiée, et $\beta(v) = E$.

Intuitivement, un arbre de capture peut être vu comme une décomposition arborescente relachée, c'est-à-dire ne vérifiant pas nécessairement la condition **C3** des décompositions arborescentes. En effet, à chaque sommet v de T, nous pouvons associer le sous-ensemble χ_v des sommets u de G tel que soit u est incident à une arête dans $\beta(v)$, soit il appartient à la frontière de la partition $\big\{\beta(v)\big\} \cup \mu(v)$. Il est alors facile de vérifier que pour toute arête e de E, il existe un sommet $v \in V(T)$ tel que $e \in \beta(v)$. Donc, le couple $(T, (\chi_v)_{v \in V(T)})$ satisfait les conditions **C1** et **C2** des décompositions arborescentes.

Nous étendons la définition de β à un sous-arbre T' de T en posant $\beta(T') = \cup_{v \in T'} \beta(v)$. La *largeur* d'un arbre de capture est défini par $\mathrm{width}(T, \alpha, \beta) = \max_{v \in V(T)}\{|\chi_v|\}$ avec $\chi_v = V\big[\beta(v)\big] \cup \delta\big(\mu(v)\big)$, et $|\chi_v|$ désigne la cardinalité de χ_v, appelée le *poids* du sommet $v \in V(T)$. Comme pour les décompositions arborescentes, nous considérons les arbres de capture *enracinés*, notés (T, α, β, r), où l'arbre T est enraciné en r. Pour tout $q \geq 0$, un arbre de capture *q-branché* est un arbre de capture enraciné (T, α, β, r) de G tel que T enraciné en r est q-branché. Une arête $e = \{u, v\}$ d'un arbre de capture est *monotone* si $\alpha(u, e) = E \setminus \alpha(v, e)$. Une arête qui n'est pas monotone est dite *sale*. Un arbre de capture est *monotone* si toutes ses arêtes le sont. Nous verrons que si un arbre de capture (T, α, β) d'un graphe G est monotone, la condition **C3** des décompositions arborescentes est satisfaite par $(T, (\chi_v)_{v \in V(T)})$ qui, dans ce cas, est une décomposition arborescente du graphe G.

La figure 2.2 représente un graphe $G = (V, E)$ et un arbre de capture (T, α, β) pour G. Les conventions de représentation utilisées pour cette figure sont les suivantes. Les sommets du graphe G sont désignés par les entiers de l'ensemble $V = \{1, \cdots, 7\}$. Les arêtes du graphe G sont désignées par les lettres minuscules de l'ensemble $E = \{a, b, c, d, e, f, g, h, i\}$. Les sommets de l'arbre T sont désignés par les lettres majuscules de l'ensemble $V(T) = \{U, V, W, Z, X\}$. Pour chaque sommet Y de $V(T)$, la partie de E indiquée dans le cercle représentant Y désigne l'ensemble $\beta(Y) \subseteq E$. Pour chaque arête $e \in E(T)$, incidente à un sommet $Y \in V(T)$, l'ensemble $\alpha(Y, e) \subseteq E$ est représentée le long du trait représentant e, du côté de Y. Notons que les arêtes $\{U, V\}$ et $\{X, V\}$ sont sales. Remarquons que $\chi_U = \{1, 2, 5, 6, 7\}$, $\chi_V = \{2, 3, 5, 6\}$, $\chi_W = \{5, 6, 7\}$, $\chi_X = E$ et $\chi_Z = \{3, 4, 5\}$. Nous en déduisons que la largeur de (T, α, β) est égale à $|\chi_X| = 7$. Il est facile de vérifier que $(T, (\chi_v)_{v \in V(T)})$ satisfait les conditions **C1** et **C2** des décompositions arborescentes, mais que la conditions **C3** n'est pas satisfaite.

2.3.2 Le jeu de capture non-déterministe est monotone

Cette section est dévolue à la preuve de la monotonie du jeu de capture non-déterministe. La preuve se décompose en trois étapes principales. Premièrement, nous prouvons que s'il existe un q-programme utilisant au plus q chercheurs pour nettoyer un graphe G, nous pouvons construire un arbre de capture q-branché de largeur au plus k pour G (lemme 2). Puis, par des optimisations locales, nous transformons cet arbre de capture q-branché en un arbre de capture q-branché monotone, sans augmenter sa largeur (lemme 4). Enfin, nous prouvons qu'à partir de tout arbre de capture q-branché monotone de largeur k pour G, il est possible de construire une décomposition arborescente q-branchée de largeur au plus $k-1$ pour G (lemme 6). La preuve de la monotonie du jeu de capture non-déterministe découle alors facilement du théorème 12. Plus formellement, nous prouvons le théorème suivant :

Théorème 13 *Soit G un graphe connexe, $q \geq 0$ et $k \geq 1$. Les assertions suivantes sont équivalentes :*

i. Il existe un q-programme qui nettoie G utilisant au plus k chercheurs ;

ii. Il existe un arbre de capture q-branché de largeur au plus k pour G ;

iii. Il existe un arbre de capture q-branché monotone de largeur au plus k pour G ;

iv. Il existe une décomposition arborescente q-branchée de largeur au plus $k-1$ pour G ;

v. Il existe un q-programme monotone qui nettoie G utilisant au plus k chercheurs.

Preuve. Si $k = 1$, le résultat est trivial puisque le graphe est réduit à un unique sommet. Dans la suite, nous supposons que $k \geq 2$. Nous prouvons que $i \Rightarrow ii$ (Lemme 2), $ii \Rightarrow iii$ (Lemme 4), et $iii \Rightarrow iv$ (Lemme 6). L'implication $iv \Rightarrow v$ découle du théorème 12, et $v \Rightarrow i$ est trivial. □

Lemme 2 *Soit G un graphe connexe, $q \geq 0$ et $k \geq 2$, $i \Rightarrow ii$*

i. Il existe un q-programme qui nettoie G utilisant au plus k chercheurs;

ii. Il existe un arbre de capture q-branché de largeur au plus k pour G.

Preuve. Pour cette preuve, nous considérons des programmes de capture dont la *configuration de départ* n'est pas nécessairement la configuration $(\emptyset, \emptyset)$. C'est-à-dire que nous considérons des programmes de capture qui débutent d'une configuration (Z_0, A_0) vérifiant $\delta(A_0) \subseteq Z_0$. La *longueur* d'un programme de capture est le nombre maximum d'étapes parmi toutes les stratégies que le programme calcule. Nous définissons également la *largeur partielle* d'un arbre de capture enraciné comme le maximum des poids de ses sommets, ce maximum ne prenant pas en compte le poids de la racine.

Nous prouvons l'assertion suivante par induction sur la longueur du programme de capture.

Assertion 2 *Pour tout q-programme gagnant utilisant au plus k chercheurs, avec (Z_0, A_0) comme configuration de départ ($\delta(A_0) \subseteq Z_0$), il existe un arbre de capture q-branché enraciné (T, α, β, r) de largeur partielle au plus k, et tel que, r est incident à une unique arête $e \in E(T)$, et $\alpha(r, e) = E \setminus A_0$.*

Soit $q \geq 0$ et soit $\mathcal{S}$ un q-programme gagnant pour G utilisant k chercheurs, avec (Z_0, A_0) comme configuration de départ, $\delta(A_0) \subseteq Z_0$.

- Supposons que $\mathcal{S}$ est de longueur 1.
  Dans ce cas, il n'y a qu'une stratégie qui peut être calculée par le programme. Cette stratégie consiste en une étape "placer un chercheur". Donc, $\mathcal{S}$ renvoie seulement la stratégie 0-branchée : $(Z_0, A_0), (Z_1, A_1)$ où $Z_1 = Z_0 \cup \{x_1\}$ et $A_1 = A_0 \cup B_1 = E$.
  Nous définissons l'arbre T avec une seule arête $\{r, v\}$, $\beta(v) = \alpha(r, \{r, v\}) = E \setminus A_0$ et $\beta(r) = \alpha(v, \{r, v\}) = A_0$. Puisque $V\big[\beta(v)\big] \cup \delta(\mu_v) = V[E \setminus A_0]$ qui est un sous-ensemble de Z_1, (T, α, β, r) est un arbre de capture 0-branché de largeur partielle au plus k.
- Supposons que $\mathcal{S}$ soit de longueur $l > 1$. Considérons $\mathcal{S}'$ obtenu en supprimant la première configuration du programme $\mathcal{S}$. Notons que, $\mathcal{S}'$ est strictement plus courte que $\mathcal{S}$. Nous considérons trois cas selon le type de la première étape de $\mathcal{S}$.
  a. si la première étape de $\mathcal{S}$ consiste à supprimer un chercheur, $\mathcal{S}'$ est un q-programme avec (Z_1, A_1) comme configuration de départ, $Z_1 \subseteq Z_0$ et $A_1 \subseteq A_0$. D'après l'hypothèse d'induction, il existe un arbre de capture q-branché $(T', \alpha', \beta', r')$ de largeur partielle au plus k pour G, et tel que il existe une arête e' incidente à r' avec $\alpha'(r', e') = E \setminus A_1$.
     Nous définissons le nouvel arbre de capture q-branché (T, α, β, r) à partir de $(T', \alpha', \beta', r')$ de la manière suivante :
     - ajoutons une nouvelle feuille r incidente à r' dans T', et posons r en tant que nouvelle racine,
     - posons $\alpha\big(r, \{r, r'\}\big) = E \setminus A_0$, $\alpha(r', \{r, r'\}) = A_1$ et $\alpha = \alpha'$ partout ailleurs;
     - posons $\beta(r) = A_0$, $\beta(r') = \emptyset$ et $\beta = \beta'$ partout ailleurs.

Puisque $A_1 \subseteq A_0$, $\alpha(r, \{r, r'\}) \cap \alpha(r', \{r, r'\}) = \emptyset$ et (T, α, r) est un arbre de capture q-branché. De plus, $V\big[\beta(r')\big] \cup \delta\big(\mu(r')\big) \subseteq Z_1$ et (T, α, β, r) satisfait les conditions requises.

b. si la première étape de $\mathcal{S}$ consiste à placer un chercheur, $\mathcal{S}'$ est un q-programme avec (Z_1, A_1) comme configuration de départ, $Z_1 = Z_0 \cup \{x_1\}$ et $A_1 = A_0 \cup B_1$. D'après l'hypothèse d'induction, il existe un arbre de capture q-branché $(T', \alpha', \beta', r')$ de largeur partielle au plus k, et tel que il existe une arête e' incidente à r' avec $\alpha'(r', e') = E \setminus A_1$.

   Nous définissons le nouvel arbre de capture q-branché (T, α, β, r) à partir de $(T', \alpha', \beta', r')$ de la manière suivante :
   - ajoutons une nouvelle feuille r incidente à r' dans T', et posons r en tant que nouvelle racine,
   - posons $\alpha\big(r, \{r, r'\}\big) = E \setminus A_0$, $\alpha(r', \{r, r'\}) = A_0$ et $\alpha = \alpha'$ partout ailleurs ;
   - posons $\beta(r) = A_0$, $\beta(r') = B_1$ et $\beta = \beta'$ partout ailleurs.

   Par construction, (T, α, β, r) est un arbre de capture q-branché qui satisfait les conditions requises.

c. si la première étape de $\mathcal{S}$ consiste à poser une question, il existe $p \geq 1$ distincts $(q-1)$-programmes $\mathcal{S}_1, \ldots, \mathcal{S}_p$ pour G tels que : $\{A_0, E \setminus Y_1, \ldots, E \setminus Y_p\}$ est une partition de E, et, pour tout i, $1 \leq i \leq p$, $\mathcal{S}_i$ est un $(q-1)$-programme gagnant pour G, avec (Z_i, Y_i) comme configuration de départ, et utilisant au plus k chercheurs. Pour tout i, $1 \leq i \leq p$, puisque les $(q-1)$-programmes $\mathcal{S}_i$ sont plus court que $\mathcal{S}$, d'après l'hypothèse d'induction, il existe des arbres de capture $q-1$-branchés $(T_i, \alpha_i, \beta_i, r_i)$ de largeur partielle au plus k, et tels qu'il existe une arête e_i incidente à r_i avec $\alpha_i(r_i, e_i) = E \setminus Y_i$.

   Nous définissons le nouvel arbre de capture q-branché (T, α, β, r) à partir de ces arbres de capture de la manière suivante :
   - identifions les racines r_i à un seul sommet r', ajoutons une nouvelle feuille r liée à r' dans T', et posons r en tant que nouvelle racine ;
   - posons $\alpha(r, \{r, r'\}) = E \setminus A_0$, $\alpha(r', \{r, r'\}) = A_0$ et $\alpha(u, e) = \alpha_i(u, e)$ pour toute arête e de T_i ;
   - posons $\beta(r) = A_0$, $\beta(r') = \emptyset$, et, pour tout $1 \leq i \leq p$ et pour tout sommet u de T_i, $\beta(u) = \beta_i(u)$.

   L'arbre de capture enraciné (T, α, β, r) a un sommet de branchement de plus que chacun des arbres de capture $(T_i, \alpha_i, \beta_i, r_i)$ et, puisque chacun d'eux a au plus $q-1$ sommets de branchement, (T, α, β, r) satisfait les conditions requises.

L'induction est vraie pour les programmes de longueur $l+1$. Ce qui conclut la preuve de l'assertion.

Pour conclure la preuve du lemme, il suffit de remarquer que, si $A_0 = \emptyset$, le poids de la racine de l'arbre de capture vaut 0. Donc, sa largeur partielle est égale à sa largeur. ☐

A présent, nous introduisons la notation suivante sur les partitions de E. Soit $\mu = \{E_1, \cdots, E_p\}$ une partition (éventuellement dégénérée) de E. Soient $i \in \{1, \cdots, p\}$ et

$F \subseteq E$ tel que $F \cap E_i = \emptyset$. Nous dirons que la partition $\mu_{E_i \to F}$ est obtenue en *poussant* E_i sur F, avec $\mu_{E_i \to F} = \{E_1 \cap F, \cdots, E_{i-1} \cap F, E \setminus F, E_{i+1} \cap F, \cdots, E_p \cap F\}$.

Avant de poursuivre avec la deuxième étape de notre preuve, nous avons besoin du lemme technique suivant.

Lemme 3 *Soient $G = (V, E)$ un graphe connexe, $\mu = \{E_1, \ldots, E_p\}$ et $\eta = \{F_1, \ldots, F_{p'}\}$ deux partitions (éventuellement dégénérées) de E. Soient $i \in \{1, \cdots, p\}$ et $j \in \{1, \cdots, p'\}$ tels que $E_i \cap F_j = \emptyset$. Alors,*

$$|\delta(\mu)| + |\delta(\eta)| \geq |\delta(\mu_{E_i \to F_j})| + |\delta(\eta_{F_j \to E_i})|$$

Preuve. Un sommet v de $\delta(\mu_{E_i \to F_j})$ est, soit incident à une arête de $E_a \cap F_j$ et à une arête de $E_b \cap F_j$ $(a, b \neq i)$, soit à une arête de $E_a \cap F_j$ $(a \neq i)$ et à une arête de $E \setminus F_j$. Dans le premier cas, $v \in \delta(\mu)$, et dans le second, $v \in \delta(\eta)$. Nous prouvons de même que $\delta(\eta_{F_j \to E_i}) \subseteq \delta(\mu) \cup \delta(\eta)$. Ainsi, un sommet qui contribue pour un à la somme de droite, contribue au moins pour un à la somme de gauche.

Soit $v \in \delta(\mu_{E_i \to F_j}) \cap \delta(\eta_{F_j \to E_i})$, contribuant pour deux à la somme de droite. v est incident à une arête de $E_a \cap F_j$ $(a \neq i)$ et une arête de $F_b \cap E_i$ $(b \neq j)$. Donc $v \in \delta(\mu) \cap \delta(\eta)$, et contribue pour deux à la somme de gauche. $\square$

Lemme 4 *Soient G un graphe connexe, $q \geq 0$ et $k \geq 2$, ii $\Rightarrow$ iii.*

ii. Il existe un arbre de capture q-branché de largeur au plus k pour G ;

iii. Il existe un arbre de capture q-branché monotone de largeur au plus k pour G.

Preuve. Soit $\mathcal{T} = (T, \alpha, \beta, r)$ un arbre de capture q-branché enraciné de largeur k pour G.

Soit $v \in V(T)$ incident à une arête $e = \{u, v\}$, tel que $\alpha(v, e) \cap \alpha(u, e) = \emptyset$. Soient $e = e_1, \cdots, e_p$ les arêtes incidentes à v $(p \geq 1)$. Rappelons que $\mu(v) = \{\alpha(v, e_1), \cdots, \alpha(v, e_p)\}$. La *partition associée* à v est $\eta(v) = \mu(v) \cup \{\beta(v)\}$. Nous dirons que nous *poussons, dans T*, la partition associée à v de $\alpha(v, e_1)$ vers $\alpha(u, e_1)$, si nous remplaçons la partition $\eta(v)$ associée à v par $\eta(v)_{\alpha(v,e_1) \to \alpha(u,e_1)}$. C'est-à-dire, si nous remplaçons $\alpha(v, e_1)$ par $E \setminus \alpha(u, e_1)$, $\beta(v)$ par $\beta(v) \cap \alpha(u, e_1)$, et, pour tout $2 \leq i \leq p$, $\alpha(v, e_i)$ par $\alpha(v, e_i) \cap \alpha(u, e_1)$. Nous posons $|\chi'_{\mathcal{T}}(v)|$ le poids de v dans l'arbre ainsi obtenu.

Il est important de remarquer qu'en poussant, dans T, la partition associée à v de $\alpha(v, e_1)$ vers $\alpha(u, e_1)$, nous obtenons un autre arbre de capture q-branché. De plus, si e était sale, elle devient monotone. Enfin, nous prouvons l'assertion suivante

Assertion 3 *Si $|\delta(\eta(v))| > |\delta(\eta(v)_{\alpha(v,e_1) \to \alpha(u,e_1)})|$, le poids de v dans le nouvel arbre est strictement inférieur à $|\chi_{\mathcal{T}}(v)|$.*

En effet, $|\chi_{\mathcal{T}}(v)| = |\delta(\mu(v)) \cup V[\beta(v)]| = |\delta(\eta_v) \cup (V[\beta(v)] \setminus \delta(\beta(v)))| = |\delta(\eta(v))| + |V[\beta(v)] \setminus \delta(\beta(v))| > |\delta(\eta(v)_{\alpha(v,e_1) \to \alpha(u,e_1)})| + |V[\beta(v) \cap \alpha(u, e_1)] \setminus \delta(\beta(v) \cap \alpha(u, e_1))| = |\chi'_{\mathcal{T}}(v)|$ $\diamond$

Prouvons à présent qu'il est possible de réaliser cette opération jusqu'à ce que l'arbre de capture soit monotone et sans augmenter sa largeur. Pour cela, nous définissons le *poids $wg(\mathcal{T})$* d'un arbre de capture T par $\sum_{v \in V(T)} |\chi_{\mathcal{T}}(v)|$.

Supposons que $\mathcal{T} = (T, \alpha, \beta, r)$ est choisi de poids minimum. Supposons, de plus, que T n'est pas monotone. Soit $e = \{u, v\} \in E(T)$ une arête sale. Supposons que u est plus proche de r que v.

D'après le lemme 3, $|\delta(\eta(u))| + |\delta(\eta(v))| \geq |\delta(\eta(u)_{\alpha(u,e_1)\to\alpha(v,e_1)})| + |\delta(\eta(v)_{\alpha(v,e_1)\to\alpha(u,e_1)})|$. Puisque $\mathcal{T}$ est de poids minimum, l'assertion 3 implique : $|\delta(\eta(v))| = |\delta(\eta(v)_{\alpha(v,e_1)\to\alpha(u,e_1)})|$. Sinon, en poussant, dans T, la partition associée à u, ou à v, nous obtiendrions un arbre de poids strictement inférieur. En poussant, dans T, la partition associée à v de $\alpha(v, e_1)$ vers $\alpha(u, e_1)$, nous obtenons donc un autre arbre de capture q-branché sans augmenter ni son poids, ni sa largeur. De plus, nous avons rendu e monotone.

Enfin, ce processus termine avec toutes les arêtes propres. En effet, cette opération “repousse” les arêtes sales vers les feuilles. Plus formellement, pour toute arête $e \in E(T)$, $\mathrm{dist}(e)$ désigne la distance de e à r. A chaque application de l'opération décrite plus haut, la fonction $\sum_{e\ sale} n^{-\mathrm{dist}(e)}$ décroît strictement. $\square$

Les deux lemmes suivants concluent la troisième étape de la preuve du théorème 13.

Lemme 5 *Soient G un graphe connexe et $\mathcal{T} = (T, \alpha, \beta, r)$ un arbre de capture monotone pour G. Pour toute arête $\{u, v\}$ de T, $\alpha(u, \{u, v\}) = \beta(T_v)$ avec T_v la composante connexe de $T \setminus \{u, v\}$ qui contient v.*

Preuve. La preuve est par induction sur $|V(T_v)|$.

- si $|V(T_v)| = 1$, alors $\beta(v) = E \setminus \alpha(v, \{u, v\})$ et puisque $\alpha(u, \{u, v\}) = E \setminus \alpha(v, \{u, v\})$ ($\mathcal{T}$ est monotone), nous obtenons $\alpha(u, \{u, v\}) = \beta(v) = \beta(T_v)$.
- sinon, soient $w_1, \dots, w_p$ les voisins de v dans T_v, et pour $1 \leq i \leq p$, soit T_{w_i} la composante connexe de $T_v \setminus \{v, w_i\}$ qui contient w_i. D'après l'hypothèse d'induction, $\alpha(v, \{v, w_i\}) = \beta(T_{w_i})$. Puisque T est un arbre de capture, les ensembles $\beta(v)$, $\alpha(v, \{u, v\})$ et $\beta(T_{w_1}), \dots, \beta(T_{w_p})$ induisent une partition de E, donc $\alpha(v, \{u, v\}) = E \setminus \beta(T_v)$. Puisque $\mathcal{T}$ est monotone, $\alpha(u, \{u, v\}) = E \setminus \alpha(v, \{u, v\}) = \beta(T_v)$, ce qui conclut la preuve.

$\square$

Lemme 6 *Soient G un graphe connexe, $q \geq 0$ et $k \geq 2$, iii $\Rightarrow$ iv.*

iii. Il existe un arbre de capture q-branché monotone de largeur au plus k pour G ;

iv. Il existe une décomposition arborescente q-branchée de largeur au plus $k-1$ pour G.

Preuve. Soit $\mathcal{T} = (T, \alpha, \beta, r)$ un arbre de capture q-branché monotone de largeur au plus k pour G.

Nous affirmons que $\Theta = \big(T, \mathcal{X}, r\big)$ avec $\mathcal{X} = \{\chi_v \mid v \text{ nœud de } T\}$ est une décomposition arborescente q-branchée de largeur au plus $k-1$ pour G.

Puisque G est connexe, et $|E| > 0$, la propriété **C2** des décompositions arborescentes implique la propriété **C1**.

Soit $\{x, y\} \in E$ une arête de G. Puisque $\mathcal{T}$ est monotone, pour toute arête $\{u, v\}$ de T, $\{x, y\}$ appartient soit à $\alpha(u, \{u, v\})$, soit à $\alpha(v, \{u, v\})$. Supposons $\{x, y\} \in \alpha(u, \{u, v\})$, d'après le lemme 5, $\{x, y\} \in \beta(T_v)$ avec T_v la composante connexe de $T \setminus \{u, v\}$ qui contient v. L'arête $\{x, y\}$ appartient donc à au moins un ensemble $\beta(w)$ pour un certain sommet w de T_v. Par définition de χ_w, $\{x, y\} \subseteq \chi_w$.

Soient u, v, w trois sommets de T avec v sur le chemin $\{u, u', \ldots, v, \ldots, w', w\}$ de u à w. Soit T_u (resp., T_w) la composante connexe de $T \setminus \{u, u'\}$ (resp., $T \setminus \{w, w'\}$) qui contient u (resp., w). Soit T_u^v (resp., T_w^v) la composante connexe de $T \setminus v$ qui contient u (resp., w).

Soient $u_1 = u', \ldots, u_p$ les voisins de u dans T, et $x \in \chi_u$. Soit il y a une arête de G incidente à x dans $\beta(u)$, soit il existe $1 < i \leq p$ tel qu'une arête de $\alpha(u, \{u, u_i\})$ est incidente à x. D'après le lemme 5, il existe une arête incidente à x dans $\beta(T_{u_i}) \subseteq \beta(T_u)$.

Supposons que $x \in \chi_u \cap \chi_w$. Il existe une arête incidente à x dans $\beta(T_u^v) \supseteq \beta(T_u)$ et une arête incidente à x dans $\beta(T_w^v) \supseteq \beta(T_w)$. D'après le lemme 5, il s'en suit : $x \in \delta(\mu_v)$. Donc, $x \in \chi_v$. Ceci prouve que Θ est une décomposition arborescente. De plus, par construction, $w(\Theta) = \text{width}(\mathcal{T}) - 1$. Puisque $\mathcal{T}$ et Θ utilisent le même arbre sous-jacent, Θ est une décomposition arborescente q-branchée de largeur au plus $k - 1$. □

2.4 Algorithmes et bornes

Dans cette section, nous prouvons que le problème de décision associé à l'indice d'échappement non-déterministe est NP-complet. Nous proposons un algorithme exponentiel exact résolvant ce problème. Dans un deuxième temps, nous étudions le gain, en terme de nombre de chercheurs, que nous pouvons espérer lors du nettoyage d'un graphe si nous nous autorisons une question supplémentaire. Cette étude permet de déterminer une borne sur le nombre minimum de questions qu'il est nécessaire de poser pour nettoyer un graphe G en utilisant $tw(G) + 1$ chercheurs.

2.4.1 NP-complétude et algorithme exponentiel exact

Pour tout $q \geq 0$, le problème de décision qui prend un graphe G et un entier $k \geq 1$ comme entrées, et qui détermine si $tw_q(G) \leq k$ est NP-complet. En effet, déterminer si $tw(G) \leq k$ est connu pour être NP-complet [ACP87], même si l'entrée est restreinte à des graphes co-bipartis, c'est-à-dire, le complémentaire de graphes bipartis. Puisque pour tout graphe co-biparti G, $tw(G) = pw(G)$ [ACP87], la NP-complétude de décider si $tw_q(G) \leq k$ découle alors du fait que $tw(G) \leq tw_q(G) \leq pw(G)$, pour tout $q \geq 0$. Ce résultat prouve la NP-difficulté du problème de décision correspondant à l'indice d'échappement non-déterministe. La NP-complétude découle de la monotonie que nous avons prouvé à la section précédente. En résumé, nous avons :

Théorème 14 *Le problème de décision suivant est NP-complet :*
Entrée : un graphe G, un entier $k > 0$ et un entier $q \geq 0$,
Question : $s_q(G) \leq k$?

Selon le récent résultat de Feige *et al.* [FHL05a], la largeur arborescente d'un graphe G peut être approchée avec un facteur multiplicatif $O(\sqrt{\log tw(G)})$ en temps polynomial. Cependant, aucun algorithme d'approximation n'est connu pour la largeur linéaire (si ce n'est en combinant ceux existant pour la largeur arborescente avec le fait que $pw(G) \leq tw(G) \cdot O(\log n)$. Par ailleurs, plusieurs algorithmes exponentiels exacts ont été proposés pour calculer les largeurs arborescentes et linéaires [FKT04, BFK$^+$06]. Actuellement, le meilleur algorithme exact calcule la largeur arborescente en temps $O(1.8899^n)$ [Vil06]. Dans cette section, nous proposons un algorithme exact qui calcule la largeur arborescente q-branchée, pour tout $q \geq 0$. Cet algorithme utilise la correspondance entre largeur arborescente q-branchée et indice d'échappement q-limité.

Théorème 15 *Il existe un algorithme qui, pour tout graphe G de n sommets, calcule $tw_q(G)$ et une décomposition arborescente q-branchée optimale de G, en temps $O(2^n n \log n)$.*

Preuve. Nous proposons un algorithme qui calcule $s_q(G)$ et une stratégie q-limitée pour G. Cette stratégie peut être transformée en une décomposition arborescente q-branchée optimale, en utilisant les arguments énoncés dans la preuve du théorème 12. Soit un graphe G, et fixons $k \geq 1$. Le graphe dirigé H des k-configurations est défini comme suit :

$$V(H) = \{S \subseteq V(G) \ tel\ que\ |\delta(S)| \leq k\}$$

Un ensemble S de sommets propres tel que $|\delta(S)| > k$ n'est accessible par aucune stratégie de capture utilisant k chercheurs, et ne correspond donc à aucun sommet de H. Les sommets de H sont appelés les H-configurations. L'ensemble des arêtes de H est composé de deux types d'arêtes dirigées : les *arêtes de placements* et les *arêtes de requêtes*. Une arête de placement, ou simplement une p-arête, est une arête $\{S, S'\}$ avec $|\delta(S)| < k$, et $S' = S \cup \{v\}$, $v \notin S$. Evidemment, les p-arêtes correspondent au placement d'un chercheur sur un sommet v. Une arête de requête, ou q-arête, est une arête $\{S, S'\}$ avec $S' = G \setminus C$ où C est une composante connexe de $G \setminus S$. Nous supposons qu'il existe une arête $\{S, G \setminus C\}$ seulement si $G \setminus S$ contient au moins deux composantes connexes (c'est-à-dire, nous n'autorisons pas les boucles). Donc, une q-arête $\{S, G \setminus C\}$ correspond à une question à laquelle l'oracle répond la composante C. L'objectif de notre algorithme est de trouver un chemin dans le graphe des configurations H de $S = \emptyset$ à $S = V(G)$. Ce chemin peut alors être associé à une stratégie non-déterministe.

Dans le but de trouver un tel chemin, nous étiquetons les sommets de H avec un entier positif. L'étiquetage débute de la H-configuration $V(G)$, et se propage vers la H-configuration $\emptyset$. La H-configuration $V(G)$ est étiquetée 0. Toutes les autres H-configurations sont étiquetées ∞. Toutes les H-configurations de degré sortant nul sont dites *terminales*. En particulier, $V(G)$ est terminale. Une H-configuration non terminale est dite *pendante*. L'étiquetage se déroule tant qu'il existe au moins une H-configuration pendante S satisfaisant une des deux conditions suivantes :

- **Cas 1.** S est incidente à une p-arête sortante $e = \{S, S'\}$ avec S' terminale.
- **Cas 2.** S est incidente à $d \geq 2$ q-arêtes sortantes
  $e_1 = \{S, S_1\}, e_2 = \{S, S_2\}, \cdots, e_d = \{S, S_d\}$, avec $S_1, \cdots, S_d$ terminales.

Dans le premier cas, nous mettons à jour l'étiquette de S en posant :

$$etiquette(S) = \min\{etiquette(S), etiquette(S')\}$$

et l'arête e est supprimée de H.

Dans le second cas, nous mettons à jour l'étiquette de S en posant :

$$etiquette(S) = \min\{etiquette(S), 1 + \max_{1 \leq i \leq d} etiquette(S_i)\}$$

et les arêtes $e_1, \cdots, e_d$ sont supprimées de H. Dans les deux cas, si la H-configuration pendante S n'est plus incidente à une arête sortant du fait de cette suppression, alors S devient terminale.

Assertion 4 *Le processus d'étiquetage termine.*

Pour prouver cette assertion, notons que H est un graphe dirigé acyclique puisque, pour toute arête $\{S, S'\}$, $|S'| > |S|$ (Rappelons que les boucles ne sont pas autorisées). Supprimer des arêtes conserve cette propriété. Ainsi, toute H-configuration finit par devenir terminale, et donc le processus termine.

Assertion 5 *La H-configuration $\emptyset$ est étiquetée $q < \infty$ si et seulement si q est le plus petit nombre de questions requises pour nettoyer G utilisant au plus k chercheurs. La H-configuration $\emptyset$ est étiquetée ∞ si k chercheurs ne peuvent pas nettoyer G, indépendamment du nombre de questions posées.*

Nous prouvons cette assertion en prouvant un résultat légèrement plus fort : pour toute H-configuration $S \neq V(G)$, S est terminale et étiquetée $etiquette(S) \leq q \neq \infty$ si et seulement si G peut être nettoyé, en partant de S avec au plus k chercheurs et en posant au plus q questions. Nettoyer G en partant de S signifie que nous supposons que les sommets de S sont propres et que $|\delta(S)|$ chercheurs occupent les sommets de $\delta(S)$, et par conséquent, que le fugitif occupe un sommet de $G \setminus S$. La preuve se fait par induction sur $q = etiquette(S)$.

Si $q = 0$, alors il existe un chemin P dans H de S à $V(G)$ qui n'est composé que de p-arêtes. Soit $\{S', S''\} \in P$, avec $S'' = S' \cup \{v\}$. Les étapes de la stratégie qui correspondent à cette arête consiste à supprimer un par un les chercheurs qui occupent des sommets n'appartenant pas $\delta(S')$, et à placer un chercheur sur v. Ainsi, le fugitif peut être capturé sans poser une question, en commençant de S, et en suivant les arêtes de P jusqu'à atteindre la H-configuration $V(G)$. Réciproquement, si G peut être nettoyé en partant de S avec au plus k chercheurs et sans poser de questions, alors il existe un chemin dans H, de S à $V(G)$ composé uniquement de p-arêtes. Ces arêtes sont définies par les étapes de placement dans la stratégie.

Supposons maintenant que le résultat est valide pour q, et considérons S telle que $etiquette(S) = q + 1$. Nous définissons une *bonne* arête comme étant une p-arête $\{S', S''\}$

telle que $etiquette(S) = etiquette(S') = etiquette(S'')$. A partir de S, commençons à parcourir H en empruntant uniquement les bonnes arêtes, jusqu'à atteindre une configuration S^* telle que :

$$etiquette(S) = etiquette(S^*) = 1 + \max_{1 \leq i \leq d} etiquette(S_i^*)\}$$

avec les arêtes $\{S^*, S_i^*\}$, $i = 1, \cdots, d$, sont les q-arêtes sortantes de S^*. Cette configuration S^* existe car :

1. une bonne arête $\{S', S''\}$ satisfait $|S''| > |S'|$, et
2. $etiquette(S') = etiquette(S'') = etiquette(S) < \infty$.

Ainsi, si aucune configuration S^* définie précédemment n'était rencontrée, alors, par (1) une H-configuration de degré sortant nul serait finalement atteinte, et, par (2) cette H-configuration ne pourrait qu'être $V(G)$ puisque son étiquette est finie. Puisque $etiquette(S) = q + 1 > 0$, l'hypothèse d'induction contredirait le fait qu'il n'existe pas de chemin de S à $V(G)$ composé uniquement de p-arêtes. Donc S^* existe.

Pour tout $i = 1, \cdots, d$, $etiquette(S_i^*) \leq q$. Donc, d'après l'hypothèse d'induction, G peut être nettoyé en utilisant au plus k chercheurs, en partant de S_i^*, et en posant au plus q questions. La stratégie non-déterministe partant de S débute en effectuant les étapes de placement et de suppression de chercheurs, définies d'après le chemin entre S et S^* dans H. La stratégie consiste ensuite à poser une question à l'oracle. A la suite de la réponse de l'oracle, l'une des configurations S_i^* est atteinte. La fin de la stratégie découle de l'hypothèse d'induction.

Réciproquement, supposons que G peut être nettoyé avec au plus k chercheurs, en partant de S, et en posant au plus $q + 1$ questions. Soit une stratégie non-déterministe correspondante, et soit $t \geq 0$ la première étape à laquelle une question est posée. Aux $t - 1$ premières étapes peut être associé un chemin P dans H, commençant en S, et ne contenant que des bonnes arêtes. P connecte S à une H-configuration S^*. La question à l'étape t correspond aux q-arêtes sortantes $\{S^*, S_i^*\}$, $i = 1, \cdots, d$, de S^*. A partir de chacune des configurations S_i^*, la stratégie se poursuit en posant au plus q questions. Donc, d'après l'hypothèse d'induction, $etiquette(S_i^*) \leq q$. D'où, $etiquette(S^*) \leq q + 1$. Puisque P n'est constitué que de bonnes arêtes, $etiquette(S) = etiquette(S^*) \leq q + 1$.

Pour tout k, le temps d'exécution de notre processus d'étiquetage est linéaire en le nombre d'arête de H, qui est $O(2^n n)$. Donc, par dichotomie, l'indice d'échappement non-déterministe et $tw_q(G)$ peut être calculé en temps $O(2^n n \log n)$. □

2.4.2 Compromis entre nombre de chercheurs et nombre de questions

Dans cette section, nous déterminons une borne inférieure sur le nombre d'étapes non-déterministes, c'est-à-dire de questions, qu'une stratégie de capture non-déterministe doit effectuer dans un graphe G, pour nettoyer ce graphe avec le nombre minimum de chercheurs, c'est-à-dire avec $tw(G) + 1$ chercheurs.

Nous prouvons tout d'abord que si les chercheurs disposent d'une question supplémentaires, leur nombre peut être, au mieux, divisé par deux. Intuitivement, si S est une stratégie permettant de nettoyer un graphe G en utilisant au plus $k \geq 1$ chercheurs posant au plus $q \geq 1$ questions, pour nettoyer G en posant au plus $q-1$ questions, il suffit de réaliser la stratégie S avec k agents jusqu'à ce que les $q-1$ questions soient épuisées. Nous prouvons alors que $2k$ chercheurs sont suffisants pour conclure le nettoyage de G.

Théorème 16 *Pour tout graphe G, et pour tout $q \geq 1$, $tw_{q-1}(G) \leq 2\ tw_q(G)$.*

Preuve. Nous avons d'abord besoin d'un lemme technique. Soient G un graphe et $(T, \mathcal{X}, r)$ une décomposition arborescente de G, enracinée en r. Pour tout $i \in V(T)$, le sous-arbre de T enraciné en i est noté T_i. Soit $(T', \mathcal{X}', i)$ une décomposition arborescente de $G[T_i]$ telle que $X_i = X'_i$. Soit T_{new} l'arbre obtenu à partir de T et T', en supprimant T_i de T, et en le remplaçant par T' enraciné en i. Soit $\mathcal{X}_{new} = \{Y_j, j \in V(T_{new})\}$ la famille de sous-ensembles de $V(G)$ telle que $Y_j = X'_j$ si $j \in V(T')$ et $Y_j = X_j$ si $j \in V(T_{new}) \setminus V(T')$. L'assertion suivante est triviale :

Assertion 6 *$(T_{new}, \mathcal{X}_{new})$ est une décomposition arborescente de G.*
De plus, $w(T_{new}, \mathcal{X}_{new}) \leq \max\{w(T, \mathcal{X}), w(T', \mathcal{X}')\}$.

Soit $(T, \mathcal{X}, r)$ une décomposition arborescente q-branchée optimale de G, c'est-à-dire, $w(T, \mathcal{X}) = tw_q(G)$. Nous définissons les sommets de branchement *les plus bas* de T comme étant des sommets $i \in V(T)$ qui satisfont :

1. i possède au moins deux enfants dans T, c'est-à-dire, i est un sommet de branchement, et
2. pour tout $j \in V(T_i) \setminus \{i\}$, j possède au plus un enfant.

Soit Π l'ensemble des chemins de r à une feuille de T qui contiennent exactement q sommets de branchement. Soit J le sous-ensemble des sommets de branchement les plus bas qui appartiennent à un chemin de Π. Soit $i \in J$. T_i est constitué du sommet i et de $d \geq 2$ chemins $P_1, \cdots, P_d$ attachés en i. Nous définissons le chemin T' obtenu en concaténant les chemins P_i, c'est-à-dire, $T' = \{i, P_1, \cdots, P_d\}$. Tout sommet j de T' vient d'un sommet de T, nous pouvons donc définir $X'_j = X_i \cap X_j$. Soit $\mathcal{X}' = \{X'_j \mid j \in V(T')\}$. $(T', \mathcal{X}')$ est une décomposition linéaire de $G[T_i]$, de largeur au plus $2\ tw_q(G)$. Soit $T^{(i)}_{new}$ l'arbre obtenu à partir de T et T', en supprimant T_i de T, et en le remplaçant par T' enraciné en i. Soit $\mathcal{X}^{(i)}_{new} = \{Y_j, j \in V(T^{(i)}_{new})\}$ la famille de sous-ensemble de $V(G)$ telle que $Y_j = X'_j$ si $j \in V(T')$ et $Y_j = X_j$ si $j \in V(T^{(i)}_{new}) \setminus V(T')$. D'après l'assertion ci-dessus, $(T^{(i)}_{new}, \mathcal{X}^{(i)}_{new})$ est une décomposition arborescente de G de largeur au plus $2\ tw_q(G)$.

Nous répétons ce procédé pour chaque sommet $i \in J$. Puisque, pour tous sommets $i \neq i'$ de J, $T_i \cap T_{i'} = \emptyset$, chaque application du procédé de remplacement est indépendant des précédents. Par conséquent, nous obtenons finalement une décomposition arborescente $(T^{(*)}, \mathcal{X}^{(*)}, r)$ de G, de largeur au plus $2\ tw_q(G)$. Par construction, tout chemin de r à une feuille de $T^{(*)}$ contient au plus $q-1$ sommets de branchement. Donc, $tw_{q-1}(G) \leq 2\ tw_q(G)$.
□

Le théorème 16 implique une borne inférieure sur le nombre minimum de questions $\tau(G)$ qu'il est nécessaire de poser pour nettoyer un graphe G en utilisant au plus $tw(G)+1$ chercheurs.

Corollaire 1 *Pour tout graphe G, $\tau(G) \geq \log_2(pw(G)/tw(G))$.*

Preuve. Par définition, $\tau(G)$ est le plus petit entier q tel que $tw_q(G) = tw(G)$. En appliquant $\tau(G)$ fois le théorème 16, nous obtenons $pw(G) = tw_0(G) \leq 2^{\tau(G)}\ tw(G)$. □

Le prochain théorème prouve que la borne obtenue par le théorème 16 est presque optimale.

Théorème 17 *Pour tout $q \geq 1$ et $k \geq 1$, il existe un graphe $G_{k,q}$ tel que $tw_q(G_{k,q}) = k$ et $tw_{q-1}(G_{k,q}) \geq 2\ tw_q(G_{k,q}) - 1$.*

Preuve. Pour tout $q \geq 1$, nous définissons l'arbre enraciné T_q récursivement. T_1 est isomorphe au graphe biparti complet $K_{1,3}$ et la racine de T_1 est son sommet de degré trois. Pour tout $q \geq 2$, l'arbre T_q est formé de trois copies disjointes de T_{q-1} en connectant les racines de chacune de ces copies à la nouvelle racine r. Donc, $T_q \setminus \{r\}$ consiste en trois copies de T_{q-1}. Nous définissons le graphe $G_{k,q}$ comme étant le graphe obtenu à partir de T_q en remplaçant chaque sommet $u \in V(T_q)$ par une clique K_u de k sommets, et en remplaçant chaque arête $\{u, v\}$ par un couplage parfait entre les deux cliques K_u et K_v.

Prouvons d'abord que $tw_q(G_{k,q}) = k$. D'après le théorème 12, pour prouver que $tw_q(G_{k,q}) \leq k$ il suffit de prouver que $s_q(G_{k,q}) \leq k+1$. Nous définissons une stratégie de capture q-limitée, utilisant $k+1$ chercheurs. Initialement, k chercheurs sont placés sur la clique K_r correspondant à la racine. Nous posons alors une question à l'oracle pour la première fois. Soient u, v et w les enfants de r. Après la première question, une seule des cliques K_u, K_v et K_w est contaminée. Disons, sans perte de généralité, qu'il s'agit de K_v. En utilisant les $k+1$ chercheurs, il est possible de les déplacer un par un de K_r à K_v. Le sommet v correspondant à la clique K_v est la racine d'un arbre T_{q-1} obtenu à partir de T_q en supprimant r. A présent, la situation se répète, puisque la partie contaminée du graphe, plus K_v forme $G_{k,q-1}$. Une simple induction sur q permet de conclure que $s_q(G_{k,q}) \leq k+1$. De plus, pour tout $q \geq 1$, $G_{k,q}$ admet la clique à $k+1$ sommets comme mineur. Donc $tw_q(G_{k,q}) = k$.

La preuve de $tw_{q-1}(G_{k,q}) \geq 2\ tw_q(G_{k,q}) - 1$ s'effectue par induction sur q.

Pour $q = 1$, le graphe $G_{k,1}$ est composé d'une clique centrale K_c connectée par des couplages parfaits à trois cliques externes K_u, K_v et K_w. Nous affirmons que $tw_0(G_{k,1}) \geq 2k-1$. Soit $(P, \mathcal{X} = \{X_i, i \in V(P)\})$ une décomposition linéaire de $G_{k,1}$. Il est bien connu, et découle des propriétés **C2** et **C3** des décompositions arborescents, que toute clique de $G_{k,1}$ doit être incluse dans l'un des sacs X_i de $\mathcal{X}$. Sans perte de généralité, supposons que $K_v \subseteq X_{i_1}$, $K_u \subseteq X_{i_2}$ et $K_w \subseteq X_{i_3}$, avec i_2 sur le chemin entre i_1 et i_3. Si $i_1 = i_2$ ou $i_2 = i_3$, alors X_{i_2} contient au moins $2k$ sommets, et $tw_0(G_{k,1}) \geq 2k-1$. Donc suppsons, que i_1, i_2 et i_3 sont deux à deux disjoints. Dans ce cas, en utilisant les propriétés des décompositions arborescentes, il est facile de prouver que X_{i_2} contient les sommets de la clique K_c. Donc $K_c \cup K_u \subseteq X_{i_2}$ dont la taille est au moins $2k$.

Soit $p \geq 1$. Supposons que pour tout $q \leq p$, $tw_{q-1}(G_{k,q}) \geq 2k-1$. Une fois encore, nous allons utiliser l'interprétation de la décomposition q-branchée en termes de stratégie de capture non-déterministe. Soit Π une stratégie de capture non-déterministe monotone pour $G_{k,q}$, impliquant $2k-1$ chercheurs, et posant $q = p+1$ questions. Nous prouvons que Π ne peut pas capturer le fugitif dans $G_{k,q}$.

Considérons la situation qui se produit à l'étape précédent la première question. Soit X l'ensemble des sommets propres de $G_{k,q}$ à cette étape, et S l'ensemble des sommets occupés par les chercheurs. Quelle peut être la taille de X ? Soit $l \geq 0$, et soit $K_{u_1}, K_{u_2}, \cdots, K_{u_l}$ les cliques maximales de $G_{k,q}$ contenues dans X. Nous avons déjà prouvé que, sans poser de question, $2k-1$ chercheurs ne peuvent nettoyer $G_{k,1}$. Donc chaque composante connexe du sous-arbre de T_q induit par les sommets $u_1, u_2, \cdots, u_l$ est un chemin. De plus, d'après la monotonie de la stratégie considérée, chaque chemin liant un sommet contaminé à un sommet de X doit contenir un sommet occupé par un chercheur. Puisque $|S| \leq 2k-1$, nous pouvons en déduire que $l \leq 3$. De plus, si $l = 3$, alors l'un des sommets u_1, u_2, u_3 est une feuille de T_q et un autre de ces sommets est adjacent à cette feuille.

Nous considérons les trois cas qui peuvent se produire à l'étape à laquelle la première question est posée.

- *Cas 1. $l = 0$.* Dans ce cas, les seuls sommets propres avant que la première question ne soit posée sont les sommets de S. Tout séparateur minimal de $G_{k,q}$ est de taille au moins k. Dons, si S contient un séparateur minimal P, alors toute composante connexe de $G_{k,q} \setminus P$ contient au plus $k-1$ chercheurs. Tout séparateur minimal de $G_{k,q}$ est contenu dans l'union de deux cliques adjacentes, ce qui implique que, après la question, il existe une composante C de $G_{k,q} \setminus P$ qui contient $G_{k,q-1}$ comme sous-graphe. De plus, les seuls sommets propres de C sont ceux occupés par des chercheurs. D'après l'hypothèse d'induction, $s_{q-2}(G_{k,q-1}) \geq 2k$, la stratégie utilisant $2k-1$ chercheurs ne peut donc capturer le fugitif dans C en posant au plus $q-2$ questions.
- *Cas 2. $l = 1$ et u_1 est une feuille de T_q.* Soit $K_{u'}$ la clique adjacente à K_{u_1}. La monotonie de la stratégie implique que, dans ce cas, $S \cap (K_{u'} \cup K_{u_1})| \geq k$. Donc le graphe $G_{k,q} \setminus (K_{u'} \cup K_{u_1})$ contient au plus $k-1$ chercheurs. Après la question, les seuls sommets propres de $C = G_{k,q} \setminus (K_{u'} \cup K_{u_1})$ sont les sommets occupés par des chercheurs. Donc, C contient $G_{k,q-1}$ comme sous-graphe, et, d'après l'hypothèse d'induction, $s_{q-2}(C) \geq 2k$.
- *Cas 3. Un des sommets u_i n'est pas une feuille de T_q.* Alors, par monotonie de la stratégie, les sommets d'une des cliques, disons K_{u_1}, doivent tous être occupés par des chercheurs. Encore, cela implique que, lorsque la première question est posée, il existe une composante C de $G_{k,q} \setminus K_{u_r}$ (rappelons que r est la racine de T_q) telle que, C contient $G_{k,q-1}$ comme sous-graphe, $V(C) \cap \bigcup_{1 \leq i \leq l} K_{u_i} = \emptyset$, et il y a au plus $k-1$ chercheurs dans C. Donc, $s_{q-2}(C) \geq 2k$.

Dans tous les cas, nous avons démontré que, après la première question, il existe toujours un sous-graphe C de $G_{k,q}$, tel que $2k-1$ chercheurs posant $q-2$ questions ne peuvent nettoyer C. Donc Π ne peut capturer le fugitif.

D'après le théorème 12, $tw_{q-1}(G_{k,q}) \geq 2\ tw_q(G_{k,q}) - 1$. □

2.5 Perspectives

Dans le cadre de l'étude des stratégies de capture, la classe des arbres a toujours fait l'objet d'une attention particulière. Cela est dû, en particulier, au fait que dans les variantes étudiées (jeux de capture invisible et connexe), il existe des algorithmes en temps linéaires pour calculer des stratégies de capture optimales dans la classe des arbres. En ce qui concerne les stratégies de capture non-déterministe, la question reste ouverte : existe-t-il un algorithme en temps polynomial, voire linéaire, pour calculer une stratégie q-limitée optimale, dans tout arbre T, et pour tout $q \geq 0$? Par ailleurs, la classe des graphes d'indice d'échappement q-limité borné étant close pour la relation de minoration, les résultats de Robertson et Seymour permettent d'affirmer qu'il existe un algorithme FPT (pour *Fixed Parameter Tractable* en anglais) pour calculer l'indice d'échappement q-limité dans cette classe de graphe. Cependant, peut-on concevoir explicitement un algorithme en temps polynomial pour calculer une stratégie q-limitée optimale, dans la classe des graphes d'indice d'échappement q-limité borné par k fixé, et pour tout $q \geq 0$?

Deuxième partie

Stratégies de capture connexe

Contenu de la partie

Cette partie est consacrée à l'étude des stratégies de capture connexes. Les stratégies étudiées dans cette partie doivent donc satisfaire la contrainte de connexité selon laquelle, à chaque étape de la stratégie, la partie propre du graphe doit être connexe. Nous considérons les stratégies-sommet.

Dans le chapitre 3, nous développons tout d'abord la notion de décomposition arborescente connexe d'un graphe. Nous donnons une nouvelle preuve de l'égalité entre largeur arborescente et largeur arborescente connexe. Cette preuve a l'avantage d'être constructive. En effet, nous proposons un algorithme en temps $O(n^4)$ qui transforme toute décomposition arborescente d'un graphe G en une décomposition arborescente connexe de G sans augmenter sa largeur. La suite du chapitre est consacrée à l'utilisation des décompositions arborescentes connexes pour la conception de stratégies de capture connexe dans les graphes. Les deux algorithmes que nous proposons dans ce chapitre nous permettent de donner de nouvelles bornes supérieures pour le rapport entre l'indice d'échappement connexe d'un graphe et son indice d'échappement. Nous prouvons notamment que ce rapport est au plus $\log n$ pour tout graphe de n sommets, et qu'il est borné supérieurement par deux fois la largeur arborescente dans le cas des graphes cordaux.

Le chapitre 4 présente l'étude des stratégies de capture connexe dans le cas où le fugitif est visible. Nous étudions tout d'abord le rapport entre l'indice d'échappement monotone connexe visible d'un graphe et son indice d'échappement visible. Nous prouvons qu'il existe une famille infinie de graphes pour laquelle ce rapport est au moins $\Omega(\log n)$. Dans la seconde section de ce chapitre, nous prouvons que les stratégies de capture connexes visibles ne sont pas forcément monotones.

Chapitre 3

Le coût de la connexité

Ce chapitre est consacré à l'étude mentionnée dans la question 2 de la section 1.3.4. Nous nous attachons ainsi à déterminer une borne supérieure pour le rapport entre indice d'échappement connexe et indice d'échappement dans un graphe. Tous les graphes considérés dans ce chapitre sont supposés connexes.

3.1 Introduction

Les chapitres 1 et 2 ont déjà mentionné l'étroite relation qui existe entre stratégies de capture et décomposition de graphes. Il est donc légitime de chercher à tirer partie de ces décompositions dans le cadre des stratégies de capture connexes.

Dans [FFT04], Fomin *et al.* donnent une preuve constructive du résultat de Seymour et Thomas [ST94] selon lequel tout graphe 2-connexe G admet une décomposition en branche connexe de largeur $bw(G)$. Fomin *et al.* proposent alors un algorithme qui transforme toute décomposition en branche d'un graphe 2-connexe G en une décomposition en branche connexe, sans en augmenter la largeur. Cette décomposition connexe de G est alors utilisée pour déterminer une stratégie de capture connexe monotone pour G en partant de n'importe quelle arête de G. Informellement, étant donnée une arête e de G, la décomposition en branches induit un ordre sur les arêtes de G commençant de e. La stratégie consiste à nettoyer les arêtes dans cet ordre. Fomin *et al.* prouvent qu'une telle stratégie implique au plus $es(G)(2 + \log m)$ chercheurs, ce dont découlait alors la meilleure borne connue pour le rapport entre indice d'échappement connexe et indice d'échappement dans le cas d'un graphe arbitraire (cf. question 2 de la section 1.3.4). Etant donnée l'étroite relation entre décompositions arborescentes et stratégies de capture, il est naturel d'envisager l'utilisation des décompositions arborescentes dans le but de résoudre ce même problème.

Avant de continuer, notons que la terminologie définie dans la section 1.2.4 est légèrement modifiée dans ce chapitre pour une meilleure lisibilité. En effet, dans ce chapitre, nous considérons uniquement des stratégies de capture-sommet invisible que nous désignerons simplement *stratégies de capture*. Etant donné un graphe G, *l'indice d'échappement* de G, c'est-à-dire l'indice d'échappement-sommet de G, sera noté $s(G)$ (au lieu de $ns(G)$).

Un "m" (resp., un "c") accolé à ce symbole signifiera que les stratégies considérées sont monotones (resp., connexes).

L'égalité entre largeur arborescente et largeur arborescente connexe est valide dans le cas d'un graphe quelconque, et découle d'un résultat de Parra et Scheffler [PS97]. Dans la deuxième section de ce chapitre, nous donnons une preuve constructive de ce résultat. Plus précisément, nous proposons un algorithme en temps polynomial qui, étant donnée une décomposition arborescente d'un graphe G, calcule une décomposition arborescente connexe de G, sans augmenter la largeur de la décomposition. L'algorithme que nous proposons procède en deux phases. La première se propage des feuilles à la racine pour rendre la décomposition sous-connexe (la définition formelle de la sous-connexité est donnée dans la section 3.2.1). La seconde phase se propage en sens inverse, c'est-à-dire, de la racine aux feuilles, et finit de rendre la décomposition connexe.

La troisième section de ce chapitre est consacrée à l'utilisation des décompositions arborescentes connexe pour la conception de stratégies de capture connexes dans les graphes. Nous proposons tout d'abord un algorithme en temps polynomial qui, étant donnée une décomposition arborescente connexe d'un graphe G, détermine une stratégie de capture monotone et connexe pour G, partant de n'importe quel de ses sommets. Cette stratégie implique au plus $s(G) \log n$ chercheurs. Ce résultat améliore ainsi la meilleure borne supérieure connue pour le rapport entre indice d'échappement connexe et indice d'échappement dans le cas d'un graphe arbitraire. Combiné à l'algorithme d'approximation proposé par Feige *et al.* [FHL05a], nous obtenons un algorithme d'approximation en temps polynomial pour calculer l'indice d'échappement connexe d'un graphe arbitraire G avec un rapport d'approximation $O(\log n \sqrt{\log tw(G)})$.

La dernière section de ce chapitre présente un second algorithme polynomial qui, étant donnée une décomposition arborescente connexe d'un graphe G, détermine une stratégie de capture connexe monotone pour G. La stratégie calculée implique un nombre de chercheurs qui est fonction de la cordalité du graphe, de sa largeur arborescente et de l'indice d'échappement de l'arbre de décomposition utilisé. Ce résultat nous permet de proposer un algorithme en temps polynomial permettant de calculer une stratégie de capture connexe monotone pour tout graphe cordal G, qui implique au plus $2\ s(G)\ (tw(G)+1)$ chercheurs. Ceci donne une nouvelle borne supérieure pour le rapport entre indice d'échappement connexe et indice d'échappement dans le cas de la classe des graphes cordaux.

Les résultats des sections 3.2 et 3.3 ont été réalisés en collaboration avec Pierre Fraigniaud. Ils ont donné lieu à une publication dans la conférence LATIN 2006 [FN06a]. Les résultats de la section 3.4 ont donné lieu à une publication dans le journal Discrete Applied Maths. [Nis07].

3.2 Décompositions arborescentes connexes

Cette section est consacrée à l'égalité entre largeur arborescente et largeur arborescente connexe. La preuve de cette égalité fait partie du folklore, elle est généralement admise par la communauté des chercheurs qui travaillent sur les décompositions arborescentes. Dans un but d'exhaustivité, nous donnons tout d'abord la preuve de cette égalité. Celle-ci peut être obtenue en combinant des résultats de Parra et Scheffler [PS97] et de Golumbic [Gol80].

Lemme 7 *Pour tout graphe connexe G, $ctw(G) = tw(G)$.*

Preuve. Soit G un graphe connexe, et $S \subset V(G)$. Soient $a, b \in V(G)$. S est un a, b-séparateur si a et b se trouvent dans différentes composantes connexes de $G \setminus S$. Une composante connexe $C \setminus S$ est dite *pleine* si tout sommet de S est adjacent à un sommet de C. S est un *séparateur minimal* de G si il existe $a, b \in V(G)$ tels que S est un a, b-séparateur et aucun sous-ensemble strict de S ne sépare a et b. Il est bien connu que, si S est un a, b-séparateur minimal, la composante connexe contenant a (resp., b) est pleine. Soient S, T deux séparateurs minimaux de G. S *croise* T si il existe deux composantes C, D de $G \setminus T$ qui sont intersectées par S. Si S et T ne se croisent pas, ils sont dit *parallèles*. Une *triangulation minimale* d'un graphe $G = (V, E)$ est un sur-graphe cordal $H = (V, F)$ de G tel que $E \subseteq F$ et pour tout $E \subseteq F' \subset F$, (V, F') n'est pas cordal. Soit H une triangulation minimale de G. Soit Δ_H l'ensemble des séparateurs minimaux de H. Parra et Scheffler [PS97] prouvent que Δ_H est un ensemble maximal par inclusion de séparateurs minimaux de G deux-à-deux parallèles. De plus, pour tout $S \in \Delta_H$, les composantes connexes pleines de $G \setminus S$ et de $H \setminus S$ sont les mêmes (voir [Par98] pour plus de détails). D'autre part, Golumbic [Gol80] prouve que tout arbre de cliques d'une triangulation H d'un graphe G est une décomposition arborescente de G. Soit $(T, \mathcal{X})$ l'arbre de clique d'une telle triangulation H, nous prouvons qu'il s'agit d'une décomposition arborescente connexe.

Soit $e = \{u, v\} \in E(T)$ et soient $T_1^{(e)}, T_2^{(e)}, G_1^{(e)}$ et $G_2^{(e)}$ définis comme à la section 1.2.5. Soit $i \in \{1, 2\}$. Prouvons que $G_i^{(e)}$ est connexe. Soit $S = X_u \cap X_v$. Puisque H est un graphe cordal, $S \in \Delta_H$. Puisque S est un séparateur minimal de H, et H est cordal, il existe G_1 une composante pleine de $H \setminus S$ dans $G_i^{(e)}$. Ainsi, d'après le résultat de Parra et Scheffler, S est un séparateur minimal de G, et G_1 est une composante pleine de $G \setminus S$ dans $G_i^{(e)}$. Soient $a, b \in V(G_i^{(e)})$. Puisque G est connexe, et que S est une séparateur, il existe $x \in S$ et un chemin de a à x dans $V(G_i^{(e)})$. De même, il existe $y \in S$ et un chemin entre b et y dans $V(G_i^{(e)})$. Puisque G_1 est une composante pleine de $G \setminus S$, il existe un chemin de x à y dans $V(G_1) \cup S$. Par conséquent, puisque $V(G_1) \subseteq V(G_i^{(e)})$, il existe un chemin entre a et b dans $V(G_i^{(e)})$. Donc, $G_i^{(e)}$ est connexe. □

Le reste de cette section est dévolue à la présentation de l'algorithme `rend-connexe` qui, étant donnée une décomposition arborescente d'un graphe G, calcule une décomposition arborescente connexe de G, sans augmenter la largeur de la décomposition*.

---

*Merci à Dieter Kratsch pour avoir porté à notre connaissance les travaux [BHT01, BBH+06] qui traitent également de ce sujet. Le principe de ces travaux consiste à compléter tous les sacs d'une décomposition arborescente d'un graphe G en des cliques. Soit H le graphe cordal obtenu. Il s'agit ensuite de construire H' une triangulation minimale de G qui soit sous graphe de H, et enfin de déterminer un arbre de cliques de H'.

3.2.1 Décompositions arborescentes sous-connexes

Nous définissons premièrement une version plus souple de la propriété de connexité des décompositions arborescentes. Informellement, étant donnée une décomposition arborescente $(T, \mathcal{X})$ d'un graphe G et une racine u de T, une arête $e \in E(T)$ sera dite sous-connexe, si le sous-graphe de G induit par les sommets contenus dans le sous-arbre de $T \setminus \{e\}$ ne contenant pas u est connexe. Nous présentons ensuite la sous-procédure `éclate` qui sera utilisée par notre algorithme `rend-connexe`. Cette sous-procédure permet de rendre les arêtes de la décomposition sous-connexes.

Etant donné une décomposition arborescente $(T, \mathcal{X})$ d'un graphe G, et $u \in V(T)$, nous notons $(T, \mathcal{X}, u)$ la décomposition arborescente $(T, \mathcal{X})$ enracinée en u. Pour $v \in V(T)$, nous notons T_v le sous-arbre de $(T, \mathcal{X}, u)$ enraciné en v. Le sous-graphe de G induit par les sommets contenus dans les sacs de T_v est noté $G[T_v]$.

Définition 20 *Une décomposition arborescente enracinée $(T, \mathcal{X}, u)$ est* sous-connexe *en $v \in V(T)$ si, pour tout $w \in V(T_v)$, $G[T_w]$ est un graphe connexe. $(T, \mathcal{X}, u)$ est* sous-connexe *si elle est sous-connexe en sa racine u.*

Notons que la sous-connectivité de $(T, \mathcal{X}, u)$ en v peut être définie alternativement par : (1) $G[T_v]$ est connexe, et (2) pour tout fils w de v dans T_v, $(T, \mathcal{X}, u)$ est sous-connexe en w.

Nous décrivons maintenant une procédure élémentaire, appelée `éclate`, qui, appliquée itérativement, transforme une décomposition arborescente en une décomposition arborescente sous-connexe. La procédure `éclate` est représentée sur la figure 3.1. Cette procédure prend en entrée (1) une décomposition arborescente enracinée $(T, \mathcal{X}, u)$ d'un graphe connexe G, et (2) un sommet $v \in V(T_u)$, $v \neq u$, tel que, pour tout fils w de v dans T_u, $(T, \mathcal{X}, u)$ est sous-connexe en w. La procédure `éclate` renvoie une décomposition arborescente enracinée $(T', \mathcal{X}', u)$ de G qui est égale à $(T, \mathcal{X}, u)$, sauf au niveau du sommet v. Informellement, le sommet v de $(T, \mathcal{X}, u)$ est remplacé par plusieurs sommets $v_1, \ldots, v_\ell$ dans $(T', \mathcal{X}', u)$. Les sommets v_i ont le père de v dans $(T, \mathcal{X}, u)$ pour père dans $(T', \mathcal{X}', u)$. Les fils de v dans $(T, \mathcal{X}, u)$ sont répartis parmi les sommets v_i. La procédure `éclate` vérifie que $(T', \mathcal{X}', u)$ est sous-connexe en chaque sommet v_i, $i = 1, \ldots, \ell$. A présent, nous décrivons formellement la procédure.

Utilisant les mêmes notations que dans la description de la procédure `éclate` sur la figure 3.1, nous avons :

Lemme 8 *La procédure* `éclate` *appliquée au sommet v renvoie une décomposition arborescente enracinée $(T', \mathcal{X}', u)$ de G, de largeur $\leq k$. La décomposition arborescente $(T', \mathcal{X}', u)$ diffère de $(T, \mathcal{X}, u)$ uniquement en v, qui est remplacé par les sommets $v_1, \ldots, v_\ell$. $(T', \mathcal{X}', u)$ est sous-connexe en v_i, $i = 1, \ldots, \ell$. De plus, pour tout sommet z tel que $(T, \mathcal{X}, u)$ est sous-connexe en z, $(T', \mathcal{X}', u)$ reste sous-connexe en z.*

Preuve. Le lemme est évidemment vrai dans le cas 1. Donc, nous concentrons notre attention sur le cas 2. Dans ce cas, $\ell = t$. Bien entendu, puisque la modification de T

Procédure `éclate` : Soit $(T, \mathcal{X}, u)$ une décomposition arborescente enracinée d'un graphe connexe G, de largeur k. Soit $v \in V(T_u)$, $v \neq u$, tel que :
- $(T, \mathcal{X}, u)$ n'est pas sous-connexe en v ;
- pour tout fils w de v, $(T, \mathcal{X}, u)$ est sous-connexe en w.

Puisque G est connexe, et puisque $(T, \mathcal{X}, u)$ n'est pas sous-connexe en v mais est sous-connexe en chaque fils de v, le sous-graphe de G induit par les sommets contenus dans le sac X_v n'est pas connexe. Soit $Y_i,\ i = 1, \ldots, r$ la décomposition de X_v en composantes connexes (c'est-à-dire, les $G[Y_i]$ sont les composantes connexes de $G[X_v]$). Soit v' le père de v dans T_u. La procédure `éclate` opère comme suit :

Cas 1 : v est une feuille de T_u. `éclate` remplace v par r sommets $v_1, \ldots, v_r$, tous connectés à v', et chaque sac correspondant X_{v_i} est l'ensemble Y_i.

Cas 2 : v possède s fils $w_1, \ldots, w_s,\ s \geq 1$. Soient $Z_i,\ i = 1, \ldots, t$, les composantes connexes de $G[T_v]$. Comme nous le prouverons, il existe une partition $\{I_i, i = 1, \ldots, t\}$ de $\{1, \ldots, r\}$, et une partition $\{J_i, i = 1, \ldots, t\}$ de $\{1, \ldots, s\}$ telles que, pour tout $i = 1, \ldots, t$:

$$Z_i = \Big(\cup_{j \in I_i} Y_j \Big) \cup \Big(\cup_{j \in J_i} G[T_{w_j}] \Big). \tag{3.1}$$

La procédure `éclate` remplace v par t sommets $v_1, \ldots, v_t$, tous reliés à v' (cf. Fig. 3.2). Pour tout $i = 1, \ldots, t$, le sommet v_i est le père de w_j pour tout $j \in J_i$, et le sac X_{v_i} correspondant à v_i est l'ensemble $\cup_{j \in I_i} Y_j$.

FIG. 3.1 – Procédure `éclate`.

se produit uniquement au niveau du sommet v, pour tout sommet z tel que $(T, \mathcal{X}, u)$ est sous-connexe en z, $(T', \mathcal{X}', u)$ reste sous-connexe en z. Donc, concentrons-nous sur la transformation de v en $v_1, \ldots, v_t$. Premièrement, montrons que l'équation (3.1) est valide. Soit H le graphe biparti dont une partition consiste en r sommets $Y_1, \ldots, Y_r$, et l'autre partition consiste en s sommets $w_1, \ldots, w_s$ (cf. Fig. 3.2). Il y a une arête entre Y_i et w_j si et seulement si un sommet x de G appartient à $Y_i \cap X_{w_j}$. Par construction, il existe une correspondance univoque entre les composantes connexes de $G[T_v]$ et les composantes connexes de H. L'équation (3.1) découle de cette correspondance. De la construction des sommets v_i, basée sur l'équation (3.1), $(T', \mathcal{X}', u)$ est sous-connexe en v_i, $i = 1, \ldots, t$. Donc, il reste à prouver que (T', X') est une décomposition arborescente de G, de largeur $\leq k$.

Puisque $X'_{v_i} = \cup_{j \in I_i} Y_j$, et que les ensembles I_i forment une partition de $\{1, \ldots, r\}$, nous obtenons que $\cup_{i=1,\ldots,r} X'_{v_i} = \cup_{j=1,\ldots,r} Y_j = X_v$.

Donc chaque sommet de G apparaît dans au moins un sac, c'est-à-dire, **C1** est valide. Chaque arête de G apparaît dans au moins un sac aussi, car les ensembles Y_i sont les composantes connexes de X_v, et donc, il n'existe pas d'arêtes entre les sommets qui appartiennent à deux ensembles Y_i différents. c'est-à-dire, **C2** est valide.

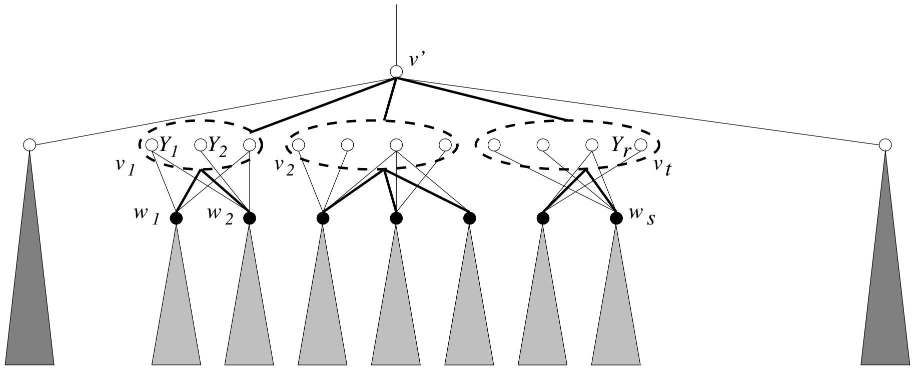

FIG. 3.2 – La procédure `éclate` remplace le sommet v par t sommets $v_1, \ldots, v_t$. Les ensembles Y_i forment une partition de X_v en composantes connexes. Les sommets w_i sont les fils de v. Le sommet v' est le père de v.

Sans surprise, la condition **C3** des décompositions arborescentes est la plus difficile à vérifier. Soient x, y, z trois sommets distincts de T' avec y sur le chemin entre x et z. Montrons que $X'_x \cap X'_z \subseteq X'_y$. Bien sûr, l'assertion est vraie si $v_i \notin \{x, y, z\}$ pour tout $i = 1, \ldots, r$, car, dans ce cas, les sacs de T' considérés sont exactement ceux de T. L'assertion est également valide si le chemin P de x à z contient deux sommets v_i, car, dans ce cas, $X'_x \cap X'_z = \emptyset$. Donc, dans la suite, nous considérons le cas où le chemin P contient exactement un sommet v_i. Il y a deux sous-cas, selon que le sommet de P qui appartient à $\{v_1, \ldots, v_t\}$ est y, ou une des deux extrémités x ou z.

Supposons que $x = v_i$ avec $i \in \{1, \ldots, r\}$, et que $v_j \notin \{y, z\}$ pour tout $j \neq i$. Alors, $X'_x \cap X'_z \subseteq X_v \cap X'_z = X_v \cap X_z \subseteq X_y = X'_y$, et donc la condition **C3** est vraie.

Supposons que $y = v_i$ avec $i \in \{1, \ldots, r\}$, et que $v_j \notin \{x, z\}$ pour tout $j \neq i$. Supposons, sans perte de généralité, que $x \in V(T'_{v_i})$. Le sommet z qui appartient soit à $V(T'_{v_i})$, soit au sous-arbre de T contenant v', obtenu après la suppression de l'arête $\{v, v'\}$ de T. Dans les deux cas, $X'_x \cap X'_z = X_x \cap X_z \subseteq X_v$. De plus, par construction du graphe biparti H (cf. Fig. 3.2), $X'_x \cap X_v \subseteq \cup_{j \in I_i} Y_j$. Par conséquent, $X'_x \cap X'_z \subseteq \cup_{j \in I_i} Y_j = X'_{v_i}$, et donc la condition **C3** est satisfaite.

Pour conclure la démonstration, remarquons que l'application de la procédure `éclate` peut seulement diminuer la largeur de la décomposition arborescente puisqu'un sac est divisé en plusieurs sacs plus petits. Ainsi, la largeur de $(T', \mathcal{X}', u)$ est $\leq k$. □

3.2.2 Construction de décompositions connexes

Nous présentons maintenant l'algorithme `rend-connexe` décrit en détails sur la figure 3.3. La procédure `rend-connexe` prend en entrée une décomposition arborescente de largeur k d'un graphe G, et calcule une décomposition arborescente connexe de largeur $\leq k$ de G. La décomposition en entrée est tout d'abord enracinée en un sommet arbitraire u. Puis l'algorithme procède en deux phases. Au cours de la première phase, la sous-procédure `éclate` est appliquée itérativement des feuilles jusqu'à la racine de manière à rendre la décomposition sous-connexe. Dans la seconde phase, la procédure `éclate` est appliquée jusqu'à ce que toutes les arêtes soient connexes, de la manière suivante. La procédure `rend-connexe` choisit une arête e non connexe telle qu'il n'y a pas d'arête non connexe entre e et la racine. La décomposition est alors ré-enracinée en l'extrémité "fille" v de e et la procédure `éclate` est appliquée à v.

Nous énonçons et prouvons à présent le théorème principal de cette section.

Théorème 18 *Il existe un algorithme en temps $O(nk^3)$ qui, étant donné une décomposition arborescente de n sommets d'un graphe connexe G, de largeur k, renvoie une décomposition arborescente connexe de $O(nk)$ sommets de G, de largeur $\leq k$.*

Corollaire 2 *Pour tout graphe connexe G, $ctw(G) = tw(G)$.*

Preuve du Théorème 18. Nous prouvons que l'algorithme `rend-connexe` décrit sur la figure 3.3 satisfait l'énoncé du théorème. Soit k la largeur de la décomposition arborescente $(T, \mathcal{X})$ en entrée. Prouvons tout d'abord :

Assertion 7 *A la fin de la phase 1, la décomposition arborescente $(T, \mathcal{X}, u)$ est sous-connexe, et sa largeur est $\leq k$.*

A chaque application de la boucle *tant que* de la Phase 1, le sommet $v \in W$ considéré satisfait la condition d'application de la procédure `éclate`. La boucle *tant que* termine après que $W = \emptyset$. Par conséquent, d'après le lemme 8, pour chaque sommet $v \neq u$, $(T, \mathcal{X}, u)$ est sous-connexe en v. Il reste à vérifier que $(T, \mathcal{X}, u)$ est sous-connexe en u. Les sacs dans T_u contiennent tous les sommets de G. Par conséquent, puisque G est connexe, $G[T_u]$ est également connexe, et donc $(T, \mathcal{X}, u)$ est sous-connexe en u, c'est-à-dire, $(T, \mathcal{X}, u)$ est sous-connexe. D'après le lemme 8, l'application de la procédure `éclate` n'augmente pas la largeur de la décomposition arborescente courante, par conséquent la largeur de la décomposition arborescente résultante de la Phase 1 est de largeur $\leq k$.

Le but de la phase 2 est de transformer la décomposition arborescente sous-connexe courante $(T, \mathcal{X}, u)$ en une décomposition arborescente connexe (éventuellement enracinée en un autre sommet r).

Assertion 8 *A la fin de la phase 2, la décomposition arborescente $(T, \mathcal{X}, r)$ est connexe, et sa largeur $\leq k$.*

Pour prouver l'assertion, nous prouverons les invariants suivants, satisfaits après chaque application de la boucle *tant que* au cours de la phase 2 :
- **I1 :** $(T, \mathcal{X}, r)$ est sous-connexe ;
- **I2 :** C est l'ensemble des arêtes e de T correspondant à des e-coupes non-connexes.

Ces deux propriétés sont satisfaites avant la première application de la boucle *tant que*. Soit $e = \{v, w\}$ définie comme dans l'algorithme `rend-connexe`. Si $v \neq r$, soit v' le père de v dans T_r, éventuellement $r = v'$. (Si $v = r$, alors il n'y a simplement pas besoin de définir le sommet v'.) Soit S le sous-arbre de T contenant v' obtenu après la suppression de $\{v, v'\}$ de T (cf. Fig. 3.4). Du choix de e, $\{v, v'\} \notin C$, et donc $G[S]$ est connexe. Soient

Entrée : Une décomposition arborescente $(T, \mathcal{X})$ d'un graphe connexe G.
début
  Choisir un sommet u de T, et enraciner T en u ;
  /* Phase 1 */
  $W \leftarrow \{v \in V(T_u) \mid v \neq u$ and $(T, \mathcal{X}, u)$ n'est pas sous-connexe en $v\}$;
  tant que $W \neq \emptyset$ **faire**
    Soit $v \in W$ tel que, pour tout fils w de v, $w \notin W$;
    $(T, \mathcal{X}, u) \leftarrow$ `éclate`$(G, (T, \mathcal{X}, u), v)$;
    $W \leftarrow W \setminus \{v\}$;
  fin tant que
  /* Phase 2 */
  $r \leftarrow u$; /* r est la racine de T */
  $C \leftarrow \{e \in E(T) \mid$ la e-coupe de T n'est pas connexe$\}$
  tant que $C \neq \emptyset$ **faire**
    Soit $e = \{v, w\} \in C$ avec v le père de w, et
      aucune arête sur le chemin de r à v est dans C ;
    $r \leftarrow w$; /* la racine est modifiée */
    $(T, \mathcal{X}, r) \leftarrow$ `éclate`$(G, (T, \mathcal{X}, r), v)$;
    $C \leftarrow C \setminus \{e\}$;
    pour tout fils $w_j \neq w$ de v tel que $\{v, w_j\} \in C$ **faire**
      Soit v_i le nouveau père de w_j après application de `éclate` ;
      Remplacer $\{v, w_j\}$ par $\{v_i, w_j\}$ dans C ;
    fin pour
  fin tant que
  renvoie $(T, \mathcal{X}, r)$;
fin.

FIG. 3.3 – Algorithme `rend-connexe`.

$w_1, \ldots, w_s$ les fils de v dans T_r, différents de w, et soit S_i le sous-arbre de T_r enraciné en w_i, $i = 1, \ldots, s$. Puisque $(T, \mathcal{X}, r)$ est sous-connexe, $G[S_i]$ est connexe pour tout i. Finalement, soit R le sous-arbre de T_r enraciné en w. Encore, puisque $(T, \mathcal{X}, r)$ est sous-connexe, $G[R]$ est connexe. Avant l'application de la procédure `éclate` en v, $(T, \mathcal{X})$ est ré-enracinée en w. Puisque $\{v, w\} \in C$, $(T, \mathcal{X}, w)$ n'est pas sous-connexe en v. Cependant, $(T, \mathcal{X}, w)$ est sous-connexe en chacun des fils $v', w_1, \ldots, w_s$ de v dans $(T, \mathcal{X}, w)$. Par conséquent, nous sommes dans les conditions d'application de la procédure `éclate`.

L'invariant **I1** est satisfait. En effet, après application de la procédure `éclate` en v dans $(T, \mathcal{X}, w)$, $(T, \mathcal{X}, w)$ devient sous-connexe en ses nouveaux sommets $v_1, \ldots, v_\ell$. Les seuls autres sommets en lesquels la sous-connectivité peut être mise en doute sont les sommets le long du chemin entre r et v'. Puisque toutes les arêtes de ce chemin correspondent à

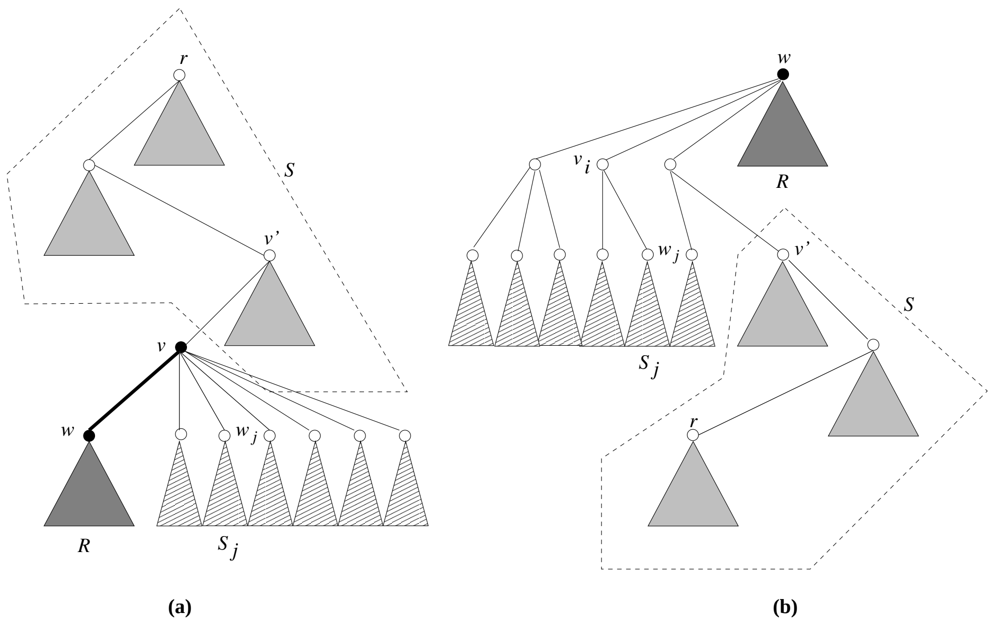

FIG. 3.4 – Dans la seconde phase de l'algorithme `rend-connexe`, la décomposition arborescente est ré-enracinée en w avant l'application de la procédure `éclate` au sommet v. (a) représente l'arbre enraciné en r, et (b) représente le même arbre, ré-enraciné en w, après application de la procédure `éclate` en v.

une coupe connexe (d'après le choix de $\{v, w\}$), la sous-connectivité reste satisfaite le long de ce chemin. ($(T, \mathcal{X}, w)$ est sous-connexe en w car G est connexe.)

Maintenant nous prouvons que l'invariant **I2** est aussi satisfait, c'est-à-dire que toutes les coupes non-connexes dans $(T, \mathcal{X}, w)$ sont dans C. Considérons d'abord l''arête $f = \{v_i, v'\}$ définie dans la procédure `éclate` dans $(T, \mathcal{X},)$. D'un côté de f, il y a S, et de l'autre côté il y a $T \setminus S$. $G[S]$ et $G[T \setminus S]$ sont connexes car ils correspondent à la coupe $\{v, v'\}$, qui est connexe.

Ensuite, nous considérons une arête $f = \{v_i, w\}$ introduite par la procédure `éclate` dans $(T, \mathcal{X}, w)$. D'un côté de f, il y a le sous-arbre T_{v_i} de $(T, \mathcal{X}, w)$ enraciné en v_i. Puisque la procédure `éclate` rend $(T, \mathcal{X}, w)$ sous-connexe en v_i, nous obtenons que $G[T_{v_i}]$ est connexe. Montrons que $G[T \setminus T_{v_i}]$ est aussi connexe. Soit $j \neq i$. Puisque G est connexe, considérons une arête $\{x, y\}$ avec $x \in G[T_{v_j}]$, et $y \notin G[T_{v_j}]$. Par la propriété **C2** des décompositions arborescentes, il y a un sac de $T \setminus T_{v_j}$ contenant $\{x, y\}$, et donc $x \in X_w$. Par conséquent, pour tout $j \neq i$, $V(T_{v_j}) \cap R \neq \emptyset$. Puisque R est connexe, nous obtenons que $G[T \setminus T_{v_i}]$ est connexe. Ainsi, la f-coupe considérée est connexe.

Finalement, nous considérons une arête $f = \{v_i, w_j\}$. D'un côté de f, il y a S_j, et de l'autre $T \setminus S_j$. $G[S_j]$ est connexe car $(T, \mathcal{X}, r)$ est sous-connexe en w_j. $G[T \setminus S_j]$ n'est pas forcément connexe. Cependant, si il n'est pas connexe, alors il n'était pas connexe avant l'application de la procédure `éclate`. En d'autres termes, si f-coupe n'est pas connexe, alors, la f'-coupe correspondant à $f' = \{v, w_j\}$ n'est pas connexe. L'algorithme `rend-connexe` prend cela en compte en remplaçant $\{v, w_j\}$ par $\{v_i, w_j\}$ dans C. Cela conclut la preuve du fait que l'invariant **I2** est satisfait.

Etudions à présent la complexité de l'algorithme `rend-connexe`. Avant d'exécuter la phase 1, les sacs de la décomposition arborescente $(T, \mathcal{X})$ peuvent être décomposés en composantes connexes en temps $(\sum_{v \in V(T)} |X_v|^2) \leq O(nk^2)$. Soit v un sommet pour lequel la procédure `éclate` est appliquée lors de la phase 1. Supposons que lorsque cela se produit, v a d_v fils dans la décomposition arborescente courante. Le graphe biparti H correspondant possède alors une partition avec d_v sommets $w_1, \ldots, w_{d_v}$. L'autre est composée de r sommets $Y_1, \ldots, Y_r$. Pour toute paire (Y_i, w_j), cela prend un temps $O(|X_{w_j}| \cdot |Y_i|)$ pour déterminer si l'arête $\{Y_i, w_j\}$ appartient à H, car nous devons seulement tester si $Y_i \cap X_{w_j} \neq \emptyset$. H peut être construit en temps $O(\sum_{i,j} |X_{w_j}| \cdot |Y_i|) \leq O(\sum_j k \cdot |X_{w_j}|) \leq O(\sum_j k^2) \leq O(k^2 d_v)$. Les composantes connexes de H peuvent être calculées en parallèle avec sa construction. Par conséquent, en appliquant la procédure `éclate` sur le sommet v prends un temps total de $O(k^2 d_v)$. Maintenant, observons que, après que v a été éclaté en ℓ sommets $v_1, \ldots, v_\ell$, nous avons $d_v = \sum_{i=1}^{\ell} d_{v_i}$. Par conséquent, ajouter le coût de toutes les applications de la procédure `éclate` au cours de la phase 1 résulte en un temps $O(k^2 \sum_{v \in V(T)} d_v) = O(|V(T)|k^2)$ où T est la décomposition arborescente après la phase 1. Partant d'une décomposition arborescente de n nœuds, la phase 1 produit une décomposition arborescente avec au plus kn nœuds. Par conséquent, la phase 1 prend un temps $O(nk^3)$. Par des arguments similaires, nous pouvons montrer que la phase 2 prend également un temps $O(nk^3)$, ce qui conclut la démonstration. □

3.3 Approximation de l'indice d'échappement connexe

Dans cette section, nous appliquons le théorème 18 pour concevoir un algorithme d'approximation calculant une stratégie de capture connexe. Il en découle une nouvelle borne pour le rapport entre l'indice d'échappement connexe et l'indice d'échappement dans le cas de graphes arbitraires.

Théorème 19 *Pour tout graphe connexe G de n sommets, $mcs(G) \leq s(G) \cdot \log n$.*

Preuve. Plus précisément, nous prouvons que pour tout graphe de n sommets, et pour tout sommet $v_0 \in V(G)$, il existe une stratégie de capture connexe monotone pour G, utilisant au plus $(tw(G) + 1) \log n$ chercheurs et partant de v_0. De plus, cette stratégie peut être calculée à partir d'une décomposition arborescente connexe optimale de G.

La preuve est par induction sur n. Si $n \in \{1, 2\}$, le théorème est vrai. Soit $n \geq 3$ et supposons que, pour tout $n' < n$, le résultat est vrai. Soit G un graphe connexe de n sommets et $v_0 \in V$. Soit $(T, \mathcal{X})$ une décomposition arborescente connexe G, de largeur $tw(G)$. Pour tout sous-arbre T' de T, nous notons $G[T']$ le sous-graphe de G induit par les sommets contenus dans les sacs de T'. Le théorème 2.5 de [RS86] stipule que pour toute décomposition arborescente :

- soit il existe $u \in V(T)$ tel que la suppression de u dans T résulte en q sous-arbres $T_1, \cdots, T_q$ de T, avec $|V(G[T_i])| \leq n/2$ pour tout $1 \leq i \leq q$,

– soit il existe $\{u, u'\} \in E(T)$ telle que la suppression de $\{u, u'\}$ dans T résulte en q sous-arbres $T_1, \cdots, T_q$ de T, avec $|V(G[T_i])| \leq n/2$ pour tout $1 \leq i \leq q$.

Nous traitons uniquement le second cas, dit "arête-centroïde", le premier cas en découle facilement. Soit $\{u, u'\} \in E(T)$ tel que pour tout $1 \leq i \leq q$, $|V(G[T_i])| \leq n/2$. Par définition d'une décomposition arborescente connexe, $G[T_i]$ est un sous-graphe connexe de G. Supposons, sans perte de généralité, qu'il existe $1 \leq p < q$ tel que u est adjacent à chacun des sous-arbres $T_1, \cdots, T_p$, et u' est adjacent à chacun des sous-arbres $T_{p+1}, \cdots, T_q$. Supposons, sans perte de généralité, que $v_0 \in V(G[T_1]) \cup X_u$.

Nous proposons une stratégie de capture connexe S utilisant au plus $(tw(G) + 1) \log n$ chercheurs, en partant de v_0. Ces $(tw(G)+1) \log n$ chercheurs sont divisés en deux équipes : une équipe $\mathcal{A}$ composée de $tw(G) + 1$ chercheurs, et une équipe $\mathcal{B}$ composée de $(tw(G) + 1) \log(n/2)$ chercheurs. La stratégie S est divisée en q phases.

Si $v_0 \in X_u \setminus V(G[T_1])$, la connexité de $(T, \mathcal{X})$ implique l'existence de $X \subseteq X_u$ tel que $v_0 \in X$, $G[X]$ est connexe et il existe $1 \leq i \leq p$ $X \cap V(G[T_i]) \neq \emptyset$ (sans perte de généralité, $i = 1$). Dans ce cas, la stratégie débute par le placement de chercheurs de $\mathcal{A}$ sur les sommets de X. Puis, la stratégie se poursuit par le nettoyage de $G[T_1]$, en partant d'un sommet de $X \cap V(G[T_1])$ comme décrit ci-dessous. Si $v_0 \in V(G[T_1])$, la stratégie débute directement par le nettoyage de $G[T_1]$, en partant de v_0.

D'après l'hypothèse d'induction, il existe S_1 une stratégie de capture connexe monotone pour $G[T_1]$, utilisant les chercheurs de $\mathcal{B}$, et partant de v_0. La première phase de S consiste à appliquer la stratégie S_1, tout en ajoutant les opérations suivantes. A chaque étape à laquelle S_1 consiste à placer un chercheur de $\mathcal{B}$ sur un sommet $v \in X_u$, et tel que aucun chercheur de $\mathcal{A}$ n'occupe déjà v, un chercheur de l'équipe $\mathcal{A}$ est placé sur v. Il est facile de vérifier, qu'à la fin de cette première phase de S, $G[T_1]$ a été nettoyé de façon monotone et connexe. De plus, les sommets de $X_u \cap V(G[T_1])$ sont occupés par des chercheurs de l'équipe $\mathcal{A}$ empêchant toute recontamination issue du reste du graphe. Enfin, tous les chercheurs de l'équipe $\mathcal{B}$ ont été supprimés du graphe.

Pour tout $1 \leq i \leq p$, soit G_i le sous-graphe de G induit par les sommets contenus dans les sacs de $T_1, \cdots, T_i$, et soit $\mathcal{P}_i$ la propriété qui vérifie que (1) G_i est propre, (2) chaque sommet de $X_u \cap V(G_i)$ est occupé par un chercheur de l'équipe $\mathcal{A}$, et (3) les chercheurs de $\mathcal{B}$ ont tous été supprimés de G. Supposons qu'après la phase $i-1$ de S, la propriété $\mathcal{P}_{i-1}$ est satisfaite. Prouvons que nous pouvons poursuivre la stratégie S de manière monotone et connexe, et telle qu'après la phase i de S, la propriété $\mathcal{P}_i$ soit satisfaite. Deux cas se présentent :

– Soit il existe j, $i \leq j \leq p$, et un sommet $x \in V(G[T_j])$ tels que x est occupé par un chercheur de l'équipe $\mathcal{A}$ après l'étape $i-1$ de S. Notons que $x \in X_u$. Sans perte de généralité (quitte à renuméroter les sous-arbres de la décomposition arborescente), nous pouvons supposer que $i = j$. Dans ce cas, d'après l'hypothèse d'induction, il existe S_i une stratégie de capture connexe pour $G[T_i]$, utilisant les chercheurs de $\mathcal{B}$, et partant de x. La phase i de S consiste à appliquer la stratégie S_i, tout en ajoutant les opérations suivantes. A chaque étape à laquelle S_i consiste à placer un chercheur de $\mathcal{B}$ sur un sommet $v \in X_u$, et tel que aucun chercheur de $\mathcal{A}$ n'occupe déjà v, un chercheur de l'équipe $\mathcal{A}$ est placé sur v.

– Soit pour tout j, $i \leq j \leq p$, aucun sommet de $V(G[T_j])$ n'est occupé par un chercheur après l'étape $i-1$ de S. Puisque $(T, \mathcal{X})$ est connexe, G_p est connexe. Donc il existe un sommet de $y \in V(G_p) \setminus V(G_{i-1})$ et un sommet $x \in V(G_{i-1})$ tels que $\{x, y\} \in E(G_p)$. Supposons, sans perte de généralité, que $y \in V(G[T_i])$. D'après les propriétés **C2** et **C3** des décompositions arborescentes, $\{x, y\} \in X_u$. Dans ce cas, d'après l'hypothèse d'induction, il existe S_i une stratégie de capture connexe pour $G[T_i]$, utilisant les chercheurs de $\mathcal{B}$, et partant de y. La phase i de la stratégie S est similaire à celle décrite dans le cas précédent.

Dans les deux cas, il est facile de vérifier qu'à la fin de l'étape i, la propriété $\mathcal{P}_i$ est vérifiée. Après la phase p, G_p est donc propre et chaque sommets de $X_u \cap V(G_p)$ est donc occupé par un chercheur de l'équipe $\mathcal{A}$. Avant la phase suivante de S, nous plaçons des chercheurs de l'équipe $\mathcal{A}$ sur les sommets de X_u qui sont encore innocupés, puis les chercheurs de $\mathcal{A}$ qui occupent les sommets de $X_u \setminus X_{u'}$ sont supprimés du graphe. Les phases $p+1$ à q de la stratégie S se déroulent comme les phases 1 à p, à la différence près que u' joue le rôle de u. C'est à dire que S nettoie successivement les sous-graphes $G[T_j]$, $p+1 \leq j \leq q$, et que les chercheurs de l'équipe $\mathcal{A}$ sont successivement placés sur les sommets de $X_{u'}$.

Il est facile de conclure que la stratégie S ainsi calculée est une stratégie de capture connexe monotone pour G, utilisant au plus $(tw(G)+1)\log n$ chercheurs et partant de v_0. Nous concluons la preuve du théorème en remarquant que $(tw(G)+1) \leq s(G)$. □

Soit $t(n)$ la complexité temporelle du plus rapide algorithme d'approximation pour calculer la largeur arborescente d'un graphe de n sommets, à un facteur $O(\sqrt{\log OPT})$ (le meilleur facteur d'approximation connu actuellement [FHL05a]).

Théorème 20 *Il existe un algorithme en temps $O(t(n) + nk^3 \log^{3/2} k + m \log n)$ et de rapport d'approximation $O(\log n \sqrt{\log OPT})$ pour calculer l'indice d'échappement connexe d'un graphe de n sommets, m arêtes et de largeur arborescente k. Plus précisément, étant donné un graphe connexe G, l'algorithme renvoie une stratégie de capture connexe monotone pour G utilisant au plus $O(cs(G) \log n \sqrt{\log tw(G)})$ chercheurs, en partant de n'importe quel sommet.*

Preuve. La preuve du théorème est obtenue en combinant l'algorithme de Feige et al. [FHL05a], qui permet d'obtenir une décomposition arborescente de largeur approchée, avec l'algorithme `rend-connexe` et le théorème 18, de manière à obtenir une décomposition arborescente connexe de largeur approchée. A partir de cette décomposition, la stratégie proposée dans la preuve du théorème 19 permet de conclure. □

3.4 Cas des graphes de cordalité bornée

L'objectif de cette section est la conception d'un algorithme d'approximation pour calculer une stratégie de capture connexe pour tout graphe cordal G avec un rapport d'approximation $tw(G)$. Pour cela, nous proposons tout d'abord un algorithme qui, à partir d'une décomposition arborescente connexe, détermine une stratégie de capture connexe pour tout graphe de cordalité bornée.

3.4.1 Lemmes techniques

Nous commençons par prouver deux lemmes techniques relatifs aux décompositions arborescentes connexes des graphes de cordalité bornée. Etant donnée une décomposition arborescente connexe d'un graphe de cordalité bornée, ces deux lemmes sont destinés à déterminer des sur-graphes connexes "pas trop grands" de chaque sac de la décomposition.

Lemme 9 *Soit G un graphe connexe de cordalité k, et soit $(T,\mathcal{X})$ une décomposition arborescente connexe de G. Soit $v \in V(T)$. Il existe un sous-graphe connexe $\mathcal{F}_v$ de G, tel que $X_v \subseteq \mathcal{F}_v$, et $|\mathcal{F}_v| \leq (|X_v|-1)\ \lfloor k/2 \rfloor + 1$.*

Preuve. Soit X une composante connexe de $G[X_v]$. Nous construisons $\mathcal{F}_v$ de la façon suivante :

$\mathcal{F}_v \leftarrow X$;
$Y \leftarrow X_v \setminus \mathcal{F}_v$;
tant que $Y \neq \emptyset$ **faire**
  Soient $x \in \mathcal{F}_v$ et $y \in Y$ tels que $\mathrm{dist}_G(x,y) = \mathrm{dist}_G(\mathcal{F}_v, Y)$;
  Soit P un plus court chemin entre x et y ;
  Soit Z la composante connexe de X_v qui contient y ;
  $\mathcal{F}_v \leftarrow \mathcal{F}_v \cup P \cup Z$;
  $Y \leftarrow X_v \setminus \mathcal{F}_v$;
fin tant que

Clairement, $\mathcal{F}_v$ est un sous-graphe connexe de G tel que $X_v \subseteq \mathcal{F}_v$. Nous prouvons que, à chaque exécution de la boucle *tant que*, la longueur du chemin P considéré à cette étape est au plus $\lfloor k/2 \rfloor$. Puisque $\mathcal{F}_v$ et Y sont deux composantes connexes disjointes de $G[\mathcal{F}_v \cup Y]$, il existe $z \in P$ et $w \in V(T)$ tel que $e = \{v,w\} \in E(T)$ et $z \in X_w \setminus \mathcal{F}_v$. De la suppression de e dans T résultent deux sous-arbres T_1 et T_2. Sans perte de généralité, nous pouvons supposer que $v \in V(T_1)$ et $w \in V(T_2)$. Puisque $G[T_1]$ est connexe, soit $a \in \mathcal{F}_v$ et $b \in Y$ tels que $\mathrm{dist}_{G[T_1]}(a,b) = \mathrm{dist}_{G[T_1]}(\mathcal{F}_v, Y)$, et soit P' un plus court chemin entre a et b dans $G[T_1]$. Soit Q un plus court chemin entre x et a dans $\mathcal{F}_v$, et soit Q' un plus court chemin entre y et b dans $G[T_1]$. Soit C le cycle qui résulte de la concaténation de P, Q, P' et Q'. C est un cycle simple. Montrons que C est un cycle sans corde. Puisque P est un plus court chemin, par **C3** : $P \setminus \{x,y\} \subseteq G[T_2]$. Par construction, $P' \subseteq G[T_1]$. Donc, il n'y a pas de cordes entre P et P'. Puisque $\mathcal{F}_v$ et Y sont des composantes connexes disjointes de $G[\mathcal{F}_v \cup Y]$, il n'y a pas de corde entre Q et Q'. Pour finir, si il y a une corde entre $L \in \{P, P'\}$ et $M \in \{Q, Q'\}$, cela contredirait le fait que P et P' sont des plus courts chemins. Donc C est sans corde. Par conséquent, $|C| \leq k$, et donc $\mathrm{dist}_G(x,y) = \mathrm{dist}_G(X,Y) \leq \lfloor k/2 \rfloor$. Ainsi, $|P \setminus \{x,y\}| \leq \lfloor k/2 \rfloor - 1$. Puisqu'il y a au plus $|X_v|$ composantes connexes dans X_v, au plus $|X_v| - 1$ chemins sont inclus dans $\mathcal{F}_v$. Donc, $|\mathcal{F}_v| \leq (|X_v|-1)\ \lfloor k/2 \rfloor + 1$. □

Soit G un graphe connexe de cordalité k, et soit $(T, \mathcal{X})$ une décomposition arborescente connexe optimale de G. Soit $v \in V(T)$. Dans ce qui suit, $\mathcal{F}_v$ désigne le sous-graphe connexe de G tel que défini dans le lemme 9. D'après le lemme 9, $|\mathcal{F}_v| \leq tw(G)\lfloor k/2 \rfloor + 1$. Notons que si G est un graphe cordal, $\mathcal{F}_v = X_v$ pour tout $v \in V(T)$.

Lemme 10 *Soient G un graphe connexe, et $(T, \mathcal{X})$ une décomposition arborescente connexe optimale de G. Soient S un sous-arbre de T, et $\delta_T(S)$ la frontière de S dans T. Soit $H = G[(\cup_{v \in V(S) \setminus \delta_T(S)} X_v) \bigcup (\cup_{v \in \delta_T(S)} \mathcal{F}_v)]$. H est un sous-graphe connexe de G.*

Preuve. Soient $x, y \in V(H)$. Soit $P = \{x = x_1, x_2, \cdots, x_r = y\}$ un chemin entre x et y dans G, et soit $p = |V(P) \setminus V(H)|$. Nous prouvons par induction sur p, qu'il existe un chemin entre x et y dans H. Si $p = 0$, la propriété est satisfaite. Soit $n > 0$, et supposons que le résultat est valide pour tout $p < n$. Supposons que $|V(P) \setminus V(H)| = n$. Soit $i > 1$ le plus petit indice supérieur à 1 tel que : $x_i \in V(P) \setminus V(H)$. Soit j le plus petit indice supérieur à i tel que : $x_j \in V(P) \cap V(H)$. D'après **C3**, il existe $v \in \delta(S)$ tel que x_{i-1} et $x_j \in \mathcal{F}_v$. Puisque $\mathcal{F}_v \subseteq V(H)$, il existe un chemin $\{x_{i-1}, x'_1, \cdots, x'_s, x_j\} \subseteq \mathcal{F}_v$ entre x_{i-1} et x_j. Donc $P' = \{x_1, \cdots, x_{i-1}, x'_1, \cdots, x'_s, x_j \cdots, x_r\}$ est un chemin entre x et y dans G avec $|V(P') \setminus V(H)| < n$. Donc, en utilisant l'hypothèse d'induction, il existe un chemin entre x et y dans H. □

3.4.2 Stratégie de capture connexe

L'algorithme que nous présentons dans cette section utilise une stratégie de capture connexe d'un arbre de décomposition d'un graphe G pour calculer une stratégie de capture connexe pour G lui-même.

La figure 3.5 décrit l'algorithme `Stratégie_connexe` qui, étant donnés un graphe connexe G de cordalité k, et une décomposition arborescente connexe optimale $(T, \mathcal{X})$ de G, calcule une stratégie de capture connexe pour G utilisant au plus $(tw(G) \lfloor k/2 \rfloor + 1)\ cs(T)$ chercheurs.

Dans la suite, une stratégie de capture connexe pour un graphe G est appelée *basique* si elle implique $|V(G)|$ chercheurs et consiste à placer un chercheur sur un sommet $v \in V(G)$, puis à placer successivement un chercheur sur tout sommet inoccupé adjacent à un sommet occupé.

Nous expliquons à présent les principales caractéristiques de l'algorithme `Stratégie_connexe`. Nous prouverons ensuite que la stratégie définie par l'algorithme `Stratégie_connexe` est connexe. Pour éviter toute confusion, nous emploierons *sommet* pour désigner les sommets du graphe, et *nœud* pour désigner les sommets de la décomposition arborescente. L'entrée de l'algorithme `Stratégie_connexe` est un graphe connexe G de cordalité k et une décomposition arborescente optimale connexe $(T, \mathcal{X})$ de G. L'algorithme `Stratégie_connexe` calcule d'abord une stratégie de capture connexe optimale S' de T en utilisant l'algorithme linéaire de [BFFS02]. Ensuite, il utilise cette stratégie pour définir une stratégie de capture connexe optimale S de G. Informellement, la stratégie S "suit" la stratégie S', en nettoyant le sous-graphe $G[\mathcal{F}_v]$ à chaque fois que S' consiste à placer un chercheur sur le nœud $v \in V(T)$. Plus précisément, soit $S' = \{s'_1, \cdots, s'_r\}$ une stratégie de capture connexe optimale de T, où $r > 1$ est le nombre d'étapes de S', et pour tout $1 \leq j \leq r$, s'_j est l'opération effectuée à l'étape j par S'. Soit $v \in V(T)$ le premier nœud de T nettoyé par S'. Puisque S' est connexe et monotone, il existe une séquence $T_1, \cdots, T_r$ de sous-arbres de T tels que :

- $T_1 = (\{v\}, \emptyset)$ et $T_r = T$;
- pour tout $1 \leq j \leq r-1$, T_j est un sous-arbre de T_{j+1} ;
- pour tout $1 \leq j \leq r$, T_j est le sous-arbre propre de T à l'étape j de S'.

Notons que, pour tout $1 \leq j \leq r$, $G[\cup_{w \in V(T_j)} X_w]$ n'est pas nécessairement connexe alors que, d'après le lemme 10,

$$G_j = G[(\cup_{w \in V(T_j) \setminus \delta(T_j)} X_w) \bigcup (\cup_{w \in \delta(T_j)} \mathcal{F}_w)]$$

est un sous-graphe connexe de G.

La stratégie S pour G calculée par l'algorithme `Stratégie_connexe` satisfait $S = Q_1|Q_2|\cdots|Q_r$, telle que, pour tout $1 \leq j \leq r$, Q_j est une séquence d'opérations qui dépend du type de s'_j (placer ou supprimer), et "|" représente la concaténation de ces séquences. Plus précisément, S consiste à appliquer les étapes de Q_1, puis les étapes de Q_2, et ainsi de suite. Pour tout chercheur $i \in \{1, \cdots, s(T)\}$ impliqué dans la stratégie S', nous associons une équipe H_i de $tw(G)\lfloor k/2 \rfloor + 1$ chercheurs qui vont servir dans la stratégie S.

L'étape s'_1 consiste à placer un chercheur, disons le chercheur 1, sur le sommet v. L'algorithme `Stratégie_connexe` détermine la stratégie Q_1 nettoyant $G_1 = G[\mathcal{F}_v]$, qui nettoie $G[X_v]$ plus un certain nombre de chemins de longueur au plus $\lfloor k/2 \rfloor$ connectant les différentes composantes connexes de $G[X_v]$. La stratégie Q_1 est la stratégie basique pour $G[\mathcal{F}_v]$ (telle que définie ci-dessus) partant de n'importe quel sommet de $G[\mathcal{F}_v]$ et impliquant les chercheurs de l'équipe H_1. Après la phase $j \leq r$ de l'algorithme `Stratégie_connexe`, la stratégie $S = Q_1|\cdots|Q_j$ déjà calculée est une stratégie de capture connexe pour G_j, un sous-graphe de G composé de $G[T_j]$ plus un certain nombre de chemins connectant les différentes composantes connexes de $G[T_j]$. La phase $j+1$ de l'algorithme `Stratégie_connexe` consiste alors à concaténer la stratégie S qui résulte des phases précédentes avec une stratégie Q_{j+1} qui nettoie $G_{j+1} \setminus G_j$ de manière à ce que G_j ne soit pas recontaminé. Q_{j+1} dépend de l'opération s_{j+1} :

- Si s_{j+1} consiste à placer un chercheur, disons i, sur le nœud u de $V(T)$, alors Q_{j+1} est une stratégie de capture basique pour $G[\mathcal{F}_u]$ en partant de n'importe quel sommet de $G[\mathcal{F}_u]$ qui a déjà été nettoyé, et en utilisant les chercheurs de l'équipe H_i.
- Si s_{j+1} consiste à supprimer un chercheur, disons le chercheur i, du nœud u de $V(T)$, alors Q_{j+1} consiste à supprimer tous les chercheurs de l'équipe H_i.

Théorème 21 *Etant donné un graphe connexe G de cordalité k, et une décomposition arborescente connexe optimale $(T, \mathcal{X})$ de G, l'algorithme* `Stratégie_connexe` *retourne une stratégie de capture connexe pour G, utilisant au plus $(tw(G)\ \lfloor k/2 \rfloor + 1)\ cs(T)$ chercheurs, en temps polynomial.*

Entrée : une décomposition arborescente connexe optimale $(T, \mathcal{X})$ d'un graphe connexe G de cordalité k.
Sortie : une stratégie de capture connexe $\mathcal{S}$ de G utilisant au plus $(tw(G)\ \lfloor k/2 \rfloor + 1)\ cs(T)$ chercheurs.
début
Déterminer une stratégie connexe monotone optimale S' pour T,
utilisant l'algorithme de [BFFS02] ;
Soit $S' = \{s'_1, \cdots, s'_r\}$, avec r le nombre d'étapes de S' ;
/ pour tout $1 \leq j \leq r$, s_j est l'opération effectuée par S' à l'étape j.*
Soit H un ensemble de $(tw(G)\ \lfloor k/2 \rfloor + 1)\ cs(T)$ chercheurs, divisé en $cs(T)$
équipes $H_1, \cdots, H_{cs(T)}$, chacune contenant $(tw(G)\ \lfloor k/2 \rfloor + 1)$ chercheurs ;
Soit v le nœud de T sur lequel un chercheur, disons le chercheur i, est placé à
l'étape 1 de S' ;
Soit Q_1 une stratégie de capture connexe basique pour $G[\mathcal{F}_v]$ impliquant les chercheurs de
l'équipe H_i, en partant d'un sommet quelconque de $G[\mathcal{F}_v]$;
$\mathcal{S} \leftarrow Q_1$;
/ Après cette phase, il y a un chercheur de l'équipe H_i placé sur chaque sommet de $\mathcal{F}_v$.*/*
pour tout $j = 2, \cdots, r$ **faire**
cas 1 : s'_j consiste à placer le chercheur i sur le nœud $v \in V(T)$.
Soit Q la stratégie connexe basique qui nettoie $G[\mathcal{F}_v]$ en partant d'un sommet propre
de X_v, et en utilisant les chercheurs de H_i ;
$\mathcal{S} \leftarrow \mathcal{S}|Q$;
cas 2 : s'_j consiste à supprimer le chercheur i du nœud $v \in V(T)$.
Soit Q la stratégie consistant à supprimer les chercheurs de l'équipe H_i du graphe ;
$\mathcal{S} \leftarrow \mathcal{S}|Q$;
fin pour tout
renvoie $\mathcal{S}$.
fin.

FIG. 3.5 – Algorithme `Stratégie_connexe`.

Preuve. Nous prouvons par induction sur j, que la stratégie obtenue après la j^{eme} boucle de l'algorithme `Stratégie_connexe` (c'est-à-dire $Q_1|Q_2|\cdots|Q_j$) est une stratégie de capture connexe pour G_j utilisant au plus $(tw(G)\ \lfloor k/2 \rfloor + 1)\ cs(T)$ chercheurs. De plus, nous prouvons qu'après la j^{eme} boucle, il y a au plus $\lambda_j\ (tw(G)\ \lfloor k/2 \rfloor + 1)$ chercheurs sur les sommets de G_j, où λ_j est le nombre de chercheurs placés sur les nœuds de T à l'étape j de S'. Pour $j = 1$, le résultat est évident par définition d'une stratégie basique. Soit $j > 1$ et supposons que le résultat est valide pour tout $j' < j$.

D'après l'hypothèse d'induction, $Q_1|Q_2|\cdots|Q_{j-1}$ est une stratégie de capture connexe pour G_{j-1} qui implique au plus $(tw(G)\ \lfloor k/2 \rfloor + 1)\ cs(T)$ chercheurs, et telle que, à la fin, au plus $\lambda_{j-1}\ (tw(G)\ \lfloor k/2 \rfloor + 1)$ chercheurs occupent des sommets de G_{j-1}. Nous considérons les deux types d'opération :

- **Cas 1 :** L'opération s'_j consiste à placer le chercheur i, sur le nœud $v \in V(T)$. Premièrement, nous prouvons qu'il existe un sommet $x \in X_v$ qui a été nettoyé par la stratégie $Q_1|Q_2|\cdots|Q_{j-1}$. Puisque S' est une stratégie connexe pour T, v est adjacent à un nœud $w \in V(T_{j-1})$. Par l'hypothèse d'induction, les sommets de $X_w \in V(G_{j-1})$ sont propres. Puisque G est connexe, $X_v \cap X_w \neq \emptyset$. Donc, les sommets de $X_v \cap X_w$ sont propres. Soit $x \in X_v \cap X_w$. Ce sommet a été nettoyé par $Q_1|Q_2|\cdots|Q_{j-1}$. Q_j est une stratégie monotone connexe qui nettoie $G[\mathcal{F}_v]$ en partant de x. De plus, aucune recontamination de G_{j-1} n'a lieu puisque aucun chercheur n'est supprimé durant l'exécution de la stratégie Q_j. Par conséquent, $Q_1|Q_2|\cdots|Q_j$ est une stratégie de capture connexe pour G_j.
  Comme s'_j consiste à placer un chercheur, il y a exactement $\lambda_j = \lambda_{j-1}+1$ chercheurs placés sur les nœuds de T à la fin de l'étape j, donc $\lambda_{j-1} < cs(T)$. Donc, d'après l'hypothèse d'induction, à la fin de la stratégie $Q_1|\cdots|Q_{j-1}$, au plus $(tw(G)\ \lfloor k/2\rfloor + 1)\ (cs(T)-1)$ chercheurs occupent les sommets du graphe G. Puisque Q_j est une stratégie basique, elle implique $|\mathcal{F}_v|$ chercheurs supplémentaires. C'est-à-dire, par le lemme 9, Q_j implique au plus $(tw(G)\ \lfloor k/2\rfloor+1)$ chercheurs supplémentaires. Donc, $Q_1|Q_2|\cdots|Q_j$ utilise au plus $(tw(G)\ \lfloor k/2\rfloor+1)\ cs(T)$ chercheurs au total, et, à la fin de la j^{eme} boucle de l'algorithme `Stratégie_connexe`, seulement $\lambda_j\ (tw(G)\ \lfloor k/2\rfloor + 1)$ chercheurs occupent les sommets de G_j.
- **Cas 2 :** l'opération s'_j consiste à supprimer le chercheur i de $v \in V(T)$. Nous prouvons qu'aucun sommet de G_j n'est recontaminé lorsque les chercheurs de H_i sont supprimés au cours de la stratégie Q_j. Supposons que l'on supprime un chercheur de l'équipe H_i du sommet $x \in X_v$. Soit $y \in V(G)$ un sommet encore contaminé. Soit $u \in V(T)$ tel que $y \in X_u$. L'hypothèse d'induction implique que $u \in V(T) \setminus V(T_j)$. Soit P le chemin de v à u dans T et $z \in V(P) \cap \delta(T_j)$. Puisque $z \in \delta(T_j)$, un chercheur occupe le nœud z à l'étape j de la stratégie S'. Donc, à cette étape de la stratégie $Q_1|\cdots|Q''_j$, des chercheurs occupent chacun des sommets de $\mathcal{X}_z$. Finalement, la propriété **C3** implique que tout chemin entre x et y contient un sommet contenu dans X_z. Donc, aucun sommet de X_v n'est recontaminé lorsque les chercheurs de H_i sont supprimés. Par conséquent, G_j reste propre. Ainsi, $Q_1|Q_2|\cdots|Q_{j-1}|Q_j$ est une stratégie de capture connexe de G_j.
  Puisque Q_j consiste à supprimer les chercheurs de H_i, à la fin de la j^{eme} boucle de l'algorithme `Stratégie_connexe`, au plus $\lambda_j\ (tw(G)\ \lfloor k/2\rfloor+1)$ chercheurs occupent les sommets de G_{j-1}.

Soit $n = |V(G)|$. Selon [Gol80], il existe une décomposition arborescente connexe optimale $(T, \mathcal{X})$ de G, telle que $|V(T)| \leq n$. Donc, puisque S' est monotone, $r = O(n)$. La complexité de chaque phase de l'algorithme `Stratégie_connexe` est dominée par le calcul de $\mathcal{F}_v$ pour tout $v \in V(T)$, ce qui peut être effectué en temps $O(tw(G)\ k)$. Donc, la complexité de l'algorithme `Stratégie_connexe` est $O(n\ tw(G)\ k)$. Par conséquent, notre algorithme `Stratégie_connexe` est linéaire pour les graphes de largeur arborescente et de cordalité bornées. □

Corollaire 3 *Soit G un graphe connexe de cordalité k, et soit $(T, \mathcal{X})$ une décomposition* *arborescente connexe optimale de G. Alors,* $cs(G) \leq (2\ s(T) - 2)\ (tw(G)\ \lfloor k/2 \rfloor + 1)$*. De plus, il existe un algorithme en temps polynomial qui, étant donnée une décomposition arborescente connexe optimale de G, calcule une stratégie de capture connexe pour G, impliquant au plus* $(tw(G)\ \lfloor k/2 \rfloor + 1)(2\ s(T) - 2)$ *chercheurs.*

Preuve. Barrière *et al.* [BFST03] ont prouvé que pour tout arbre T, $cs(T) \leq 2\ s(T) - 2$. Le résultat découle donc directement du théorème 21. □

Bodlaender and Thilikos [BT97] proposent une borne supérieure de la largeur arborescente d'un graphe en fonction de son degré maximum et de sa cordalité. Ils ont prouvé que, pour tout graphe connexe G de cordalité k et de degré maximum Δ, $tw(G) \leq \Delta\ (\Delta-1)^{k-3}$. Le corollaire suivant découle de ce résultat.

Corollaire 4 *Soit G un graphe connexe de cordalité k, et de degré maximum* Δ*. Soit* $(T, \mathcal{X})$ *une décomposition arborescente connexe optimale de G. Alors,*

$$cs(G) \leq (2\ s(T) - 1)\ (\Delta\ (\Delta - 1)^{k-3}\ \lfloor k/2 \rfloor + 2).$$

3.4.3 Algorithme dans le cas des graphes cordaux

Cette section donne une borne supérieure pour le rapport entre l'indice d'échappement connexe et l'indice d'échappement dans le cas des graphes cordaux. Pour cela, nous proposons un algorithme en temps linéaire qui calcule une décomposition arborescente connexe optimale pour tout graphe cordal G. Notons que des algorithmes calculant une décomposition arborescente optimale des graphes cordaux étaient déjà connus dans la littérature [Heg06, BP92]. La valeur ajoutée de notre algorithme vient de ce qu'il vérifie de plus que l'arbre de décomposition est isomorphe à un arbre couvrant de G. Associé à l'algorithme précédent, nous en déduisons une borne sur l'indice d'échappement connexe monotone d'un graphe cordal en fonction de sa largeur arborescente et de son indice d'échappement.

Un sommet est dit *simplicial* si l'ensemble de ses voisins induit un graphe complet. Une permutation $\sigma = (v_1, \cdots, v_n)$ des sommets d'un graphe de n sommets est appelée un *ordre d'élimination parfait* si tout sommet v_i est simplicial dans le sous-graphe de G induit par $\{v_i, \cdots, v_n\}$. Fulkerson et Gross [FG65, Gol80] ont prouvé qu'un graphe est cordal si et seulement si il admet un ordre d'élimination parfait. De plus, cet ordre peut être calculé en temps linéaire.

Lemme 11 *Soit G un graphe cordal. Soit* $(T, \mathcal{X})$ *la décomposition arborescente calculée par l'algorithme* `Décomposition_pour_Cordaux` *de la figure 3.6.* $(T, \mathcal{X})$ *est une décomposition arborescente connexe optimale de G telle que T est isomorphe à un arbre couvrant de G. De plus,* $(T, \mathcal{X})$ *est calculée en temps linéaire.*

Preuve. Nous prouvons, par induction sur i, que, à chaque phase i de l'algorithme `Décomposition_pour_Cordaux`, $(T, \mathcal{X})$ est une décomposition arborescente connexe du

Entrée : Un graphe cordal G de n sommets.
Sortie : Une décomposition arborescente connexe optimale $(T, \mathcal{X})$ de G telle que T est isomorphe à un arbre couvrant de G.
début
  Soit $\{u_1, \ldots, u_n\}$ un ordre d'élimination parfait de G ;
  Posons $v_i = u_{n-i+1}$ pour tout $1 \leq i \leq n$;
  $(V(T), E(T)) = (\{v_1\}, \emptyset)$;
  $X_{v_1} = \{v_1\}$;
  pour tout $i = 2, \cdots, n$ **faire**
    Soit G_i le sous-graphe de G induit par $\{v_1, \cdots, v_i\}$;
    Soit N l'ensemble des voisins de v_i dans G_i ;
    $V(T) \leftarrow V(T) \cup \{v_i\}$;
    $X_{v_i} = N \cup \{v_i\}$;
    Soit $j < i$ le plus grand indice tel que $v_j \in X_{v_i}$;
    $E(T) \leftarrow E(T) \cup (v_i, v_j)$;
    $X \leftarrow X \cup X_{v_i}$;
  fin pour tout
fin.

FIG. 3.6 – Algorithme `Décomposition_pour_Cordaux`.

sous-graphe G_i de G induit par $\{v_1, \cdots, v_i\}$, et que T est isomorphe à un arbre couvrant de G_i. Soit $(T^{(i)}, X^{(i)})$ la construction partielle de $(T, \mathcal{X})$ à la fin de la phase i. Si $i = 1$, le résultat est évidemment valide. Supposons que, à la fin de la phase $i - 1$, $(T^{(i-1)}, X^{(i-1)})$ est une décomposition arborescente connexe de G_{i-1}, et $T^{(i-1)}$ est isomorphe à un arbre couvrant de G_{i-1}. Puisque $T^{(i-1)}$ est isomorphe à un arbre couvrant de G_{i-1}, et puisque $(V(T^{(i)}), E(T^{(i)})) = (V(T^{(i-1)}) \cup \{v_i\}, E(T^{(i-1)}) \cup (v_i, v_j))$ avec $(v_i, v_j) \in E(G_i)$ et $v_i \in V(G_i) \setminus V(G_{i-1})$, $T^{(i)}$ est bien isomorphe à un arbre couvrant de G_i. Nous obtenons :

$$\bigcup_{v \in V(T^{(i)})} X_v^{(i)} = \bigcup_{v \in V(T^{(i-1)})} X_v^{(i-1)} \cup \{v_i\} = V(G_{i-1}) \cup \{v_i\} = V(G_i)$$

et

$$\begin{aligned}
\bigcup_{v \in V(T^{(i)})} E(G[X_v^{(i)}]) &= \bigcup_{v \in V(T^{(i-1)})} E(G[X_v^{(i-1)}]) \cup E(G[X_i^{(i)}]) \\
&= E(G_{i-1}) \cup E(G[X_i^{(i)}]) \\
&= E(G_i)
\end{aligned}$$

Par conséquent, les conditions **C1** et **C2** des décompositions arborescentes sont satisfaites. Soit N' l'ensemble des voisins de v_j dans G_j. Puisque v_i est simplicial dans G_i, N est une clique. Donc, $N \setminus \{v_j\} \subseteq N'$ et $X_i^{(i)} \setminus \{v_i\} \subseteq X_j^{(i)}$. Nous en déduisons que la condition **C3** est également vérifiée, et $(T^{(i)}, X^{(i)})$ est une décomposition arborescente. Finalement, pour tout $v \in V(T^{(i)})$, $X_v^{(i)}$ induit un sous-graphe complet de G_i et donc $(T^{(i)}, X^{(i)})$ est connexe et optimale. Puisque un ordre d'élimination parfait d'un graphe cordal peut être obtenu en temps linéaire, l'algorithme `Décomposition_pour_Cordaux` est linéaire. □

Corollaire 5 *Soit G un graphe cordal. Il existe une décomposition arborescente connexe et optimale $(T, \mathcal{X})$ de G qui peut être calculée en temps linéaire et telle que $s(T) \leq s(G)$.*

Théorème 22 *Pour tout graphe cordal G de largeur arborescente $tw(G)$, une stratégie de capture connexe utilisant au plus $(2\ s(G) - 2)(tw(G) + 1)$ chercheurs peut être calculée en temps polynomial.*

Preuve. Découle directement des corollaires 3 et 5. □

Corollaire 6 *Pour tout graphe cordal G de largeur arborescente $tw(G)$,*

$$cs(G)/s(G) \leq 2\ (tw(G) + 1)$$

.

3.5 Perspectives

Ce chapitre a été consacré à l'étude de la question suivante. Existe-t-il une constante α telle que, pour tout graphe connexe G, $cs(G)/s(G) \leq \alpha$? Si nous améliorons la meilleure borne connue pour ce rapport dans le cas général et dans le cas de la classe des graphes cordaux, la réponse à cette question reste malgré tout un problème ouvert. Nous conjecturons que $\alpha = 2$. Plusieurs raisons nous poussent à cette conjecture. Tout d'abord, pour tout arbre T, Barrière *et al.* ont prouvé que :

$$cs(T) \leq 2\ s(T) - 2,$$

et cette borne est atteinte. La raison pour laquelle nous pensons que $\log n$ n'est pas la borne optimale, vient des contraintes fortes que satisfait la stratégie que nous avons exhibée pour prouver cette borne. En effet, celle-ci est non seulement monotone, mais peut également commencer de n'importe quel sommet. C'est pourquoi nous pensons que $\log n$ n'est pas la borne optimale.

Un autre problème ouvert concerne l'existence d'un algorithme en temps polynomial pour calculer une stratégie de capture connexe optimale dans le cas de la classe des graphes d'indice d'échappement connexe borné. En d'autres termes, le problème de déterminer l'indice d'échappement connexe d'un graphe est-il FPT (pour *Fixed Parameter Tractable* en anglais) ?

Chapitre 4

Stratégies de capture connexe visible

Ce chapitre est consacré à l'étude de la question 2 de la section 1.3.4 dans le cas d'un fugitif visible. C'est-à-dire que nous étudions le rapport entre l'indice d'échappement connexe visible d'un graphe et son échappement visible. Nous étudions également la question suivante : le jeu de capture visible connexe est-il monotone ? Nous répondons par la négative aux deux questions. Plus précisément, nous prouvons d'une part qu'il existe des graphes pour lesquels imposer la contrainte de connexité lors de la capture d'un fugitif visible est très pénalisant en terme de nombre de chercheurs, et d'autre part que le jeu de capture visible connexe n'est pas monotone. La situation dans le cas visible est donc très différente que celle dans le cas invisible du chapitre précédent.

4.1 Introduction

Dans ce chapitre, nous étudions le jeu de capture connexe visible. Aucun résultat concernant cette variante n'était encore connu. Comme le chapitre précédent, ce chapitre ne s'intéresse qu'aux stratégies de capture-sommet. Ainsi, $vs(G)$ désignera l'indice d'échappement-sommet visible du graphe G.

Avant d'entrer dans les détails des résultats présentés dans ce chapitre, faisons quelques remarques visant à prévenir d'une confusion qu'il convient d'éviter. Rappelons tout d'abord, qu'une décomposition linéaire connexe peut être utilisée pour en déduire une stratégie de capture connexe monotone d'un graphe. Cela n'est plus du tout vrai dans le cas où le fugitif est visible. Ainsi, si une décomposition arborescente correspond à une stratégie de capture visible monotone, une décomposition arborescente connexe (cf., le chapitre précédent) ne représente en rien une stratégie de capture visible connexe monotone. En effet, les stratégies de capture visible connexes monotones peuvent être associées à des décompositions arborescentes connexes qui satisfont des contraintes supplémentaires. En bref, une telle décomposition $(T, \mathcal{X}, r)$ est connexe et enracinée en $r \in V(T)$, et, pour tout sommet $v \in V(T)$, $G[V \setminus (V(G[T_v]) \setminus X_v)]$ est connexe. Cependant, de telles décompositions se révèlent difficiles à appréhender, et leur utilisation peu fructueuse. Nous n'avons donc pas développé plus avant cet outil.

Dans la section 4.2, nous répondons à la question 2 de la section 1.3.4 lorsque la stratégie considérée vise à capturer un fugitif visible. L'algorithme proposé dans la section 3.3, dans le cas d'un fugitif invisible, peut, bien entendu, être appliqué à la capture d'un fugitif visible. Il en découle aisément que $cvs(G)/vs(G) \leq \log n$ pour tout graphe G de n sommets. L'essentiel de cette section est consacrée à la preuve de l'existence de graphes pour lesquels cette borne est atteinte. Plus précisément, nous prouvons qu'il existe une constante a et une famille infinie de graphes de taille croissante, tels que, pour tout graphe G de cette famille, $cvs(G)/vs(G) \geq a \ \log n$.

La seconde section de ce chapitre répond quant à elle à la question relative à la monotonie du jeu de capture visible connexe. Nous prouvons ainsi que le jeu de capture visible connexe n'est pas monotone. Plus précisemment, nous exhibons une famille infinie de graphes, d'indice d'échappement visible connexe arbitrairement grand, pour lesquels capturer un fugitif visible de manière connexe et optimale requiert au moins une étape de recontamination.

Les résultats de ce chapitre ont été réalisés en collaboration avec Pierre Fraigniaud. Ils ont fait l'objet d'une publication dans les actes de la conférence WG 2006 [FN06b].

4.2 Le coût de la connexité

Cette section est consacrée à la preuve de l'existence de graphes pour lesquels la contrainte de connexité se révèle être très pénalisante dans le cadre de la traque d'un fugitif visible. La preuve du théorème 23 se décompose de la manière suivante. Nous décrivons tout d'abord la construction récursive d'une famille infinie de graphes $\{G_i, i \geq 1\}$ dont nous prouvons qu'ils ont un indice d'échappement visible (c'est-à-dire, une largeur arborescente) borné par une constante (Assertion 9 dans la preuve du théorème 23). Puis, nous prouvons que ces graphes ont un indice d'échappement visible connexe monotone logarithmique en leur taille (Assertion 10) lorsque le point de départ est imposé en deux sommets adjacents particuliers. Enfin, nous prouvons qu'en “accolant” deux copies du graphe G_i (le graphe obtenu est appelé le symétrique de G_i), toute stratégie optimale peut être transformée de façon à débuter en les deux sommets cités ci-dessus (Assertion 11). Ainsi, le graphe symétrique de G_i possède un indice d'échappement visible constant et un indice d'échappement visible connexe monotone logarithmique, ce qui achève la preuve.

Nous commençons par prouver que la borne supérieure de $\log n$ dans le cas invisible pour le rapport entre indice d'échappement connexe et indice d'échappement, vaut également dans le cas d'un fugitif visible.

Nous avons prouvé à la section 3.3 (cf., [FN06a]) que pour tout graphe connexe G de n sommets, il existe une stratégie de capture connexe monotone utilisant au plus $(tw(G)+1)\log n$ chercheurs pour capturer un fugitif invisible. Donc, il existe une stratégie de capture connexe monotone utilisant au plus $(tw(G)+1)\log n$ chercheurs pour capturer un fugitif visible. Puisque $vs(G) = tw(G) + 1$, il en découle qu'il existe une stratégie de capture connexe monotone pour capturer un fugitif visible, en utilisant au plus $vs(G)\log n$ chercheurs. Nous prouvons que cette borne est optimale asymptotiquement.

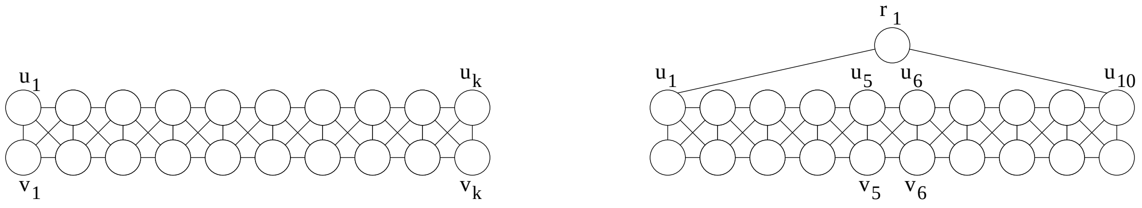

FIG. 4.1 – Une échelle de longueur $k = 10$ (gauche), et le graphe G_1 (droite)

Théorème 23 *Pour tout n_0, il existe $n \geq n_0$ et un graphe G de n sommets tels que toute stratégie de capture connexe monotone pour G utilise au moins $C \cdot vs(G) \cdot \log n$ chercheurs pour capturer un fugitif visible, où C est une constante indépendante de G, de n_0 et de n.*

Preuve. Nous construisons une famille infinie de graphes connexes telle que toute stratégie de capture connexe monotone capturant un fugitif visible dans un graphe de n sommets appartenant à cette famille, utilise au moins $C \cdot vs(G) \cdot \log n$ chercheurs avec $C > 0$ une constante. Dans ce but, nous construisons la famille infinie de graphes connexes $\{G_i, i \geq 1\}$ comme suit.

Nous définissons une *échelle* de longueur $k > 0$ comme étant le graphe de $2k$ sommets $u_1, \ldots, u_k, v_1, \ldots, v_k$ où les sommets u_i sont appelés les sommets du *haut*, et les sommets v_i sont appelés les sommets du *bas* (cf. Fig. 4.1). Il y a une arête entre u_i et u_{i+1} pour tout $i = 1, \ldots, k-1$; il y a une arête entre v_i et v_{i+1} pour tout $i = 1, \ldots, k-1$; et il y a une arête entre u_i et v_j pour tout i, j tels que $|i - j| \leq 1$. Le *centre* d'une échelle de longueur paire $2k$ est le sous-graphe induit par les quatre sommets $u_k, u_{k+1}, v_k, v_{k+1}$. Les *extrémités* d'une échelle de longueur k sont les quatre sommets u_1, v_1, et u_k, v_k, qui sont appelés respectivement les extrémités de gauche et les extrémités de droite.

G_1 est défini comme étant une échelle de longueur $k = 10$, plus un sommet r_1 appelé la *racine*, et qui connectée aux deux extrémités u_1 et u_k de l'échelle (cf. Fig. 4.1). Pour tout $i \geq 1$, la *base* de G_i est le sous-graphe de G_i qui est une échelle de longueur paire, et le *noyau* de G_i est le centre de sa base. Par exemple, la base de G_1 est l'échelle de longueur 10, et le noyau de G_1 est l'ensemble $\{u_5, v_5, u_6, v_6\}$, où v_5 et v_6 sont les sommets du bas du noyau de G_1.

Etant donné G_i avec $i \geq 1$, nous construisons G_{i+1} comme suit (cf. Fig. 4.2). Intuitivement, G_{i+1} est obtenu en insérant deux copies de G_i dans une troisième copie de G_i. Les racines des deux premières copies sont connectées à la racine de la troisième. Des connections supplémentaires sont établies entre le noyau des deux premières copies de G_i et le noyau de la troisième. Dans ce but, le noyau de la troisième copie est légèrement élargie en ajoutant 8 sommets au milieu.

Plus formellement, soit S_i la base de G_i (c'est-à-dire, une échelle de longueur paire $2k$), et soit r_i la racine de G_i. Premièrement, soit H une copie de G_i. Soient $u_k, u_{k+1}, v_k, v_{k+1}$ les quatre sommets du noyau de H (c'est-à-dire, le centre de la base de H). Ce noyau est remplacé par une échelle de longueur 6, c'est-à-dire : les arêtes $\{u_k, u_{k+1}\}$, $\{u_k, v_{k+1}\}$, $\{v_k, v_{k+1}\}$, et $\{v_k, u_{k+1}\}$ sont supprimées, u_k et v_k sont identifiées aux extrémités gauches de l'échelle de longueur 6, et u_{k+1} et v_{k+1} sont identifiées aux extrémités droites de l'échelle

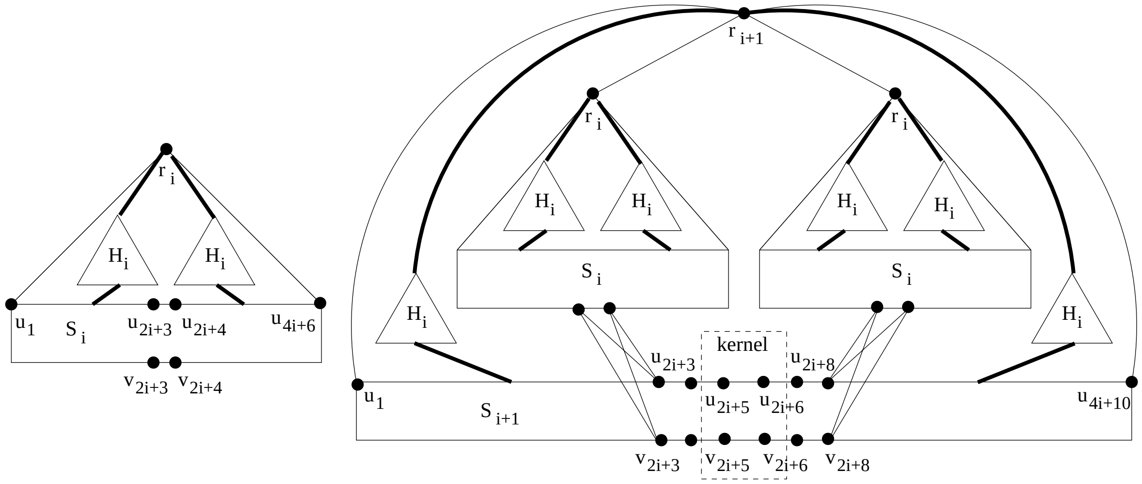

FIG. 4.2 – Construction récursive de G_{i+1} (droite) à partir de G_i (gauche). Les lignes représentées en gras représentent des ensembles d'arêtes.

de longueur 6. Cette opération résulte en une échelle S_{i+1} de longueur $2k+4$, qui devient la base de G_{i+1}. Ensuite, soient H_1 et H_2 deux copies de G_i. Connectons les deux copies de r_i à la racine de H, qui devient la racine r_{i+1} de G_{i+1}. Pour finir, nous ajoutons toutes les arêtes possibles entre les deux sommets u_k et v_k de H, et les deux sommets du bas du noyau de H_1, et toutes les arêtes possibles entre les deux sommets u_{k+1} et v_{k+1} de H, et les deux sommets du bas du noyau de H_2.

Nous obtenons $|V(G_{i+1})| = 1 + 2|V(G_i)| + (|V(G_i)| - 1 + 8) = 3|V(G_i)| + 8$. Donc $|V(G_i)| = 25 \cdot 3^{i-1} - 4$.

Pour résumer, la base de G_i consiste en une échelle de longueur $2k$ avec $k = 2i+3$, dont les sommets du haut sont $u_1, \ldots, u_{2k}$, et les sommets du bas $v_1, \ldots, v_{2k}$. Le noyau de G_i est le centre $\{u_k, v_k, u_{k+1}, v_{k+1}\}$ de cette base. Donc, les sommets du bas du noyau sont les deux sommets v_k et v_{k+1}. Les deux sommets u_1 et u_{2k} sont les extrémités du haut de la base de G_i.

Assertion 9 *Pour tout $i \geq 1$, $tw(G_i) \leq 4$.*

Nous prouvons cette assertion en prouvant la propriété $\mathcal{P}_i$, $i \geq 1$: il existe une décomposition arborescente $(T, \mathcal{X})$ de G_i telle que : (1) la largeur de $(T, \mathcal{X})$ est au plus 4, (2) $(T, \mathcal{X})$ contient un sac $B = \{u_k, v_k, u_{k+1}, v_{k+1}, r_i\}$ où $k = 2i+3$, c'est-à-dire $u_k, v_k, u_{k+1}, v_{k+1}$ est le centre de la base S_i de G_i, et r_i est la racine de G_i, (3) B (c'est-à-dire, le sommet de T associé au sac B) est un sommet de degré deux de T, et (4) les deux voisins B' et B'' de B dans T satisfont $B \cap B' = \{u_k, v_k, r_i\}$ et $B \cap B'' = \{u_{k+1}, v_{k+1}, r_i\}$.

Le sac B est appelé le sac *racine* de T, et B' et B'' sont appelés les voisins de *gauche* et de *droite* de B. Sur la Figure 4.3, la décomposition arborescente de G_i est décrite à gauche : F est le sac racine, et E' et E'' sont respectivement les sacs de gauche et de droite.

Pour $i = 1$, il existe une décomposition arborescente $(T, \mathcal{X})$ (qui, en fait, est une décomposition linéaire) de G_1 composée de 9 sacs, $\{u_i, u_{i+1}, v_i, v_{i+1}, r_1\}$, $i = 1, \cdots, 9$. Chaque sac contient exactement cinq sommets (une clique K_4 plus la racine r_1). $(T, \mathcal{X})$ vérifie $\mathcal{P}_1$.

Supposons que $\mathcal{P}_i$ est satisfaite, et prouvons que cela vaut aussi pour $\mathcal{P}_{i+1}$. G_{i+1} est obtenu en insérant deux copies H_1 et H_2 de G_i dans une troisième copie H de G_i. Soient $(T_1, \mathcal{X}_1)$ et $(T_2, \mathcal{X}_2)$ des décompositions arborescentes de H_1 et H_2, vérifiant $\mathcal{P}_i$, et soit $(T_3, \mathcal{X}_3)$ une décomposition arborescente de H, vérifiant $\mathcal{P}_i$. Nous construisons une décomposition arborescente $(T, \mathcal{X})$ de G_{i+1} vérifiant $\mathcal{P}_{i+1}$ (cf. Fig. 4.3). Soit $k = 2(i+1)+3$. Nous définissons le sac $B = \{u_k, v_k, u_{k+1}, v_{k+1}, r_{i+1}\}$ où $u_k, v_k, u_{k+1}, v_{k+1}$ est le centre de la base S_{i+1} de G_{i+1}, et r_{i+1} est la racine de G_{i+1}. B a deux voisins $B' = \{u_{k-1}, v_{k-1}, u_k, v_k, r_{i+1}\}$ et $B'' = \{u_{k+1}, v_{k+1}, u_{k+2}, v_{k+2}, r_{i+1}\}$ dans T. Ces deux voisins sont aussi de degré 2, c'est-à-dire, chacun d'eux a exactement un voisin différent de B. Nous décrivons la décomposition du côté de B'. La décomposition du côté de B'' est similaire par symétrie de la construction de G_{i+1}. Le voisin de B' distinct de B est $C = \{u_{k-2}, v_{k-2}, u_{k-1}, v_{k-1}, r_{i+1}\}$. Au niveau de C, il y a un branchement dans T, c'est-à-dire, C est un sommet de degré 3 dans T.

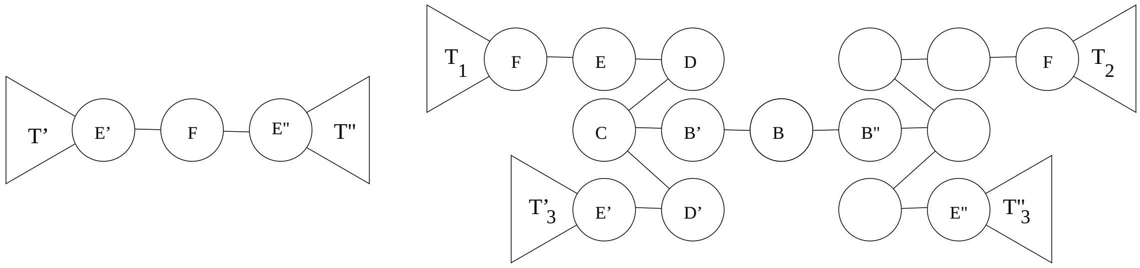

FIG. 4.3 – Construction récursive de la décomposition arborescente de G_{i+1} (droite) à partir de la décomposition de G_i (gauche)

Un des deux voisins de C distinct de B' est $D = \{u_{k-2}, v_{k-2}, r_i, r_{i+1}\}$ où r_i est la racine de H_1. D est de degré 2 dans T, et son voisin distinct de C est $E = \{u_{k-2}, v_{k-2}, x, y, r_i\}$ où x et y sont les deux sommets du bas du noyau de H_1. E est de degré 2 dans T, et son voisin distinct de D est $F = \{x, y, z, t, r_i\}$ où x, y, z, t est le noyau de H_1. D'après l'hypothèse d'induction, F est le sac racine de $(T_1, \mathcal{X}_1)$. Nous attachons T_1 en F dans T.

L'autre voisin de C distinct de B' est $D' = \{u_{k-2}, v_{k-2}, r_{i+1}\}$. Puisque une échelle de longueur 6 a été insérée dans la base de H, le sac D' est, d'après l'hypothèse d'induction, égale à l'intersection du sac racine de $(T_3, \mathcal{X}_3)$ avec son voisin de gauche dans T_3. Dans T, D' est de degré 2, et son voisin distinct de C est E' défini comme le voisin de gauche du sac racine de $(T_3, \mathcal{X}_3)$. A E', nous attachons la partie T_3' de la décomposition $(T_3, \mathcal{X}_3)$ qui résulte de la suppression de l'arête entre son sac racine et son voisin de gauche. Par symétrie, la décomposition du côté de B'' utilise la décomposition T_2 et la partie T_3'' de la décomposition $(T_3, \mathcal{X}_3)$ qui résulte de la suppression de l'arête entre son sac racine et son voisin de droite.

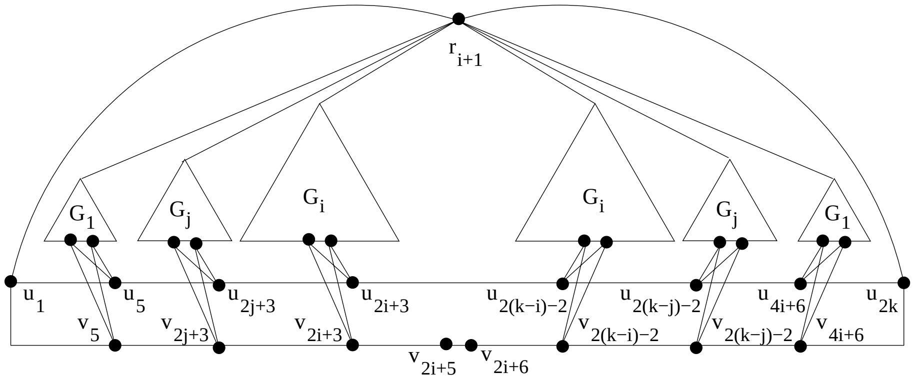

FIG. 4.4 – Définition alternative pour G_{i+1}

Par construction, $(T, \mathcal{X})$ est une décomposition arborescente de G_{i+1} vérifiant $\mathcal{P}_{i+1}$. Sa largeur est 4, et donc $tw(G_i) \leq 4$ pour tout $i \geq 1$. $\diamond$

Puisque pour tout graphe G, $vs(G) = tw(G) + 1$, une conséquence de l'assertion 9 est que $vs(G_i) \leq 5$ pour tout $i \geq 1$. Avant de continuer la preuve du théorème 23, nous avons besoin de donner une autre représentation des graphes G_i. D'après la définition de G_{i+1}, nous pouvons vérifier qu'il résulte de la connexion de deux copies de G_j, pour $j = 1, \ldots, i$, à une échelle de longueur $2k = 4i + 10$ et à une racine r_{i+1} (cf. Fig. 4.4). (Cela est vrai aussi pour $i = 0$). Plus précisément, les copies de G_j sont placées à la suite les unes des autres, dans l'ordre $G_1, G_2, \ldots, G_i, G_i, \ldots, G_2, G_1$. Pour tout j, la racine r_j de chacune des deux copies de G_j est connectée à la racine r_{i+1} de G_{i+1}. Les deux sommets du bas du noyau de la première copie de G_j sont connectés aux sommets u_{2j+3} et v_{2j+3} de la base de G_{i+1}, et les deux sommets du bas du noyau de la deuxième copie de G_j sont connectés aux sommets $u_{2k-(2j+2)}$ et $v_{2k-(2j+2)}$ de la base de G_{i+1}. Finalement, les deux extrémités u_1 et u_{4i+10} de la base de G_{i+1} sont connectées à r_{i+1}. Cette représentation alternatives des graphes G_i nous permet de prouver l'assertion suivante.

Assertion 10 *Pour tout $i \geq 1$, toute stratégie de capture monotone connexe pour capturer un fugitif visible dans G_i, et dont les deux premières étapes consistent à placer un chercheur sur chacun des sommets v_k et v_{k+1} du noyau de G_i, utilise au moins $2i + 4$ chercheurs.*

La preuve est par induction sur $i \geq 1$. En fait, nous prouvons que toute stratégie de capture visible connexe monotone partant de v_k et v_{k+1} dans G_i a au moins $2i+4$ chercheurs placés dans G_i à l'étape avant qu'elle ne nettoie la racine r_i de G_i. Il est facile de vérifier (cf. Fig. 4.1) que le résultat est valide pour G_1, c'est-à-dire que toute stratégie de capture visible connexe monotone partant de v_5 et v_6 dans G_1 a au moins 6 chercheurs placés dans G_1 avant qu'elle ne nettoie la racine r_1. La principale raison pour laquelle ce résultat est vrai, est que progresser de manière connexe de v_5 et v_6 jusqu'à un voisin de r_1, tout en préservant v_5 et v_6 de la recontamination, requiert au moins 6 chercheurs.

Soit $i \geq 1$ et supposons que le résultat est valide pour tout j, $1 \leq j \leq i$. Soit S une stratégie de capture visible connexe monotone pour G_{i+1} partant des deux sommets

$v_k = v_{2(i+1)+3}$ et $v_{k+1} = v_{2(i+1)+4}$ du noyau de G_{i+1}. Considérons G_{i+1} comme décrit sur la figure 4.4. Pour accéder à r_{i+1} de v_k et v_{k+1} de manière connexe, S doit nettoyer la racine r_j d'une des deux copies de G_j pour un certain $1 \leq j \leq i$, ou l'une des deux extrémités u_1 ou u_{2k} de la base de G_{i+1}. Soit R l'ensemble de $2i+2$ sommets composé de toutes les racines des copies des graphes G_j dans G_{i+1}, plus les deux extrémités u_1 et u_{2k}. Soit v le premier sommet de R qui est nettoyé par S. Nous considérons deux cas.

Le premier cas suppose que v est une des deux extrémités de la base de G_{i+1}. Par symétrie de G_{i+1}, nous pouvons supposer, sans perte de généralité, que $v = u_1$. Considérons chaque G_j qui est connecté aux sommets de la base entre u_1 et u_k (Rappelons que les deux sommets du bas du noyau de G_j sont connectés aux sommets u_{2j+3} et v_{2j+3} de la base). Il existe deux chemins sommets-disjoints entre la racine r_j de la copie de G_j considérée et les sommets u_{2j+3} et v_{2j+3} de la base. Par conséquent, si moins de deux sommets de $V(G_j) \cup \{u_{2j+3}, v_{2j+3}\}$ sont occupés par des chercheurs, alors un chercheur doit occuper soit u_{2j+3}, soit v_{2j+3} sans quoi la stratégie S ne serait pas connexe. En effet, u_{2j+3} et v_{2j+3} pourraient être recontaminés par r_j. De plus, si un seul chercheur occupe u_{2j+3} ou v_{2j+3}, alors, un autre chercheur doit occuper soit u_{2j+4}, soit v_{2j+4}, sans quoi la stratégie S ne serait pas connexe. En conséquence, pour tout $1 \leq j \leq i$, au moins deux sommets de $V(G_j) \cup \{u_{2j+3}, v_{2j+3}, u_{2j+4}, v_{2j+4}\}$ sont occupés par des chercheurs. De plus, deux chercheurs doivent occuper des sommets dans $\{u_j, k \leq j \leq 2k\} \cup \{v_j, k \leq j \leq 2k\}$ pour éviter la recontamination de v_k et v_{k+1} depuis u_{2k}. Finalement, au moins quatre chercheurs doivent occuper des sommets dans $\{u_1, v_1, u_2, v_2, u_3, v_3, u_4, v_4\}$ pour connecter u_1 avec la partie propre de G_{i+1}. Cela nous amène à un total d'au moins $2i+6$ chercheurs dans le graphe à l'étape où u_1 est nettoyé, donc S utilise au moins $2(i+1)+4$ chercheurs dans G_{i+1}.

Le second cas suppose que le premier sommet $v \in R$ qui est nettoyé par S est la racine d'une copie de G_j, $1 \leq j \leq i$. Encore, par symétrie de G_{i+1}, nous pouvons supposer, sans perte de généralité, que $v = r_j$ avec r_j la racine de la copie de G_j attachée aux sommets u_{2j+3} et v_{2j+3} de la base de G_{i+1}. Par le même argument que dans le premier cas, nous prouvons que, pour tout $j < t \leq i$, au moins deux sommets de $V(G_t) \cup \{u_{2t+3}, v_{2t+3}, u_{2t+4}, v_{2t+4}\}$ sont occupés par des chercheurs, ce qui fait un total de $2(i-j-1)$ chercheurs dans cette partie du graphe G_{i+1}. D'après l'hypothèse d'induction, quand r_j est nettoyé, $2j+4$ chercheurs occupent des sommets de G_j. De plus, deux chercheurs doivent occuper des sommets dans $\{u_t, 1 \leq t \leq 2j+4\} \cup \{v_t, 1 \leq t \leq 2j+4\}$ pour éviter la recontamination de G_j depuis u_1. Finalement, deux chercheurs doivent occuper des sommets de $\{u_t, k \leq t \leq 2k\} \cup \{v_t, k \leq t \leq 2k\}$ pour éviter la recontamination de v_k et v_{k+1} depuis u_{2k}. Cela nous amène à un total d'au moins $2i+6$ chercheurs dans le graphe quand r_j est nettoyé, donc S utilise au moins $2(i+1)+4$ chercheurs dans G_{i+1}. Cela conclut l'induction et la démonstration de l'assertion. ◇

L'assertion précédente démontre que les graphes G_i occupent beaucoup de chercheurs pour être nettoyés de manière connexe, lorsque le point de départ est l'un des deux sommets v_k et v_{k+1}. La construction que nous présentons maintenant a pour but de forcer ces deux sommets comme point de départ de toute stratégie.

Soit G un graphe connexe, et soit $e = \{u, v\} \in E(G)$. Nous définissons le graphe *symétrique* de G par rapport à e comme étant le graphe obtenu à partir de deux copies de G liées par toutes les arêtes possibles entre les quatre sommets issus des deux copies de $\{u,v\}$ (cf. Fig. 4.5). Le symétrique de G par rapport à $e = \{u,v\}$ est noté $G^*_{u,v}$. La clique K_4 liant les deux copies de G dans $G^*_{u,v}$ est appelée le *centre* de $G^*_{u,v}$.

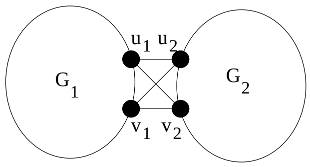

FIG. 4.5 – Le graphe symétrique $G^*_{u,v}$ de G par rapport à l'arête $\{u,v\}$ (les deux copies de G sont indexées par 1 et 2)

Assertion 11 *Soit G un graphe connexe, et soit $\{u,v\} \in E(G)$. Soit k le nombre minimum de chercheurs pour capturer un fugitif visible dans $G^*_{u,v}$ de manière connexe monotone. Il existe une stratégie de capture visible connexe monotone pour G utilisant au plus k chercheurs, et dont les deux premières étapes consistent à placer un chercheur sur u et un chercheur sur v.*

Puisque $G^*_{u,v}$ contient un graphe complet de 4 sommets (son centre), $k \geq 4$. Soit S une stratégie de capture visible connexe monotone pour $G^*_{u,v}$ utilisant k chercheurs. $G^*_{u,v}$ est composé de deux copies G_1 et G_2 de G. Les sommets u_1 et u_2 (resp., v_1 et v_2) sont les copies du sommet u (resp., v), correspondant à G_1 et G_2, respectivement. Sans perte de généralité, supposons que la première étape de S consiste à placer un chercheur sur un sommet de G_1. Puisque S capture tout fugitif, S doit considérer le cas où le fugitif se place dans G_2. Alors, soit $t > 1$ la première étape de S à laquelle un chercheur est placé sur un sommet de G_2. S est connexe monotone, donc ce sommet doit être u_2 ou v_2. Supposons que c'est u_2. A l'étape t, il doit y avoir un chercheur sur u_1 ou v_1 car la stratégie S est connexe. Supposons, sans perte de généralité, qu'il s'agit de u_1. Soit $t' > t$ la première étape de S à laquelle v_2 est propre. Deux cas peuvent se produire. Soit v_2 est nettoyé parce que l'étape t' de S consiste à placer un chercheur sur v_2, ou v_2 est nettoyé car la position des chercheurs forme un séparateur entre v_2 et la position du fugitif à l'étape t' de S. Notons que, entre les étapes t et t', les chercheurs doivent occuper u_2 et u_1 pour éviter la recontamination depuis v_2, pour assurer la monotonie de la stratégie. Donc, entre les étapes t et t', au plus $k-1$ chercheurs occupent les sommets de $G_2 \setminus \{v_2\}$. Soit S' la sous-séquence S obtenue en gardant uniquement les opérations de S qui consistent à, soit placer un chercheur sur un sommet de G_2, soit supprimer un chercheur d'un sommet de G_2.

S' a été définie pour G_2 mais peut, bien entendu, être également appliquée à G puisque G_2 est une copie isomorphe du graphe G. Il en découle que S' est une stratégie visible connexe monotone pour G utilisant au plus k chercheurs, et partant de u. La première étape de S' est exactement l'étape t de S. Soit t'' l'étape de S' qui correspond à l'étape

t' dans S. Puisque, entre les étapes t et t', la stratégie S utilise au plus $k-1$ chercheurs dans G_2, alors, entre les étapes 1 et t'', la stratégie S' utilise au plus $k-1$ chercheurs.

A partir de S', nous définissons une stratégie S_0 pour nettoyer G. Soit S_0 la stratégie suivante :

1. Placer un chercheur sur chacun des sommets u et v.
2. Appliquer les étapes 2 à $t''-1$ de la stratégie S' dans G.
3. Si l'étape t'' de S' consiste à placer un chercheur sur v,

   - alors, appliquer la stratégie S' à partir de l'étape $t''+1$. (L'étape t'' n'a pas besoin d'être réalisée puisqu'un chercheur occupe déjà v)

   Sinon,

   - supprimer le chercheur de v et appliquer la stratégie S' à partir de l'étape $t''+1$. (Le sommet v n'a plus besoin d'être occupé puisqu'il est propre dans S')

S_0 est une stratégie de capture visible connexe monotone pour G partant de u et v, et utilisant au plus k chercheurs. Dans la preuve, nous avons supposé que u_2 était le premier sommet de G_2 à être nettoyé par S. Le cas où v_2 est le premier sommet de G_2 nettoyé par S peut être traité de la même manière en échangeant les rôles des sommets u et v dans S_0. $\diamond$

Nous pouvons à présent combiner les assertions précédentes pour prouver le théorème. Pour tout $i \geq 1$, soit $\mathcal{G}_i$ le symétrique de G_i par rapport à $\{v_k, v_{k+1}\}$ où v_k et v_{k+1} sont les deux sommets du bas du noyau de G_i. Nous avons $|V(\mathcal{G}_i)| = 2|V(G_i)| = 2(25 \cdot 3^{i-1} - 4)$. Nous obtenons $tw(\mathcal{G}_i) \leq tw(G_i)$ en connectant un sac contenant $\{v_k, v_{k+1}\}$ dans la décomposition de la première copie de G_i avec un sac contenant $\{v_k, v_{k+1}\}$ dans la décomposition de la deuxième copie de G_i en insérant entre ces deux sacs, un sac de taille quatre contenant les deux copies v_k et les deux copies de v_{k+1}. Ainsi, d'après l'assertion 9, $tw(\mathcal{G}_i) \leq 4$, et par conséquent $vs(\mathcal{G}_i) \leq 5$. Par ailleurs, en combinant l'assertion 10 avec l'assertion 11, nous obtenons que toute stratégie de capture visible connexe monotone pour $\mathcal{G}_i$ utilise au moins $2i+4$ chercheurs. Par conséquent, toute stratégie de capture visible connexe monotone pour le graphe $\mathcal{G}_i$ de n_i sommets utilise au moins $2\log_3(\frac{\frac{n_i}{2}+4}{25}) + 6$ chercheurs. $\square$

4.3 Monotonie

Dans cette section, nous prouvons que le jeu de capture visible connexe ne satisfait pas la propriété de monotonie. Plus précisément, nous prouvons que le symétrique du graphe représenté sur la figure 4.6 ne peut pas être nettoyé de manière connexe monotone tout en utilisant le moins de chercheurs possible. L'idée de la preuve est la suivante. Nous prouvons tout d'abord que pour nettoyer ce graphe avec le nombre optimal de chercheurs, en partant de la clique H_1 le sommet B doit être nettoyé avant C et D. Pour cela, la seule possibilité pour que la partie propre reste connexe est de "passer" par la clique centrale. Nous prouvons ensuite, que le sous-graphe E peut être nettoyé en partant de

B “vers” la clique A (Assertion 12 du théorème 24). Enfin, nous prouvons que, pour que le nettoyage puisse se poursuivre, le chemin propre entre A et B passant par la clique centrale (c'est-à-dire le chemin nettoyé afin d'assurer la connexité de la partie propre lors du nettoyage du sommet B) doit être recontaminé. L'assertion 14 prouve que la stratégie non monotone décrite succinctement ci-dessus est valide, et l'assertion 15 prouve que si l'on impose la monotonie, alors au moins un chercheur supplémentaire est nécessaire.

Nous énonçons à présent le théorème principal de cette section.

Théorème 24 *Pour tout $k \geq 4$, il existe un graphe G tel que $cvs(G) = 4k+1$ et toute stratégie de capture visible connexe monotone utilise au moins $4k+2$ chercheurs.*

Preuve. La preuve est constructive. Pour construire les graphes mentionnés dans l'énoncé du théorème, nous utilisons à nouveau la famille $\{G_i, i \geq 1\}$ définie dans la preuve du théorème 23. L'intuition de la preuve est la suivante. Considérons le graphe $I^{(k)}$ décrit sur la figure 4.6. Nous allons montrer que le symétrique de ce graphe par rapport à $\{u, v\}$ ne peut pas être nettoyé par une stratégie de capture visible connexe monotone utilisant au plus $4k+1$ chercheurs. Sur cette figure, les graphes E et F sont deux copies du graphe $G_{\lceil 3k/2 \rceil}$. Brièvement, le placement de ces graphes force toute stratégie utilisant au plus $4k+1$ chercheurs à les nettoyer en partant des sommets D et B. Nous prouvons qu'il n'est pas possible de procéder ainsi de façon monotone.

Nous étudions tout d'abord une stratégie de capture particulière pour G_i. L'assertion suivante doit être considérée en opposition avec l'assertion 10. Cette assertion montre que le point de départ de la stratégie a un impact non négligeable sur le nombre de chercheurs nécessaires pour nettoyer ce graphe.

Assertion 12 *Pour tout $i \geq 1$, il existe une stratégie de capture visible connexe monotone pour G_i, utilisant au plus 5 chercheurs, et partant de r_i (c'est-à-dire, la première étape de la stratégie consiste à placer un chercheur sur r_i).*

Introduisons tout d'abord une nouvelle terminologie. Soit $i \geq 1$. Supprimer la racine r_i de G_i ainsi que les arêtes liant les sommets du noyau de G_i résulte en deux composantes (cf. Fig 4.2). Soient L_i la composante qui contient l'extrémité gauche u_1 de la base de G_i, et R_i la composante qui contient l'extrémité droite u_{2k} de la base de G_i, $k = 2i+3$. Une induction triviale sur i permet de prouver que, pour tout $i \geq 1$, L_i et R_i sont 2-sommet-connexes. De plus, pour tout $i \geq 1$ et pour tout j, $1 \leq j \leq i$, L_j et R_j sont deux sous-graphes de G_i. Cela vient du fait que G_i contient tous les graphes G_j's en tant que sous-graphes, avec $j \leq i$. En fait, comme nous l'avons déjà mentionné dans la preuve du théorème 23, supprimer r_i et la base de G_i, résulte en $2(i-1)$ composantes $G_1, G_2, \ldots, G_{i-1}, G_{i-1}, \ldots, G_2, G_1$ (cf. Fig. 4.4). Le graphe G_j inclut dans L_i (resp., R_i) est appelé la j^{eme} *branche* de L_i (resp., R_i). Les sommets u_{2j+3} et v_{2j+3} (resp., $u_{2k-(2j+3)}$ et $v_{2k-(2j+3)}$) liant la j^{eme} branche de L_i (resp., R_i) à la base de G_i sont appelés les *sommets d'accès* à cette branche.

Pour tout $i \geq 1$, nous prouvons la propriété $\mathcal{P}_i$ suivante. Supposons que 5 chercheurs soient placés sur r_i et les quatre sommets du noyau de G_i ; supposons de plus que L_i ou

R_i est propre ; alors, il existe une stratégie de capture visible connexe pour G_i partant de cette situation, et utilisant au plus 5 chercheurs, qui capture le fugitif.

Notons que, bien que la position des 5 chercheurs n'induise pas un sous-graphe connexe, la partie du graphe qui est supposée propre est, elle, bien connexe.

La preuve de $\mathcal{P}_i$ se fait par induction sur $i \geq 1$. $\mathcal{P}_1$ est satisfaite. Supposons que $\mathcal{P}_j$ est satisfaite pour tout $1 \leq j \leq i$. Considérons $\mathcal{P}_{i+1}$. Nous prouvons comment terminer le nettoyage de G_{i+1} en utilisant 5 chercheurs. Par symétrie de G_{i+1}, supposons, sans perte de généralité, que le fugitif se trouve dans R_{i+1}, c'est-à-dire, L_{i+1} est propre. Tout d'abord, les quatre chercheurs sur les sommets du noyau de G_{i+1} peuvent atteindre les sommets d'accès à la première branche de R_{i+1}, ce qui les mène à G_i. Ce déplacement peut facilement être effectué de façon connexe monotone. Si le fugitif n'est pas dans la première branche, alors les chercheurs se déplacent jusqu'aux sommets d'accès à la branche suivante menant à G_{i-1}. Et ainsi de suite. Si le fugitive ne se trouve dans aucune des branches, alors, les chercheurs finissent par le capturer à l'extrémité de R_{i+1}. Donc, supposons que le fugitif se trouve dans la j^{eme} branche lorsque les chercheurs occupent les sommets d'accès à cette branche.

Nous sommes dans une situation telle que deux chercheurs gardent l'accès à la branche alors qu'un troisième chercheur occupe toujours la racine de G_{i+1}. Deux chercheurs sont libres. L'un d'entre eux est placé sur la racine r_j de G_j. Le chercheur qui occupe la racine de G_{i+1} est alors supprimé de r_{i+1}. Les deux chercheurs libres sont alors placés sur les sommets du bas du noyau de G_j. Alors, les deux chercheurs occupant les sommets d'accès à la j^{eme} branche sont supprimés, et placés sur les sommets du haut du noyau de G_j. Puisque le fugitif est visible, L_j ou R_j est propre. La stratégie se conclut en utilisant la propriété d'induction $\mathcal{P}_j$. Ainsi $\mathcal{P}_{i+1}$ est satisfaite.

A présent, décrivons une stratégie qui vérifie les hypothèses de l'assertion. Evidemment, il existe une stratégie de capture visible et connexe S_1 pour G_1 qui utilise au plus 5 chercheurs, et commence à partir de r_1. Soit $i \geq 1$. Considérons la stratégie visible connexe S_i pour G_i définit de la manière suivante. Un chercheur est placé sur r_i. Alors, deux chercheurs sont placés sur les extrémités gauches de la base de G_i. Deux autres chercheurs sont placés sur les deux sommets adjacents aux deux chercheurs précédents. Alors, ces quatre chercheurs se déplacent le long de la base de G_i en direction du noyau de G_i. Ce faisant, ils regardent à chaque fois qu'ils occupent les sommets d'accès à une branche si le fugitif s'y trouve ou non. Deux cas peuvent se produire.

Si les chercheurs croisent l'accès à une branche menant à au graphe G_j et que le fugitif s'y trouve, alors la stratégie consiste à atteindre la situation dans laquelle un chercheur occupe la racine r_j de G_j et quatre autres chercheurs occupent les sommets du noyau de G_j. Pour conclure la stratégie, il suffit alors d'appliquer la propriété $\mathcal{P}_j$.

Sinon, les quatre chercheurs atteignent les sommets à l'extrémité droite de la base de G_i, alors que, dans le même temps, le cinquième chercheur protège la racine. Le fugitif n'a aucun moyen de fuir et il est capturé. ◇

Etudions à présent les autres parties du graphe $I^{(k)}$ (cf. Fig. 4.6), qui consistent essentiellement en des chemins de cliques liées par des couplages parfaits. Plus formellement, rappelons que P_n désigne un chemin de n sommets. Soit $P_{k,n}$ le graphe obtenu en remplaçant chaque sommet de P_n par un graphe complet de k sommets, et chaque arête de P_n par un couplage parfait entre les cliques correspondant aux deux extrémités de l'arête. Nous appelerons le graphe $P_{k,n}$ un *chemin de cliques*. Les *extrémités* de $P_{k,n}$ sont les deux cliques correspondant aux deux extrémités de P_n.

Assertion 13 *Pour tout $n \geq 1$ et tout $k \geq 1$:*
- *Il existe une stratégie de capture visible connexe pour $P_{k,n}$ utilisant au plus $k+1$ chercheurs, et commençant de n'importe quel sommet d'une des extrémités de $P_{k,n}$.*
- *Si $n \geq k+1$, alors, aucune stratégie de capture visible connexe monotone pour $P_{k,n}$, utilisant au plus k chercheurs, et partant d'un sommet quelconque d'une extrémité de $P_{k,n}$, ne peut nettoyer un sommet de l'autre extrémité de $P_{k,n}$.*

La preuve de l'assertion précédente est facile et laissée aux soins du lecteur (qui pourra s'aider de la preuve du théorème 2 du chapitre 1).

Pour $k \geq 1$, soit $I^{(k)}$ le graphe représenté sur la figure 4.6. Cette représentation utilise les notations suivantes :
- Un point noir représente un sommet.
- Un cercle représente une clique dont le nombre de sommets est le nombre indiqué à l'intérieur du cercle.
- Une ligne simple entre deux sommets représente une arête.
- Une ligne simple entre un sommet x et une clique représente l'ensemble des arêtes entre x et les sommets de la clique.
- Une ligne double entre deux cliques représente un couplage parfait entre ces deux cliques si leurs tailles sont identiques, ou entre la plus petite des cliques et une sous-clique de la plus grande si leurs tailles diffèrent.
- Une ligne double en pointillés entre deux cliques de même taille s représente un chemin de cliques, dont les cliques sont de tailles s.
- Les graphes K_A, K_B, K_C et K_D sont des k-cliques deux-à-deux sommet-disjoints, qui sont les extrémités de chemins de cliques, et des sous-graphes de la clique K de taille $4k+1$.
- Les sous-graphes E et F sont isomorphes à $G_{\lceil 3k/2 \rceil}$ (les sommets représentés sont la racine r, et les deux sommets du bas du noyau de $G_{\lceil 3k/2 \rceil}$).

Assertion 14 *Pour tout $k \geq 2$, il existe une stratégie de capture visible connexe pour $I^{(k)}$, partant de u et v, et utilisant au plus $4k+1$ chercheurs.*

La stratégie (non monotone) suivante utilise $4k+1$ chercheurs. Placer un chercheur sur chacun des sommets u et v, et utiliser $k+1$ chercheurs pour nettoyer le chemin de cliques menant à A. Soit P un plus court chemin de A à B passant à travers la clique centrale K. Placer un chercheur sur chaque sommet de P, utilisant $2k+3$ chercheurs, en plus des k chercheurs occupant les sommets de A (cela est possible puisque $k \geq 2$, et donc, $3k+3 \leq 4k+1$).

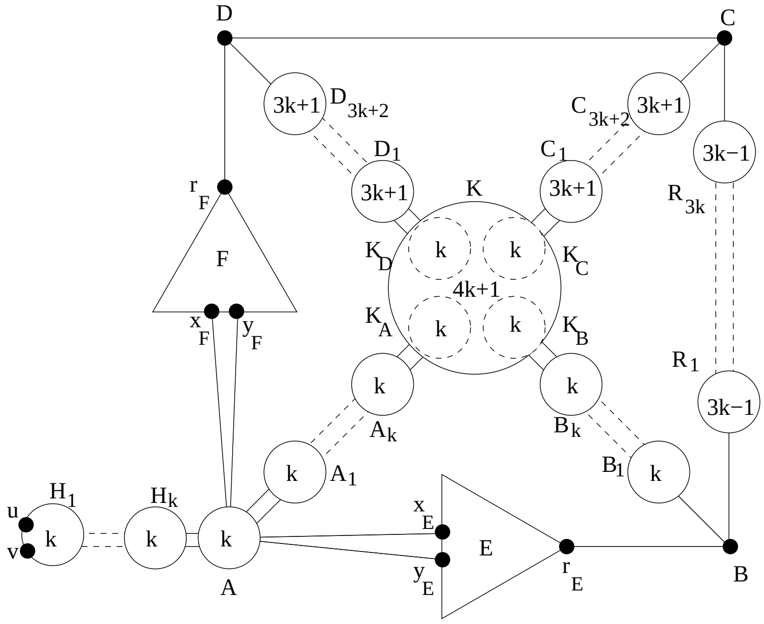

FIG. 4.6 – Le graphe $I^{(k)}$

Si le fugitif est dans le sous-graphe E, alors, supprimer tous les chercheur exceptés ceux sur les sommets de A et celui sur B. D'après l'assertion 12, 5 chercheurs sont suffisants pour nettoyer E en partant de sa racine.

Si le fugitif n'occupe pas un sommet de E, alors supprime tous les chercheurs exceptés les $k+1$ chercheurs occupant A et B. La partie propre du graphe est connexe puisque E reste propre. Notons que la stratégie n'est pas monotone puisque, à cette étape, les sommets de P sont recontaminés. Utiliser les $3k$ chercheurs restant pour nettoyer le chemin de cliques entre B et C (cf., le point 1 de l'assertion 13). Après cette étape, k chercheurs occupent les sommets de A, un chercheur occupe B et un chercheur occupe C. Placer un chercheur sur D.

Si le fugitif est dans le sous-graphe F, alors, supprimer le chercheur sur B. D'après l'assertion 12, 5 chercheurs sont suffisants pour nettoyer F en partant de sa racine.

Si le fugitif n'occupe pas un sommet de F, alors, utiliser les k chercheurs de A, plus un chercheur supplémentaire, pour nettoyer le chemin de cliques entre A et K_A. Après cette étape, k chercheurs occupent les sommets de K_A, trois chercheurs occupent B, C et D. Placer k chercheurs sur les sommets de K_D.

Si le fugitif se trouve sur un sommet dans l'une des cliques D_i, alors, supprimer tous les chercheurs exceptés ceux occupant les sommets de K_D et D. Puis, utiliser les k chercheurs occupant les sommets de K_D et les $3k$ chercheurs restant pour nettoyer le chemin de cliques entre K_D et D.

Si le fugitif ne se trouve pas sur un sommet de l'une des cliques D_i, alors, placer k chercheurs sur les sommets de K_C. Si le fugitif occupe un sommet de l'une des cliques C_i, alors, supprimer tous les chercheurs exceptés ceux occupant les sommets de K_C et C. Utiliser les k chercheurs sur les sommets de K_C, et les $3k$ chercheurs restant pour nettoyer le chemin de cliques entre K_C et C.

Finalement, si le fugitif n'occupe pas un sommet appartenant à l'une des cliques C_i, alors, utiliser le chercheur sur B, et les k chercheurs restant pour nettoyer le chemin de cliques entre B et K_B. Après cette étape, $4k$ chercheurs occupent les sommets de K_A, K_B, K_C et K_D. Utiliser le chercheur restant pour nettoyer le dernier sommet de K.

Dans tous les cas, le fugitif est capturé. La stratégie convient donc. $\diamond$

Notons que, puisque $I^{(k)}$ contient une clique de taille $(4k+1)$, la stratégie décrite dans la preuve de l'assertion 14 est optimale.

Assertion 15 *Pour tout $k \geq 4$, toute stratégie de capture visible connexe et monotone pour $I^{(k)}$ partant des sommets u et v doit utiliser au moins $4k+3$ chercheurs.*

La preuve de cette assertion s'inspire de la preuve de non-monotonie des stratégies de capture (invisible) connexe proposée dans [YDA04]. Dans la suite, nous dirons que deux chemins P et P' entre un sommet v et un ensemble de sommets X, $v \notin X$, sont indépendants si $P \cap P' = \{v\}$. Considérons une stratégie de capture visible connexe et monotone S pour $I^{(k)}$, partant de u et v.

Supposons tout d'abord que la racine r_E de E est nettoyée avant que le sommet B ne le soit. Soit s l'étape de S à laquelle r_E est nettoyée. Soit P un chemin propre entre u et r_E, et soit P' le sous-chemin de P entre u et un sommet dans A. Puisqu'il y a k chemins indépendants entre B et P', tous passant par la K de $I^{(k)}$, k chercheurs doivent garder ces chemins jusqu'à l'étape s, pour éviter la recontamination depuis B. De plus, d'après l'assertion 10, $G_{\lceil 3k/2 \rceil}$ ne peut être nettoyer par une stratégie de capture visible connexe monotone, en partant de x_E ou y_E, et utilisant moins de $3k+3$ chercheurs. Donc, si r_E est nettoyé avant B, alors S utilise au moins $4k+3$ chercheurs.

Le résultat est identique si r_F est nettoyé avant D. Dans ce cas, S utilise au moins $4k+3$ chercheurs.

Par conséquent, pour éviter que S n'utilise trop de chercheurs, B doit être nettoyé avant r_E, et D doit être nettoyé avant r_F. Nous pouvons en déduire qu'il existe un sommet de K_A qui est nettoyé avant chacun des sommets B, C, et D. Soit x le premier sommet de K_A à être nettoyé par S. Supposons que x est nettoyé à l'étape s' de S. (Notons que, tant qu'aucun des sommets B, C et D ne sont nettoyés, ils appartiennent à la même composante de la partie contaminée de $I^{(k)}$, et donc, le fait que le fugitif soit visible n'aide pas à nettoyer l'un de ces sommets).

Soit P_0 un chemin propre entre u et x à l'étape s'. Soit P_1 (resp., P_2) le sous-chemin de P_0 qui relie u à A (resp., A_1 à x).

Supposons que, parmi B, C, et D, D est le premier sommet nettoyé par S. Soit $s'' > s'$ l'étape de S à laquelle D est nettoyé. Soient P'_1 et P'_2 deux chemins indépendants de r_E à deux sommets distincts de P_1. Soient P'_3 et P'_4 deux chemins indépendants de r_F à deux sommets distincts de P_1 qui sont aussi deux-à-deux distincts des extrémités de P'_1 et P'_2. Finalement, Soient $P'_5, \ldots, P'_{k+4}$ k chemins indépendants de B à k sommets distincts de P_2. Puisque $k \geq 4$, ces $k+4$ chemins peuvent être choisis deux-à-deux sommet-disjoints, excepté le fait que P'_1 et P'_2 partagent r_E, P'_3 et P'_4 partagent r_F, et $P'_5, \ldots, P'_{k+4}$ partagent B. De plus, tous ces chemins peuvent être choisis disjoints de toute clique D_i. Donc, pour tout i, $1 \leq i \leq k+4$, et pour toute étape dans $[s', s'']$, il doit y avoir un chercheur distinct occupant un sommet de P'_i à cette étape pour éviter la recontamination de P_0 depuis r_E, r_F, ou B. Le point 2 de l'assertion 13 stipule que, partant d'un sommet de D_1, une stratégie de capture visible connexe et monotone doit utiliser au moins $3k+2$ chercheurs pour nettoyer un sommet de D_{3k+2}. Donc, le nombre total de chercheurs utilisés par S est au moins $4k+6$.

Donc, pour que S utilise moins de $4k+3$ chercheurs, D ne doit pas être le premier sommet à être nettoyé parmi B, C, et D.

Nous prouvons de façon similaire, que, pour que S utilise moins de $4k+3$ chercheurs, C ne doit pas être le premier sommet à être nettoyé parmi B, C, et D.

Donc, pour que S utilise moins de $4k+3$ chercheurs, B doit être le premier sommet à être nettoyé parmi B, C, et D. Soit $s'' > s'$ la première étape à laquelle B est nettoyé par S. A cette étape, il existe un chemin propre de x à B, qui traverse les cliques B_i. Rappelons que nous avons supposé que B est nettoyé avant r_E. (Notons que tant que C et D ne sont pas nettoyés, chacun de ces deux sommets appartient à la même composante connexe de la partie contaminée, et par conséquent, le fait que le fugitif est visible n'aide pas au nettoyage de ces sommets).

Soit P_3 un chemin propre de x à B après l'étape s''.

Considérons à présent deux cas selon lequel des deux sommets C et D est nettoyé avant l'autre.

Supposons tout d'abord que D est nettoyé avant C par S. Soit $s''' > s''$ la première étape à laquelle un chercheur est placé sur D. Soient P'_1 et P'_2 deux chemins indépendants entre r_F et deux sommets distincts de P_1, et soient $P'_3, \ldots, P'_{k+2}$, k chemins indépendants entre C et k sommets distincts de P_2. Puisque $k \geq 2$, les $k+2$ chemins $P'_1, \ldots, P'_{k+2}$ peuvent être choisis deux-à-deux sommet-disjoints, excepté le fait que P'_1 et P'_2 partagent r_F, et $P'_3, \ldots, P'_{k+2}$ partagent C. De plus, tous ces chemins peuvent être choisis disjoints de chaque clique D_i. Donc, pour tout $1 \leq i \leq k+2$, et pour toute étape dans $[s', s''']$, il doit y avoir un chercheur occupant un sommet de P'_i à cette étape, pour éviter la recontamination de P_0 depuis r_F ou C. Le point 2 de l'assertion 13 stipule que, en partant d'un sommet de D_1, nettoyer un sommet de D_{3k+2} de manière visible connexe et monotone requièrt au moins $3k+2$ chercheurs. Ainsi, le nombre total de chercheurs utilisés par S est au moins $4k+4$.

Deuxièmement, supposons C est nettoyé avant D par S. Soit $s''' > s''$ la première étape à laquelle un chercheur est placé sur C. Le sommet C peut être atteint de deux façons différentes : soit le long du chemin de cliques entre C_1 et C_{3k+2}, ou le long du chemin de cliques entre R_1 et R_{3k}. Nous considérons ces deux cas séparément.

- Supposons que C est atteint en passant par le chemin de cliques entre C_1 et C_{3k+2}. Soient P'_1 et P'_2 deux chemins indépendants entre r_F et deux sommets distincts de P_1. Soient $P'_3, \ldots, P'_{k+2}$, k chemins indépendants entre D et k sommets distincts de P_2. Puisque $k \geq 2$, les $k+2$ chemins $P'_1, \ldots, P'_{k+2}$ peuvent être choisis sommet-disjoints, excepté le fait que P'_1 et P'_2 partagent r_F, et $P'_3, \ldots, P'_{k+2}$ partagent D. De plus, tous ces chemins peuvent être choisis disjoints de chaque clique C_i. Par conséquent, pour tout i, $1 \leq i \leq k+2$, et pour toute étape dans $[s', s''']$, il doit y a avoir un chercheur occupant un sommet de P'_i pour éviter la recontamination de P_0 depuis r_F ou D. Le point 2 de l'assertion 13 stipule que, en partant d'un sommet de C_1, nettoyer un sommet de C_{3k+2} de manière visible connexe et monotone requiert au moins $3k+2$ chercheurs. Ainsi, le nombre total de chercheurs utilisés par S est au moins $4k+4$.

– Supposons que C est atteint en passant par le chemin de cliques entre R_1 et R_{3k}. Pour tout i, il existe un sommet $y \in C_i$ qui n'est pas propre à l'étape s'''. Soient P'_1 et P'_2 deux chemins indépendants entre r_F et deux sommets distincts de P_1. Soient $P'_3, \ldots, P'_{k+2}$, k chemins indépendants entre D et k sommets distincts de P_2. Soit P'_{k+3} un chemin de y à P_3. Puisque $k \geq 2$, les $k+3$ chemins $P'_1, \ldots, P'_{k+3}$ peuvent être choisis sommet-disjoints, excepté le fait que P'_1 et P'_2 partagent r_F, et $P'_3, \ldots, P'_{k+2}$ partagent D. De plus, tous ces chemins peuvent être choisis disjoints de chaque clique R_i. Par conséquent, pour tout i, $1 \leq i \leq k+3$, et pour toute étape dans $[s', s''']$, il doit y a avoir un chercheur occupant un sommet de P'_i pour éviter la recontamination de P_0 depuis y, D, ou r_F. Le point 2 de l'assertion 13 stipule que, en partant d'un sommet de R_1, nettoyer un sommet de R_{3k} de manière visible connexe et monotone requiert au moins $3k+1$ chercheurs. Ainsi, le nombre total de chercheurs utilisés par S est au moins $4k+3$.

Ainsi, toute stratégie de capture visible connexe et monotone S pour $I^{(k)}$ utilise au moins $4k+3$ chercheurs. ◇

Soit $k \geq 4$. Soit $G = I^{(k)*}_{u,v}$ le graphe symétrique de $I^{(k)}$ par rapport à l'arête $\{u, v\}$. D'après l'assertion 14, il existe une stratégie de capture visible et connexe pour $I^{(k)}$, partant de u et v, et utilisant au plus $4k+1$ chercheurs. Par conséquent $cvs(G) \leq 4k+1$. Par ailleurs, l'assertion 15 stipule que toute stratégie de capture visible connexe et monotone pour $I^{(k)}$, partant de u et v, utilise au moins $4k+3$ chercheurs. D'après l'assertion 11, cela implique que toute stratégie de capture visible connexe et monotone pour G utilise au moins $4k+3$ chercheurs, c'est-à-dire un nombre de chercheurs strictement supérieur à $cvs(G)$. Ceci conclut la preuve du théorème. □

L'indice d'échappement visible connexe du graphe décrit dans la preuve du théorème 24 est égal à $4k+1$ pour tout $k \geq 4$, c'est-à-dire, au moins 17. Nous pouvons cependant construire des exemples dont l'indice d'échappement visible connexe (non-monotone) est plus petit. Par exemple, la proposition suivante peut être vérifiée facilement :

Proposition 1 *Soit G le graphe représenté sur la figure 4.7. Alors, $cvs(G) = 4$ et toute stratégie de capture visible connexe et monotone pour G utilise au moins 5 chercheurs.*

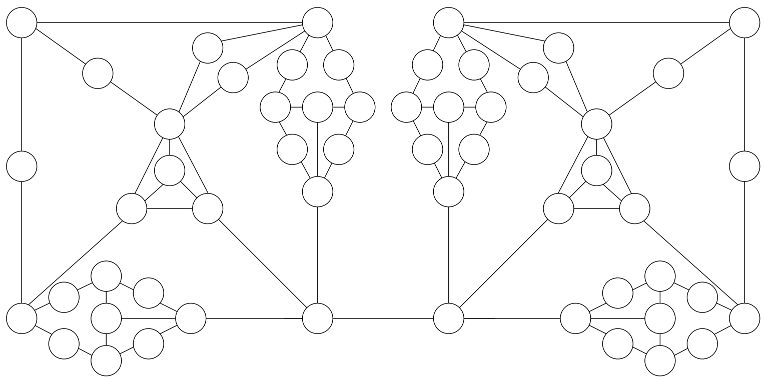

FIG. 4.7 – Un graphe G tel que $cvs(G) = 4$, mais pour lequel toute stratégie de capture visible connexe et monotone pour G requiert au moins 5 chercheurs.

4.4 Perspectives

Dans ce chapitre, nous avons répondu à deux des principales questions qui se posaient relativement au jeu de capture visible connexe. Il reste toutefois de nombreux problèmes ouverts. Parmi les questions qui se posent encore citons celle liée à l'existence d'un algorithme en temps polynomial calculant l'indice d'échappement visible connexe dans le cas de la classe des graphes d'indice d'échappement visible connexe borné. D'autre part, la question de l'appartenance des problèmes de décision relatifs aux stratégies de capture connexes à la classe NP reste ouverte.

Par ailleurs, nous savons que pour tout graphe G tel que $cvs(G) = 2$, il existe une stratégie visible connexe monotone qui nettoie G en utilisant deux chercheurs (cela vient du fait que les graphes d'indice d'échappement visible connexe égal à 2 sont exactement les arbres). La proposition 1 prouve que ce n'est pas le cas si $cvs(G) = 4$. Qu'en est-il si $cvs(G) = 3$? En d'autre termes, est-il vrai que, pour tout graphe G tel que $cvs(G) = 3$, on a $mcvs(G) = 3$? Nous conjecturons que la réponse à cette question est affirmative.

Enfin, le problème précédent nous incite à poser cette même question dans le cas d'un fugitif invisible. En effet, Alspach *et al.* ont prouvé qu'il existe un graphe G pour lequel la recontamination aide à capturer un fugitif invisible lorsque la stratégie est contrainte d'être connexe [YDA04]. Cependant le graphe G en question (cf., figure 1.3) possède près de 400.000 sommets, et $mcs(G) = 290 > cs(G) = 281$. Les questions qui se posent sont donc : (1) quel est le plus petit entier k tel qu'il existe un graphe G avec $cs(G) = k < mcs(G)$? (2) quel est le plus petit graphe pour lequel la recontamination aide à capturer un fugitif invisible, lorsque la stratégie est contrainte d'être connexe ?

Troisième partie

Stratégies de capture dans un contexte réparti

Contenu de la partie

Cette partie est consacrée à l'étude des stratégies de capture dans un contexte décentralisé. La variante de capture qu'il est le plus naturel de traiter dans un tel contexte concerne les stratégies de capture-arête monotones et connexes. En effet, ces stratégies peuvent être décrites ainsi : tous les chercheurs partent du même sommet, la base, et se déplacent le long des arêtes du graphe (représentant un réseau) si ce mouvement n'induit aucune recontamination de la base, et si la partie propre reste connexe.

Dans le chapitre 5, nous proposons un algorithme décentralisé qui permet de nettoyer tout réseau asynchrone. La stratégie est connexe et optimale dans un cadre décentralisé. Nous prouvons que bien que la stratégie effectivement réalisée ne soit pas monotone, notre algorithme permet de calculer une stratégie connexe monotone. La difficulté du problème vient du fait que l'on cherche à nettoyer tout graphe connexe sans que les chercheurs n'aient aucune connaissance *a priori* de l'environnement dans lequel ils se trouvent. Rappelons que le problème de déterminer de telles stratégies est NP-complet dans le cas centralisé. Tout naturellement, notre algorithme réalise donc le nettoyage en temps exponentiel.

Le but du chapitre 6 est d'améliorer l'algorithme proposé dans le chapitre 5. Comme nous l'avons dit, la stratégie réalisée par ce premier algorithme n'est pas monotone, et par conséquent le temps d'exécution peut être exponentiel. Une question se pose alors : de quelle(s) information(s) les chercheurs doivent-ils disposer *a priori* sur le graphe pour qu'il existe un algorithme utilisant cette information leur permettant de nettoyer le graphe de manière monotone ? Nous répondons à cette question de manière quantitative. Nous déterminons ainsi la quantité minimum d'information nécessaire aux chercheurs pour qu'il existe un algorithme décentralisé de nettoyage qui puisse, en utilisant cette information, nettoyer tous les graphes.

Chapitre 5

Nettoyer un réseau inconnu

Ce chapitre est consacré à la présentation d'un algorithme qui permet à une équipe de chercheurs de nettoyer un réseau de façon répartie. La stratégie réalisée par les chercheurs est connexe, et elle est effectuée de façon complètement décentralisée, dans un réseau asynchrone et dont la structure n'est pas connue *a priori*.

5.1 Introduction

Parmi les principales applications qui motivent l'étude des jeux de capture dans les graphes, examinons le cas du nettoyage d'un réseau contaminé (par exemple un réseau de type Internet contaminé par un virus) et celui de la recherche d'une personne (par exemple égarée dans un réseau de souterrains). Dans ces deux cas, les chercheurs qui réalisent le nettoyage du réseau ne peuvent pas toujours se référer à une entité centralisée pour coordonner leurs mouvements. Il est donc nécessaire de concevoir des algorithmes qui permettent aux chercheurs de nettoyer un graphe de manière décentralisée. Une autre caractéristique des réseaux "réels" est que leur topologie est souvent inconnue des chercheurs qui doivent les nettoyer. Dans le contexte décentralisé, nous considérerons les stratégies connexes car elles assurent l'intégrité des communications entre les chercheurs. Enfin, tant que la téléportation n'aura pas été inventée, il est impossible pour les chercheurs de sauter d'un sommet à l'autre comme c'est le cas dans le cadre des jeux de capture-sommet. Dans cette section, nous nous focaliserons donc sur les stratégies de capture-arête où les chercheurs sont contraints de se déplacer au sein du graphe sans téléportation d'un sommet à un autre.

Avant de poursuivre, notons que la terminologie définie dans la section 1.2.4 est légèrement modifiée dans ce chapitre pour une meilleure lisibilité. En effet, dans ce chapitre, nous considérons uniquement des stratégies de capture-arête invisible que nous désignerons simplement *stratégies de capture*. Etant donné un graphe G, *l'indice d'échappement* de G, c'est-à-dire l'indice d'échappement-arête de G, sera noté $s(G)$ (au lieu de $es(G)$). Un "m" (resp., un "c") accolé à ce symbole signifiera que les stratégies considérées sont monotones (resp., connexes).

On trouve dans la littérature plusieurs algorithmes décentralisés réalisant le nettoyage de graphe (cf. section 1.3.5). Ils traitent cependant tous de graphes ayant une topologie particulière [FHL05b, FHL06, FLS05, Luc07]. Dans ce chapitre, nous présentons un algorithme qui traite tous les graphes, indépendamment de leur structure. Cet algorithme prend un graphe G et un sommet $u \in V$ en entrée, et permet à une équipe de $mcs(G, u) + 1$ chercheurs de nettoyer le graphe G en partant de u. La stratégie est réalisée de façon décentralisée, asynchrone et connexe. Le chercheur supplémentaire (par rapport à $mcs(G, u)$) est nécessaire du fait de l'asynchronisme (cf. explication relative à la figure 1.4). Pour prouver la correction de notre algorithme décentralisé, nous proposons également un algorithme semi-décentralisé. Nous prouvons la correction de ce dernier, puis nous prouvons que les deux algorithmes (semi-décentralisé et décentralisé) sont équivalents.

La section 5.2 est consacrée à la définition du modèle que nous emploierons. Y est également présentée l'idée du déroulement de notre algorithme. La section 5.3 présente notre algorithme semi-décentralisé, ainsi que sa preuve. La section 5.4 définit les structures de donnée utilisées par notre algorithme décentralisé, puis l'algorithme lui-même. Enfin, la section 5.5 prouve l'équivalence entre les deux algorithmes présentés.

Les résultats de ce chapitre ont été réalisés en collaboration avec Lélia Blin, Pierre Fraigniaud et Sandrine Vial. Ils ont donné lieu à une publication dans la conférence SIROCCO 2006 [BFNV06].

5.2 Préliminaires

Dans cette section, nous présentons le modèle utilisé dans ce chapitre. Puis, nous exposons formellement les caractéristiques de l'algorithme `Nettoyage_réparti`, dont nous décrivons informellement le principe de fonctionnement.

5.2.1 Modèle

Le modèle auquel nous nous référons dans ce chapitre est celui généralement utilisé dans les études des jeux de capture dans un contexte réparti (cf., [FLS05, FHL05b, FHL06, Luc07] et section 1.3.5). Ce modèle est celui défini à la section 1.3.5. Nous commençons par en rappeler les principales caractéristiques.

Un réseau est ici modélisé par un graphe dont les arêtes sont étiquetées en chaque nœud par des numéros de port. Les nœuds ne sont pas étiquetés (le réseau est dit anonyme). Les chercheurs sont modélisés par des entités mobiles, avec des identifiants distincts, qui peuvent se déplacer le long des arêtes du graphe. Chaque nœud du réseau est équipé d'un tableau blanc, accessible en lecture et en écriture par les chercheurs. L'accès aux tableaux blancs est supposé effectué grâce à un processus d'exclusion mutuelle équitable (c'est-à-dire que chaque chercheur présent sur un nœud accédera ultimement au tableau blanc de ce nœud). La décision prise par un chercheur sur un nœud dépend de l'état du chercheur, du contenu du tableau blanc du nœud), et du numéro de port par lequel le chercheur est arrivé. Cette décision peut être soit de quitter le sommet par un certain port, soit d'attendre sur le sommet jusqu'à avoir à nouveau accès au tableau blanc, soit encore d'appeler un chercheur supplémentaire (nous verrons comment cela peut être réalisé). Les chercheurs sont asynchrones dans le sens où chaque action réalisée par un chercheur prend un temps fini mais non borné. Quand les chercheurs sont lancés dans le réseau, ils ignorent tout de sa structure et n'ont aucune information le concernant. Ils ignorent même la taille du réseau.

Un protocole résolvant le *problème du nettoyage réparti* est un programme réparti exécuté par les chercheurs pour nettoyer un graphe, en partant d'un sommet quelconque fixé, appelé la base, et ce de manière connexe. Initialement, un chercheur est spontanément généré à la base, et toutes les arêtes du graphe sont contaminées. Les chercheurs peuvent se déplacer le long des arêtes du graphe, ou appeler un nouveau chercheur. Notons que lorsqu'un nouvel chercheur est appelé, il apparaît à la base, qui doit impérativement être occupée par un autre chercheur : celui qui a appelé le nouveau chercheur. Le i^{eme} chercheur généré par la base prend l'identifiant i. L'exécution du protocole aboutit à une équipe de chercheurs se déplaçant dans le graphe pour le nettoyer. Nous insistons sur le fait que, dans ce chapitre, le protocole est exécuté sans aucune connaissance *a priori* sur le réseau. Notons également que, de par son statut particulier de point d'accès au réseau, la base ne doit jamais être recontaminée.

5.2.2 L'algorithme `Nettoyage_réparti`

Notre résultat principal est la conception d'un algorithme, `Nettoyage_réparti`, qui résout le problème du nettoyage réparti dans tous les graphes. Les performances de notre protocole sont à mettre en comparaison avec celles des stratégies de capture-arête connexes monotones. Le théorème suivant résume les principales caractéristiques de `Nettoyage_réparti`.

Théorème 25 *Pour tout réseau connexe, anonyme et asynchrone G, et pour tout $u_0 \in V(G)$,* `Nettoyage_réparti` *permet de nettoyer G, de manière connexe, en partant de la base u_0, et sans que les chercheurs n'aient de connaissance a priori concernant la structure de G. Les principales caractéristiques de* `Nettoyage_réparti` *sont*
- *il utilise au plus $k = mcs(G, u_0)+1$ chercheurs si $mcs(G, u_0) > 1$, et $k = 1$ chercheur si $mcs(G, u_0) = 1$;*
- *Tout chercheur impliqué dans la stratégie utilise $O(\log k)$ bits de mémoire ;*
- *Lors de l'exécution de* `Nettoyage_réparti`*, au plus $O(m \log n)$ bits d'information sont stockés sur chaque tableau blanc.*

Remarques.
- Notons que le théorème ci-dessus implique que pour tout réseau qui peut être nettoyé par un nombre constant de chercheurs, le protocole `Nettoyage_réparti` peut être implémenté en utilisant des automates finis.
- La stratégie effectuée par les chercheurs est connexe mais pas forcément monotone. Cependant, il est facile de vérifier que, une fois qu'un graphe G a été nettoyé par les chercheurs appliquant `Nettoyage_réparti`, la description d'une stratégie S est stockée de manière distribuée sur les tableaux blancs des nœuds de G. S est une

stratégie de capture connexe monotone pour G, partant de u_0, et utilisant au plus $mcs(G, u_0) + 1$ chercheurs. De plus, un chercheur avec au plus $O(\log n)$ bits de mémoire peut collecter S, en supposant qu'aucun intrus ne peut plus corrompre les tableaux blancs entre le moment où S a été écrite sur les tableaux blancs et celui où elle est collectée.

– Notons aussi que la stratégie S calculée par le protocole `Nettoyage_réparti` est optimale au sens suivant. Pour tout $k \geq 1$, il existe un graphe G et $u_0 \in V(G)$ tel que $k = mcs(G, u_0)$ mais tout protocole de nettoyage réparti $\mathcal{P}$ résolvant le problème du nettoyage réparti dans G en partant de u_0 requiert $k + 1$ chercheurs [FHL05b].

5.2.3 Idée du protocole `Nettoyage_réparti` et de sa preuve

Soit un réseau G, et $k \geq 1$. Une k-configuration est un ensemble $X \subseteq E(G)$ tel que $|\delta(X)| \leq k$. Intuitivement, une k-configuration représente une zone propre protégée par au plus k chercheurs. Le *graphe des k-configuration* $\mathcal{C}_k$ de G est défini comme suit. $V(\mathcal{C}_k)$ est l'ensemble de toutes les k-configurations possibles. Il y a un arc de X à X' dans $\mathcal{C}_k$ si la configuration X' peut être atteinte de X par une étape d'une stratégie de capture monotone connexe (place, déplace ou supprime un chercheur), utilisant k chercheurs. L'objectif du protocole `Nettoyage_réparti` est principalement d'essayer successivement pour $k = 1, 2, \ldots$, si le graphe des k-configurations $\mathcal{C}_k$ peut être traversée de $\emptyset$ à $E(G)$ de telle manière que les chercheurs partent de u_0. Si cette traversée est possible, alors `Nettoyage_réparti` termine après avoir nettoyé le graphe en utilisant au plus k chercheurs. Sinon, `Nettoyage_réparti` essaie avec $k + 1$ chercheurs.

Remarque. Cette approche est similaire à plusieurs algorithmes centralisés existants dans la littérature (cf., e.g., [ACP87, FFN05, FKT04] et voir section 2.4). Cependant, la difficulté de notre approche est de découvrir de façon *décentralisée* si le graphe des configurations $\mathcal{C}_k$ peut être traversé de $\emptyset$ à $E(G)$.

Pour k fixé, l'objectif de `Nettoyage_réparti` est d'organiser les mouvements des chercheurs de façon à ce qu'ils réalisent un DFS de $\mathcal{C}_k$ (en ignorant tout de la structure de G, et dans un environement asynchrone). Cet objectif est atteint en suivant un ordre spécifié par une pile *virtuelle* dans laquelle sont stockées les informations relatives aux mouvements des chercheurs. Informellement, le protocole `Nettoyage_réparti` construit tous les états possibles de la pile virtuelle, dans un ordre lexicographique défini sur les états de cette pile. La difficulté du protocole est de distribuer la pile virtuelle sur les tableaux blancs de telle sorte que, lorsqu'un chercheur arrive sur un nœud, il trouve sur le tableau blanc l'information qui lui est nécessaire pour calculer le prochain pas de la stratégie qu'il doit effectuer. Puisque l'intrus peut corrompre les tableaux blancs, les étapes durant lesquelles les chercheurs se retirent des nœuds préalablement visités doivent être traitées de manière à être sûr qu'aucune information ne soit perdue. Notons ici que, bien que la stratégie qui est ultimement calculée par les chercheurs est monotone (c'est-à-dire que les informations stockées sur tableaux décrivent une stratégie monotone), les stratégies qui ont été testées au préalable (en suivant l'ordre lexicographique des états de la pile virtuelle), mènent parfois à des retours en arrière, et donc à des étapes de recontamination. Si toutes les stratégies avec k chercheurs échouent, alors les chercheurs reviennent à la base, un chercheur supplémentaire est appelé, et le protocole commence à tester les stratégies utilisant $k+1$ chercheurs.

Le chercheur supplémentaire utilisé par le protocole `Nettoyage_réparti`, en comparaison avec $mcs(G, u_0)$, est utilisé pour éviter les interblocages comme ceux décrits dans [FHL05b]. Il est aussi utilisé pour coordonner les autres chercheurs et transmettre des informations entre les chercheurs. Il pourrait être remplacé par de simples moyens de communication entre les chercheurs. Par exemple, si les chercheurs avaient la capacité d'envoyer et de lire des messages à partir de la base, cela pourrait éviter l'emploi de ce chercheur supplémentaire. Dans Internet par exemple, chaque chercheur n'aurait qu'à conserver en mémoire l'adresse IP de la base pour communiquer avec elle.

La preuve de la correction du protocole `Nettoyage_réparti` s'effectue en deux phases. D'abord, nous prouvons la validité d'un algorithme, noté $\mathcal{A}$, qui utilise une pile centralisée pour traverser le graphe des configurations $\mathcal{C}_k$. La seconde partie de la preuve consiste à prouver la correspondance univoque entre toute exécution de `Nettoyage_réparti` utilisant une pile virtuelle (c'est-à-dire décentralisée), et une exécution de $\mathcal{A}$ utilisant une pile centralisée.

5.3 Un algorithme semi-décentralisé

Dans cet section, nous décrivons l'algorithme $\mathcal{A}$ permettant à une équipe de chercheurs lancés dans un environnement inconnu, de capturer un intrus caché dans le réseau. L'algorithme $\mathcal{A}$ n'est pas complètement décentralisé puisqu'il utilise une pile centralisée, accessible par les chercheurs à partir de n'importe quel nœud.

5.3.1 Algorithme $\mathcal{A}$

L'algorithme $\mathcal{A}$ utilise le concept de *mouvements larges*, définis par des triplets (a_i, a_j, p) où a_i et a_j représentent des chercheurs, et p est un numéro de port. Intuitivement, à chaque mouvement large sera associé le nettoyage d'une arête.

Définition 21 *Un* mouvement large (a_i, a_j, p) *correspond à : (1) le chercheur a_i rejoint le chercheur a_j, et (2) le chercheur avec le plus petit identifiant parmi a_i et a_j quitte le nœud à présent occupé par les deux chercheurs par le port p. (Notons que $i = j$ est autorisé, auquel cas, a_i quitte le nœud qu'il occupe par le port p).*

La pile centralisée stocke les mouvements larges, et par conséquent décrit une séquence d'opérations réalisées par les chercheurs. Plus précisément, lire la pile de bas en haut définit une séquence d'opérations qui décrit l'exécution partielle d'une stratégie de capture.

Définition 22 *A $k \geq 1$ fixé, un état de la pile virtuelle est* valide *s'il existe une stratégie de capture connexe monotone utilisant au plus k chercheurs, dont cet état de la pile décrit une exécution partielle.*

Par abus de langage, nous dirons parfois qu'une pile Q est valide, pour signifier que l'état courant S de la pile Q est valide. Etant donné un état valide S de la pile Q, nous noterons X_S la configuration induite par S, c'est-à-dire X_S est l'ensemble des arêtes propres après l'exécution des mouvements larges dans S.

Le principe de l'algorithme $\mathcal{A}$ est le même que celui décrit dans la section 5.2.3. C'est-à-dire qu'il essaie, pour tout $k = 1, 2 \ldots,$ toutes les stratégies de capture connexes monotones possibles avec k chercheurs, jusqu'à atteindre soit une situation dans laquelle le graphe est propre, soit une étape de l'exécution à laquelle toutes les stratégies possibles ont été testées. Dans ce dernier cas, l'algorithme $\mathcal{A}$ recommence avec $k+1$ chercheurs en appelant un nouveau chercheur à la base.

A partir de maintenant, nous supposons que k est fixé. Les k chercheurs sont désignés par $a_1, \ldots, a_k$, tel que l'identifiant de a_i est simplement l'indice i. L'algorithme $\mathcal{A}$ est décrit sur la figure 5.1. Nous détaillons ici son comportement. L'algorithme $\mathcal{A}$ renvoie un booléen *possible*. Si *possible* vaut vrai, alors le nettoyage du réseau avec k chercheurs est possible, auquel cas la pile Q calculée par l'algorithme $\mathcal{A}$ est valide, et contient une stratégie de capture connexe monotone qui nettoie G avec k chercheurs.

Nous dirons qu'un chercheur est *libre* s'il ne préserve pas la partie propre du graphe de la recontamination, donc un chercheur qui occupe un sommet dont toutes les arêtes incidentes sont propres. Si un sommet de la frontière de la partie propre (un sommet incident, à la fois à une arête propre et à une arête sale) est occupé par plusieurs chercheurs, tous sauf l'un d'entre eux deviennent libres. Au cours de l'exécution de l'algorithme $\mathcal{A}$, la pile Q est initialement vide, et seul a_1 est placé sur u_0. Les autres chercheurs $a_2, \ldots, a_k$ sont libres. En plus de la pile centralisée Q, l'algorithme $\mathcal{A}$ utilise une variable locale *etat* qui peut prendre les deux valeurs NETTOIE ou RETOUR dont la signification deviendra claire dans la suite. Finalement, l'algorithme $\mathcal{A}$ utilise une variable booléenne *decide* qui est fausse jusqu'à ce que soit une stratégie de capture connexe monotone utilisant k chercheurs nettoyant le réseau est découverte, soit toutes les stratégies de capture connexes monotones utilisant k chercheurs ont été testées. Ainsi, la boucle principale *tant-que* de l'algorithme $\mathcal{A}$ est basée sur la valeur de *decide* (cf. Figure 5.1). Cette boucle principale *tant-que* est essentiellement composée de deux blocs d'instructions. Ces blocs sont exécutés selon la valeur de *etat* (NETTOIE ou RETOUR).

L'algorithme exécute l'un de ces blocs à moins que tous les chercheurs soient libres. Dans ce cas, une stratégie a été trouvée. Initialement, a_1 est placé sur u_0 et donc n'est pas libre. Le cas NETTOIE correspond à la situation dans laquelle l'algorithme $\mathcal{A}$ vient de nettoyer une arête, c'est-à-dire la dernière exécution de la boucle principale *tant-que* a résulté en l'insertion d'un mouvement large dans Q. Le cas RETOUR correspond à la situation dans laquelle la dernière exécution de la boucle principale *tant-que* a résulté en la suppression du dernier mouvement large de la pile Q, c'est-à-dire cette exécution a résulté en la recontamination d'une arête.

Concentrons nous sur le cas *etat* = NETTOIE. L'algorithme $\mathcal{A}$ ne considère qu'un certain type de mouvements larges, ceux qui n'entraînent aucune recontamination (c'est pourquoi $\mathcal{A}$ finit par déterminer une stratégie monotone). Plus formellement, considérons un état valide S de la pile Q, c'est-à-dire S est une séquence de mouvements larges notée

$M_1|\dots|M_r$. Empiler un mouvement large M dans Q résulte en un nouvel état, noté $S|M$. Nous disons qu'un mouvement M est *valide par rapport à* Q si $S' = S|M$ est un état valide. Notons que $\mathcal{A}$ ne maintient pas l'ensemble X des arêtes propres, ni l'ensemble des chercheurs libres. En effet, étant donné un état valide S d'une pile Q, il est facile de construire X_S en exécutant la stratégie de capture partielle décrite par S. Un chercheur est donc *libre* s'il occupe un sommet de $\delta(X_S)$ ou un sommet occupé par un autre chercheur d'identifiant plus petit. Il existe donc une caractérisation très simple d'un mouvement large M valide par rapport à un état valide S de Q :

- Si $S = \emptyset$, alors M est valide par rapport à Q si et seulement si u_0 est de degré 1 et $M = (a_1, a_1, 1)$, ou $k > 1$ et $M = (a_2, a_1, 1)$.
- Si $S \neq \emptyset$, $M = (a_i, a_j, p)$ est valide par rapport à Q si et seulement si ou bien $i = j$, a_i occupe un sommet $u \in \delta(X_S)$, et p est le seul port contaminé du sommet u, ou bien $i \neq j$, a_i est libre, a_j occupe un sommet $u \in \delta(X_S)$, et p est un port contaminé du sommet u.

La première instruction du cas $etat$ = NETTOIE consiste à tester s'il existe un mouvement large valide par rapport à Q. Le point crucial est alors de choisir quel mouvement large doit être choisi, parmi tous ceux qui sont valides. Pour déterminer ce choix, nous ordonnons les mouvements larges dans l'ordre lexicographique.

Définition 23 *Soient $M = (a_i, a_j, p)$ et $M' = (a_{i'}, a_{j'}, p')$ deux mouvements larges. Nous définissons $M \prec M'$ si et seulement si ($i < i'$), ou ($i = i'$, et $j < j'$), ou ($i = i'$, $j = j'$, et $p < p'$).*

Si il y a un mouvement large qui est valide par rapport à Q alors l'algorithme $\mathcal{A}$ choisit celui qui est le plus petit dans l'ordre lexicographique parmi tous ceux qui sont valides par rapport à Q. S'il n'existe pas de mouvement large valide par rapport à Q, alors $\mathcal{A}$ passe dans l'état RETOUR. Pour cela, le dernier mouvement de Q est supprimé et stocké dans la variable globale M_{last}. Si $Q = \emptyset$, cette suppression n'est pas possible, et $\mathcal{A}$ décide que k chercheurs ne sont pas suffisants pour nettoyer le réseau.

Concentrons nous sur le cas $etat$ = RETOUR. $\mathcal{A}$ considère le dernier mouvement M_{last}. Si il existe un mouvement large $M \succ M_{last}$ valide par rapport à l'état de la pile, alors $\mathcal{A}$ effectue le plus petit de ces mouvements, en l'empilant sur la pile, et il passe dans l'état NETTOIE. Sinon $\mathcal{A}$ recommence à supprimer le mouvement large situé en haut de la pile et à le stocker dans M_{last}.

5.3.2 Correction de l'algorithme $\mathcal{A}$

Dans cette section, nous prouvons que l'algorithme $\mathcal{A}$ calcule une stratégie de capture connexe monotone pour G, partant de u_0, et utilisant au plus $mcs(G)$ chercheurs. Plus précisément, nous prouvons le théorème suivant :

Théorème 26 *L'algorithme $\mathcal{A}$ termine pour $k = mcs(G, u_0)$. Lorsque $\mathcal{A}$ termine, la pile Q décrit une stratégie de capture connexe monotone partant de u_0 et utilisant k chercheurs.*

Entrée : $k \geq 1$ chercheurs $a_1, a_2, \cdots, a_k$ et un sommet u_0 d'un graphe G.
    a_1 est placé sur u_0 ; $a_2, \cdots, a_k$ sont libres.
Sortie : un booléen *possible*, et une pile Q de mouvements larges.
début
  $Q \leftarrow \emptyset$;
  etat ← Nettoie ;
  decide ← *faux* ;
  tant que *decide* est faux **faire**
    si tous les chercheurs sont libres **alors**
      decide ← *vrai* ;
      possible ← *vrai* ;
    sinon
      /* cas *etat* = Nettoie */
      si *etat* = Nettoie **alors**
        si il existe un mouvement large valide par rapport à Q **alors**
          $(a_i, a_j, p) \leftarrow$ le plus petit mouvement large valide par rapport à Q ;
          empile(a_i, a_j, p) ;
        sinon
          si $Q \neq \emptyset$ **alors**
            $M_{last} \leftarrow$ *depile*() ;
            etat ← Retour ;
          sinon
            decide ← *vrai* ;
            possible ← *faux* ;
          fin si
        fin si
      /* cas *etat* = Retour */
      sinon
        Soit $M_{last} = (a_i, a_j, p)$;
        si il existe un mouvement large valide par rapport à Q plus grand que (a_i, a_j, p) **alors**
          $(a'_i, a'_j, p') \leftarrow$ le plus petit mouvement large valide par rapport à Q, et
            plus grand que (a_i, a_j, p) ;
          empile(a'_i, a'_j, p') ;
          etat ← Nettoie ;
        sinon
          si $Q \neq \emptyset$ **alors** $M_{last} \leftarrow$ *depile*() ;
          sinon
            decide ← *vrai* ;
            possible ← *faux* ;
          fin si
        fin si
      fin si
    fin si
  fin tant que
  renvoie (*possible*, Q) ;
fin.

Fig. 5.1 – Algorithme $\mathcal{A}$

La validité de ce théorème découle facilement des cinq lemmes suivants.

Lemme 12 *Après chaque exécution de la boucle* tant-que *de l'algorithme $\mathcal{A}$, l'état de la pile est valide.*

Preuve. Initialement, la pile est vide, et correspond à la stratégie telle que a_1 occupe le sommet u_0, et donc la pile est valide. Supposons que l'état de Q avant d'exécuter la boucle *tant-que* est valide, et considérons l'état de la pile Q après la boucle. Deux cas doivent être considérés selon qu'un mouvement large est empilé ou dépilé. Ces deux cas ne dépendent pas de la valeur de *etat*. Le résultat d'une insertion est valide car seul un mouvement large valide par rapport à Q peut être empilé dans Q. Le résultat d'une suppression est aussi valide puisqu'il correspond à une stratégie partielle décrite par Q avant la boucle, moins le mouvement large qui vient d'être supprimé. □

Le lemme suivant nécessite d'ordonner les états de la pile de la même manière que nous avons ordonné les mouvements larges.

Définition 24 *Etant donné deux états de la pile Q, $S = M_1|\cdots|M_r$ et $S' = M'_1|\cdots|M'_{r'}$, nous définissons $S \prec S'$ si et seulement si il existe $i \leq \min\{r, r'\}$ tel que $M_i \prec M'_i$ et, pour tout $j < i$, $M_j = M'_j$.*

L'ordre sur les piles défini ci-dessus est un ordre total. Puisque le mouvement large empilé sur la pile dans le cas NETTOIE de l'algorithme $\mathcal{A}$ est le plus petit mouvement par rapport à l'état courant de la pile, nous obtenons que la séquence des piles construites par l'algorithme $\mathcal{A}$ respecte cet ordre total. Plus précisément :

Lemme 13 *L'ordre dans lequel les piles valides sont construites par l'algorithme $\mathcal{A}$ est compatible avec l'ordre total de la définition 24, dans le sens où, lors de la première exécution de la boucle* tant que *à laquelle un certain état S apparaît, tous les états valides $S' \prec S$ sont apparus auparavant, et aucun état valide $S'' \succ S$ n'est encore apparu.*

Nous disons qu'une séquence valide de mouvements larges est *complète* si la stratégie de capture correspondante nettoie complètement le réseau. Le lemme suivant découle directement du lemme 13

Lemme 14 *Soit $S = M_1|\ldots|M_r$ une séquence de mouvements larges correspondant à l'exécution partielle d'une stratégie de capture utilisant au plus k chercheurs. Soit il existe une séquence complète S' de mouvements larges avec $S' \prec S$, soit l'algorithme $\mathcal{A}$ est tel qu'à un moment de son exécution S est l'état de la pile.*

Lemme 15 *Si $mcs(G, u_0) > k$, alors l'algorithme $\mathcal{A}$ renvoie $(faux, \emptyset)$ pour k.*

Preuve. Soit S la plus grande séquence valide de mouvements larges selon l'ordre de la définition 24. Puisque le graphe ne peut être nettoyé par k chercheurs partant de u_0, pour toute séquence valide $S' \preceq S$, S' n'est pas complète. D'après le lemme 14, l'algorithme $\mathcal{A}$ finit par atteindre l'état S de la pile. Après cette étape, l'algorithme reste toujours dans l'état RETOUR et supprime successivement tous les mouvements larges de la pile. Ainsi, il atteint la situation $Q = \emptyset$ telle qu'il n'existe plus de mouvements larges. Alors, l'algorithme $\mathcal{A}$ renvoie $(faux, \emptyset)$. □

Lemme 16 *Supposons que $mcs(G, u_0) = k$. Soit S la plus petite séquence complète de mouvements larges correspondant à une stratégie de capture connexe partant de u_0. L'algorithme $\mathcal{A}$ renvoie $(vrai, Q)$ pour k, avec Q dans l'état S.*

Preuve. D'après le lemme 14, puisque S est la plus petite séquence complète de mouvements larges valides, l'algorithme $\mathcal{A}$ calcule S. A cette étape de l'algorithme $\mathcal{A}$, tous les sommets du graphe sont propres. Donc, tous les chercheurs sont libres, et par conséquent l'algorithme $\mathcal{A}$ renvoie $(vrai, Q)$. □

5.4 Algorithme décentralisé

Dans cette section, nous décrivons les principales caractéristiques du protocole `Nettoyage_réparti`. Dans cette description, nous supposons que les chercheurs sont capables de communiquer en échangeant des messages de taille $O(\log k)$ bits où k est le nombre de chercheurs impliqués dans la stratégie. En utilisant ces moyens de communications, nous allons montré que `Nettoyage_réparti` capture l'intrus avec $mcs(G, u_0)$ chercheurs. L'utilisation d'un chercheur supplémentaire sert à implémenter les communications entre les $mcs(G, u_0)$ autres chercheurs. Ainsi, `Nettoyage_réparti` capture l'intrus avec $mcs(G, u_0) + 1$ chercheurs. Nous supposons ici que les chercheurs peuvent communiquer grâce à l'échange de messages, uniquement dans le but de simplifier la présentation. Le fait qu'un chercheur supplémentaire puisse implémenter les communications entre les chercheurs apparaîtra clairement lors de la description du protocole `Nettoyage_réparti`. La principale raison pour laquelle cela peut être fait provient du fait que trouver son chemin dans la partie propre du réseau est facilement réalisable grâce aux informations stockées sur les tableaux blancs. L'émetteur d'un message est toujours le chercheur qui a réalisé la dernière action, et une action résulte toujours de la reception d'un message.

De plus, pour simplifier, nous supposons que deux chercheurs sur un même sommet peuvent se voir l'un l'autre, et peuvent échanger leurs états. Cette hypothèse n'est pas restrictive puisque cela peut être implémenté grâce aux tableaux blancs, mais cela compliquerait inutilement la présentation.

Tout d'abord, nous décrivons les structures de données utilisées par `Nettoyage_réparti`.

5.4.1 Structures de données de `Nettoyage_réparti`

Chaque chercheur a une variable d'état qui peut prendre $k + 2$ valeurs différentes où k est le nombre courant de chercheurs. Ces $k + 2$ états sont : NETTOIE, RETOUR, et (AIDE, j), pour $j = 1, \ldots, k$. Initialement, tous les chercheurs sont dans l'état NETTOIE. Au cours de l'exécution du protocole

- Un chercheur est dans l'état NETTOIE s'il vient juste de nettoyer une arête ;
- Un chercheur est dans l'état RETOUR s'il vient de se retirer d'une arête qu'il avait nettoyé au préalable (la laissant être recontaminée) ;

- Un chercheur est dans l'état (Aide, j) si il est sensé rejoindre le chercheur j pour l'aider à nettoyer le réseau (un des deux chercheurs gardera un sommet, pendant que l'autre nettoiera une arête incidente à ce sommet).

Le chercheur a_1 possède une variable booléenne supplémentaire *termine* qu'il utilise pour déterminer si le graphe est propre. Cela se produit lorsque le chercheur a_1 a essayé d'aider tous les autres chercheurs et aucun d'entre eux n'avait besoin de cette aide (tous occupaient des sommets incident à des arêtes propres uniquement).

Les messages que les chercheurs peuvent échanger sont de quatre types : `départ`, `bouge`, `aide` and `désolé`.

- `départ` est un message d'initialisation qui est seulement utilisé au démarrage du protocole `Nettoyage_réparti` (seul le chercheur a_1 reçoit ce message, au tout début de l'exécution du protocole).
- Si le chercheur a_i reçoit un message ($\texttt{bouge}, j$) du chercheur a_j alors c'est au tour du chercheur a_i d'agir. (Il apparaîtra clairement plus tard que les chercheurs s'organisent de manière à ce que exactement un chercheur agisse à la fois).
- Si le chercheur a_i reçoit un message ($\texttt{aide}, j$) du chercheur a_j, alors a_j vient juste d'arriver sur le même sommet que a_i pour aider a_i. (Notons que a_i et a_j pourraient utiliser les tableaux blancs pour communiquer, et ce type de messages est simplement utilisé dans un but d'unification avec les autres messages).
- Si le chercheur a_i a reçu un message ($\texttt{bouge}, j$) ou ($\texttt{aide}, j$) du chercheur a_j et, après avoir éventuellement effectué plusieurs actions, il se révèle que ces actions étaient inutiles, alors a_i envoie un message ($\texttt{désolé}, i$) en retour au chercheur a_j.

Le tableau blanc de chaque sommet contient une pile locale, et deux vecteurs `direction` et `port_propre`. Le protocole assure qu'après qu'un sommet ait été visité par un chercheur, `direction`[0] indique le numéro de port à prendre pour atteindre la base à partir de ce sommet, et, pour $i > 0$, `direction`[i] est le numéro de port que le chercheur a_i a emprunté pour quitter le sommet la dernière fois qu'il était présent sur ce sommet. Sur le sommet v, pour tout p, $1 \leq p \leq \deg(v)$, `port_propre`[p] $= 1$ si et seulement si l'arête correspondante au numéro de port p est propre, sinon `port_propre`[p] $= 0$.

Quand un chercheur sur un sommet v décide d'effectuer une action, il sauve une *trace* de cette action dans la pile locale. Une trace est un triplet (X, ID, x) tel que X est un symbole, ID l'identifiant d'un chercheur, et x est soit un numéro de port, soit l'identifiant d'un chercheur, selon le symbole X. Plus précisément :

- (CC, i, p) signifie que p est le seul numéro de port contaminé (C), et le chercheur a_i décide de nettoyer (C pour Clear) l'arête qui correspond à p ;
- (CJ, i, p) signifie qu'un certain chercheur a rejoint (J pour Join) a_i sur ce sommet, et a_i décide de nettoyer (C) l'arête qui correspond à p ;
- (JJ, i, j) signifie que le chercheur a_i décide de rejoindre (J) le chercheur a_j ;
- (RT, i, j) signifie que le chercheur a_i a reçu (R) un message ($\texttt{bouge}, j$) ou ($\texttt{aide}, j$) du chercheur a_j ;
- (ST, i, j) signifie que le chercheur a_i décide d'envoyer (S pour Send) un message au chercheur a_j ;
- (AC, i, p) signifie que le chercheur a_i est arrivé (A) sur v par le port p après avoir

nettoyé (C pour Clear) l'arête correspondante ;
- (AH, i, p) signifie que le chercheur a_i est arrivé (A) sur v par le port p dans le but de rejoindre un autre chercheur (H pour Help).

5.4.2 L'algorithme `Nettoyage_réparti`

Le protocole `Nettoyage_réparti` organise les mouvements des chercheurs, et les messages échangés entre eux, dans un ordre spécifique. En se reposant sur un ordre lexicographique des actions des chercheurs, `Nettoyage_réparti` les ordonne de manière à toujours exécuter la plus petite action possible. Comme pour l'algorithme $\mathcal{A}$, le principe de `Nettoyage_réparti` est d'essayer toutes les stratégies monotones et connexes possibles qui utilisent k chercheurs, jusqu'à ce que le graphe sont complètement propre, ou qu'aucun chercheur ne puisse plus bouger sans entraîner de recontamination. Dans le dernier cas, le chercheur qui a effectué le dernier mouvement revient sur ces pas, et `Nettoyage_réparti` essaie l'action suivante dans l'ordre lexicographique sur les actions.

Le fait que `Nettoyage_réparti` termine est prouvé de la manière suivante. Le graphe est nettoyé à l'étape t si et seulement si tous les chercheurs occupent des sommets propres à cette étape, c'est-à-dire des sommets dont toutes les arêtes incidentes sont propres. Cette configuration est identifiée par les chercheurs lorsque a_1 essaie d'aider chacun des autres chercheurs, de a_2 à a_k, mais aucun d'entre eux n'a besoin d'aide. Réciproquement, les chercheurs identifient le fait que k chercheurs ne sont pas suffisants pour nettoyer le graphe lorsqu'ils occupent tous la base, et le protocole essaie de supprimer le mouvement au sommet de la pile locale, alors que celle-ci est vide. Dans ce cas, a_1 appelle un nouveau chercheur, et les $k+1$ chercheurs sont prêts à tenter de capturer le fugitif en partant de la base.

Le squelette du protocole `Nettoyage_réparti` est présenté sur les figures 5.2-5.4. Plus précisément, la figure 5.2 décrit le comportement global d'un chercheur, utilisant les sous-protocoles décrits sur les figures 5.3-5.4. Un chercheur agit soit à la réception d'un message (cf. partie gauche de la figure 5.2), ou à son arrivée sur un sommet (cf. partie droite de la figure 5.2). Le message de type `départ` est uniquement utilisé au cours de l'initialisation : initialement, le chercheur a_1 reçoit un message `départ` (et appelle alors la procédure $decide()$).

Nous décrivons maintenant le protocole `Nettoyage_réparti` comme il est décrit sur la figure 5.2

Si le chercheur a_i reçoit un message $(\texttt{bouge}, j)$, alors, par définition d'un tel message, cela signifie simplement que c'est au tour de a_i d'agir. Par conséquent, a_i écrit sur le tableau blanc du sommet qu'il occupe, qu'il vient de recevoir un message du chercheur a_j lui ordonnant d'agir. Dans ce but, a_i empile (RT, i, j) sur la pile locale. La nature des prochaines actions réalisées par a_i dépend du résultat de la procédure $decide()$. Avant de décrire cette procédure, listons les autres cas, qui dépendent du message reçu par a_i. Si a_i reçoit un message $(\texttt{aide}, j)$ alors, cela signifie que a_j vient juste d'arriver sur le même sommet que a_i pour l'aider. Donc, a_i empile (RT, i, j) sur la pile locale, et nettoie l'arête ayant le plus petit numéro de port p parmi toutes les arêtes contaminées incidentes au

```
Programme d'un chercheur i sur le sommet v.

début
/* Le chercheur i reçoit un message */
Cas :
   message = départ
      decide() ;
   message = (bouge, j)
      empile(RT, i, j) ;
      decide() ;
   message = (aide, j)
      empile(RT, i, j) ;
      p ← plus petit numéro de port contaminé ;
      nettoie(CJ, i, p)
   message = (désolé, j)
      retour() ;

/* Le chercheur i arrive sur le sommet v par le port p */
Cas :
   état = NETTOIE
      si aucun autre chercheur n'est présent
        sur v alors
          effacer le tableau blanc ;
      direction[0] ← p ;
      port_propre[p] ← 1 ;
      empile(AC, i, p) ;
      sinon si i ≠ 1 alors
         empile(ST, i, 1) ;
         envoyer le message (bouge, i) to 1 ;
      sinon decide() ;
   état = (AIDE, j)
      empile(AH, i, p) ;
      rejoins(j) ;
   état = RETOUR
      retour() ;
fin
```

FIG. 5.2 – Squelette du protocole Nettoyage_réparti

sommet occupé par a_i. Cette action est réalisée en appelant la procédure *nettoie*(CJ, i, p). Finalement, si a_i reçoit un message (désolé, j), alors, cela signifie que a_i vient d'envoyer un message (bouge, i) ou un message (aide, i) à a_j mais a_j ne pouvait rien faire, ou toutes les actions que a_j a tentées ont été annulée (a_i est revenu sur ses pas). Par conséquent, a_i appelle la procédure *retour*() pour déterminer quel autre chercheur il doit essayer d'aider.

L'action d'un chercheur a_i arrivant sur un sommet v par le port p dépend de son état. Dans l'état (AIDE, j), a_i doit rejoindre a_j pour l'aider à nettoyer le réseau. Ainsi, a_i empile (AH, i, p) sur la pile locale pour indiquer qu'il est arrivé par le port p dans le but de rejoindre un autre chercheur, et il appelle la procédure *rejoins*() pour déterminer quoi faire pour rejoindre a_j. Dans l'état RETOUR, a_i appelle simplement la procédure *retour*() pour poursuivre l'annulation des actions préalables (pour revenir encore une fois sur ses pas). Le cas où a_i arrive sur un sommet v dans l'état NETTOIE est plus évolué. Si il n'y a aucun autre chercheur sur v alors a_i efface le tableau blanc puisque il était accessible par l'intrus, et donc son contenu n'a aucun sens (quand un chercheur efface un tableau blanc, il réinitialise toutes les variables locales à 0, et la pile locale à $\emptyset$). Alors a_i met direction[0] à la valeur p pour indiquer qu'il est arrivé par le port p, et met port_propre[p] à 1 pour indiquer que l'arête correspondant au port p est propre. a_i empile alors (AC, i, p) dans la pile locale de v pour indiquer que a_i est arrivé sur v par le port p après avoir nettoyé l'arête correspondante. A cette étape, le comportement de a_i dépend de si $i = 1$ ou non. Alors que a_1 appelle simplement *decide*() pour déterminer quoi faire ensuite, a_i $(i > 1)$

propose à a_1 d'agir ensuite. Pour cela, a_i envoie un message (bouge, i) à a_1. Bien sur, pour garder une trace de cette action, a_i empile $(ST, i, 1)$ dans la pile locale.

Remarque. Avant de détailler les procédures mentionnées ci-dessus, notons que les actions sont ordonnées. Par exemple, si plusieurs arêtes incidentes peuvent être nettoyées, celle qui l'est effectivement, est celle avec le plus petit numéro de port. De même, après le nettoyage d'une arête, a_i propose au chercheur avec le plus petit identifiant a_1 d'agir ensuite. Comme nous le verrons dans le détail des procédures *decide*() et *retour*(), le protocole `Nettoyage_réparti` essaie toujours de réaliser la plus petite action. C'est, entre autre, le rôle de la procédure *prochain_chercheur* décrite sur le côté droit de la figure 5.3.

Procédure *prochain_chercheur*

La procédure *prochain_chercheur* vise à déterminer quel chercheur a_j va être le prochain à agir. Dans le cas où a_i est le chercheur avec le plus petit identifiant occupant le sommet, $j = i+1$. Sinon, c'est-à-dire a_i n'est pas le chercheur avec le plus petit identifiant occupant le sommet, j est le plus petit identifiant supérieur à i tel que a_j n'occupe pas le même sommet que a_i. Lorsque j est déterminé, a_i propose à a_j d'agir ensuite, en lui envoyant un message (bouge, i). Comme toujours, une trace de cette action est conservée sur le sommet courant en empilant (ST, i, j) sur la pile locale. Si il n'y a aucun a_j avec $j > i$ qui n'occupe pas le même sommet que a_i, alors a_i appelle *retour*() dans le but de revenir en arrière.

Les procédures *nettoie*() et *bouge*() décrites sur le côté gauche de la figure 5.3 exécute le nettoyage d'une arête, et la traversée d'une arête, respectivement. (Bien sur, le nettoyage d'une arête nécessite sa traversée).

```
nettoie(action X, identifiant i, port p)
/* X ∈ {CC;CJ} */
début
    empile(X, i, p) ;
    port_propre[p] ← 1 ;
    etat ← NETTOIE ;
    bouge(p, i) ;
fin

bouge(port_number p, identifiant i)
début
    direction[i] ← p ;
    quitter le sommet courant par le port p ;
fin
```

```
prochain_chercheur (identifiant i)
début
    j ← i + 1 ;
    si i n'est pas le chercheur avec le plus petit ID
      sur le sommet v alors
       tant que (j est sur le sommet v)
         et (j ≤ k) faire
          j ← j + 1 ;
    si j ≤ k alors
       empile(ST, i, j) ;
       envoyer (bouge, i) to j ;
    sinon
       retour()
fin
```

FIG. 5.3 – Procédures `nettoie`, `prochain_chercheur` and `bouge`.

```
retour()
début
    etat ← RETOUR ;
    msg ← depile() ;
    cas :
       msg = (RT, i, j)
          envoyer (désolé, i) à j ;
       msg = (JJ, i, j)
          si (i = k and j = i − 1) alors retour() ;
          sinon si (i ≠ k and j = k) alors
                prochain_chercheur(i) ;
             sinon
                si j + 1 ≠ i alors ℓ ← j + 1 ;
                   sinon ℓ ← j + 2 ;
                empile(JJ, i, ℓ) ;
                rejoins(ℓ) ;
       msg = (CC, i, p)
          port_propre[p] ← 0 ;
          si i = k alors retour()
          sinon prochain_chercheur(i) ;
       msg = (CJ, i, p)
          port_propre[p] ← 0 ;
          si ∃q le plus petit port contaminé
          tel que q > p alors clean_edge(CJ, i, q) ;
          sinon
             msg2 ← depile() ;
             si msg2 = (AH, i, p) alors
                bouge(p, i) ;
             sinon msg2 = (RT, i, j) alors
                envoyer (désolé, i) to j ;
       msg = (AC, i, p)
          port_propre[p] ← 0 ;
          bouge(p, i) ;
       msg = (AH, i, p)
          bouge(p, i) ;
       msg = (ST, i, j)
          retour() ;
       msg = ∅
          k ← k + 1 ;
          initialisation(k) ;
fin
```

```
decide()
début
    si le sommet v est propre
    ou il y a un autre chercheur d'ID ℓ < i sur v
      alors
       si i = 1 alors
          j ← 2 ;
          termine ← vrai ;
       sinon j ← 1 ;
       empile(JJ, i, j) ;
       rejoins(j) ;
    sinon si ∃ un unique port contaminé p alors
          nettoie(CC, i, p) ;
       sinon si i ≠ k alors
             prochain_chercheur(i) ;
          sinon retour() ;
fin

rejoins(identifiant j)
début
    etat ← (AIDE, j) ;
    si j est présent sur v alors
       si v est propre alors
          si i = 1 et termine et j = k
             alors "Le graphe est propre" ;
          sinon retour() ;
       sinon
          si i = 1 alors termine ← faux ;
          Soit q le plus petit port contaminé ;
          si j < i alors
             empile(ST, i, j) ;
             envoyer (aide, i) à j ;
          sinon
             nettoie(CJ, i, q) ;
    sinon
       p ← direction[j] ;
       si p = 0 faire p ← direction[0] ;
       bouge(p, i) ;
fin
```

FIG. 5.4 – Procédures `retour`, `decide`, and `rejoins`.

Procédure *nettoie*() **:**

Le chercheur a_i exécutant la procédure $nettoie(X, i, p)$ empile d'abord la trace (X, i, p) sur la pile locale, met `port_propre`$[p]$ à 1 pour spécifier que l'arête correspondant au port p est propre, passe dans l'état NETTOIE, et finalement quitte le sommet par le port p pour nettoyer l'arête correspondante.

Procédure *bouge*() **:**

Le chercheur a_i exécutant la procédure $bouge(p, i)$ quitte simplement le sommet courant par le sommet p. Mais avant cela, il met `direction`$[i] = p$ pour spécifier que, dans le but de joindre a_i à partir de ce sommet, il faudra emprunter le port p.

Nous décrivons maintenant les procédures *decide*(), *retour*(), et *rejoins*() détaillés sur la figure 5.4.

Procédure *decide*()

La procédure *decide*() est appelée sur un sommet lorsque le chercheur concerné doit décider quelle est la prochaine action qu'il doit effectuer. Soit v le sommet sur lequel le chercheur a_i applique *decide*().

Si le sommet v est propre, ou au moins un autre chercheur a_ℓ, $\ell < i$, occupe v, alors a_i n'a pas besoin de garder le sommet v. Donc a_i essaie d'aider un autre chercheur. Selon l'ordre défini précédemment, a_i essaie d'aider le chercheur avec le plus petit identifiant. Alors, a_i applique $rejoins(2)$ si $i = 1$, et $rejoins(1)$ sinon. (La variable booléenne *termine* de a_1 est mise à *vrai* si $i = 1$; rappelons que cette variable est utilisée pour détecter la terminaison du protocole `Nettoyage_réparti`).

Si il y a une seule arête incidente à v contaminée, alors le chercheur a_i la nettoie en appliquant la procédure *clear_edge*.

Sinon (c'est-à-dire a_i est le chercheur avec le plus petit identifiant qui occupe le sommet v, et v est incident à plus d'une arête contaminée), a_i ne peut pas bouger puisque le protocole assure que le chercheur avec le plus petit identifiant sur un sommet le préserve de la recontamination. Par conséquent, si $i = k$ (c'est-à-dire tous les chercheurs ont essayé de progresser, mais aucun d'eux ne le peut) alors le chercheur a_i applique *retour*() dans le but de revenir en arrière. Sinon (si $i < k$) a_i applique $prochain\_chercheur(i)$ pour donner à un autre chercheur une chance de progresser.

Procédure *retour*()

La procédure *retour*() est appelée dans le but de revenir en arrière, ce qui entraîne parfois la recontamination de certaines arêtes. Soit v un sommet où le chercheur a_i applique la procédure *retour*(). Le chercheur a_i commence par passer dans l'état RETOUR, et dépile le sommet de la pile locale qu'il stocke dans la variable locale *msg*. Le comportement de a_i dépend alors de *msg*, ce qui mène à huit cas. Ces huit cas correspondent aux différentes traces qui peuvent se trouver en haut de la pile.

- Cas $msg = (RT, i, j)$: cela signifie que le chercheur a_j a envoyé un message à a_i pour lui donner une chance de progresser. Puisque a_i applique $retour()$, cela signifie que a_i ne peut en fait effectuer aucune action (notons cependant que a_i pourrait avoir effectué une action entre temps, et l'avoir annulée depuis). Donc, a_i envoie le message $(\texttt{désolé}, i)$ à a_j dans le but de décliner "l'invitation" de a_j, et lui laisser la possibilité d'effectuer une autre action.
- Cas $msg = (JJ, i, j)$: cela signifie que, lors d'une étape précédente de la stratégie, le chercheur a_i, occupant le sommet v, a décidé d'aider le chercheur a_j. Puisque a_i applique $retour()$, cela signifie que sa tentative d'aider a_j n'a pas abouti. Plusieurs situations doivent être prises en compte :
  Si il y a un autre chercheur que a_i n'a pas encore essayé d'aider ($j < k$ et $i \neq k$, ou, $i = k$ et $j < k - 1$), alors a_i essaie d'aider parmi ceux-ci le chercheur avec le plus petit identifiant (désigné par a_ℓ), en appliquant $rejoins(\ell)$.
  Sinon, si $i = k$ (c'est-à-dire tous les chercheurs ont essayé de progresser, mais aucun d'entre eux n'a réussi) alors, le chercheur a_i applique encore $retour()$ pour revenir en arrière de nouveau. Si $i < k$, le chercheur a_i applique $prochain\_chercheur(i)$ pour donner à un autre chercheur une chance de progresser.
- Cas $msg = (CC, i, p)$: cela signifie que a_i est le chercheur avec le plus petit identifiant sur le sommet v, et v est incident à une unique arête contaminée, associée port p. Puisque, a_i applique $retour()$, cela signifie que a_i vient juste d'annuler le nettoyage d'une arête, la laissant être recontaminée. Ainsi, a_i ne peut faire autre chose. Donc, soit $i = k$ (c'est-à-dire tous les chercheurs ont essayé de progresser, mais aucun d'eux n'a réussi) et donc le chercheur a_i applique encore $retour()$ pour revenir en arrière de nouveau, soit $i < k$ et le chercheur a_i applique $prochain\_chercheur(i)$ pour donner à un autre chercheur une chance de progresser.
- Cas $msg = (CJ, i, p)$: cela signifie que a_i vient juste d'annuler le nettoyage de l'arête correspondant au numéro de port p, la laissant être recontaminée. De plus, ce nettoyage implique un autre chercheur a_j (avec $j > i$). Deux cas sont possibles selon que le chercheur a_i est arrivé de v pour aider le chercheur a_j ou inversement. Le premier cas sera désigné comme le cas 1, et le second comme le cas 2.
  Si il y a une arête que a_i n'a pas encore essayée de nettoyer (c'est-à-dire une arête contaminée ayant pour numéro de port $q > p$), alors, le chercheur a_i applique la procédure $nettoie(CJ, i, q)$ pour nettoyer cette arête (CJ indique qu'un tel mouvement est possible grâce à la présence d'un autre chercheur sur v).
  Sinon, p est le numéro de port le plus grand correspondant à une arête contaminée. Par conséquent, dans le cas 1, le chercheur a_i a essayé d'aider a_j (resp., dans le cas 2, a_j a essayé d'aider a_i) sans succès. Dans les deux cas, a_i doit encore revenir en arrière, et donc, il dépile le sommet de la pile locale qu'il associe à la variable $msg2$. Si $msg2 = (AH, i, q)$, nous sommes dans le cas 1, et donc, le chercheur a_i retraverse l'arête d'où il est venu (i.e., l'arête de numéro de port q). Si $msg2 \neq (AH, i, q)$, le seul cas possible est $msg2 = (RT, i, j)$, qui correspond au cas 2. C'est-à-dire que le chercheur a_j est venu sur v pour aider le chercheur a_i, et, puisque $i < j$, le chercheur a_j a envoyé le message $(\texttt{aide}, j)$ à a_i (cf. Procédure $rejoins()$). Dans ce dernier cas, le

chercheur a_i informe le chercheur a_j que son aide n'a pas été fructueuse, en envoyant le message $(\texttt{désolé}, i)$ à a_j.

- Cas $msg = (AC, i, p)$: cela signifie que a_i est venu sur ce sommet par l'arête de numéro de port p, après avoir nettoyé cette arête. Puisque le chercheur a_i applique $retour()$, a_i revient sur ses pas, c'est-à-dire retourne par cette arête, la laissant être recontaminée.
- Cas $msg = (AH, i, p)$: cela signifie que a_i est venu sur ce sommet par l'arête de numéro de port p, dans le but d'aider un chercheur (i.e., cette arête était déjà propre). Puisque le chercheur a_i applique $retour()$, a_i revient sur ses pas, en partant par l'arête d'où il vient.
- Cas $msg = (ST, i, j)$: cela signifie que a_i a envoyé un message à le chercheur a_j, et que a_j vient d'envoyer à a_i le message $(\texttt{désolé}, j)$, signifiant que a_j ne peut plus rien faire. Donc, a_i applique $retour()$ pour revenir en arrière.
- Cas $msg = \emptyset$: cela signifie que toutes les actions que les chercheurs ont effectuées auparavant ont été annulées. Notons que seul le chercheur a_1 peut se trouver dans une telle situation. Puisqu'il est dans l'état RETOUR, cela signifie que toutes les stratégies utilisant k chercheurs ont été testées sans succès. Donc le protocole recommence avec un chercheur de plus.

Procédure $rejoins()$

Soit v un sommet où le chercheur a_i applique $rejoins(j)$. Appliquer cette procédure signifie que le chercheur a_i a décidé d'aider le chercheur a_j. Premièrement, a_i passe dans l'état (AIDE, j).

Si a_j occupe v, alors le comportement de a_i dépend de si v est propre ou non. Si v est propre, $i = 1$, $termine$ est vrai, et $j = k$, alors le chercheur a_1 a essayé d'aider tous les chercheurs mais aucun d'eux n'avait besoin de son aide. Donc le graphe est propre. Sinon, toujours sous l'hypothèse selon laquelle v est propre, a_j n'a pas besoin d'aide. Donc, le chercheur a_i annule sa tentative d'aide de a_j en appliquant $retour()$. Le dernier sous-cas se produit lorsque a_j occupe un sommet v qui n'est pas propre. Dans ce cas, le chercheur de plus petit identifiant entre a_i et a_j doit nettoyer l'arête contaminée avec le plus petit numéro de port (disons q) incidente à v. Si $i < j$, alors le chercheur a_i applique $nettoie(CJ, i, q)$ pour nettoyer l'arête (CJ signifie que le nettoyage peut être réalisé grâce à la présence d'un autre chercheur). Si $i > j$, alors le chercheur a_i envoie $(\texttt{aide}, i)$ à a_j, pour indiquer au chercheur a_j qu'il peut nettoyer une arête grâce à la présence de a_i.

Si a_j n'occupe pas v, alors a_i essaie de rejoindre le chercheur a_j en le suivant (si a_j a déjà visité le sommet v), ou en allant à la base, ce qui est possible grâce au vecteur local `direction`. La procédure $rejoins()$ utilise les indications présentes sur les tableaux blancs. Rappelons que si a_j était sur un sommet, le tableau blanc contient, dans $\texttt{direction}[j]$, le numéro de port par lequel a_j a quitté ce sommet. Le chercheur a_i retourne à la base en utilisant $\texttt{direction}[0]$ jusqu'à ce qu'il croise un sommet où $\texttt{direction}[j]$ est défini, auquel cas a_i commence à suivre cette direction pour trouver a_j.

5.5 Correction de Nettoyage_réparti

A chaque étape de Nettoyage_réparti, une seule opération est réalisée, sur une unique des piles distribuées sur tous les sommets du réseau. En effet, seul le chercheur qui vient de recevoir un message peut réaliser une action, et en particulier modifier une pile. Donc nous pouvons définir une *pile virtuelle* centralisée, $Q_{virtual}$, où nous empilons ou dépilons tous les mouvements effectués par les chercheurs , en même temps qu'ils sont empilés ou dépilés sur les piles distribuées. Précisément, un *mouvement* est une paire $(a_i \to a_j, p)$, qui peut être interprétée comme suit.

- Si $i \neq j$, alors $(a_i \to a_j, p)$ signifie que a_i quitte sa position courante par le port p dans le but de rejoindre a_j ;
- Le mouvement $(a_i \to a_i, p)$ signifie que a_i quitte sa position courante par le port p, en nettoyant l'arête correspondante.

Un mouvement large correspond à une séquence de mouvements. De l'interprétation ci-dessus, le mouvement large (a_i, a_i, p) est *équivalent* au mouvement $(a_i \to a_i, p)$, et si $i \neq j$ le mouvement large (a_i, a_j, p) est équivalent à la séquence de mouvements

$$(a_i \to a_j, p_1), (a_i \to a_j, p_2), \ldots, (a_i \to a_j, p_\ell), (\min\{a_i, a_j\} \to \min\{a_i, a_j\}, p)$$

où $p_1, \ldots, p_\ell$ est la séquence de numéros de port correspondant à un chemin (dans la partie propre du graphe) entre le sommet occupé par a_i et le sommet occupé par a_j au moment où le mouvement large (a_i, a_j, p) est considéré.

$Q_{virtual}$ est mis à jour de la manière suivante. A chaque exécution de la procédure $bouge()$, nous empilons ou dépilons un mouvement dans $Q_{virtual}$ comme suit. Si a_i applique $bouge(p, i)$ pendant l'exécution de la procédure $nettoie(X, i, p)$, alors le mouvement $(a_i \to a_i, p)$ est empilé sur $Q_{virtual}$. Si a_i applique $bouge(p, i)$ pendant l'exécution de la procédure $rejoins(j)$, alors le mouvement $(a_i \to a_j, p)$ est empilé sur $Q_{virtual}$, où p est le numéro de port emprunté pendant l'exécution de $rejoins()$, avant d'appeler la procédure $bouge()$. Finalement, si le chercheur applique $bouge(p, i)$ pendant l'exécution de la procédure $retour()$, alors $Q_{virtual}$ est dépilé.

Avec cette définition de $Q_{virtual}$, nous montrons que la pile Q de l'algorithme centralisé $\mathcal{A}$, et la pile virtuelle $Q_{virtual}$ sont équivalentes au sens suivant. Soit $Q = M_1|\cdots|M_r$ une séquence valide (éventuellement vide) de mouvements larges. Nous définissons les notions suivantes :

- $Q_{virtual}$ est *fortement équivalente* à Q si $Q_{virtual} = S_1|\cdots|S_r$ tel que, pour tout j, $1 \leq j \leq r$, S_j est une séquence de mouvements équivalente à M_j.
- $Q_{virtual}$ est *faiblement équivalente* à Q si $Q_{virtual} = S_1|\cdots|S_r|S_{r+1}$ tel que, pour tout j, $1 \leq j \leq r$, S_j est une séquence de mouvements équivalente à M_j, et $S_{r+1} = (a_i \to a_{i'}, p_1), (a_i \to a_{i'}, p_2), \ldots, (a_i \to a_{i'}, p_\ell)$ où $p_1, \cdots, p_\ell$ est la séquence de numéros de port correspondant à un chemin entre le sommet occupé par a_i et le sommet occupé par $a_{i'}$ dans la partie propre du graphe (dans la configuration associée à Q dans l'état $M_1|\cdots|M_r$).

Il est facile de vérifier que deux piles fortement équivalentes correspondent exactement à la même stratégie (c'est-à-dire à la fin des deux stratégies, l'ensemble des arêtes propres et la positions des chercheurs sont les mêmes). Si Q et $Q_{virtual}$ sont seulement faiblement équivalentes, alors la stratégie associée à $Q_{virtual}$ consiste à réaliser la stratégie associée à Q, puis à déplacer un chercheur occupant d'un sommet à un autre sommet occupé par un autre chercheur (le déplacement étant effectué dans la partie propre du graphe, et sans recontamination). Nous verrons plus tard pourquoi cette version plus faible de l'équivalence est importante dans notre démonstration. Nous dirons que les deux piles $Q_{virtual}$ et Q sont *équivalentes* si elles sont soit fortement équivalentes, soit faiblement équivalentes.

La preuve de `Nettoyage_réparti` se fait en considérant l'algorithme étape après étape, où une *étape* est un moment de l'exécution où une arête est soit nettoyée, soit recontaminée. C'est-à-dire, une étape de `Nettoyage_réparti` désigne une étape de son exécution où un mouvement du type $(a_i \to a_i, p)$ est empilé ou dépilé sur $Q_{virtual}$.

Formellement, nous prouvons que, pour tout $t \geq 0$, la pile virtuelle $Q_{virtual}$ après l'étape t de `Nettoyage_réparti` est équivalente à la pile Q construite par $\mathcal{A}$. En d'autre termes, nous prouvons que, à chaque étape $t \geq 0$, les deux algorithmes construisent la même stratégie partielle. C'est-à-dire, à chaque étape, la sous-graphe propre et les positions des chercheurs qui gardent la frontière de ce sous-graphe propre sont identiques pour les deux stratégies. Simultanément, nous prouvons que pour toute étape, lorsqu'un mouvement large est dépilé par $\mathcal{A}$, toutes les traces de la séquence équivalente de mouvements dans `Nettoyage_réparti` sont supprimés des tableaux blancs distribués.

Notre preuve est par induction sur le nombre d'étapes. Nous supposons que la pile centralisée Q et la pile virtuelle $Q_{virtual}$ sont équivalentes à l'étape t, et nous considérons l'étape suivante pour prouver qu'elles restent équivalentes. La difficulté de la preuve est due au nombre de cas différents à considérer. Il y a exactement quatorze cas à prendre en compte, regroupés en deux groupes :

- Groupe A : Q et $Q_{virtual}$ viennent de nettoyer une arête e. Le premier cas correspond au nettoyage complet du graphe. Trois autre cas doivent être considérés : (1) un chercheur peut nettoyer une nouvelle arête, ou (2) un chercheur peut joindre un autre chercheur et l'un des deux peut nettoyer une nouvelle arête, ou (3) aucune autre arête ne peut être nettoyée et le nettoyage de e doit être annulé. Ce dernier cas doit être combiné avec trois autres cas qui dépendent de la manière dont e a été nettoyée. Donc le groupe A contient 7 cas au total.
- Groupe B : Q et $Q_{virtual}$ viennent de laisser une arête être recontaminée. Alors, soit une autre arête e peut être nettoyée, ou aucune autre arête ne peut être nettoyée (et le dernier nettoyage d'arête, disons l'arête e', doit être annulé). Dans le premier cas, il y a trois sous-cas, qui dépendent du type de mouvement qui doit être dépilé (i.e., le mouvement qui doit être annulé). Dans le second cas, il y a quatre sous-cas qui dépendent de la manière dont e' a été nettoyée. Donc le groupe B contient 7 cas.

La preuve consiste en une analyse attentive de chacun des 14 cas. Avant d'analyser ces 14 cas, nous prouvons que les piles calculées à la première étape des deux algorithmes sont équivalentes. Initialement, Q et $Q_{virtual}$ sont vides. Dans `Nettoyage_réparti`, a_1 exécute la fonction *decide*().

- Si $deg(u_0) = 1$, alors l'algorithme `Nettoyage_réparti` empile $(CC, 1, 1)$ et $(AC, 1, p)$ sur les tableaux blancs répartis, alors que $(a_1 \rightarrow a_1, 1)$ est empilé sur $Q_{virtual}$. Après la première exécution de la boucle *tant que* de l'algorithme $\mathcal{A}$, puisque $\deg(u_0) = 1$, $Q = ((a_1, a_1, 1))$. De plus, dans les deux cas, le sous-graphe propre est l'arête (u_0, w) incidente à u_0 avec a_1 sur le sommet w, et tous les autres chercheurs sont sur le sommet u_0.
- Si $deg(u_0) > 1$ et $k = 1$, alors les deux algorithmes appellent un chercheur supplémentaire. Les deux piles restent vide et seul u_0 est propre et occupé par deux chercheurs.
- Si $deg(u_0) > 1$ et $k > 1$, alors l'algorithme `Nettoyage_réparti` empile $(ST, 1, 2)$, $(RT, 2, 1)$, $(JJ, 2, 1)$, $(ST, 2, 1)$, $(RT, 1, 2)$, $(CC, 1, 1)$ et $(AC, 1, p)$ sur les tableaux blancs répartis, alors que $(a_1 \rightarrow a_1, 1)$ est empilé sur $Q_{virtual}$. L'algorithme $\mathcal{A}$ empile $(a_2, a_1, 1)$ sur Q. Donc, les deux piles sont fortement équivalentes. En effet, a_2 et a_1 étaient déjà sur le même sommet, et donc, il n'y a aucun mouvement associé avec le fait que a_2 rejoint a_1. Ainsi, dans les deux piles, a_1 nettoie l'arête ayant pour numéro de port 1 de u_0.

Supposons que après l'étape t des deux algorithmes, les deux piles Q et $Q_{virtual}$ sont équivalentes. Nous prouvons que pour les 14 cas énumérés précédemment, après l'étape $t+1$ des deux algorithmes, les stratégies de capture restent les mêmes pour les deux algorithmes, c'est-à-dire les deux piles restent équivalentes, et les mêmes configurations sont atteintes par les deux algorithmes. Les deux prochaines sous-sections étudient séparément cas par cas les groupes A et B.

Groupe A

Le groupe A suppose que Q et $Q_{virtual}$ ont été atteintes en nettoyant une arête. Soient S et $S_{virtual}$ les états de Q et $Q_{virtual}$ à cette étape des deux algorithmes. Puisque, Q et $Q_{virtual}$ ont été atteintes en nettoyant une arête, elles sont fortement équivalentes. Donc, il existe une séquence S' de mouvements larges valides, et une séquence $S'_{virtual}$ de mouvements, avec S' et $S'_{virtual}$ fortement équivalents, et il existe un mouvement large M, et une séquence M' de mouvements, avec M' fortement équivalent à M, tel que $S = S'|M$ et $S_{virtual} = S'_{virtual}|M'$.

Nous prouvons d'abord que la prochaine étape de l'exécution de l'algorithme `Nettoyage_réparti` commence avec a_1 appliquant la procédure *decide*(). Soit j, $1 \leq j \leq k$, l'identifiant du chercheur qui vient de nettoyer une arête. Le chercheur a_j arrive sur un sommet dans l'état NETTOIE. Soit $j = 1$, et a_j applique la procédure *decide*(), soit a_j envoie $(\texttt{bouge}, j)$ à a_1, qui reçoit $(\texttt{bouge}, j)$ de j. Dans les deux cas, a_1 applique la procédure *decide*().

Nous considérons à présent les cas du groupe A.

Cas A.1 Dans le cas A.1, le graphe est supposé complètement propre. Dans ce cas, d'après le lemme 16, l'algorithme $\mathcal{A}$ termine. Prouvons que c'est aussi le cas pour l'algorithme `Nettoyage_réparti`. Le chercheur a_1 applique la procédure *decide*(). Puisque le graphe est propre, le sommet v_1 occupé par a_1, est propre. Donc, a_1 empile $(JJ, 1, 2)$ et applique la procédure $rejoins(2)$ après avoir mis $termine$ à $vrai$. Appliquant la procédure $rejoins()$, a_1 calcule un numéro de port p_1 qui vaut $direction[2]$ si a_2 a déjà visité le sommet v_1, et $direction[0]$ sinon (rappelons que $direction[0]$ est la direction de la base). Le premier cas est identifié par le fait que $direction[2] \neq 0$. Nous empilons $(1 \to 2, p_1)$ sur $Q_{virtual}$. Alors, a_1 emprunte l'arête correspondant au port p_1 de v_1, et arrive sur un nouveau sommet v_2 par le port q_1, dans l'état $(\textsc{Aide}, 2)$. Sur v_2, le chercheur a_1 écrit $(AH, 1, q_1)$ sur le tableau blanc, et applique à nouveau la procédure $rejoins()$. Ceci est répété jusqu'à ce que a_1 rejoigne a_2, sur un sommet v_t. Soit $P = v_1, v_2, \ldots, v_t$ le chemin suivi par a_1 de v_1 jusqu'à ce qu'il atteigne a_2 sur v_t. Soit p_i (resp., q_i) le numéro de port de l'arête $\{v_i, v_{i+1}\}$ sur v_i (resp., v_{i+1}). Sur chaque sommet v_i, $i \geq 2$, le chercheur a_1 écrit $(AH, 1, q_{i-1})$ pendant l'exécution de $rejoins()$. Sur $Q_{virtual}$, nous empilons $(a_1 \to a_2, p_i)$ pour $i = 1, \ldots, t-1$. Puisque v_t est propre, le chercheur a_2 n'a pas besoin d'aide, et donc, a_1 applique la procédure $retour()$. Par conséquent, il dépile $(AH, 1, q_{t-1})$ du tableau blanc de v_t, et retourne sur v_{t-1}. Sur chaque sommet v_i, pour $i = t-1, \ldots, 2$, le chercheur a_1 arrive dans l'état $\textsc{Retour}$, et donc, dépile la pile locale, qui contient $(AH, 1, q_{i-1})$. Il en résulte qu'il se rend sur v_{i-1} utilisant le port q_{i-1}. Simultanément, nous dépilons $(a_1 \to a_2, p_i)$ qui avait été empilé précédemment sur $Q_{virtual}$. Finalement, a_1 est de retour sur v_1 dans l'état $\textsc{Retour}$. Sur v_1, le chercheur a_1 applique la procédure $retour()$, et donc dépile $(JJ, 1, 2)$ de la pile locale. Cette procédure stipule que a_1 doit essayer d'aider tous les chercheurs a_i, pour $3 \leq i \leq k$. Dans ce but, a_1 applique successivement la procédure $rejoins(i)$ pour $i = 3, \ldots, k$. Puisque le graphe est complètement propre, aucun chercheur n'a besoin d'aide, et par conséquent, la même situation que pour a_2 se produit pour tout $i = 3, \ldots, k-1$, c'est-à-dire a_1 rejoint a_i, et revient sur v_1 puisque a_i n'a pas besoin d'aide. La séquence d'empilements et de suppressions est la même pour a_i que pour a_2. Quand a_1 finit par atteindre a_k, la variable $termine$ de a_1 est toujours égale à $vrai$, et donc, l'algorithme `Nettoyage_réparti` termine. La pile virtuelle satisfait $Q_{virtual} = S_{virtual}|(a_1 \to a_k, r_1)| \cdots |(a_1 \to a_k, r_\ell)$ où $r_1, \ldots, r_\ell$ est la séquence de numéros de port entre v_1 et le sommet sur lequel a_1 rencontre a_k. La pile Q est toujours dans l'état S puisque aucun mouvement large n'a été empilé. Puisque, d'après l'hypothèse d'induction, les deux piles Q et $Q_{virtual}$ étaient fortement équivalentes avant cette séquence de mouvements, le nouvel état S de la pile Q, et le nouvel état $S_{virtual}|(a_1 \to a_k, p_1)| \cdots |(a_1 \to a_k, p_\ell)$ de la pile $Q_{virtual}$, sont faiblement équivalents.

Cas A.2 Le cas A.2 suppose qu'un mouvement large valide peut être réalisé dans la configuration courante de la stratégie. Dans ce cas, l'algorithme $\mathcal{A}$ empile sur Q le plus petit mouvement large valide M (donc, l'état de Q devient $S|M$). Prouvons que c'est aussi le cas de l'algorithme `Nettoyage_réparti`, indépendamment du type de M. Nous prouvons qu'il existe une séquence M' de mouvements qui est fortement équivalente à M, et tel que, après la prochaine étape, l'état de $Q_{virtual}$ devient $S_{virtual}|M'$.

- Cas A.2.1 : M est du type (a_i, a_i, p).
  Nous considérons seulement le cas $i > 1$, le cas $i = 1$ peut être déduit facilement de cette étude. Dans ce cas, pour tout j, $1 \leq j < i$, a_j garde un sommet v_j qui est incident à plus d'une arête contaminée, et a_j est le chercheur de plus petit

identifiant sur v_j. De plus, le sommet v_i occupé par le chercheur a_i est incident à une unique arête contaminée. Soit p le numéro de port correspondant à cette unique arête contaminée. Exécutant l'algorithme `Nettoyage_réparti`, a_1 applique la procédure *decide*(). Appliquant cette procédure, le chercheur a_1 écrit $(ST, 1, 2)$ et envoie $(\texttt{bouge}, 1)$ à a_2. Pour tout j, $2 \leq j \leq i$, le chercheur a_j reçoit $(\texttt{bouge}, j-1)$ de a_{j-1} et écrit $(RT, j, j-1)$. Appliquant la procédure *decide*(), le chercheur a_j écrit $(ST, j, j+1)$ et envoie $(\texttt{bouge}, j)$ à a_{j+1}. Lorsque a_i reçoit le message $(\texttt{bouge}, i-1)$ de a_{i-1}, il applique la procédure *decide*() qui appelle la procédure $nettoie(CC, i, p)$. Donc, a_i écrit (CC, i, p) sur le tableau blanc de v_i et emprunte l'arête correspondant au port p of v_i, la nettoyant. Alors, le chercheur a_i arrive sur un nouveau sommet v. Finalement, a_i écrit (AC, i, q) et $(ST, i, 1)$ sur le tableau blanc de v. Nous empilons le mouvement $(a_i \to a_i, p)$ dans $Q_{virtual}$. Donc l'état de $Q_{virtual}$ devient $S_{virtual}|(a_i \to a_i, p)$ qui est fortement équivalent à l'état $S|(a_i, a_i, p)$ de Q.

- Cas A.2.2 : M est du type (a_1, a_j, p) avec $j > 1$.
  Dans ce cas, le chercheur a_j est le chercheur avec le plus petit identifiant, occupant un sommet incident à une arête contaminée, appelons le v_j. En particulier, le chercheur a_1 occupe un sommet propre, disons v_1, et doit aider le chercheur a_j. a_1 applique la procédure *decide*(). Puisque le sommet v_1 est propre, a_1 empile $(JJ, 1, 2)$ et applique la procédure $rejoins(2)$ après avoir mis *termine* à *vrai*. De manière identique au cas A.1, cette procédure stipule que a_1 doit essayer d'aider tous les chercheurs a_i, pour tout i, $3 \leq i \leq j$. Pour tout $i = 3, \ldots, j-1$, puisque le chercheur a_i n'a pas besoin d'aide, le chercheur a_1 applique la procédure *retour*() après avoir atteint a_i. Alors a_1 revient sur v_1 et applique $rejoins(i+1)$ (cf., cas A.1). Lorsque a_1 finit par rencontrer a_j sur v_j, la variable *termine* de a_1 passe à $faux$. La pile virtuelle satisfait $Q_{virtual} = S_{virtual}|(a_1 \to a_j, r_1)| \cdots |(a_1 \to a_j, r_\ell)$ avec $r_1, \ldots, r_\ell$ est la séquence de numéros de port de v_1 jusqu'au sommet où a_1 rencontre a_j. Soit p le plus petit numéro de port d'une arête contaminée incidente à v_j. Alors, le chercheur a_1 applique la procédure $nettoie(CJ, 1, p)$, c'est-à-dire qu'il écrit $(CJ, 1, p)$ et nettoie l'arête correspondante. Le mouvement $(a_1 \to a_1, p)$ est empilé sur $Q_{virtual}$. Donc le plus petit mouvement large est effectué par les deux algorithmes. De plus, après cette étape, l'état de $Q_{virtual}$ est $S_{virtual}|(a_1 \to a_j, p_1)| \cdots |(a_1 \to a_j, p_{|P|})|(a_1 \to a_1, p)$, qui est fortement équivalent à l'état $S|(a_1, a_j, p)$ de Q.
- Cas A.2.3 : M est du type (a_i, a_1, p) avec $i > 1$.
  Dans ce cas, pour tout $\ell < i$, le chercheur a_ℓ est seul sur le sommet v_ℓ qui estr incident à plus d'une arête contaminée. De plus, le chercheur a_i occupe v_i, un sommet propre ou un sommet occupé par un chercheur a_ℓ, avec $\ell < i$. Dans l'algorithme `Nettoyage_réparti`, a_1 applique la procédure *decide*(). Appliquant cette procédure, le chercheur a_1 écrit $(ST, 1, 2)$ et envoie $(\texttt{bouge}, 1)$ to a_2. Pour tout j, $2 \leq j \leq i$, le chercheur a_j reçoit $(\texttt{bouge}, j-1)$ de a_{j-1} et écrit $(RT, j, j-1)$. Appliquant la procédure *decide*(), le chercheur a_j écrit $(ST, j, j+1)$ et envoie $(\texttt{bouge}, j)$ à a_{j+1}. Lorsque a_i reçoit le message $(\texttt{bouge}, i-1)$ de a_{i-1}, il applique la procédure *decide*() qui appelle la procédure $rejoins(1)$. Cette procédure est appelée jusqu'à ce que a_i rejoigne a_1. Soit $P = w_1, w_2, \ldots, w_r$ le chemin suivi par a_i de $w_1 = v_i$

jusqu'à $w_r = v_1$. Soit p_j (resp., q_j) le numéro de port de l'arête $\{w_j, w_{j+1}\}$ sur w_j (resp., w_{j+1}). Sur chaque sommet w_j, $j \geq 2$, le chercheur a_i écrit (AH, i, q_{j-1}) au cours de l'exécution de $rejoins()$. Dans $Q_{virtual}$, nous empilons $(a_i \to a_1, p_j)$ pour $j = 1, \ldots, r-1$. Sur w_r, le chercheur a_i écrit $(ST, i, 1)$ et envoie le message $(\texttt{aide}, i)$ au chercheur a_1. Alors, a_1 écrit $(RT, 1, i)$ et $(CJ, 1, p)$, et nettoie l'arête correspondante. Le mouvement $(a_1 \to a_1, p)$ est empilé sur $Q_{virtual}$. Donc, le plus petit mouvement large est réalisé par les deux algorithmes. De plus, après cette étape, l'état de $Q_{virtual}$ est $S_{virtual}|(a_i \to a_1, p_1)| \cdots |(a_i \to a_1, p_r)|(a_1 \to a_1, p)$, qui est fortement équivalent à l'état $S|(a_i, a_1, p)$ de Q.

Cas A.3 Le cas A.3 suppose qu'il n'existe pas de mouvement large valide par rapport à l'état courant de la pile Q. Par conséquent, l'algorithme $\mathcal{A}$ dépile le dernier mouvement large M de $S = S'|M$. Prouvons que l'algorithme `Nettoyage_réparti` fait la même chose. Supposons que pour réaliser ce dernier mouvement large, a_i ($i \geq 1$) a nettoyé l'arête $e = (v, w)$ en empruntant le port p de v. Rappelons que $S_{virtual} = S'_{virtual}|M'$ avec M' une séquence de mouvements équivalente à M. Trois cas doivent être pris en compte :

- **Cas A.3.1** : $M = (a_i, a_i, p)$. Dans ce cas, M' est la séquence de longeur 1 : $(a_i \to a_i, p)$.
- **Cas A.3.2** : Il existe $k \geq s > i$ tel que $M = (a_i, a_s, p)$. Dans ce cas, le chercheur a_i quitte un sommet, disons v_i, pour rejoindre le chercheur a_s sur le sommet v. Alors, le chercheur a_i nettoie l'arête correspondant au port p de v. Soit $P = w_1, w_2, \ldots, w_r$ le chemin suivi par a_i de $w_1 = v_i$ jusqu'à $w_r = v$. Soit p_j (resp., q_j) le numéro de port de l'arête $\{w_j, w_{j+1}\}$ sur w_j (resp., w_{j+1}). D'après l'hypothèse d'induction, $M' = (a_i \to a_s, p_1)| \cdots |(a_i \to a_s, p_r)|(a_i \to a_i, p)$
- **Cas A.3.3** : Il existe $k \geq s > i$ tel que $M = (a_s, a_i, p)$. Dans ce cas, le chercheur a_s quitte un sommet, disons v_s pour rejoindre le chercheur a_i sur le sommet v. Alors, le chercheur a_i nettoie l'arête correspondant au port p de v. Soit $P = w_1, w_2, \ldots, w_r$ le chemin suivi par a_s de $w_1 = v_s$ à $w_r = v$. Soit p_j (resp., q_j) le numéro de port de l'arête $\{w_j, w_{j+1}\}$ sur w_j (resp., w_{j+1}). D'après l'hypothèse d'induction, $M' = (a_s \to a_i, p_1)| \cdots |(a_s \to a_i, p_r)|(a_i \to a_i, p)$

Le chercheur a_i est arrivé sur le sommet w par le port, disons q, et a_i a empilé (AC, i, q). Si $i > 1$, a_i a aussi empilé $(ST, i, 1)$ et envoyé $(\texttt{bouge}, i)$ à a_1, qui a empilé $(RT, 1, i)$ sur la pile du sommet qu'il occupe. Alors, le chercheur a_1 applique la procédure $decide()$. Puisque, il n'existe pas de mouvement large valide, cela signifie que, pour tout j, $1 \leq j \leq k$, le chercheur a_j est sur un sommet v_j qui est incident à plus d'une arête contaminée, et pour tout j, $1 \leq j < \ell \leq k$, $v_j \neq v_\ell$. Pour tout $j < k$, a_j écrit $(ST, j, j+1)$ et envoie $(\texttt{bouge}, j)$ à a_{j+1} en appliquant la procédure $prochain\_chercheur(j)$ dans la procédure $decide()$. Alors, le chercheur a_{j+1} empile $(RT, j+1, j)$ sur la pile du sommet qu'il occupe avant d'appliquer la procédure $decide()$ aussi. Lorsque a_k reçoit le message $(\texttt{bouge}, k-1)$ de a_{k-1}, il applique la procédure $decide()$ qui appelle la procédure $retour()$. Alors, a_k dépile $(RT, k, k-1)$ et envoie $(\texttt{désolé}, k)$ au chercheur a_{k-1}. Pour tout $j > 1$, a_j reçoit $(\texttt{désolé}, j+1)$ de a_{j+1}. Alors le chercheur a_j applique la procédure $retour()$ qui dépile $(ST, j, j+1)$, puis dépile $(RT, j, j-1)$, et envoie $(\texttt{désolé}, j)$ à a_{j-1}. Lorsque a_1 reçoit

$(\texttt{désolé}, 2)$, a_1 applique la procédure $retour()$ qui dépile $(ST, 1, 2)$, puis dépile $(RT, 1, i)$, et envoie $(\texttt{désolé}, 1)$ à a_i. En appliquant la procédure $retour()$, a_i dépile $(ST, i, 1)$, puis (AC, i, q). Finalement, a_i assigne $cleared\_port[q]$ à $faux$ et revient sur v (laissant l'arête e être recontaminée). Le chercheur a_i arrives dans l'état RETOUR par le numéro de port p. Donc, le mouvement $(a_i \to a_i, p)$ est supprimé de $Q_{virtual}$. Alors a_i assigne $cleared\_port[p]$ à $faux$. Donc, l'arête e est bien marquée comme étant contaminée et a_i est revenu à sa position précédente. Donc, les deux algorithmes annulent le nettoyage de la dernière arête. Notons que dans les trois sous-cas, nous supprimons seulement le mouvement $(a_i \to a_i, p)$. Donc, le nouvel état de $Q_{virtual}$ dépend du cas :

- Cas A.3.1 : $S'_{virtual}$
- Cas A.3.2 : $S'_{virtual}|(a_i \to a_s, p_1)| \cdots |(a_i \to a_s, p_\ell)$
- Cas A.3.3 : $S'_{virtual}|(a_s \to a_i, p_1)| \cdots |(a_s \to a_i, p_\ell)$

Par conséquent, $S'_{virtual}$ est équivalent à l'état S' de Q (fortement équivalent dans le cas A.3.1, et faiblement équivalent dans les deux autres cas).

Groupe B

Les cas du groupe B supposent que Q et $Q_{virtual}$ ont été atteintes en annulant le nettoyage d'une arête. Soit M le mouvement large dépilé par l'algorithme $\mathcal{A}$ au cours de l'étape précédente. Soient S et $S_{virtual}$ les états de Q et $Q_{virtual}$ à cette étape des deux algorithmes. Donc, il existe $i \geq 1$, un sommet v, un port p de v correspondant à une arête e, tels que le chercheur a_i vient d'arriver dans l'état RETOUR, sur le sommet v, par le port p, laissant l'arête e se faire recontaminée. Donc, dans ces cas, la prochaine étape de l'exécution de l'algorithme `Nettoyage_réparti` débute avec a_i appliquant la procédure $retour()$.

Cas B.1 Les cas B.1 supposent qu'il existe un mouvement large valide plus grand que M. Dans ce cas, l'algorithme $\mathcal{A}$ empile le plus petit mouvement large valide $M' \succ M$ sur Q. Dans ce qui suit, M' peut être de trois types différents, définis ci-dessous. Prouvons que l'algorithme `Nettoyage_réparti` exécute une séquence de mouvements équivalente à M'. Il y a 3 cas qui dépendent du type du mouvement large M.

- **Cas B.1.1** : $M = (a_i, a_i, p)$. Ce cas se produit après l'opération de suppression dans le cas $A.3.1$. Donc S et $S_{virtual}$ sont en fait fortement équivalents. Dans ce cas, il existe $i < j \leq \ell \leq k$ et $0 \leq q \leq n$ tels que il existe un mouvement large $M' = (a_j, a_\ell, q)$ plus grand que M. C'est-à-dire, j est le plus petit identifiant plus grand que i tel que a_j peut effectuer un mouvement large valide. En appliquant la procédure $retour()$, a_i dépile (CC, i, p) de la pile locale sur v. Donc, a_i appelle la procédure $prochain\_chercheur(i)$, puis empile $(ST, i, i{+}1)$ sur v, et envoie $(\texttt{bouge}, i)$ à a_{i+1}. De la même manière que dans le cas A.2, le message $(\texttt{bouge}, j-1)$ est reçu par a_j qui peut réaliser un mouvement large valide. Comme dans le cas A.2, le chercheur a_j effectue ce mouvement et nous empilons sur $Q_{virtual}$ une séquence de mouvement équivalente à M'. Donc les deux piles restent fortement équivalentes.
- **Cas B.1.2** : $M = (a_i, a_j, p)$ avec $i < j$.

Ce cas se produit après l'opération de suppression dans le cas $A.3.2$. Donc S et $S_{virtual}$ sont faiblement équivalents. Plus précisément, il existe un état $S'_{virtual}$ qui est fortement équivalent à S et une séquence $(p_1, \ldots, p_{t-1})$ de numéros de port, tels que $S_{virtual} = S'_{virtual}|(a_i \to a_j, p_1)| \cdots |(a_i \to a_j, p_{t-1})$. Soit v_i (resp., v_j) le sommet que le chercheur a_i (resp., a_j) occupe dans la configuration associée à $S'_{virtual}$. Notons que $v_j = v$. $(p_1, \ldots, p_{t-1})$ est exactement la séquence de numéros de port que le chercheur a_i a suivi le long d'un chemin de v_i à v_j. Soit $P = w_1, \cdots, w_t$ ce chemin, avec $w_1 = v_i$ et $w_t = v_j$. Plus précisément, la configuration associée à $S_{virtual}$ est obtenu à partir de la configuration associée à $S'_{virtual}$ (qui est aussi la configuration associée à S) en déplaçant le chercheur a_i le long d'un chemin de v_i à v_j en suivant la séquence $(p_1, \ldots, p_{t-1})$ de numéros de port. Soit q le numéro de port de v_j correspondant à l'arête $\{w_{t-1}, v_j\}$. Rappelons que, lorsque il a rejoint a_j sur v_j, le chercheur a_i y avait écrit (AH, i, q). Alors, puisque $i < j$, a_i avait écrit (CJ, i, p) et avait nettoyé l'arête.

Pour prouver que les deux piles restent équivalentes, nous considérons le type du mouvement large M'. Il y a trois sous-cas :

- Cas B.1.2.a : il y a un numéro de port $r \leq n$ de v_j, plus grand que p tel que l'arête correspondante est contaminée. Dans ce cas, $M' = (a_i, a_j, r)$.
- Cas B.1.2.b : il y a un chercheur d'identifiant $\ell \leq k$, plus grand que j, sur le sommet v_ℓ, et un numéro de port $r \leq n$ de v_ℓ tel que l'arête correspondante est contaminée. Dans ce cas, $M' = (a_i, a_\ell, r)$.
- Cas B.1.2.c : il y a un chercheur d'identifiant $\ell \leq k$, plus grand que i, sur le sommet v_ℓ, qui peut effectuer un mouvement large valide. C'est-à-dire, qu'il existe $\ell < u \leq k$ et $r \leq n$ tel que $M' = (a_\ell, a_u, r)$.

Notons que le mouvement large considéré dans le cas B.1.2.a est plus petit que le mouvement large du cas B.1.2.b, lui même plus petit que le mouvement large du cas B.1.2.c.

Maintenant, considérons l'exécution du protocole `Nettoyage_réparti` après avoir annulé le nettoyage de e. En appliquant la procédure $retour()$, a_i dépile (CJ, i, p). Alors, l'algorithme `Nettoyage_réparti` teste d'abord si il existe un numéro de port $r > p$ d'une arête contaminée incidente à v_j. Supposons qu'un tel numéro de port existe. Cela correspond au cas B.1.2.a :

- L'algorithme $\mathcal{A}$ empile $M' = (a_i, a_j, r)$ sur Q. Le chercheur a_i empile (CJ, i, r) sur v_j et nettoie l'arête correspondante, arrive sur un nouveau sommet par le port, disons o. Le chercheur a_i empile (AC, i, o) et $(ST, i, 1)$ sur le nouveau sommet, puis envoie le message $(\texttt{bouge}, i)$ à a_1. Nous empilons $(a_i \to a_i, r)$ sur $Q_{virtual}$. Donc, l'état de Q est $S|M'_1$ et l'état de $Q_{virtual}$ est $S'_{virtual}|(a_i \to a_j, p_1)| \cdots |(a_i \to a_j, p_{t-1})|(a_i \to a_i, r)$. Par conséquent, les deux piles sont fortement équivalentes.

Maintenant, supposons qu'il n'existe pas de numéro de port de v_j, plus grand que p, correspondant à une arête contaminée. Dans ce cas, a_i applique la procédure $retour()$. Par conséquent, il dépile (AH, i, q) du tableau blanc de $w_t = v_j$, et retourne sur w_{t-1}. Sur chaque sommet w_f, pour $f = t-1, \ldots, 2$, le chercheur a_i arrive dans l'état RETOUR, et donc, dépile le sommet de la pile locale qui contient (AH, i, q_f) où

q_f est le numéro de port menant à w_{f-1}. Il en résulte qu'il va sur w_{f-1} en utilisant le port q_{f-1}. Simultanément, nous supprimons $(a_i \to a_j, p_f)$ qui avait été empilé sur $Q_{virtual}$ précédemment. a_i finit par revenir sur v_i dans l'état RETOUR. A cette étape de l'exécution de `Nettoyage_réparti`, l'état courant de $Q_{virtual}$ est $S'_{virtual}$ qui est fortement équivalent à S. Alors, en appliquant la procédure $retour()$, a_i dépile (JJ, i, j). Alors, l'algorithme `Nettoyage_réparti` teste si le chercheur a_i peut aider un chercheur d'identifiant supérieur à j. En appliquant la procédure $retour()$, a_i empile $(JJ, i, j+1)$ et applique la procédure $rejoins(j+1)$. De la même manière que dans le cas A.1, cette procédure stipule que a_i doit essayer d'aider chaque chercheur a_t, pour tout t, $j+1 \leq t \leq k$. Supposons que il y a un chercheur d'identifiant $\ell \leq k$, plus grand j, sur le sommet v_ℓ, et un numéro de port $r \leq n$ sur v_ℓ tel que l'arête correspondante est contaminée. Cela correspond au cas B.1.2.b :

- L'algorithme $\mathcal{A}$ empile $M' = (a_i, a_\ell, r)$ sur Q. Pour tout $f = j+1, \ldots, \ell-1$, puisque le chercheur a_f n'a pas besoin d'aide, le chercheur a_i applique la procédure $retour()$ après avoir atteint a_f. Alors a_i revient sur v_i et applique $rejoins(f+1)$ (cf., cas A.1). Lorsque a_i finit par atteindre a_ℓ sur v_ℓ, la pile virtuelle satisfait $Q_{virtual} = S_{virtual}|(a_i \to a_\ell, p_1)|\cdots|(a_i \to a_\ell, p_t)$ avec $p_1, \ldots, p_t$ la séquence de numéros de port de v_i à v_ℓ. Alors, le chercheur a_i applique la procédure $nettoie(CJ, i, r)$, c'est-à-dire qu'il écrit (CJ, i, r) et nettoie l'arête correspondante. Le mouvement $(a_i \to a_i, r)$ est empilé sur $Q_{virtual}$. Donc, le plus petit mouvement large est réalisé par les deux algorithmes. De plus, après cette étape, l'état de $Q_{virtual}$ est $S_{virtual}|(a_i \to a_\ell, p_1)|\cdots|(a_i \to a_\ell, p_t)|(a_i \to a_i, r)$, qui est fortement équivalent à l'état $S|(a_i, a_\ell, r)$ de Q.

Considérons à présent le cas où il n'existe pas $\ell > j$ tel que le chercheur a_ℓ occupe le sommet v_ℓ incident à une arête contaminée. Donc, a_i revient sur v_i après avoir essayé d'aider chaque chercheur a_ℓ, pour tout ℓ, $j < \ell \leq k$ (en appliquant itérativement la procédure $rejoins()$ comme dans le cas précédent). A cette étape de l'exécution `Nettoyage_réparti`, l'état courant de $Q_{virtual}$ est $S'_{virtual}$ qui est fortement équivalent à S (l'état courant de Q). Lorsque a_i revient sur v_i, il dépile (JJ, i, k). Donc, la procédure $retour()$ appelle la procédure $prochain\_chercheur(i)$. Par conséquent, a_i empile $(ST, i, i+1)$ et envoie $(\texttt{bouge}, i)$ au chercheur a_{i+1}. Supposons qu'il y a un chercheur d'identifiant $\ell \leq k$, plus grand que i, sur le sommet v_ℓ, qui peut réaliser un mouvement large valide. Cela correspond au cas B.1.2.c :

- Dans ce cas, il existe $\ell \leq u \leq k$ et $r \leq n$ tels que l'algorithme $\mathcal{A}$ empile $M' = (a_\ell, a_u, r)$ sur Q. De la même façon que dans le cas $A.2.3$, pour tout f, $i \leq f \leq \ell$, le chercheur a_f reçoit $(\texttt{bouge}, f-1)$ de a_{f-1} et écrit $(RT, f, f-1)$. Appliquant la procédure $decide()$, le chercheur a_f écrit $(ST, f, f+1)$ et envoie $(\texttt{bouge}, f)$ à a_{f+1}. Lorsque a_ℓ reçoit le message $(\texttt{bouge}, \ell-1)$ de $a_{\ell-1}$, il applique la procédure $decide()$. Alors le mouvement M' est réalisé comme dans le cas A.2. Ainsi, les deux piles deviennent fortement équivalentes.

- **Cas B.1.3** : $M = (a_j, a_i, p)$ avec $i < j$.

Ce cas se produit après l'opération de suppression dans le cas $A.3.3$. Donc S et $S_{virtual}$ sont faiblement équivalents. Plus précisément, il existe un état $S'_{virtual}$ qui est fortement équivalent à S et une séquence $(p_1, \ldots, p_{t-1})$ de numéros de port, tels que $S_{virtual} = S'_{virtual}|(a_j \to a_i, p_1)| \cdots |(a_j \to a_i, p_{t-1})$. Soit v_i (resp., v_j) le sommet occupé par le chercheur a_i (resp., a_j) dans la configuration associée à $S'_{virtual}$. Notons que $v_i = v$. $(p_1, \ldots, p_{t-1})$ est exactement la séquence de numéro de port que le chercheur a_j a suivi le long du chemin de v_j à v_i. Soit $P = w_1, \cdots, w_t$ ce chemin, avec $w_1 = v_j$ et $w_t = v_i$. Plus précisément, la configuration associée à $S_{virtual}$ est obtenu à partir de la configuration associée à $S'_{virtual}$ (qui est aussi la configuration associée à S) en déplaçant le chercheur a_j le long du chemin de v_j à v_i en suivant la séquence de numéros de port $(p_1, \ldots, p_{t-1})$. Soit q le numéro de port de v_i correspondant à l'arête $\{w_{t-1}, v_i\}$. Rappelons que, lorsqu'il a rejoint a_i sur v_i, le chercheur a_j a écrit (AH, j, q). Alors, puisque $i < j$, a_j a empilé (ST, j, i) sur v_i et envoyé $(\texttt{aide}, j)$ au chercheur a_i. Alors, le chercheur a_i a empilé (RT, i, j) et (CJ, i, p) sur v_i, et a nettoyé l'arête.

Pour prouver que les deux piles restent équivalentes, nous considérons le type du mouvement large M'. Il y a trois cas :

– Cas B.1.3.a : il existe un numéro de port $r \leq n$ de v_i, plus grand que p tel que l'arête correspondante est contaminée. Dans ce cas, $M' = (a_j, a_i, r)$.
– Cas B.1.3.b : il y a un chercheur d'identifiant $\ell \leq k$, plus grand que i, sur un sommet v_ℓ, et un numéro de port $r \leq n$ de v_ℓ tel que l'arête correspondante est contaminée. Dans ce cas, $M' = (a_j, a_\ell, r)$.
– Cas B.1.3.c : il y a un chercheur d'identifiant $\ell \leq k$, plus grand que j, sur un sommet v_ℓ, qui peut réaliser un mouvement large valide. C'est-à-dire, il existe $\ell < u \leq k$ et $r \leq n$ tels que $M' = (a_\ell, a_u, r)$.

Notons que le mouvement large considéré dans le cas B.1.3.a est plus petit que le mouvement large du cas B.1.3.b, lui même plus petit que le mouvement large du cas B.1.3.c.

Maintenant, considérons l'exécution du protocole `Nettoyage_réparti` après avoir annulé le nettoyage de e. En appliquant la procédure $retour()$, a_i dépile (CJ, i, p). Alors, l'algorithme `Nettoyage_réparti` teste d'abord si il existe un numéro de port $r > p$ d'une arête contaminée incidente à v_j. Supposons qu'un tel numéro de port existe. Cela correspond au cas B.1.3.a.

– Comme dans le cas B.1.2..a, le chercheur a_i nettoie l'arête correspondant au numéro de port r et les deux piles restent fortement équivalentes.

Si il n'existe pas de numéro de port v_j, plus grand que p, correspondant à l'arête contaminée, a_i applique la procédure $retour()$. a_i dépile (CJ, i, p), alors (RT, i, j), et envoie $(messsorry, i)$ au chercheur a_j. Alors, le chercheur a_j applique la procédure $retour()$. Par conséquent, il dépile (ST, j, i) et (AH, i, q) du tableau blanc de $w_t = v_i$, et retourne sur w_{t-1}. Sur chaque sommet w_f, pour tout $f = t-1, \ldots, 2$, le chercheur a_j arrive dans l'état RETOUR, et donc, dépile la pile locale, qui contient (AH, j, q_f) où q_f est le numéro de port de w_f menant à w_{f-1}. Il en résulte qu'il va sur w_{f-1} utilisant le port q_{f-1}. Simultanément, nous supprimons $(a_j \to a_i, p_f)$ que nous avions empilé précédemment sur $Q_{virtual}$. a_j finit par revenir sur v_j dans l'état RETOUR. A cette étape de l'exécution de `Nettoyage_réparti`, l'état courant de $Q_{virtual}$ est

$S'_{virtual}$ qui est fortement équivalent à S. Alors, en appliquant la procédure $retour()$, a_j dépile (JJ, j, i). Alors, l'algorithme `Nettoyage_réparti` teste si le chercheur a_j peut aider le chercheur avec un identifiant plus grand que i. Alors, le cas B.1.3.b est similaire au cas B.1.2.b, et le cas B.1.3.c est similaire au cas B.1.2.c. Donc, les deux piles deviennent fortement équivalentes.

Cas B.2 Le cas B.2 suppose qu'il n'existe pas de mouvement large valide plus grand que M. Dans ce cas, soit $S = \emptyset$ ou il existe un mouvement valide M' et une séquence de mouvements larges valides S' tels que $S = S'|M'$. Dans le premier cas, l'algorithme $\mathcal{A}$ appelle un autre chercheur. Dans le second cas, l'algorithme $\mathcal{A}$ dépile M' de Q. Prouvons que c'est également le cas pour l'algorithme `Nettoyage_réparti`. Il y a quatre cas selon que $S = \emptyset$ ou selon le type de M'.

- **Cas B.2.1** Si $S = \emptyset$, il y a deux cas. Soit $k = 1$ et u_0 est de degré supérieur à un, ou $k > 1$. Dans le premier cas, le chercheur a_1 applique la procédure $decide()$, alors la procédure $retour()$ appelle un second chercheur. Dans le second cas, M doit être le mouvement large (a_k, a_{k-1}, p) où p est le plus grand numéro de port de u_0. En effet, si M n'est pas ce mouvement large, alors un mouvement large plus grand que M serait valide. Dans ce cas, le chercheur a_{k-1} vient d'arriver dans l'état RETOUR, sur le sommet u_0 par le port p. De plus, tous les chercheurs occupent u_0. De plus, le tableau blanc de u_0 contient exactement la séquence $((ST, 1, 2), (RT, 2, 1), \cdots, (ST, i, i+1), (RT, i+1, i), \cdots, (ST, k-1, k), (RT, k, k-1), (JJ, k, k-1), (ST, k, k-1), (RT, k-1, k), (CJ, k-1, p))$. Donc, $S_{virtual} = \emptyset$. Ainsi, Q et $Q_{virtual}$ sont fortement équivalentes. De plus, il est facile de vérifier que la procédure $retour()$ appelle un $(k+1)^{eme}$ chercheur.

Supposons que $S \neq \emptyset$. Rappelons que dans le cas B., un chercheur a_i vient d'arriver dans l'état RETOUR, sur le sommet v, par le port p, laissant l'arête e être recontaminée. Donc, dans ces cas, la prochaine étape de l'exécution de l'algorithme `Nettoyage_réparti` commence avec a_i appliquant la procédure $retour()$.

Soit $f = (v', w')$ l'arête nettoyée grâce au mouvement M'. Soit $s \leq k$ l'identifiant du chercheur qui a nettoyé f, en arrivant du port r' de w'. Soit r le numéro de port de v' correspondant à f. Considérons les trois types possibles pour le mouvement M' :

- **Cas B.2.2** $M' = (a_s, a_s, r)$. Dans ce cas, il y a une séquence de mouvements valides $S'_{virtual}$ fortement équivalente à S', et une séquence de mouvements valides $M_{virtual}$ équivalente à M telles que $S_{virtual} = S'_{virtual}|(a_s \to a_s, r)|M_{virtual}$.
- **Cas B.2.3** Il existe $s' < s$ tel que $M' = (a_s, a_{s'}, r)$. Dans ce cas, il y a une séquence de mouvements valides $S'_{virtual}$ fortement équivalente à S', une séquence de mouvements valides $M_{virtual}$ équivalente à M, et une séquence $(p_1, \dots, p_{t-1})$ de numéros de port, telles que $S_{virtual} = S'_{virtual}|(a_s \to a_{s'}, p_1)|\cdots|(a_s \to a_{s'}, p_{t-1})|(a_s \to a_s, r)|M_{virtual}$.
- **Cas B.2.4** Il existe $s' < s$ tel que $M' = (a_{s'}, a_s, r)$. Dans ce cas, il y a une séquence de mouvements valides $S'_{virtual}$ fortement équivalente à S', une séquence de mouvements valides $M_{virtual}$ équivalente à M, et une séquence $(p_1, \dots, p_{t-1})$ de numéros de port, telles que $S_{virtual} = S'_{virtual}|(a_{s'} \to a_s, p_1)|\cdots|(a_{s'} \to a_s, p_{t-1})|(a_s \to a_s, r)|M_{virtual}$.

Après avoir nettoyé f, le chercheur a_s a empilé (AH, s, r'), puis $(ST, s, 1)$, et envoyé $(\texttt{bouge}, s)$ au chercheur a_1. Alors a_1 applique la procédure $decide()$. Appliquant cette procédure, le chercheur a_1 écrit $(ST, 1, 2)$ et envoie $(\texttt{bouge}, 1)$ à a_2. Pour tout $2 \leq j \leq i-1$, le chercheur a_j reçoit $(\texttt{bouge}, j-1)$ de a_{j-1} et écrit $(RT, j, j-1)$. Appliquant la procédure $decide()$, le chercheur a_j écrit $(ST, j, j+1)$ et envoie $(\texttt{bouge}, j)$ à a_{j+1}. Lorsque a_i reçoit le message $(\texttt{bouge}, i-1)$ de a_{i-1}, il empile $(RT, i, i-1)$ sur le sommet v_i qu'il occupe, et applique la procédure $decide()$. Soit v_i le sommet que a_i occupe à cette étape de l'exécution du protocole `Nettoyage_réparti`.

Considérons le type du mouvement large M. Soit $p \leq n$ le numéro de port de v correspondant à e. Puisqu'il n'y a pas de mouvement large valide plus grand que M, seuls trois cas sont possibles :

- $M = (a_i, a_i, p)$ et pour tout j, $i < j \leq k$, le chercheur a_j est seul à occuper un sommet, disons v_j. En annulant un tel mouvement, le protocole `Nettoyage_réparti` assure que Q et $Q_{virtual}$ sont fortement équivalentes (cf., cas A.3.1). Donc, $v = v_i$. Dans ce cas, le chercheur a_i retourne sur v dans l'état RETOUR. Appliquant la procédure $retour()$, a_i dépile (CC, i, p), empile $(ST, i, i+1)$ sur v_i, et envoie $(\texttt{bouge}, i)$ au chercheur a_{i+1}. Pour tout j, $i+1 \leq j \leq k$, le chercheur a_j reçoit $(\texttt{bouge}, j-1)$ de a_{j-1} et écrit $(RT, j, j-1)$. Appliquant la procédure $decide()$, le chercheur a_j écrit $(ST, j, j+1)$ et envoie $(\texttt{bouge}, j)$ à a_{j+1}.Lorsque a_k reçoit le message $(\texttt{bouge}, k-1)$ de a_{k-1}, le chercheur a_k applique la procédure $decide()$, puis la procédure $retour()$. Le chercheur a_k dépile $(ST, k, k-1)$ et envoie $(\texttt{désolé}, k)$ à a_{k-1}. Pour tout j, $k > j > i$, le chercheur a_j reçoit $(\texttt{désolé}, j+1)$ de a_{j+1} et dépile $(ST, j, j+1)$. Appliquant la procédure $retour()$, le chercheur a_j dépile $(RT, j-1, j)$ et envoie $(\texttt{désolé}, j)$ à a_{j-1}. Lorsque le chercheur a_i reçoit $(\texttt{désolé}, i+1)$, il dépile $(ST, i, i+1)$, et alors dépile $(RT, i, i-1)$ de la pile locale de v_i.
- $i < k$, $M = (a_i, a_k, p)$ et pour tout j, $i < j \leq k$, le chercheur a_k occupe seul un sommet, disons v_j. Dans ce cas, il existe un état $S'_{virtual}$ qui est fortement équivalent à S et une séquence $(p_1, \ldots, p_{t-1})$ de numéros de port, tels que $S_{virtual} = S'_{virtual}|(a_i \to a_k, p_1)| \cdots |(a_i \to a_k, p_{t-1})$. Notons que, dans la configuration associée à $S'_{virtual}$ (resp., à $S_{virtual}$), le chercheur a_i occupe v_i (resp., v). Le chercheur a_k occupe v dans les deux configurations. $(p_1, \ldots, p_{t-1})$ est exactement la séquence de numéros de port que le chercheur a_i a suivi le long d'un chemin de v_i à v. Soit $P = w_1, \cdots, w_t$ ce chemin, avec $w_1 = v_i$ et $w_t = v$. Pour tout f, $2 \leq f \leq t$, soit q_f le numéro de port de w_f correspondant à l'arête $\{v_{f-1}, w_f\}$. Rappelons que, a_i a suivi le chemin P pour rejoindre a_k. Puis, le chercheur a_1 a écrit (AH, i, q_t). Alors, puisque $i < j$, a_i a écrit (CJ, i, p) et a nettoyé l'arête.

  Maintenant, discutons de l'exécution du protocole `Nettoyage_réparti` après avoir annulé M. Arrivant sur v, par le port p, dans l'état RETOUR, a_i applique la procédure $retour()$. Par conséquent, il dépile (AH, i, q_t) du tableau blanc de v, et retourne sur w_{t-1}. Pour tout $f = t-1, \ldots, 2$, le chercheur a_i arrive dans l'état RETOUR sur chaque sommet w_f. Alors, il dépile la pile locale qui contient (AH, i, q_f). Il en résulte qu'il se dirige vers w_{f-1} utilisant le port q_{f-1}. Simultanément, nous supprimons $(a_i \to a_k, p_f)$ que nous avions empilé préalablement sur $Q_{virtual}$. a_i finit par revenir sur v_i dans l'état RETOUR. A cette étape de l'exécution de `Nettoyage_réparti`, l'état courant de $Q_{virtual}$ est $S'_{virtual}$ qui est fortement équivalent à S. Alors, en appliquant la procédure $retour()$, a_i dépile (JJ, i, k), empile $(ST, i, i+1)$ sur v_i, et envoie `bouge`$, i)$ au chercheur a_{i+1}. Pour tout j, $i+1 \leq j \leq k$, le chercheur a_j reçoit (`bouge`$, j-1)$ de a_{j-1} et écrit $(RT, j, j-1)$. Appliquant la procédure $decide()$, le chercheur a_j écrit $(ST, j, j+1)$ et envoie (`bouge`$, j)$ à a_{j+1}. Lorsque a_k reçoit le message (`bouge`$, k-1)$ de a_{k-1}, le chercheur a_k applique la procédure $decide()$, puis la procédure $retour()$. Le chercheur a_k dépile $(ST, k, k-1)$ et envoie (`désolé`$, k)$ à a_{k-1}. Pour tout j, $k > j > i$, le chercheur a_j reçoit (`désolé`$, j+1)$ de a_{j+1} et dépile $(ST, j, j+1)$. Appliquant la procédure $retour()$, le chercheur a_j dépile $(RT, j-1, j)$ et envoie (`désolé`$, j)$ à a_{j-1}. Lorsque le chercheur a_i reçoit (`désolé`$, i+1)$, il dépile $(ST, i, i+1)$, puis dépile $(RT, i, i-1)$ de la pile locale of v_i.

– $i = k$ et $M' = (a_i, a_{k-1}, p)$. De manière similaire au cas précédent, nous pouvons prouver qu'il existe une étape de l'exécution de `Nettoyage_réparti` où a_i dépile $(RT, i, i-1)$ de la pile locale de v_i.

Donc, quel que soit le type de M, il y a une étape de l'exécution de `Nettoyage_réparti` quand a_i dépile $(RT, i, i-1)$ de la pile locale de v_i. De plus, à cette étape, Q et $Q_{virtual}$ sont fortement équivalentes.

Si $i = k$, le chercheur a_k applique la procédure $retour()$. Sinon, a_i appelle la procédure $prochain\_chercheur(i)$, empile $(ST, i, i+1)$ et envoie (`bouge`$, i)$ à a_{i+1}. Alors, pour tout j, $i < j < k$, a_j empile $(RT, j, j-1)$ et $(ST, j, j+1)$ sur le sommet qu'il occupe, et envoie (`bouge`$, j)$ à a_{j+1}. Lorsque a_k reçoit le message (`bouge`$, k-1)$, il applique la procédure $retour()$. Le chercheur a_k envoie (`désolé`$, k)$ au chercheur a_{k-1}. Alors, pour tout j, $k \geq j > i$, a_j dépile $(ST, j, j+1)$ et $(RT, j, j-1)$, et envoie (`désolé`$, j)$ à a_{j-1}. Finalement, a_i reçoit (`désolé`$, i+1)$ du chercheur a_{i+1}, et applique la procédure $retour()$. Pour tout j, $i \geq j > s$, a_j dépile $(ST, j, j+1)$ et $(RT, j, j-1)$, et envoie (`désolé`$, j)$ à a_{j-1}. Finalement, a_s reçoit (`désolé`$, s+1)$ du chercheur a_{s+1} et applique la procédure $retour()$. Alors, le chercheur a_s dépile (AC, s, r') de la pile locale de w'. Alors, il retourne sur v' dans l'état RETOUR, laissant l'arête f être recontaminée. Nous supprimons $(a_s \rightarrow a_s, r)$ du sommet de $Q_{virtual}$. Donc, dans le cas B.2.2 (resp., B.2.3 et B.2.4), Q et $Q_{virtual}$ deviennent fortement équivalentes (resp., faiblement équivalentes).

Nous avons prouvé que, dans tous les cas, les deux piles restent équivalentes après une étape de l'exécution du protocole `Nettoyage_réparti` (C'est-à-dire qu'elles représentent la même stratégie de capture). De plus, les deux algorithmes terminent dans le même état. Donc, la preuve du théorème 25 découle directement du théorème 26.

Nous étudions à présent la taille des tableaux blancs nécessaire à l'exécution du protocole `Nettoyage_réparti`.

Lemme 17 *Soit G un graphe connexe de n sommets. Lors de l'exécution de* `Nettoyage_réparti` *au plus $O(m \ \log n)$ bits sont stockées sur les tableaux blancs de chaque sommet.*

Preuve. Soient t_1 et t_2 deux étapes de l'exécution du protocole `Nettoyage_réparti` qui satisfont (1) une arête f est nettoyée lors de l'étape t_1, ou t_1 est la première étape, (2) une arête e est nettoyée lors de l'étape t_2, et (3) pour toute arête x qui a été nettoyée entre les étapes t_1 et t_2, x a été recontaminée (i.e., le nettoyage de x a été annulé). Soit $taille(v,t)$ le nombre de traces qui sont écrites sur le tableau blanc du sommet v à l'étape t de l'exécution du protocole `Nettoyage_réparti`. Nous prouvons que, pour tout sommet $v \in V(G)$, $taille(v,t_2) - taille(v,t_1) = O(\log n)$.

Supposons que le nettoyage de e consiste en la séquence de mouvements suivante. Le chercheur a_i, $1 \leq i \leq k$, rejoint un autre chercheur a_j ($1 \leq j < i$), et le chercheur a_j nettoie une arête e. Bien sûr, ce cas est celui pour lequel le nombre de traces est le plus grand possible. Soit v_i (resp., v_j) le sommet occupé par a_i (resp., a_j) après le nettoyage de f (notons que, si t_1 est la première étape, $v_i = v_j = v_0$). Si t_1 n'est pas la première étape, soit a_ℓ, $\ell \geq 1$, le chercheur qui a nettoyé f. Après avoir nettoyé l'arête f, a_ℓ envoie le message $(\mathtt{bouge}, \ell)$ à a_1. Alors, pour tout t, $1 \leq t \leq i-1$, le message $(\mathtt{bouge}, t)$ est transis de l' chercheur a_t au chercheur a_{t+1} jusqu'à ce que le message $(\mathtt{bouge}, i-1)$ atteigne a_i. En modifiant légèrement la procédure *prochain_chercheur*, si plus de deux chercheurs sont sur le même sommet, nous pouvons nous assurer que seuls les chercheurs avec les deux plus petits identifiants reçoivent le message. Ce n'est en effet pas nécessaire d'envoyer le message aux autres chercheurs. En effet, si les deux plus petits ne peuvent rien faire, les autres ne le pourront pas non plus. Donc, entre le nettoyage de deux arêtes f et e, au plus deux traces de type (RT, ℓ, s) et deux traces de type (ST, ℓ, s) sont écrites sur chaque tableaux blancs.

Soit s_1 l'étape de l'exécution du protocole `Nettoyage_réparti` lorsque a_i reçoit $(\mathtt{bouge}, i-1)$ du chercheur a_{i-1}. Soit s_2 l'étape à laquelle a_i décide d'essayer d'aider a_j. Après l'étape s_1, le chercheur a_i essaie d'abord d'aider tous les chercheurs a_p, $p < j$. Puisque tous ces mouvements ont été annulés, toutes les traces correspondantes qui ont été écrites par les chercheurs entre les étapes s_1 et s_2 ont été effacées de même.

A l'étape s_2, le chercheur a_i décide de rejoindre a_j et empile (JJ, i, j) sur v_i. En rejoignant a_j, a_i empile une trace (AH, i, j) sur chaque tableau blanc sur le chemin entre v_i et v_j. Finalement, a_i envoie le message $(\mathtt{aide}, i)$ à a_j qui nettoie l'arête e. C'est-à-dire que le chercheur a_i empile (ST, i, j) sur v_j. Alors, le chercheur a_j empile (RT, j, i) et (CJ, j, p) sur v_j. Finalement a_j nettoie l'arête e et, il empile (AC, j, q) et $(ST, j, 1)$ à l'autre extrémité de e.

Pour résumer, sur chaque tableau blanc, ont été écrites au plus trois traces de type (RT, ℓ, s), trois de type (ST, ℓ, s), une de chacun ds types (JJ, i, j), (AH, i, j), (CJ, j, p) et (AC, j, q). Donc, lorsqu'un mouvement large est réalisé, au plus $O(1)$ traces sont écrites sur chaque tableau blanc.

Puisque il y a m mouvements larges, au plus $O(m \ \log n)$ bits sont écrits sur chaque tableau blanc. □

5.6 Perspectives

Les questions que pose cette section visent toutes à l'amélioration des performance de l'algorithme `Nettoyage_réparti`. Tout d'abord, notre algorithme implique des chercheurs dont la mémoire est de taille $\log k$ bits avec k le nombre maximum de chercheurs impliqués. Est-il possible de concevoir un algorithme qui résout le problème du nettoyage réparti en impliquant des automates finis ? Il serait également intéressant d'améliorer notre protocole de manière à ce que la mémoire qui doit être disponible sur les tableaux blancs ait une taille inférieure, par exemple, une taille logarithmique en la taille du réseau. Enfin, le dernier problème, et non le moindre, que nous posons ici, serait de concevoir un algorithme plus rapide que l'algorithme `Nettoyage_réparti`. Ce dernier peut en effet avoir un temps d'exécution exponentiel. La meilleure borne supérieure que nous ayons en ce qui concerne ce temps d'exécution est la borne triviale qui consiste en le nombre de piles possibles à tester.

Chapitre 6

Stratégies de capture répartie avec conseil

Ce chapitre est comme le précédent consacré au problème du nettoyage réparti. Les chercheurs doivent nettoyer un graphe de manière connexe monotone, de façon décentralisée. La différence avec le chapitre précédent vient donc de ce que les chercheurs doivent nettoyer le graphe de manière monotone. Pour les y aider, il leur est fourni une certaine information relative au graphe.

6.1 Introduction

Dans le chapitre précédent, nous avons prouvé qu'il était possible de calculer une stratégie de capture connexe monotone optimale d'un graphe inconnu de manière répartie. Cependant, le nettoyage proprement dit du graphe n'est pas monotone et peut prendre un temps exponentiel. Pour pallier cet inconvénient, nous nous proposons, dans ce chapitre, de fournir aux chercheurs certaines informations concernant le graphe.

Dans de nombreux problèmes répartis, la qualité des solutions dépend des informations relatives au réseau données *a priori* aux sommets du réseau (voir [FR03] pour un tour d'horizon). La comparaison de deux algorithmes avec connaissance préalable sur le réseau se révèle cependant être ardue dès que ces informations diffèrent qualitativement : une borne sur la taille du réseau [BFR$^+$02] par exemple, ou la topologie complète du réseau [CMS01], etc. Dans [FIP06], Fraigniaud *et al.* définissent la notion de conseil, de façon à mesurer quantitativement la difficulté d'une tâche distribuée. Comme application, Fraigniaud *et al.* étudient la quantité d'information qui doit être distribuée sur les sommets d'un graphe de manière à réaliser efficacement la diffusion et le réveil (i.e., en utilisant un nombre minimum de messages). Ils prouvent que le nombre minimum de bits d'information permettant de résoudre le problème du réveil (resp., de diffusion) en un nombre linéaire de messages, est $\Theta(n \log n)$ (resp., $O(n)$) bits. Cette approche permet donc de différencier quantitativement la difficulté des deux problèmes pourtant très semblables. Dans ce chapitre, nous répondons à la question suivante. Quel est le nombre minimum de bits d'information nécessaire pour résoudre le problème du nettoyage connexe monotone réparti dans les graphes ?

Le modèle que nous utiliserons dans ce chapitre est le même que celui présenté à la section 1.3.5, et utilisé au chapitre précédent (cf., section 5.2.1). Dans le but de simplifier la présentation, les mouvements des chercheurs sont synchrones. De la même façon que dans le chapitre précédent, le cas asynchrone peut être traité en utilisant un chercheur supplémentaire pour coordonner les chercheurs. La seule différence avec le chapitre précédent vient du fait que pour décider de leur prochaine action à effectuer, les chercheurs peuvent s'aider d'une information fournit par un oracle sur chaque sommet du graphe. Dans le paragraphe suivant, nous définissons formellement la notion de conseil, ainsi que le problème du nettoyage connexe monotone réparti avec conseil.

Une instance du problème consiste en un couple (G, v_0), avec G un graphe connexe et $v_0 \in V$ la base. Un *oracle* [FIP06] est une fonction $\mathcal{O}$ qui, à toute instance $(G, v_0, \mathcal{L})$, associe une fonction $f : V \to \{0,1\}^*$ attribuant une chaîne binaire, appelée *chaîne de bits d'information*, à chaque sommet de G. La *quantité d'information*, ou *taille de conseil*, ou encore *le nombre de bits d'information*, pour une instance donnée, est la somme des longueurs de toutes les chaînes attribuées aux sommets. Intuitivement, un oracle fournit de la connaissance supplémentaire aux sommets du réseau. Résoudre le problème du nettoyage connexe monotone réparti avec conseil consiste à concevoir un oracle $\mathcal{O}$ et un protocole $\mathcal{P}$ utilisant $\mathcal{O}$, avec les caractéristiques suivantes. Pour toute instance (G, v_0), tout sommet $v \in V$ est fourni avec la chaîne $f(v)$, $f = \mathcal{O}(G, v_o)$. Le protocole $\mathcal{P}$ doit alors permettre au nombre optimal de chercheurs de nettoyer G, en partant de v_0, de manière connexe monotone. De plus, la stratégie des chercheurs est calculée de façon distribuée. La seule information relative au graphe dont dispose les chercheurs est modélisée par la chaîne de bits disponible localement sur chaque sommet.

Dans ce chapitre, nous prouvons que $O(n \log n)$ bits d'information suffisent pour résoudre le problème du nettoyage connexe monotone réparti avec conseil dans tous les graphes de n sommets. Nous prouvons de plus que cette quantité d'information est nécessaire pour nettoyer certains graphes. La section 6.2 est consacrée à la description d'un oracle de taille $O(n \log n)$ bits et d'un protocole utilisant cet oracle qui résout le problème de l'encerclement réparti avec conseil. La section 6.3 est consacrée à la preuve de l'optimalité de cette borne. Plus précisément, nous prouvons qu'aucun protocole utilisant $o(n \log n)$ bits de conseil ne peut résoudre le problème du nettoyage connexe monotone réparti dans tous les graphes.

Les résultats de ce chapitre ont été réalisés en collaboration avec David Soguet. Ils ont donné lieu à une publication dans la conférence SIROCCO 2007 [NS07].

6.2 L'algorithme et l'oracle

Dans cette section, nous résolvons le problème du nettoyage connexe monotone réparti avec conseil. Plus précisément, nous prouvons le théorème suivant :

Théorème 27 *Le problème du nettoyage connexe monotone réparti peut être résolu en utilisant $O(n \log n)$ bits d'information.*

Pour prouver le théorème 27, nous décrivons un oracle $\mathcal{O}$ de taille $O(n \log n)$ bits, et un protocole réparti `Nettoyeur` qui résout le problème du nettoyage connexe monotone réparti dans un environnement synchrone. Le protocole `Nettoyeur` réalise l'exécution d'une stratégie de capture divisée en n phases, chacune divisée en deux tours de $O(n^2)$ rondes. Le nettoyage de G est donc réalisé en temps $O(n^3)$.

6.2.1 L'oracle

Nous décrivons tout d'abord l'oracle $\mathcal{O}$. Pour toute instance $(G = (V, E), v_0)$ du problème du nettoyage connexe monotone réparti, nous considérons une stratégie S qui est solution du problème (S est donc connexe monotone et optimale). La fonction $f = \mathcal{O}(G, v_0)$ est définie à partir de S. Informellement, les chaînes de bits d'information fournies par $\mathcal{O}$ permettent à des chercheurs utilisant le protocole `Nettoyeur`, de nettoyer l'ensemble des sommets de G dans le même ordre que S. De plus, elles permettent aux chercheurs de circuler dans la partie propre du graphe tout en évitant la recontamination. Définissons les notations que nous allons utiliser dans la suite.

Rappelons que $n = |V|$ et $m = |E|$. La stratégie S peut être définie par l'ordre dans lequel S nettoie les arêtes. Soit $(e_1, \cdots, e_m)$ cet ordre. Une arête e_i est plus petite qu'une arête e_j, noté $e_i \preceq e_j$, si $i \leq j$. S induit également un ordre sur les sommets de G. Pour tout $v, w \in V$, nous dirons que v est plus petit que w, noté $v \preceq w$, si la première arête incidente à v qui est nettoyée, est plus petite que la première arête nettoyée incidente à w. Soit $(v_0, \cdots, v_{n-1})$ cet ordre, c'est-à-dire $v_i \preceq v_j$ si et seulement si $i \leq j$.

Pour tout i, $0 \leq i \leq n-1$, soit $f_i \in E$ la première arête incidente à v_i qui est nettoyée par S. Du fait de la connexité et la monotonie de S, $f_0 = f_1 \prec f_2 \cdots \prec f_{n-1}$. Pour tout i, $1 \leq i \leq n-1$, le *père* de v_i, noté $parent(v_i)$, est défini comme étant le voisin v de v_i tel que, $\{v, v_i\} = f_i$. Notons que $parent(v_i) \prec v_i$, et pour tout voisin v de v_i, $f_i = \{parent(v_i), v_i\} \preceq \{v, v_i\}$. Intuitivement, pour tout i, $1 \leq i \leq n-1$, $f_i = \{parent(v_i), v_i\}$ est l'arête par laquelle un chercheur est arrivé sur v_i pour la première fois. Inversement, les fils de $v \in V$ sont les sommets w tels que $v = parent(w)$. Pour tout i, $0 \leq i \leq n-1$, soit T_i le sous-graphe de G dont l'ensemble de sommets est $\{v_0, \cdots, v_i\}$, et l'ensemble d'arêtes est $\{f_1, \cdots, f_i\}$. Pour tout i, $0 \leq i \leq n-1$, T_i est un arbre couvrant de $G[v_0, \cdots, v_i]$. Intuitivement, à la phase i de l'exécution du protocole `Nettoyeur`, T_{i-1} est un arbre couvrant de la partie propre du graphe. Cet arbre sera utilisé pour permettre aux chercheurs de se déplacer dans la partie propre, en réalisant un parcourt en profondeur d'abord (DFS pour Deep First Search) de T_{i-1}.

Nous définissons maintenant un étiquetage local des sommets de G. Insistons une fois de plus sur le fait que cet étiquetage dépend de la stratégie S qui est considérée. Soit $v \in V(G)$. L'étiquette du sommet v est composée des variables locales suivantes : une variable booléenne $\texttt{TYPE}_v$, quatre entiers $\texttt{TCU}_v$, $\texttt{TTL}_v$, $\texttt{Dernier\_Port}_v$, $\texttt{PARENT}_v$ et une liste ordonnée $\texttt{Fils}_v$ de paires d'entiers. L'indice sera omis lorsqu'aucun risque de confusion n'aura lieu.

Intuitivement, $\texttt{PARENT}_v$ et $\texttt{Fils}_v$ permettent aux chercheurs d'effectuer un DFS d'un sous-arbre couvrant la partie propre. Pour éviter la recontamination, les chercheurs doivent non seulement savoir quels ports ils peuvent emprunter ou non, mais également le moment auquel de tels mouvements sont possibles, c'est-à-dire la phase du protocole à laquelle un chercheur peut prendre un certain port. Les informations concernant les numéros de port (possible ou non) sont portées par $\texttt{PARENT}_v$, $\texttt{Fils}_v$, et $\texttt{Dernier\_Port}_v$. $\texttt{Fils}_v$, $\texttt{TCU}_v$ et $\texttt{TTL}_v$ portent les informations concernant les phases auxquelles ces ports peuvent être empruntés. De plus, si un chercheur préserve un sommet de la recontamination, nous dirons que ce chercheur *garde* le sommet, sinon, nous dirons qu'il est *libre*. Un chercheur qui garde un sommet v quittera v par la plus grande arête qui lui est incidente. La phase à laquelle un tel mouvement sera effectué est telle que toutes les autres arêtes incidentes à v seront propres. Donc aucune recontamination n'aura lieu. Dans ce but, nous avons besoin de distinguer deux types de sommets grâce à $\texttt{TYPE}_v$.

Dans ce qui suit, nous dirons d'un numéro de port p d'un sommet v (resp., d'une arête incidente à v, correspondant à p) qu'il (elle) est *étiqueté(e)* s'il existe $\ell \leq n-1$ tel que $(p, \ell) \in \texttt{Fils}_v$, ou $p = \texttt{Dernier\_Port}_v$, ou $p = \texttt{PARENT}_v$. Notons qu'une arête peut avoir deux étiquettes différentes, ou une étiquette à l'une de ses extrémités et pas à l'autre, ou encore aucune étiquette du tout. Soit $0 \leq i \leq n-1$, l'entier tel que $v = v_i$. Soit e la plus grande arête incidente à v qui n'appartient pas à $E(T_{n-1})$, et soit f la plus grande arête incidente à v qui n'appartient pas à $(T_{n-1}) \cup \{e\}$.

- $\texttt{PARENT}_v$ est le numéro de port de v menant à $parent(v)$ par une arête de $E(T_{n-1})$ (nous posons $\texttt{PARENT}_{v_0} = -1$).
- $\texttt{Fils}_v$ est une liste ordonnée de paires d'entiers. Soit $1 \leq p \leq \deg(v)$ et $0 < j \leq n-1$. $(p, j) \in \texttt{Fils}_v$ si et seulement si $v = parent(v_j)$ et p est le numéro de port de v menant à v_j. Dans la suite, $\texttt{Fils}_v(j)$ désigne le numéro de port p de v tel que $(p, j) \in \texttt{Fils}_v$.
- $\texttt{TYPE}_v$ est une variable booléenne. Elle vaut 0 si la plus grande arête incidente à v appartient à T_{n-1}. Sinon, la variable $\texttt{TYPE}_v$ vaut 1. Dans la suite, nous dirons qu'un sommet est de type 0 (resp., type 1) si $\texttt{TYPE}_v = 0$ (resp., $\texttt{TYPE}_v = 1$). Intuitivement, un sommet est de type 0 si, dans S, le chercheur qui nettoie la dernière arête contaminée incidente à ce sommet le fait en atteignant un nouveau sommet qui n'avait pas encore été occupé.
- $\texttt{Dernier\_Port}_v = -1$ si $\texttt{TYPE}_v = 0$, sinon $\texttt{Dernier\_Port}_v$ est le numéro de port correspondant à e.
- $\texttt{TCU}_v$ (pour *Time to Clean Unlabelled port*), représente la phase à laquelle les chercheurs libres doivent nettoyer tous les ports non étiquetés de v. Cas $\texttt{TYPE}_v = 0$: si e n'existe pas, alors $\texttt{TCU}_v = -1$, sinon $\texttt{TCU}_v$ est le plus grand entier k tel que $f_{k-1} \preceq e$. Cas $\texttt{TYPE}_v = 1$: si f n'existe pas, alors $\texttt{TCU}_v = -1$, sinon $\texttt{TCU}_v$ est le plus grand entier k tel que $f_{k-1} \preceq f$.
- $\texttt{TTL}_v$ (pour *Time To Leave*), représente la phase à laquelle un chercheur qui garde v le quittera. Cas $\texttt{TYPE}_v = 0$: $\texttt{TTL}_v = j$ tel que v_j est le plus grand fils de v. Cas $\texttt{TYPE}_v = 1$: $\texttt{TTL}_v$ est le plus grand entier k tel que $f_{k-1} \leq e$.

Nous définissons maintenant la fonction d'étiquetage $\mathcal{O}(G, v_0)$ fournie par l'oracle $\mathcal{O}$

à G, utilisant $\mathcal{L}(S)$. Pour tout i, $0 \leq i \leq n-1$,

$$\mathcal{O}(G, v_0)(v_i) = (i, n, \mathtt{TYPE}_{v_i}, \mathtt{PARENT}_{v_i}, \mathtt{Dernier\_Port}_{v_i}, \mathtt{TCU}_{v_i}, \mathtt{TTL}_{v_i}, \mathtt{Fils}_{v_i}).$$

Le lemme suivant découle directement de la définition de l'oracle.

Lemme 18 *Pour tout graphe de n sommets, le conseil fournit par $\mathcal{O}$ est de taille $O(n \log n)$ bits.*

6.2.2 Le protocole `Nettoyeur`

Dans cette section, nous définissons le protocole de nettoyage connexe monotone réparti `Nettoyeur` utilisant l'oracle $\mathcal{O}$, qui permet de nettoyer tout réseau synchrone G de n sommets, en partant de la base v_0. Le protocole `Nettoyeur` est décrit formellement sur les figures 6.1 et 6.2.

Décrivons succinctement notre protocole. Les chercheurs peuvent être dans sept états différents : DFS_TEST, DFS_RETOUR, NETTOIE-NON-ÉTIQUETÉE, RETOUR-NON-ÉTIQUETÉE, NETTOIE, ATTENDRE, GARDER. Initialement, tous les chercheurs se situent sur v_0. Chacun d'eux lit n sur l'étiquette $\mathcal{O}(G, v_0)(v_0)$ de v_0 pour initialiser ses compteurs. Alors, le chercheur avec le plus grand identifiant est élu pour garder v_0 et passe dans l'état GARDER, les autres chercheurs deviennent libres et passent dans l'état DFS_TEST. Après la phase $1 \leq i \leq n-1$, notre protocole assure les propriétés suivantes.

1. Un sous-graphe G' de $G[v_0, \cdots, v_i]$ contenant T_i comme sous-graphe est propre.
2. Pour tout sommet v de la frontière de G', un chercheur garde v (dans l'état GARDER).
3. Tous les autres chercheurs sont libres et se trouvent sur des sommets de G'.

Premier tour de la phase $i+1$. Le but des chercheurs libres est de nettoyer les arêtes non-étiquetées des sommets v de $V(G[v_0, \cdots, v_i])$ tels que la plus grande arête non-étiquetée e incidente à v satisfait $f_i \prec e \prec f_{i+1}$. Notons qu'une telle arête e appartient à $E(G[v_0, \cdots, v_i])$. Dans ce but, tout chercheur libre effectue un DFS de T_i guidé par les étiquettes `PARENT` et `Fils`. Le chercheur est dans l'état DFS_TEST si il “descend” dans l'arbre couvrant, c'est-à-dire s'il se déplace vers les feuilles, et dans l'état DFS_RETOUR sinon.

Au cours de ce DFS, si le chercheur rencontre un sommet v_j avec $j \leq i$ et étiqueté par $\mathtt{TCU}_{v_j} = i+1$ (rappelons que `TCU` signifie *Time to Clear Unlabelled edges*, i.e., l'heure de nettoyer les arêtes non-étiquetées), alors le chercheur nettoie toutes les arêtes non-étiquetées de v_j, et poursuit ensuite le DFS. Pour nettoyer les arêtes non-étiquetées incidentes à v_j, le chercheur prend successivement, dans l'ordre croissant des numéros de port, tous les ports non-étiquetés. Il effectue alors un aller-retour sur chacune de ces arêtes, en étant dans l'état NETTOIE-NON-ÉTIQUETÉE à l'aller, et dans l'état RETOUR-NON-ÉTIQUETÉE au retour.

Programme d'un chercheur A.

Initialisation : /* tous les chercheurs partent de v_0 */
Lire n sur $\mathcal{O}(G, v_0)$ pour initialiser le conteur ;
si A est le chercheur avec le plus grand identifiant sur v_0 **alors**
Passer dans l'état GARDER ;
sinon
A la première ronde du second tour de la phase 1,
Passer dans l'état DFS_TEST ;
fin si

Programme d'un chercheur A lors de toute ronde du tour $s \in \{0,1\}$ de la phase $1 \leq i \leq n$.

/* Cas où le chercheur A arrive sur le sommet v_j, provenant de l'arête $\{v_\ell, v_j\}$, de numéro de port p_ℓ de v_j */

Soit p_{debut} le plus petit numéro de port non-étiqueté de v_j.
$p_{debut} = -1$ si il n'existe pas de telle arête.
Soit $p_{suivant}$ le plus petit numéro de port p de v_j, tel que $p > p_\ell$.
$p_{suivant} = -1$ si il n'existe pas de telle arête.

Soit $p_{firstchild}$ le numéro de port p de v_j tel qu'il existe $1 \leq k \leq n-1$ avec p étiqueté `Fils`(k), et, pour tout k', $1 \leq k' < k$, aucune arête incidente à v_j n'est étiquetée `Fils`(k'). $p_{firstchild} = -1$ si il n'existe pas de telle arête.
Si $p_{firstchild} \neq -1$, $premierFils$ désigne le voisin de v_j correspondant.

Soit $p_{nextchild}$ le numéro de port p de v_j tel qu'il existe $\ell < k \leq n-1$ avec p étiqueté `Fils`(k), et, pour tout $\ell \leq k' < k$, aucune arête incidente à v_j n'est étiquetée `Fils`(k'). Si $p_{nextchild} \neq -1$, $filsSuivant$ désigne le voisin de v_j correspondant.

`Cas :`
`état` = DFS_TEST
si $s = 1$ et il existe un numéro de port p étiqueté `Fils`(i) **alors**
Emprunter le port p dans l'état NETTOIE ;
sinon si $s = 0$ et `TCU` $= i$ **alors**
Emprunter le port p_{debut} dans l'état NETTOIE-NON-ÉTIQUETÉE ;
sinon si $p_{premierFils} \neq -1$ et $premierFils \preceq v_{i-1}$ **alors**
Emprunter le port $p_{premierFils}$ dans l'état DFS_TEST ;
sinon Emprunter le port étiqueté `PARENT` dans l'état DFS_RETOUR ; **fin si**
fin si
fin si

`état` = RETOUR-NON-ÉTIQUETÉE
si $p_{suivant} \neq -1$ **alors**
Emprunter le port $p_{suivant}$ dans l'état NETTOIE-NON-ÉTIQUETÉE ;
sinon si $p_{premierFils} \neq -1$ et $premierFils \preceq v_{i-1}$ **alors**
Emprunter le port $p_{premierFils}$ dans l'état DFS_TEST ;
sinon Emprunter le port étiqueté `PARENT` dans l'état DFS_RETOUR ; **fin si**
fin si

FIG. 6.1 – Protocole `Nettoyeur` (1/2)

`état` = NETTOIE-NON-ÉTIQUETÉE
Emprunter le port p_ℓ dans l'état RETOUR-NON-ÉTIQUETÉE ;

`état` = DFS_RETOUR
si $s = 1$ et il existe un numéro de port p étiqueté `Fils`(i) **alors**
Emprunter le port p dans l'état NETTOIE ;
sinon si $p_{filsSuivant} \neq -1$ and $filsSuivant \preceq v_{i-1}$ **alors**
Emprunter le port $p_{filsSuivant}$ dans l'état DFS_TEST ;
sinon si `PARENT` $\neq -1$ **alors**
Emprunter le port étiqueté `PARENT` dans l'état DFS_RETOUR ;
sinon Emprunter le port `Fils`(1) dans l'état DFS_TEST ; **fin si**
fin si
fin si

`état` = NETTOIE
si $v_j \prec v_i$ ou $\deg(v_j) = 1$ **alors**
si $j > 0$ **alors**
Emprunter le port étiqueté `PARENT` dans l'état DFS_RETOUR ;
sinon Emprunter le port étiqueté `Fils`(1) dans l'état DFS_TEST ;
fin si
sinon Passer dans l'état ATTENDRE ;
fin si

/* Cas où le chercheur est situé sur le sommet v_j */

`état` = GARDER
si `TYPE` $= 1$ **alors**
si `TCU` = `TTL` **alors**
A la première ronde du second tour de la phase `TTL`,
Emprunter le port étiqueté `Dernier_Port` dans l'état NETTOIE ;
sinon A la première ronde du second tour de la phase `TTL`,
Emprunter le port étiqueté `Dernier_Port` dans l'état NETTOIE ;
fin si
sinon A la première ronde du second tour de la phase`TTL`
Emprunter le port étiqueté `Fils(TTL)` dans l'état NETTOIE ;
fin si

`état` = ATTENDRE
A la dernière ronde de cette phase :
si A est le chercheur avec le plus grand identifiant sur v_j **alors**
Passer dans l'état GARDER ;
sinon Emprunter le port étiqueté `PARENT` dans l'état DFS_RETOUR ;
fin si

fin

FIG. 6.2 – Protocole `Nettoyeur` (2/2)

De plus, pendant ce tour, tout chercheur qui est en train de garder un sommet étiqueté $\texttt{TYPE} = 1$ avec $\texttt{TCU} < \texttt{TTL} = i+1$, doit prendre (et nettoyer), dans l'état NETTOIE, l'arête ayant pour numéro de port `Dernier_Port` sur le sommet considéré (rappelons que `TTL` signifie *Time To Leave*, i.e., l'heure de partir). Le protocole `Nettoyeur` assure que l'arête correspondante est la seule arête incidente à ce sommet qui est contaminée à ce tour.

Avant la première ronde du second tour de la phase $i+1$, les deux propriétés suivantes sont satisfaites : (1) s'il existe un sommet v étiqueté avec $\texttt{TYPE}_v = 0$ et $\texttt{TTL}_v = i+1$, alors f_{i+1} est la seule arête incidente à $v = parent(v_{i+1})$ qui est encore contaminée, et (2) pour tout sommet v étiqueté $\texttt{TYPE}_v = 1$ avec $\texttt{TCU}_v = \texttt{TTL}_v = i+1$, l'arête ayant pour numéro de port $\texttt{Dernier\_Port}_v$ est la seule arête incidente à v qui est encore contaminée.

Second tour de la phase $i+1$. Le protocole `Nettoyeur` effectue le nettoyage de f_{i+1} (incidente à $parent(v_{i+1}) \in V(G')$) ainsi que le nettoyage de toute arête correspondant au numéro de port $\texttt{Dernier\_Port}_{v_j}$ des sommets v_j pour $j \leq i$, étiquetés $\texttt{TYPE}_{v_j} = 1$ avec $\texttt{TCU}_{v_j} = \texttt{TTL}_{v_j} = i+1$. Dans ce but, tout chercheur libre réalise un DFS de T_i. Lorsqu'un chercheur rencontre le sommet $parent(v_{i+1})$ dont un port est étiqueté $\texttt{Fils}(i+1)$, il emprunte l'arête correspondante dans l'état NETTOIE. De plus, le chercheur qui garde le sommet $parent(v_{i+1})$ emprunte également l'arête ayant pour numéro de port $\texttt{Fils}(i+1)$ dans l'état NETTOIE si $\texttt{TTL} = i+1$ et $\texttt{TYPE} = 0$. Finalement, tout chercheur qui garde un sommet étiqueté $\texttt{TYPE} = 1$ avec $\texttt{TCU} = \texttt{TTC} = i+1$ emprunte l'arête ayant pour numéro de port `Dernier_Port` dans l'état NETTOIE. Pendant ce tour, tout chercheur qui arrive sur v_{i+1} attend (dans l'état ATTENDRE) la dernière ronde de la phase si $\deg(v_{i+1}) > 1$. Sinon, il devient libre. Pendant cette dernière ronde, si $\deg(v_{i+1}) > 1$, alors le chercheur avec le plus grand identifiant se trouvant sur v_{i+1} est élu pour garder v_{i+1} pendant que les autres chercheurs deviennent libres et empruntent le port étiqueté `PARENT` dans l'état DFS_RETOUR.

Nous prouvons à présent que le protocole `Nettoyeur` et l'oracle $\mathcal{O}$ sont bien solution du problème du nettoyage monotone réparti. Brièvement, nous prouvons par induction sur $1 \leq i \leq n$ que, après la phase i, les sommets nettoyés (dont toutes les arêtes incidentes sont propres) par S (la stratégie à partir de laquelle l'oracle est défini) avant le nettoyage de f_i, sont aussi nettoyés par le protocole `Nettoyeur` de même.

Preuve du Théorème 27. Dans ce qui suit, un chercheur sera dit *libre* s'il n'est ni dans l'état GARDER ni dans l'état ATTENDRE. Pour énoncer l'assertion suivante, nous avons besoin d'une nouvelle définition. Pour tout i, $0 \leq i \leq n-1$, soit $M_i = \{v \in V(G) \mid \textit{pour toute arête } e \textit{ incidente à } v,\ e \preceq f_i\}$. $M_i \subseteq V(T_i)$ est l'ensemble des sommets dont toutes les arêtes incidentes, sauf f_i, ont été nettoyées par S avant l'étape correspondant au nettoyage de f_i. Nous posons de plus $M_n = V$. Donc, après cette étape, aucun sommet de M_i n'a besoin d'être gardé dans la stratégie S. Notons que, pour tout j, $0 \leq j \leq n-1$, l'ensemble $M_j \setminus M_{j-1}$ est exactement l'ensemble des sommets v tels que $\texttt{TTL} = j$. Nous pouvons l'assertion suivante.

Assertion 16 *Soit G un graphe connexe et $v_0 \in V(G)$. Soit S une stratégie de capture monotone et connexe pour G, partant de v_0, et utilisant $mcs(G, v_0)$ chercheurs. Soit $\mathcal{O}(G, v_0)$ l'étiquetage des sommets de G, fournit par un oracle utilisant S. Après* *la dernière ronde de la phase $i \geq 1$ de l'exécution du protocole* `Nettoyeur`*, la partie propre du graphe G satisfait les propriétés suivantes :*

1. *toute arête dans $\{f_0, \cdots, f_i\}$ est propre,*
2. *toute arête incidente à un sommet dans M_i est propre,*
3. *il y a exactement un chercheur dans l'état* GARDER *sur chaque sommet de $V(T_i) \setminus M_i$,*
4. *tous les autres chercheurs sont libres et occupent un sommet de T_i,*
5. *pour tout sommet v tel que* `TCU` *$\leq i$, toute arête non-étiquetée incidente à v est propre.*

Nous prouvons cette assertion par induction sur $1 \leq i \leq n$. Le cas $i = 1$ peut être facilement vérifié. Supposons que le résultat est valide pour tout i, $1 \leq i \leq n-1$. Nous prouvons qu'il reste vrai après la dernière ronde de la phase $i+1$. Nous considérons deux cas, selon qu'un chercheur est libre ou non.

Cas 1 : *Supposons qu'aucun chercheur n'est libre.* C'est-à-dire, tout chercheur occupe seul un sommet de $V(T_i) \setminus M_i$ dans l'état GARDER. Donc, d'après le point 3 de l'hypothèse d'induction, $|V(T_i) \setminus M_i| = mcs(G, v_0)$. Soit s_i l'étape de la stratégie S à laquelle l'arête f_i est nettoyée. Après l'étape s_i, chaque sommet de $V(T_i) \setminus M_i$ est occupé par au moins un chercheur. Puisque $|V(T_i) \setminus M_i| = mcs(G, v_0)$, pour tout sommet v de $V(T_i) \setminus M_i$, exactement un chercheur occupe v dans la situation atteinte à l'étape s_i de S. Soit e l'arête nettoyée par S à l'étape $s_i + 1$ en déplaçant un chercheur à partir d'un sommet v le long de e. A l'étape s_i de la stratégie S, e est la dernière arête contaminée incidente à v. Sinon, e ne pourrait pas avoir été nettoyée puisque un seul chercheur occupe chaque sommet de la frontière de la partie propre du graphe. Encore, $v \in V(T_i) \setminus M_i$, donc le point 3 de l'hypothèse d'induction stipule qu'il y a un chercheur, appelons le A, dans l'état GARDER sur le sommet v après la dernière ronde de l'exécution de la phase i du protocole `Nettoyeur`. Deux cas peuvent se produire :

– $e = f_{i+1}$: c'est-à-dire que $v = parent(v_{i+1})$. Dans ce cas, puisque e est la dernière arête contaminée, incidente à v, qui est nettoyée par S, le sommet v est de type 0. De plus, $\texttt{TCU}_v < i+1$ et pour tout j, $i+1 < j \leq n-1$, f_j n'est pas incidente à v. Donc, d'après le point 5, toute arête non-étiquetée incidente à v est propre. De plus, d'après le point 1, pour tout $0 \leq j \leq i$, f_j a déjà été nettoyée. Donc, après la dernière ronde de la phase i de l'exécution du protocole `Nettoyeur`, e est la seule arête incidente à v qui soit contaminée. Au cours de l'exécution de la phase $i+1$ du protocole `Nettoyeur`, un seul mouvement est opéré : le chercheur A sur v nettoie l'arête f_{i+1} lors de la première ronde du tour 2 de cette phase. Puisque e est la seule arête incidente à v qui était contaminée, aucune recontamination ne se produit. Si $\deg(v_{i+1}) = 1$, $M_{i+1} = M_i \cup \{v, v_{i+1}\}$. Dans ce cas, v_{i+1} a été nettoyé par le protocole `Nettoyeur` et le chercheur A devient libre. Remarquons que, en devenant libre, ce chercheur effectue un DFS de T_{i+1}, et donc n'entraîne aucune recontamination. Les points $1, 2, 3$ et 4 de l'assertion sont vérifiés. Si $\deg(v_{i+1}) \neq 1$, $M_{i+1} = M_i \cup \{v\}$, et, lors de la dernière ronde de la phase $i+1$, le chercheur A passe dans l'état GARDER sur le sommet v_{i+1}. Encore, les points $1, 2, 3$ et 4 de l'assertion sont vérifiés. De plus, puisqu'il n'y a pas d'arête ℓ telle que $f_i \prec \ell \prec f_{i+1}$, aucun sommet n'est étiqueté avec $\texttt{TCU} = i+1$. Ainsi, le point 5 de l'assertion est vérifié.

– $e \prec f_{i+1}$: soit W l'ensemble de sommets de T_i tels que $\texttt{TYPE} = 1$ avec $\texttt{TCU} < \texttt{TTL} = i+1$. Dans ce cas, le sommet v est de type 1 avec $\texttt{TTL}_v = i+1$ et $\texttt{Dernier\_Port}_v \neq -1$ est le numéro de port correspondant à e. Puisqu'il n'y a aucune arête $f_i \prec \ell \prec e$ et e est la dernière arête incidente à v nettoyée par S, $\texttt{TCU}_v < i+1$. Donc, $v \in W \neq \emptyset$. Soit $w \in W$.

Pour tout j, $i+1 \leq j \leq n-1$, f_j n'est pas incidente à w. D'après le point 5, toutes les arêtes non-étiquetées incidentes à w sont propres. D'après le point 1, pour tout j, $0 \leq j \leq i$, f_j a été nettoyée. Donc, après la dernière ronde de la phase i de l'exécution du protocole `Nettoyeur`, il y a exactement une arête contaminée qui est incidente à w et le numéro de port correspondant est étiqueté $\texttt{Dernier\_Port}_w \neq -1$. Soit e_w cette arête. Notons que $f_i \prec e_w \prec f_{i+1}$. D'après le point 3, après la dernière ronde de la phase i, il y a un chercheur A_w dans l'état GARDER sur w. Pendant la première ronde du tour 1 de la phase $i+1$, le chercheur A_w nettoie l'arête e_w. Aucune recontamination ne se produit puisque e_w est la seule arête contaminée incident à w. Le chercheur A_w arrive dans l'état NETTOIE sur un sommet u de T_i. Puisque $f_i \prec e_w$, $u \notin M_i$. Donc, lors de la dernière ronde de la phase i, le sommet u était gardé par un autre chercheur A_u dans l'état GARDER. Nous considérons deux cas :

– $u \notin W$. Le chercheur A_u est toujours dans l'état GARDER sur u.
– $u \in W$. Alors, $e_u = e_w$. Donc, lors de la première ronde du premier tour de la phase $i+1$, le chercheur A_u s'est déplacé le long de cette arête. Dans ce cas, u est propre.

Dans les deux cas, le chercheur A_w devient libre et quitte le sommet u par le port $\texttt{PARENT}_u$ dans l'état DFS_RETOUR, ou par le port $\texttt{Fils}_u(1)$ dans l'état DFS_TEST si $u = v_0$. Puisque u est propre ou gardé, aucune recontamination ne se produit.

Prouvons que durant les rondes suivantes du premier tour de la phase $i+1$, A_w effectue un DFS de T_i. De plus, nous prouvons que le chercheur A_w nettoie toutes les arêtes non-étiquetées incidentes aux sommets qui satisfont $\texttt{TCU} = i+1$. En effet, lorsque le chercheur A_w arrive sur un sommet $u \in V(T_i)$, venant d'un sommet $v \in V(T_i)$, dans l'état DFS_RETOUR, le chercheur teste si le plus petit fils v_t de u (si u possède un fils) est tel que $v \prec v_t \prec v_{i+1}$. Si u possède un tel fils v_t, le chercheur emprunte l'arête correspondante (Notons que cette arête est en fait f_t étiquetée $\texttt{Fils}_u(t)$) dans l'état DFS_TEST. Sinon, soit $u \neq v_0$ et le chercheur emprunte le port $\texttt{PARENT}_u$ dans l'état DFS_RETOUR, ou $u = v_0$ et le chercheur emprunte le port $\texttt{Fils}_u(1)$ dans l'état DFS_TEST. Par ailleurs, quand le chercheur A_w arrive sur un sommet $u \in V(T_i)$, en venant du sommet $parent(u)$, dans l'état DFS_TEST, il teste si u satisfait $\texttt{TCU} = i+1$. Si c'est le cas, le chercheur A_w nettoie toutes les arêtes non-étiquetées, passant alternativement de l'état NETTOIE-NON-ÉTIQUETÉE à l'état RETOUR-NON-ÉTIQUETÉE. Soit e_u l'une de ces arêtes.

Montrons qu'aucune recontamination ne se produit lors des mouvements de A_w le long de e_u. Notons que toutes les arêtes non-étiquetées e_u d'un tel sommet u appartiennent à $E(G[v_0, \cdots, v_i])$. De plus, d'après le point 3 de l'assertion, il y a un

chercheur dans l'état GARDER sur u à cette étape. Soit t l'autre extrémité de e_u. Si $t \in M_i \cup W$, toute arête incidente à t a déjà été nettoyée, donc, les mouvements de A_w le long de e_u n'entraîne aucune recontamination. Si $t \in V(T_i) \setminus (M_i \cup W)$, d'après le point 3, il y avait un chercheur dans l'état GARDER sur t à la fin de la phase i et il est toujours dans cet état sur t. Là encore, aucune recontamination ne se produit. Par conséquent, le nettoyage des arêtes non-étiquetées incidentes à u est réalisé sans recontamination. Après avoir nettoyé les arêtes non-étiquetées incidentes à u, le chercheur A_w continue le DFS en testant si u a au moins un fils plus petit que v_{i+1}. Si c'est le cas, le chercheur A_w emprunte le port correspondant au plus petit fils de u dans l'état DFS_TEST. Sinon, le chercheur A_w emprunte le port $\texttt{PARENT}_u$ dans l'état DFS_RETOUR. Pendant ce tour, pour tout $u \in V(T_i)$, soit $u \in M_i \cup W$ auquel cas u est propre, soit un chercheur dans l'état GARDER occupe u. *Donc, aucun mouvement de A_w n'entraîne de recontamination.*
Les sommets satisfaisant $\texttt{TCU} = i + 1$, sont dans $V(T_i) \setminus M_i$. Donc, il y a au plus $k = mcs(G, v_0)$ de ces sommets. Le premier tour de la phase $i+1$ est divisé en $O(n^2)$ rondes. Donc, après la dernière ronde de ce tour, *toutes les arêtes non-étiquetées incidentes aux sommets satisfaisant* $\texttt{TCU} = i + 1$ *sont propres.* Nous avons prouvé que lors du premier tour de la $i + 1$, au moins un chercheur s'est libéré (puisque $W \neq \emptyset$), et que chaque chercheur libre a nettoyé certaines arêtes.
Considérons à présent l'exécution du second tour de la phase $i + 1$ du protocole `Nettoyeur`.
Soit U l'ensemble des sommets de T_i de type 1 et tels que ($\texttt{TCU} = i+1$ et $\texttt{TTL} = i+1$). Si $U \neq \emptyset$, soit $w \in U$. Après la dernière ronde de la phase i, w était occupé par un chercheur, appelons le A_w, dans l'état GARDER. Pendant le premier tour de la phase $i + 1$, ce chercheur reste dans l'état GARDER sur ce sommet. Nous avons prouvé ci-dessus que toute arête non-étiquetée incidente à w a été nettoyée lors du premier tour de cette phase. Puisque $\texttt{TTL} = i + 1$, pour tout j, $i + 1 \leq j \leq n - 1$, f_j n'est pas incidente à w. Finalement, d'après le point 1, pour tout j, $0 \leq j \leq i$, f_j a été nettoyée. Donc, après la dernière ronde du premier tour de la phase $i + 1$, $\texttt{Dernier\_Port}_w \neq -1$ et l'arête correspondante e_w est la dernière arête contaminée incidente à w. Lors de la première ronde du second tour de la phase $i+1$, le chercheur A_w sur w nettoie l'arête e_w. Aucune recontamination ne se produit puisque e_w était la seule arête contaminée incidente à w. Le chercheur A_w arrive dans l'état NETTOIE sur un sommet u de T_i. Puisque $f_i \prec e_w$, $u \notin M_i$. Donc, lors de la dernière ronde de la phase i, le sommet u était gardé par un autre chercheur A_u dans l'état GARDER. Trois cas doivent être pris en compte :

- $u \in W$. Dans ce cas, e_w avait été nettoyée lors du premier tour de cette phase. Toutes les arêtes incidentes à u avaient déjà été nettoyées.
- $u \in U$. Par conséquent, lors de la première ronde du second tour de la phase $i+1$, le chercheur A_u s'est déplacé le long de $e_w = e_u$. Toutes les arêtes incidentes à u avaient déjà été nettoyées.
- $u \notin U \cup W$. Ainsi, le chercheur A_u est toujours dans l'état GARDER sur u.

Dans tous les cas, le chercheur A_w se libère et quitte sa position courante u par le port $\texttt{PARENT}_u$ dans l'état DFS_RETOUR, ou par le port $\texttt{Fils}_u(1)$ dans l'état DFS_TEST si le sommet courant est en fait v_0. Puisque u est propre ou gardé, aucune recontamination ne se produit.

Pendant le second tour de la phase $i+1$, chaque chercheur libre A effectue un DFS de T_i. Lors de ce tour, chaque chercheur libre doit nettoyer f_{i+1}. En effet, en effectuant un DFS de T_i, A finit par rencontrer le sommet $parent(v_{i+1})$. Lorsque le chercheur A arrive sur $parent(v_{i+1})$ dans l'état DFS_TEST ou DFS_RETOUR, il emprunte le port étiqueté $\texttt{Fils}(i+1)$ dans l'état NETTOIE, nettoyant l'arête f_{i+1}. Lorsqu'il arrive sur v_{i+1}, soit v_{i+1} est de degré un, et donc, il devient propre, et le chercheur le quitte en repartant par f_{i+1} dans l'état DFS_RETOUR, ou le chercheur A passe dans l'état ATTENDRE.

Finalement, considérons le sommet $w = parent(v_{i+1})$. Après la dernière ronde du premier tour de la phase $i+1$, il y a un chercheur A dans l'état GARDER sur w (point 3 de l'assertion). Si $\texttt{TYPE}_w = 0$ et $\texttt{TTL}_w = i+1$, le chercheur A emprunte le port $\texttt{Fils}_w(i+1)$ (correspondant à l'arête f_{i+1}) dans l'état NETTOIE lors de la première ronde du second tour de la phase $i+1$. Lorsqu'il arrive sur v_{i+1}, soit v_{i+1} est de degré un, et donc, il devient propre, et le chercheur le quitte en repartant par f_{i+1} dans l'état DFS_RETOUR, ou le chercheur A passe dans l'état ATTENDRE. Là encore, aucune recontamination ne peut se produire. Si $\texttt{TYPE}_w = 1$ ou $\texttt{TTL} > i+1$, le chercheur A reste sur w dans l'état GARDER.

Soit $J = \{parent(v_{i+1})\}$ si $\texttt{TYPE}_{parent(v_{i+1})} = 0$ et $\texttt{TTL}_{parent(v_{i+1})} = i+1$, sinon $J = \emptyset$. Soit $I = \{v_{i+1}\}$ si $\deg(v_{i+1}) = 1$, sinon $I = \emptyset$. Par définition, $M_{i+1} = \{v_j \mid \texttt{TTL}_{v_j} \leq i+1 \text{ ou } (j \leq i+1 \text{ avec } \deg(v_j) = 1)\}$. Remarquons que $M_{i+1} = M_i \cup W \cup U \cup I \cup J$. Alors, à la dernière ronde de la phase $i+1$, toutes les arêtes incidentes aux sommets de M_{i+1} sont propres. De plus, tout chercheur sur un sommet de $T_i \setminus M_{i+1}$ reste dans l'état GARDER. Par ailleurs, lors de la dernière ronde de la phase $i+1$, tous les chercheurs libres (rappelons que il y a au moins un chercheur libre après la première ronde du tour 1 de cette phase) sont sur v_{i+1} si $\deg(v_{i+1}) > 1$. Le chercheur avec le plus grand identifiant sur v_{i+1} passe dans l'état GARDER alors que les autres chercheurs quittent v_{i+1} par f_{i+1} dans l'état DFS_RETOUR. Donc, tous les points de l'assertion sont vérifés.

De plus, nous avons prouvé que lors de la phase $i+1$, aucune recontamination ne se produit.

Cas 2 : Si il y a un chercheur libre, la preuve est similaire au second cas du cas 1. $\diamond$

Pour conclure la preuve du théorème 27, il suffit de noter qu'après la dernière ronde de la phase n, tout sommet de M_n n'est incident qu'à des arêtes propres. De plus, nous avons prouvé que le nettoyage est réalisé de manière monotone et connexe. $\square$

6.3 Quantité minimale de conseil

Dans cette section, nous prouvons que la borne établie dans le théorème 27 est optimale asymptotiquement. Plus précisément, nous prouvons que :

Théorème 28 *Tout protocole réparti requiert $\Omega(n \log n)$ bits de conseil pour résoudre le problème du nettoyage connexe monotone réparti.*

Preuve. Pour prouver le théorème, nous construisons un graphe $\mathcal{G}$ de $O(n)$ sommets. Alors, nous prouvons que tout protocole de nettoyage connexe monotone réparti requiert $\Omega(n \log n)$ bits d'information pour nettoyer $\mathcal{G}$ de manière monotone et connexe, utilisant le nombre minimum de chercheurs.

Soit $n \geq 4$. Soit $t = 2n + 7$. Soient $P = \{v_1, \cdots, v_t\}$ un chemin, et K_{n-2}, resp. K_n, un graphe complet de $(n-2)$ sommets, resp. un graphe complet de n sommets. Nous construisons le graphe $\mathcal{G}$ en ajoutant toutes les arêtes entre v_i et les sommets de K_{n-2}, pour tout i, $1 \leq i \leq t$. Puis, nous identifions v_t et un sommet de K_n. Finalement, nous choisissons un sommet de K_{n-2} comme base v_0.

Les trois assertions suivantes décrivent plusieurs propriétés de toute stratégie qui nettoie $\mathcal{G}$ partant de v_0.

Assertion 17 $mcs(\mathcal{G}, v_0) = n$

Puisque $\mathcal{G}$ admet K_n en tant que sous-graphe, $mcs(G, v_0) \geq n$. Décrivons maintenant une stratégie qui nettoie $\mathcal{G}$ utilisant n chercheurs. En partant de v_0, déplacer un chercheur sur chaque sommet de K_{n-2} pour garder ces sommets. Utiliser les deux chercheurs restants pour nettoyer toutes les arêtes de $E(K_{n-2})$. Alors, déplacer de ces deux chercheurs sur v_1. L'autre chercheur nettoie toues les arêtes entre v_1 et K_{n-2}. Puis, déplacer le chercheur occupant v_1 sur v_2 et le second chercheur nettoie toutes les arêtes entre v_2 et K_{n-2}. Et ainsi de suite, jusqu'à ce que tous les sommets de P aient été nettoyés. A cette étape, il y a un chercheur sur chaque sommet de K_{n-2} et un chercheur sur v_t. Pour conclure, utiliser tous les chercheurs pour nettoyer K_n. $\diamond$

Assertion 18 *Pour toute stratégie monotone optimale pour $\mathcal{G}$, partant de v_0, le dernier sommet de $\mathcal{G}$ dont toutes les arêtes incidentes sont nettoyées, appartient à $V(K_n)$.*

Au cours du nettoyage de K_n, les n chercheurs doivent occuper les sommets de K_n. Donc, v_0 n'est plus occupé par un chercheur. Pour éviter qu'il ne soit recontaminé, Les arêtes incidentes à tout sommet de P et K_{n-2} doivent être propres. $\diamond$

Assertion 19 *Pour toute stratégie connexe monotone optimale pour $\mathcal{G}$, partant de v_0, le premier sommet de $\mathcal{G}$ à avoir toutes ses arêtes incidentes nettoyées est v_1 ou v_2. De plus, à cette étape, les sommets de K_{n-2} sont tous occupés, et aucun sommet de $\{v_4, \cdots, v_t\}$ n'est occupé.*

Soit u le premier sommet de $\mathcal{G}$ dont toutes les arêtes incidentes sont nettoyées. Soit s la première étape à laquelle toutes les arêtes incidentes à u sont propres. Après l'étape s, il doit y a voir un chercheur sur chaque voisin de u. De plus, après l'étape s, il doit y avoir un chercheur sur tout sommet d'un chemin entre u et v_0. Par conséquent, $u \in V(P)$. Soit

$1 \leq j \leq t$ tel que $u = v_j$. Supposons, par l'absurde, que $j \geq 3$. Après l'étape s, il y a un chercheur sur chaque sommet de K_{n-2}, sur v_{j-1} et sur v_{j+1}. Notons que, à cette étape, toutes les arêtes incidentes à v_{j-2} et v_{j+2} sont contaminées. Alors, le seul mouvement que le chercheur sur v_{j-1} (resp., sur v_{j+1}) peut effectuer est un déplacement vers v_{j-2} (resp., vers v_{j+2}). La stratégie atteint alors une situation telle que tout chercheur se trouve sur un sommet dont au moins deux arêtes incidentes sont contaminées. La stratégie échoue, et nous obtenons une contradiction. Par conséquent, $u \in \{v_1, v_2\}$. Si $u = v_2$, à l'étape s, les chercheurs occupent les sommets de K_{n-2}, v_1 et v_3. Donc, dans ce cas, l'assertion est vraie. Si $u = v_1$, à l'étape s, les chercheurs occupent les sommets de K_{n-2} et v_2. Supposons, par l'absurde que le chercheur restant occupe v_j, avec $j > 3$. Dans ce cas, le chercheur sur v_2 peut, au mieux, se déplacer sur v_3. La stratégie échoue alors car tout chercheur se trouve sur un sommet dont au moins deux arêtes incidentes sont contaminées. Ainsi, si $u = v_1$, l'assertion est vraie aussi. $\diamond$

L'assertion suivante décrit la manière dont toute stratégie connexe monotone optimale nettoie $\mathcal{G}$ en partant de v_0.

Assertion 20 *Soit S une stratégie de nettoyage connexe monotone optimale pour $\mathcal{G}$, partant de v_0. Pour tout i, $5 \leq i \leq t-2$, après l'étape à laquelle un chercheur atteint v_i au cours de l'éxécution de S, les propriétés suivantes sont satisfaites :*

- *toutes les arêtes incidentes à un sommet de $V(K_n) \cup \{v_{i+1}, \cdots, v_t\}$ sont contaminées ;*
- *tout sommet de K_{n-2} est occupé par un chercheur ;*
- *toutes les arêtes incidentes à un sommet de $\{v_1, \cdots, v_{i-2}\}$ sont propres ;*
- *Soit toutes les arêtes incidentes à v_{i-1} sont propres, ou il y a un chercheur sur v_{i-1} et une seule arête incidente à v_{i-1} est contaminée. Dans le second cas, le prochain mouvement consiste à déplacer un chercheur le long de la seule arête contaminée incidente à v_{i-1}.*

Soit s la première étape de la stratégie telle que, après cette étape, un chercheur occupe v_i. Considérons la situation juste avant cette étape. Puisque $i \geq 5$, d'après l'assertion 19, juste avant l'étape s, v_1 ou v_2 est uniquement incident à des arêtes propres, et un chercheur occupe chaque sommet de K_{n-2} pour préserver ces arêtes de la recontamination. De plus, il existe un sommet d'un chemin P entre v_1 et v_i, qui est occupé par un chercheur pour préserver v_1 ou v_2 de la recontamination. Soit j, $1 \leq j < i$, l'entier minimum tel qu'un chercheur occupe v_j. Notons que, pour tout k, $1 \leq k < j$, toutes les arêtes incidentes à v_k sont propres.

Premièrement, montrons que pour tout $\ell > i$, v_ℓ n'est pas occupé avant l'étape s. Par l'absurde, supposons que v_ℓ est occupé. Puisque toutes les arêtes incidentes à v_i sont contaminées, pour tout k, $j < k < \ell$, toutes les arêtes incidentes à v_k sont contaminées. D'après l'assertion 18 un sommet de K_n a au moins une arête incidente qui est contaminée. Donc, pour tout k, $\ell < k \leq t$, toutes les arêtes incidentes à v_k sont contaminées, puisqu'il n'y a pas de chercheurs sur le chemin entre v_k et K_n. Donc, il existe $k \neq i$ tel que toutes les arêtes incidentes à v_k sont contaminées. Donc, les chercheurs sur les sommets de K_{n-2} ne peuvent pas se déplacer, puisqu'il préserve la recontamination depuis v_i et v_k. Le chercheur sur v_ℓ ne peut se déplacer en évitant la recontamination depuis v_i et K_n. Le chercheur sur v_j peut éventuellement se déplacer sur v_{j+1}, mais il ne peut ensuite plus bouger. Donc, la stratégie échoue ce qui est une contradiction. Ceci conclut la preuve de la première propriété annoncée dans l'énoncé de l'assertion.

Donc, avant l'étape s, un chercheur occupe chaque sommet de K_{n-2}. Ces chercheurs évitent la recontamination depuis v_i et v_t. Donc, ils ne peuvent pas bouger. Ceci prouve la deuxième propriété de l'assertion.

La première propriété stipule que v_{i-1} doit avoir été atteint avant v_i. La monotonie de la stratégie implique qu'à l'étape précédente s, un chercheur occupe v_{i-1}. Considérons deux cas :

- Si l'étape s consiste à déplacer ce chercheur de v_{i-1} à v_i, la monotonie de la stratégie implique que soit toutes les arêtes incidentes à v_{i-1} sont propres, soit le chercheur restant occupe v_{i-1}. Dans le premier cas, l'assertion est vraie. Supposons donc qu'il existe au moins une arête incidente à v_{i-1} qui est toujours contaminée après l'étape s. Les sommets de K_{n-2} et v_i sont tous occupés par des chercheurs et sont tous incidents à au moins deux arêtes contaminées. Pour que la stratégie puisse se poursuivre, une seule arête incidente à v_{i-1} peut être contaminée et le prochain mouvement (la seule opération alors possible) consiste à déplacer le chercheur qui occupe v_{i-1} le long de cette arête.
- Sinon, l'étape s consiste à déplacer un chercheur d'un sommet u de K_{n-2} vers v_i. Comme $i \leq t-2$, u doit être occupé par deux chercheurs avant l'étape s, pour que l'un deux préserve u de la recontamination au cours de l'étape s. Après l'étape s, chacun des sommets de K_{n-2} et v_i est occupé par un chercheur et incident au moins deux arêtes contaminées (car $i \leq t-2$), et un chercheur occupe v_{i-1}. Le seul déplacement possible est alors de déplacer le chercheur qui occupe v_{i-1}. L'arête $e = \{v_{i-1}, v_i\}$ est contaminée. Pour que la stratégie aboutisse, e doit être la seule arête incidente à v_{i-1} qui soit contaminée.

Ceci conclut la preuve de l'assertion. $\diamond$

Rappelons qu'une orientation locale d'un graphe G est une fonction qui assigne des numéros de port en chaque sommet de G. Considérons à présent l'ensemble $\mathcal{C}$ des instances suivantes $\{(\mathcal{G}, v_0, \ell o) \mid \ell o \text{ est une orientation locale de } \mathcal{G}\}$. Soit $\mathcal{I} = |\mathcal{C}|$. L'assertion suivante prouve que tout protocole de nettoyage connexe monotone réparti, utilisant une chaîne de bits d'information arbitraire, ne peut nettoyer qu'une partie des instances de $\mathcal{C}$.

Assertion 21 *Soit $\mathcal{P}$ un protocole de nettoyage connexe monotone réparti solution du problème du nettoyage connexe monotone réparti. Soit f une chaîne de bits d'information arbitraire fournie par un oracle. En utilisant f, $\mathcal{P}$ peut nettoyer au plus $\mathcal{I}*(\frac{1}{n-2})^n$ instances de $\mathcal{C}$.*

Soit $\mathcal{I}_{k,j}$ le nombre d'instances telles que $(\mathcal{P}, f)$ permet à un chercheur de nettoyer j arêtes entre v_k et K_{n-2}. Nous prouvons que, pour tout k, $5 \leq k \leq n+5$ et pour tout j, $1 \leq j \leq n-3$, $\mathcal{I}_{k,j} \leq \mathcal{I}_{k,j-1}\frac{n-j-1}{n-j}$.

Considérons la dernière étape à laquelle exactement $0 \leq j \leq n-3$ arêtes entre v_k et K_{n-2} sont propres, juste avant qu'une $j+1^{eme}$ arête entre v_k et K_{n-2} ne soit nettoyée.

D'après l'assertion ci-dessus, à cette étape, un chercheur occupe v_k et un chercheur occupe chaque sommet de K_{n-2}. De plus, d'après le lemme 20, le chercheur restant (qui est le seul chercheur libre) ne doit pas se déplacer sur un sommet de $\{v_{k+1}, \cdots, v_t\}$. Soit v le sommet occupé par ce chercheur. Utilisant f, le protocole $\mathcal{P}$ choisit un numéro de port p que le chercheur libre doit emprunter. Il y a deux cas selon que le chercheur libre occupe v_k ou un sommet de K_{n-2}.

- Si le chercheur libre occupe v_k, il reste $n-j-1$ arêtes contaminées incidentes à ce sommet, et la stratégie échoue si p mène à v_{k+1}. Donc, la stratégie échoue dans au moins $\mathcal{I}_{k,j}\frac{1}{n-j-1}$ instances. Par conséquent, $\mathcal{I}_{k,j+1} \leq \mathcal{I}_{k,j}\frac{n-j-2}{n-j-1}$.
- Si le chercheur libre occupe un sommet de K_{n-2}, il reste au plus $n-3+t-k+1$ arêtes contaminées incidentes à ce sommet, et la stratégie échoue si p mène à un sommet de $\{v_{k+1}, \cdots, v_t\}$. Donc, la stratégie échoue dans au moins $\mathcal{I}_{k,j-1}(\frac{t-k}{n-3+t-k+1})$ instances. Ainsi, $\mathcal{I}_{k,j} \leq \mathcal{I}_{k,j-1}\frac{n-2}{t+n-2-k}$. Pour conclure, il suffit de remarquer que, puisque $n \geq 4$, $t = 2n+7$, $1 \leq j \leq n-3$ et $5 \leq k \leq n-5$, nous obtenons $\frac{n-2}{t+n-2-k} \leq \frac{n-2}{2n}$ et $\frac{n-j-2}{n-j-1} \geq \frac{n-3}{2}$. Donc, $\frac{n-2}{t+n-2-k} \leq \frac{n-j-2}{n-j-1}$.

Ainsi, $\mathcal{I}_{k,n-2} \leq \mathcal{I}_{k-1,n-2} \prod_{j=1..n-3}(\frac{n-j-2}{n-j-1}) = \mathcal{I}_{k-1,n-2}(\frac{1}{n-2})$. Utilisant f, $\mathcal{P}$ peut nettoyer au plus $\mathcal{I}_{n-5,n-2} \leq \mathcal{I}_{5,n-2}(\frac{1}{n-2})^n$. Puisque, $\mathcal{I}_{5,n-2} \leq \mathcal{I}$, l'assertion est vérifiée. $\diamond$

Soit $N = |V(\mathcal{G})| = 4n+4$. Pour prouver le théorème, il suffit de remarquer que, pour tout $\alpha < 1/4$, et pour tout oracle $\mathcal{O}$ qui fournit moins de $q = \alpha N \log N$ bits d'information, aucun protocole de nettoyage connexe monotone réparti utilisant $\mathcal{O}$ ne permet de nettoyer toutes les instances de $\mathcal{C}$. Soit $\mathcal{O}$ un tel oracle. Le nombre de fonction f que l'oracle $\mathcal{O}$ peut fournit à $\mathcal{G}$ est au plus $(q+1)2^q\binom{N+q}{N}$ [FIP06]. Donc, il existe un ensemble $\mathcal{S} \subseteq \mathcal{C}$ d'au moins $B = \frac{\mathcal{I}}{(q+1)2^q\binom{N+q}{N}}$ instances de $\mathcal{C}$ pour lesquels $\mathcal{O}$ fournit la même chaîne de bits d'information.

Soit $\mathcal{P}$ un protocole de nettoyage connexe monotone réparti qui utilise l'oracle $\mathcal{O}$ pour résoudre le problème du nettoyage connexe monotone réparti. D'après l'assertion 21, $\mathcal{P}$ ne peut nettoyer plus de $\mathcal{I} * (\frac{1}{n-2})^n$ instances de $\mathcal{C}$ en utilisant la même chaîne de bits d'information.

Pour conclure, il reste à prouver que $B > \mathcal{I} * (\frac{1}{n-2})^n$. En effet,

$$B * (\frac{(n-2)^n}{\mathcal{I}}) = \frac{(n-2)^n}{(q+1)2^q\binom{N+q}{N}}$$

En utilisant la formule de Stirling, nous obtenons que pour n suffisamment grand,

$$B * (\frac{(n-2)^n}{\mathcal{I}}) \sim \frac{(n-2)^n}{2^{\alpha N \log N}(1+\alpha \log N)^N} * (\frac{\alpha \log N}{1+\alpha \log N})^{\alpha N \log N}$$

Puisque $N = 4n+4$, nous obtenons :

$$\log[B * (\frac{(n-2)^n}{\mathcal{I}})] \sim (1-4\alpha)n \log n$$

Puisque $\alpha < 1/4$, $B > \mathcal{I} * (\frac{1}{n-2})^n$. Ceci conclut la preuve du théorème. $\square$

6.4 Perspectives

Dans ce chapitre, nous avons prouvé que $O(n \log n)$ bits d'information sont nécessaires pour qu'un protocole de nettoyage connexe réparti nettoie un graphe de manière connexe monotone. Une question qui se pose est de savoir comment réduire cette quantité d'information, en relachant, par exemple, certaines contraintes que doivent satisfaire les stratégies de capture. Quelle est la quantité minimum d'information nécessaire pour nettoyer un graphe en temps polynomial (sans forcement que la stratégie soit monotone) ? Une autre question serait de minimiser la quantité d'information disponible sur chaque sommet, plutôt que de minimiser la somme des longueurs des chaînes de bits d'information.

Chapitre 7

Conclusion et perspectives

7.1 Conclusion

Dans cette thèse, nous avons présenté plusieurs résultats destinés à améliorer notre compréhension des jeux des gendarmes et du voleur dans les graphes. Les stratégies visant la capture d'un fugitif arbitrairement rapide, visible ou invisible, avaient déjà été intensivement étudiées pour leurs relations avec les décompositions linéaires et arborescentes des graphes. Un résultat fondamental dans ce domaine de recherche concerne la monotonie de ces jeux [BS91, LaP93, ST93]. L'un des principaux résultats de cette thèse est l'introduction des stratégies de capture non-déterministes qui établissent une approche unifiée des stratégies de capture visible et invisible. Notamment, nous avons prouvé que le jeu de capture non-déterministe satisfait la propriété de monotonie, généralisant de ce fait les preuves, pourtant radicalement différentes, des cas visible et invisible.

La seconde variante des jeux de capture que nous avons étudiée est celle des stratégies de capture connexe. Récemment, Alspach *et al.* [YDA04] ont prouvé que, dans le cas d'un fugitif invisible, ces stratégies ne satisfaisaient pas forcément la propriété de monotonie. D'autre part, la principale question qui se pose dans le cadre des stratégies de capture connexe concerne le coût de la contrainte de connexité en terme de nombre de chercheurs. Dans le chapitre 3, nous proposons de nouvelles bornes pour le rapport entre indice d'échappement connexe et indice d'échappement d'un graphe. Nous prouvons que, dans le cas général d'un graphe de n sommets, ce rapport est borné supérieurement par $\log n$. Nous proposons par ailleurs une autre borne supérieure pour ce rapport dans le cas des graphes de cordalité et de largeur arborescente bornée. Ce dernier résultat nous permet de prouver que ce rapport vaut au plus $2\ (tw(G)+1)$ pour tout graphe cordal G. Le problème général du coût de la contrainte de connexité reste cependant ouvert.

Dans le chapitre 4, nous étudions les stratégies de capture connexes dans le cas d'un fugitif visible, et répondons aux deux principales questions qui se posaient dans ce contexte. A savoir, nous prouvons que la contrainte de connexité dans le cas d'un fugitif visible peut être très pénalisante en terme de nombre de chercheurs. Nous prouvons en effet que le rapport entre l'indice d'échappement visible connexe et l'indice d'échappement visible d'un graphe de n sommets est $O(\log n)$ et que cette borne est atteinte. Par ailleurs, nous prouvons que les stratégies visibles connexes ne satisfont pas forcément la propriété de monotonie.

Les stratégies de capture connexes avaient principalement été étudiées pour leurs applications pratiques. Notamment, un certain nombre d'algorithmes répartis avaient été proposés pour nettoyer des réseaux ayant certaines topologies spécifiques. Pour ces topologies (grille, hypercube, etc.), Flocchini *et al.* ont prouvé que dans le cas asynchrone, un chercheur de plus que l'optimum centralisé est nécessaire pour nettoyer certains réseaux. Dans le chapitre 5, nous prouvons que ce chercheur supplémentaire est suffisant dans tout graphe connexe. Le protocole réparti que nous proposons dans ce chapitre nettoie n'importe quel réseau, sans que les chercheurs impliqués dans le nettoyage n'en connaissent la topologie. Cependant, pour nettoyer un graphe en temps polynomial, il faut fournir de la connaissance aux chercheurs (à moins que $P = NP$). Dans le chapitre 6, nous répondons formellement à la question de la quantité d'information minimale qu'il faut fournir aux chercheurs pour pouvoir résoudre le problème du nettoyage connexe monotone réparti.

Comme nous l'avons dit, l'un des principaux résultats de cette thèse est certainement l'introduction de la variante non-déterministe des stratégies de capture. Nous pensons notamment que la notion d'arbre de capture est une structure qui ouvre des perspectives intéressantes que nous allons développer dans la section suivante.

7.2 Perspectives

Outre les problèmes ouverts que nous avons mentionnés dans la dernière section de chaque chapitre de la thèse, le cadre des jeux de capture ouvre de nombreuses perspectives.

Très récemment, un certain nombre de résultats relatif aux jeux de capture dans les graphes orientés sont parus. Ces résultats nous paraissent très importants car ils proposent diverses définitions de ce qui pourrait être l'équivalent des décompositions arborescentes dans le cas des graphes orientés.

Dès 2001, Johnson *et al* [JRST01] proposent une variante de jeu de capture dans les graphes orientés. Dans cette variante, les chercheurs sautent d'un sommet à un autre et poursuivent un fugitif visible arbitrairement rapide mais contraint toutefois de se déplacer dans les composantes fortement connexes libre de chercheurs. En d'autres termes, le fugitif peut se déplacer d'un sommet u à un sommet v s'il existe un chemin orienté P de u à v, un autre chemin P' de v à u, et à la condition qu'aucun sommet appartenant à P ou P' ne soit occupé par un chercheur. Les auteurs prouvent que la variante monotone de ce jeu peut être mis en correspondance avec une décomposition des graphes orientés. Cette décomposition est appelée *arboreal decomposition* et la largeur associée s'appelle la *directed tree-width*. A la différence des décompositions arborescentes, ces décompositions reposent sur un digraphe acyclique (DAG) dont les sommets et les arêtes sont étiquetées par des ensembles de sommets du digraphe.

Un résultat important relatif à ce jeu est qu'il ne satisfait pas la propriété de monotonie. Plus précisément, Johnson *et al* [JRST01] prouvent qu'il existe des digraphes pour lesquels capturer un fugitif avec le nombre optimum de chercheurs nécessite qu'un sommet soit occupé plusieurs fois par un chercheur. Très récemment, Adler [Adl07] a prouvé qu'il existe des digraphes pour lesquels capturer un fugitif avec le nombre optimum de chercheurs nécessite qu'un sommet soit occupé plusieurs fois par le fugitif. Adler [Adl07] a aussi prouvé que le pendant du théorème 7 n'est pas valide dans le cas orienté.

Très récemment, un autre modèle de jeu de capture dans des digraphes a été introduit par Obdrzálek [Obd06] d'une part, et Berwanger *et al.* [BDHK06] d'autre part. Nous appellerons cette variante *le jeu de capture orientée*. Dans cette variante, le fugitif n'est plus contraint de se déplacer dans une composante fortement connexe libre de chercheurs. Le fugitif peut se déplacer d'un sommet à l'autre en suivant les chemins orientés du digraphe. Les auteurs établissent une équivalence entre la variante monotone de ce jeu et une décomposition des digraphes : la *DAG-décomposition*. Barát [Bar06] étudie la variante invisible de ce jeu et prouve que pour tout digraphe D, il existe une stratégie de capture orientée invisible monotone impliquant au maximum un chercheur de plus que l'indice d'échappement orientée invisible de D.

Deux problèmes restent ouverts relativement au jeu de capture orientée visible. Premièrement, satisfait-il la propriété de monotonie ? Et deuxièmement, existe-il un équivalent des buissons dans le cas orienté ? En collaboration avec Omid Amini, Frédéric Mazoit et Stéphan Thomassé, nous avons initié l'étude de ces problèmes. Pour cela, nous utilisons le concept d'arbre de capture introduit dans cette thèse, ce qui nous a d'ores et déjà permis de fournir une approche très générale pour établir des théorèmes de dualité relatifs aux décompositions arborescentes, linéaires et en branche, ainsi qu'à la largeur arborescente des matroïdes et la *rankwidth*.

Une autre voie de recherche concerne la conception d'algorithmes en temps polynomial pour calculer des stratégies de capture dans certaines classes de graphes. En collaboration avec Omid Amini et David Coudert, nous avons initié l'étude des stratégies non-déterministes dans la classe des arbres. Nous avons ainsi conçu un algorithme en temps linéaire qui calcule le nombre minimum de questions que doivent poser deux chercheurs pour capturer un fugitif dans un arbre. Puis, nous avons conçu un algorithme polynomial pour calculer le nombre minimum de chercheurs nécessaires à la capture d'un fugitif dans un arbre en posant au plus une question. Enfin, nous avons conçu un algorithme polynomial pour calculer le nombre minimum de chercheurs nécessaires à la capture d'un fugitif dans un arbre en posant au plus $q \geq 0$ questions, dans la classe des arbres d'indice d'échappement q-limité borné. Le problème d'un algorithme en temps polynomial pour résoudre ce problème dans la classe générale des arbres reste ouvert. Un dernier problème ouvert qui vaut la peine d'être mentionné concerne la conception d'un algorithme en temps polynomial pour calculer une stratégie de capture connexe dans la classe des graphes d'indice d'échappement connexe borné.

Outre les pistes de recherche mentionnés ci-dessus, nous pensons plus généralement que le concept des jeux des gendarmes et du voleur est un cadre riche dont l'étude peut être abordée selon de nombreux axes : théorie des jeux, combinatoire, algorithmique et complexité, algorithmique distribuée, etc. En effet, bien que la théorie des mineurs de graphes ait fourni des outils très performants pour étudier les jeux des gendarmes et du voleur, ces outils s'avèrent insuffisants dans le cadre d'applications pratiques, lorsque, par exemple, on impose aux stratégies de capture d'être connexes. Il s'agit donc de développer de nouveaux outils pour étudier les jeux des gendarmes et du voleur, dans le but de répondre à des questions comme celle concernant le rapport entre l'indice d'échappement connexe d'un graphe et son indice d'échappement, de concevoir des algorithmes calculant des stratégies de capture connexes non monotones (qui utilisent potentiellement moins de chercheurs), d'étudier les stratégies de capture lorsque les chercheurs se comportent de façon égoïste. Pour conclure, les jeux des gendarmes et du voleur pourraient être les initiateurs de nouveaux concepts très généraux, de la même manière qu'ils ont participé à l'élaboration de la théorie des mineurs de graphes.

Bibliographie

[ACP87] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Algebraic Discrete Methods*, 8(2) :277–284, 1987.

[Adl07] Isolde Adler. Directed tree-width examples. *J. Comb. Theory, Ser. B*, 2007. to appear.

[AF84] Martin Aigner and Michael Fromme. A game of cops and robbers. *Discrete Applied Mathematics*, 8 :1–12, 1984.

[AF88] Richard P. Anstee and Martin Farber. On bridged graphs and cop-win graphs. *J. Comb. Theory, Ser. B*, 44(1) :22–28, 1988.

[AKL+79] Romas Aleliunas, Richard M. Karp, Richard J. Lipton, László Lovász, and Charles Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. In *Proceedings of the 20th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 218–223, 1979.

[ALS91] Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2) :308–340, 1991.

[Als04] Brian Alspach. Searching and sweeping graphs : a brief survey. In *Le Matematiche*, pages 5–37, 2004.

[And84] Thomas Andreae. Note on a pursuit game played on graphs. *Discrete Applied Mathematics*, 9 :111–115, 1984.

[And86] Thomas Andreae. On a pursuit game played on graphs for which a minor is excluded. *J. Comb. Theory, Ser. B*, 41(1) :37–47, 1986.

[AP86] Stefan Arnborg and Andrzej Proskurowski. Characterization and recognition of partial 3-trees. *SIAM J. Algebraic Discrete Methods*, 7 :305–314, 1986.

[AP92] Stefan Arnborg and Andrzej Proskurowski. Canonical representations of partial 2- and 3-trees. *BIT*, 32(2) :197–214, 1992.

[Arc80] Dan Archdeacon. *A Kuratowski Theorem for the Projective Plan*. PhD thesis, Ohio State University, 1980.

[ARS+03] Micah Adler, Harald Räcke, Naveen Sivadasan, Christian Sohler, and Berthold Vöcking. Randomized pursuit-evasion in graphs. *Combinatorics, Probability and Computing*, 12(3), 2003.

[Bar06] János Barát. Directed path-width and monotonicity in digraph searching. *Graphs and Combinatorics*, 22(2) :161–172, 2006.

[BBH+06] Anne Berry, Jean Paul Bordat, Pinar Heggernes, Geneviève Simonet, and Yngve Villanger. A wide-range algorithm for minimal triangulation from an arbitrary ordering. *J. Algorithms*, 58(1) :33–66, 2006.

[BDHK06] Dietmar Berwanger, Anuj Dawar, Paul Hunter, and Stephan Kreutzer. Dag-width and parity games. In *Proceedings ot the 23rd Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 524–536, 2006.

[Ber73] Claude Berge. *Graphes et Hypergraphes.* Dunod Université, 1973.

[BFFS02] Lali Barrière, Paola Flocchini, Pierre Fraigniaud, and Nicola Santoro. Capture of an intruder by mobile agents. In *Proceedings of the 14th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 200–209, 2002.

[BFK+06] Hans Bodlaender, Fedor V. Fomin, Arie M. C. A. Koster, Dieter Kratsch, and Dimitrios M. Thilikos. On exact algorithms for treewidth. In *Proceedings of the 14th Annual European Symposium on Algorithms (ESA)*, number 4168 in LNCS, pages 672–683, September 2006.

[BFNV06] Lélia Blin, Pierre Fraigniaud, Nicolas Nisse, and Sandrine Vial. Distributed chasing of network intruders. In *Proceedings of the 13rd International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 70–84, 2006.

[BFR+02] Michael A. Bender, Antonio Fernandez, Dana Ron, Amit Sahai, and Salil P. Vadhan. The power of a pebble : Exploring and mapping directed graphs. *Information and Computation*, 176 :1–21, 2002.

[BFST03] Lali Barrière, Pierre Fraigniaud, Nicola Santoro, and Dimitrios M. Thilikos. Searching is not jumping. In *Proceedings of the 29th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 34–45, 2003.

[BHT01] Jean R. S. Blair, Pinar Heggernes, and Jan Arne Telle. A practical algorithm for making filled graphs minimal. *Theor. Comput. Sci.*, 250(1-2) :125–141, 2001.

[Bie91] Daniel Bienstock. Graph searching, path-width, tree-width and related problems (a survey). *DIMACS Ser. in Discrete Mathematics and Theoretical Computer Science*, 5 :33–49, 1991.

[BK96] Hans L. Bodlaender and Ton Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *J. Algorithms*, 21(2) :358–402, 1996.

[BKK95] Hans L. Bodlaender, Ton Kloks, and Dieter Kratsch. Treewidth and pathwidth of permutation graphs. *SIAM J. Discrete Math.*, 8(4) :606–616, 1995.

[BKMT04] Vincent Bouchitté, Dieter Kratsch, Haiko Müller, and Ioan Todinca. On treewidth approximations. *Discrete Applied Mathematics*, 136(2-3) :183–196, 2004.

[BM93] Hans L. Bodlaender and Rolf H. Möhring. The pathwidth and treewidth of cographs. *SIAM J. Discrete Math.*, 6(2) :181–188, 1993.

[BO95] Tamer Basar and Geert J. Olsder. *Dynamic Noncooperative Game Theory.* Academic Press, London and San Diego, 1995.

[Bod88] Hans L. Bodlaender. Dynamic programming on graphs with bounded treewidth. In *Proceedings of the 15th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 105–118, 1988.

[Bod93] Hans L. Bodlaender. A tourist guide through treewidth. *Acta Cybern.*, 11(1-2) :1–22, 1993.

[Bod96] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6) :1305–1317, 1996.

[Bod97] Hans L. Bodlaender. Treewidth : Algorithmic techniques and results. In *Proceedings of the 22nd International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 19–36, 1997.

[Bod98] Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theor. Comput. Sci.*, 209(1-2) :1–45, 1998.

[Bod06] Hans L. Bodlaender. Treewidth : Characterizations, applications, and computations. In *Proceedings of the 32nd International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 1–14, 2006.

[BP92] Jean R. S. Blair and Barry W. Peyton. An introduction to chordal graphs and clique trees. *Graph Theory and Sparse Matrix Computations*, 56 :1–27, 1992.

[Bre67] R. L. Breisch. An intuitive approach to speleotopology. *Southwestern Cavers*, 6 :72–78, 1967.

[BS91] Daniel Bienstock and Paul D. Seymour. Monotonicity in graph searching. *J. Algorithms*, 12(2) :239–245, 1991.

[BT97] Hans L. Bodlaender and Dimitrios M. Thilikos. Treewidth for graphs with small chordality. *Discrete Applied Mathematics*, 79(1-3) :45–61, 1997.

[BT04] Hans L. Bodlaender and Dimitrios M. Thilikos. Computing small search numbers in linear time. In *Proceedings of the International Workshop on Parameterized and Exact Computation (IWPEC)*, pages 37–48, 2004.

[Cha72] Ashok K. Chandra. Efficient compilation of linear recursive programs. Technical report, Stanford University, 1972.

[Che97] Victor Chepoi. Bridged graphs are cop-win graphs : An algorithmic proof. *J. Comb. Theory, Ser. B*, 69(1) :97–100, 1997.

[CHS06] David Coudert, Florian Huc, and Jean-Sébastien Sereni. Pathwidth of outerplanar graphs. *Journal of Graph Theory*, 2006. to appear.

[CK06] L. Sunil Chandran and Telikepalli Kavitha. The treewidth and pathwidth of hypercubes. *Discrete Mathematics*, 306(3) :359–365, 2006.

[CM93] Bruno Courcelle and Mohamed Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theor. Comput. Sci.*, 109(1&2) :49–82, 1993.

[CMS01] Andrea E.F. Clementi, Angelo Monti, and Riccardo Silvestri. Selectives families, superimosed codes, and broadcasting on unknown radio networks. In *Proceedings of the 12th ACM Symposium on Discrete Algorithms (SODA)*, pages 709–718, 2001.

[CN88] Wei-Pang Chin and Simeon C. Ntafos. Optimum watchman routes. *Inform. Process. Lett.*, 28 :39–44, 1988.

[CN00] Nancy E. Clarke and Richard J. Nowakowski. Cops, robber, and photo radar. *Ars Comb.*, 56 :97–103, 2000.

[CN01] Nancy E. Clarke and Richard J. Nowakowski. Cops, robber, and traps. *Utilitas Mathematica*, 60 :91–98, 2001.

[CN05] Nancy E. Clarke and Richard J. Nowakowski. Tandem-win graphs. *Discrete Mathematics*, 299(1-3) :56–64, 2005.

[Coo73] Stephen A. Cook. An observation on time-storage trade off. In *Proceedings of the 5th annual ACM symposium on Theory of computing (STOC)*, pages 29–33, 1973.

[CS74] Stephen Cook and Ravi Sethi. Storage requirements for deterministic / polynomial time recognizable languages. In *Proceedings of the 6th annual ACM symposium on Theory of computing (STOC)*, pages 33–39, 1974.

[DFHT05] Erik D. Demaine, Fedor V. Fomin, Mohammad Taghi Hajiaghayi, and Dimitrios M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and -minor-free graphs. *Journal of the ACM*, 52(6) :866–893, 2005.

[DFPS06] Stefan Dobrev, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Searching for a black hole in arbitrary networks : optimal mobile agents protocols. *Distributed Computing*, 19(1), 2006.

[DHT05] Erik D. Demaine, Mohammad Taghi Hajiaghayi, and Dimitrios M. Thilikos. Exponential speedup of fixed-parameter algorithms for classes of graphs excluding single-crossing graphs as minors. *Algorithmica*, 41(4) :245–267, 2005.

[DKT97] Nick D. Dendris, Lefteris M. Kirousis, and Dimitrios M. Thilikos. Fugitive-search games on graphs and related parameters. *Theor. Comput. Sci.*, 172(1-2) :233–254, 1997.

[EM04] Jonathan A. Ellis and Minko Markov. Computing the vertex separation of unicyclic graphs. *Inform. and Comput.*, 192(2) :123–161, 2004.

[EST94] Jonathan A. Ellis, Ivan Hal Sudborough, and Jonathan S. Turner. The vertex separation and search number of a graph. *Inf. Comput.*, 113(1) :50–79, 1994.

[Far87] Martin Farber. Bridged graphs and geodesic convexity. *Discrete Applied Mathematics*, 66 :249–257, 1987.

[FFN05] Fedor V. Fomin, Pierre Fraigniaud, and Nicolas Nisse. Nondeterministic graph searching : From pathwidth to treewidth. In *Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science (MFCS)*. Springer LNCS 3618, 2005.

[FFT04] Fedor V. Fomin, Pierre Fraigniaud, and Dimitrios M. Thilikos. The price of connectedness in expansions. Rapport Technique No. LSI-04-28-R, UPC Barcelone, 2004.

[FG65] Delbert R. Fulkerson and O.A. Gross. Incidence matrices and interval graphs. *Pacific J. Math.*, 15 :835–865, 1965.

[FG00] Fedor V. Fomin and Peter A. Golovach. Graph searching and interval completion. *SIAM J. Discrete Math.*, 13(4) :454–464, 2000.

[FHL05a] Uriel Feige, Mohammad Taghi Hajiaghayi, and James R. Lee. Improved approximation algorithms for minimum-weight vertex separators. In *Proceedings of the 37th annual ACM symposium on Theory of computing (STOC)*, pages 563–572, 2005.

[FHL05b] Paola Flocchini, Miao Jun Huang, and Flaminia L. Luccio. Contiguous search in the hypercube for capturing an intruder. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2005.

[FHL06] Paola Flocchini, Miao Jun Huang, and Flaminia L. Luccio. Decontamination of chordal rings and tori. In *Proceedings of the 20th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2006. to appear.

[FHT04] Fedor V. Fomin, Pinar Heggernes, and Jan Arne Telle. Graph searching, elimination trees, and a generalization of bandwidth. *Algorithmica*, 41(2) :73–87, 2004.

[FIP06] Pierre Fraigniaud, David Ilcinkas, and Andrzej Pelc. Oracle size : a new measure of difficulty for communication tasks. In *Proceedings of the 12th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 179–187, 2006.

[FIRT06] Pierre Fraigniaud, David Ilcinkas, Sergio Rajsbaum, and Sébastien Tixeuil. The reduced automata technique for graph exploration space lower bounds. In *Essays in memory of Shimon Even*, pages 1–26, 2006.

[FKM03] Fedor V. Fomin, Dieter Kratsch, and Haiko Müller. On the domination search number. *Discrete Applied Mathematics*, 127(3) :565–580, 2003.

[FKT04] Fedor V. Fomin, Dieter Kratsch, and Ioan Todinca. Exact (exponential) algorithms for treewidth and minimum fill-in. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP)*, pages 568–580, 2004.

[FLS05] Paola Flocchini, Flaminia L. Luccio, and Lisa Xiuli Song. Size optimal strategies for capturing an intruder in mesh networks. In *Proceedings of the 2005 International Conference on Communications in Computing (CIC)*, pages 200–206, 2005.

[FN06a] Pierre Fraigniaud and Nicolas Nisse. Connected treewidth and connected graph searching. In *Proceedings of the 7th Latin American Symposium (LATIN)*, pages 479–490, 2006.

[FN06b] Pierre Fraigniaud and Nicolas Nisse. Monotony properties of connected visible graph searching. In *Proceedings of the 32nd International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 229–240, 2006.

[Fom98] Fedor V. Fomin. Helicopter search problems, bandwidth and pathwidth. *Discrete Applied Mathematics*, 85(1) :59–70, 1998.

[Fom99] Fedor V. Fomin. Note on a helicopter search problem on graphs. *Discrete Applied Mathematics*, 95(1-3) :241–249, 1999.

[FR03] Faith E. Fish and Eric Ruppert. Hundred of impossibility results for distributed computing. *Distributed Computing*, 16 :121–163, 2003.

[Fra87] Peter Frankl. Cops and robbers in graphs with large girth and cayley graphs. *Discrete Applied Mathematics*, 17 :301–305, 1987.

[FTT05] Fedor V. Fomin, Dimitrios M. Thilikos, and Ioan Todinca. Connected graph searching in outerplanar graphs. *Electronic Notes in Discrete Mathematics*, 22 :213–216, 2005. 7th International Colloquium on Graph Theory. Short communication.

[Gav74] Fanica Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory, Ser. B*, 16(B) :47–56, 1974.

[GHW79] Henry H. Glover, John P. Huneke, and Chin San Wang. 103 graphs that are irreducible for the projective plane. *J. Combin. Theory Ser. B*, 27 :332–370, 1979.

[GLL+97] Leonidas J. Guibas, Jean-Claude Latombe, Steven M. LaValle, David Lin, and Rajeev Motwani. Visibility-based pursuit-evasion in a polygonal environment. In *Proceedings of the 5th International Workshop on Algorithms and Data Structures (WADS)*, pages 17–30, London, UK, 1997. Springer-Verlag.

[GLT79] John R. Gilbert, Thomas Lengauer, and Robert E. Tarjan. The pebbling problem is complete in polynomial space. In *Proceedings of the 11th annual ACM symposium on Theory of computing (STOC)*, pages 237–248, 1979.

[GLY98] Rajeev Govindan, Michael A. Langston, and Xudong Yan. Approximation the pathwidth of outerplanar graphs. *Inf. Process. Lett.*, 68(1) :17–23, 1998.

[Gol80] Martin C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs (Annals of Discrete Mathematics, Vol 57)*. North-Holland Publishing Co., Amsterdam, The Netherlands, 1980. second edition 2004.

[GR95] Arthur S. Goldstein and Edward M. Reingold. The complexity of pursuit on a graph. *Theor. Comput. Sci.*, 143(1) :93–112, 1995.

[Gus93] Jens Gustedt. On the pathwidth of chordal graphs. *Discrete Applied Mathematics*, 45(3) :233–248, 1993.

[Heg06] Pinar Heggernes. Minimal triangulations of graphs : A survey. *Discrete Mathematics*, 306(3) :297–317, 2006.

[Hic04] Illya V. Hicks. Branch decompositions and minor containment. *Networks*, 43(1) :1–9, 2004.

[Hic05] Illya V. Hicks. Graphs, branchwidth, and tangles ! oh my ! *Networks*, 45(2) :55–60, 2005.

[HLSW02] Gena Hahn, François Laviolette, Norbert Sauer, and Robert E. Woodrow. On cop-win graphs. *Discrete Mathematics*, 258(1-3) :27–41, 2002.

[HM03] Gena Hahn and Gary MacGillivray. A characterisation of k-cop-win graphs and digraphs., 2003.

[Ilc06] David Ilcinkas. *Complexité en espace de l'exploration de graphes*. PhD thesis, Université de Paris XI, France, 2006.

[Isa65] Rufus Isaacs. *Differential Games*. Wiley, New York, 1965.

[JRST01] Thor Johnson, Neil Robertson, Paul D. Seymour, and Robin Thomas. Directed tree-width. *J. Comb. Theory, Ser. B*, 82(1) :138–154, 2001.

[Kin92] Nancy G. Kinnersley. The vertex separation number of a graph equals its pathwidth. *Inform. Process. Lett.*, 1992.

[KP85] Lefteris M. Kirousis and Christos H. Papadimitriou. Interval graphs and searching. *Discrete Mathematics*, 55 :181–184, 1985.

[KP86] Lefteris M. Kirousis and Christos H. Papadimitriou. Searching and pebbling. *Theoretical Computer Science*, 47(2) :205–218, 1986.

[Kru60] Joseph B. Kruskal. Well-quasi-ordering, the tree theorem, and vazsonyi's conjecture. *Transactions of the American Mathematical Society*, 95(2) :210–225, 1960.

[Kur30] Kazimierz Kuratowski. Sur le problemes des courbes gauche en topologie. *Fund. Math.*, 15 :271–283, 1930.

[LaP93] Andrea S. LaPaugh. Recontamination does not help to search a graph. *Journal of the ACM*, 40(2) :224–245, 1993.

[Lav07] François Laviolette. Communication personnelle, juillet 2007.

[LH93] Steve M. LaValle and Seth Hutchinson. Game theory as a unifying structure for a variety of robot tasks. In *Proceedings of the IEEE International Symposium on Intelligent Control*, pages 429–434, 1993.

[Lin78] Andrzej Lingas. A pspace complete problem related to a pebble game. In *Proceedings of the 5th Colloquium on Automata, Languages and Programming (ICALP)*, pages 300–321, 1978.

[LLG+97] Steve M. LaValle, David Lin, Leonidas J. Guibasa, Jean-Claude Latombe, and Rajeev Motwani. Finding an unpredictable target in a workspace with obstacles. In *Proceedings of the IEEE International Conference on Robotics and Automaton (ICRA)*, pages 737–742, 1997.

[Lov06] László Lovász. Graph minor theory. *Bull. Amer. Math. Soc.*, 43 :75–86, 2006.

[LT79] Thomas Lengauer and Robert E. Tarjan. Upper and lower bounds on time-space tradeoffs. In *Proceedings of the 11th annual ACM symposium on Theory of computing (STOC)*, pages 262–277, 1979.

[LT82] Thomas Lengauer and Robert E. Tarjan. Asymptotically tight bounds on time-space trade-offs in a pebble game. *Journal of the ACM*, 29(4) :1087–1130, 1982.

[Luc07] Flaminia L. Luccio. Intruder capture in sierpinski graphs. In *Proceedings of the 4th International Conference on Fun with algorithms (FUN)*, 2007. to appear.

[MHG+88] Nimrod Megiddo, S. Louis Hakimi, Michael R. Garey, David S. Johnson, and Christos H. Papadimitriou. The complexity of searching a graph. *J. Assoc. Comput. Mach.*, 35, 1988.

[MiK07] Bojan Mohar and Ken ichi Kawarabayashi. Some recent progress and applications in graph minor theory. *Graphs Combin.*, 23 :1–46, 2007.

[MN07] Frédéric Mazoit and Nicolas Nisse. Monotonicity of non-deterministic graph searching. In *Proceedings of the 33th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, 2007. to appear.

[Nis04] Nicolas Nisse. Expansion connexe dans les réseaux. Master's thesis, Université Paris XI, Orsay, France, 2004.

[Nis07] Nicolas Nisse. Connected graph searching in chordal graphs. *Discrete Applied Maths.*, 2007. to appear.

[NN98] Stewart W. Neufeld and Richard J. Nowakowski. A game of cops and robbers played on products of graphs. *Discrete Mathematics*, 186(1-3) :253–268, 1998.

[NS07] Nicolas Nisse and David Soguet. Graph searching with advice. In *Proceedings of the 14th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 51–65, 2007.

[NW83] Richard J. Nowakowski and Peter Winkler. Vertex-to-vertex pursuit in a graph. *Discrete Mathematics*, 43 :235–239, 1983.

[Obd06] Jan Obdrzálek. Dag-width : connectivity measure for directed graphs. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 814–821, 2006.

[O'R87] Joseph O'Rourke. *Art gallery theorems and algorithms.* Oxford University Press, New York, 1987.

[Par78a] Torrence D. Parsons. Pursuit-evasion in a graph. In *Theory and applications of graphs (Proc. Internat. Conf., Western Mich. Univ., Kalamazoo, Mich., 1976)*, pages 426–441. Lecture Notes in Math., Vol. 642. Springer, Berlin, 1978.

[Par78b] Torrence D. Parsons. The search number of a connected graph. In *Proceedings of the 9th Southeastern Conference on Combinatorics, Graph Theory, and Computing (Florida Atlantic Univ., Boca Raton, Fla., 1978)*, Congress. Numer., XXI, pages 549–554, Winnipeg, Man., 1978. Utilitas Math.

[Par98] Andreas Parra. *Structural and algorithms aspects of chordal graph embeddings.* PhD thesis, Technische Universität, Berlin, 1998.

[PH70] Michael S. Paterson and Carl E. Hewitt. Comparative schematology. In *Project MAC Conference on Concurrent Systems and Parallel Computation*, pages 177–192, 1970.

[PK98] Vu Anh Pham and Ahmed Karmouch. Mobile software agents : an overview. *Communications Magazine, IEEE*, 36(7) :26–37, 1998.

[PS97] Andreas Parra and Petra Scheffler. Characterizations and algorithmic applications of chordal graph embeddings. *Discrete Applied Mathematics*, 79(1-3) :171–188, 1997.

[PTC76] Wolfgang J. Paul, Robert E. Tarjan, and James R. Celoni. Space bounds for a game on graphs. *Theory of Computing Systems*, 10 :239–251, 1976.

[Qui83] Alain Quilliot. *Thèse de doctorat d'état.* PhD thesis, Université de Paris VI, France, 1983.

[Qui85] Alain Quilliot. A short note about pursuit games played on a graph with a given genus. *J. Comb. Theory, Ser. B*, 38(1) :89–92, 1985.

[Ree97] Bruce Reed. Tree width and tangles, a new measure of connectivity and some applications. *Surveys in Combinatorics*, 1997.

[RS83] Neil Robertson and Paul D. Seymour. Graph minors. i. excluding a forest. *J. Comb. Theory, Ser. B*, 35(1) :39–61, 1983.

[RS85] Neil Robertson and Paul D. Seymour. Graph minors — a survey. *Surveys in Combinatorics*, pages 153–171, 1985.

[RS86] Neil Robertson and Paul D. Seymour. Graph minors. ii. algorithmic aspects of tree-width. *J. Algorithms*, 7(3) :309–322, 1986.

[RS90a] Neil Robertson and Paul D. Seymour. Graph minors. iv. tree-width and well-quasi-ordering. *J. Comb. Theory, Ser. B*, 48(2) :227–254, 1990.

[RS90b] Neil Robertson and Paul D. Seymour. Graph minors. viii. a kuratowski theorem for general surfaces. *J. Comb. Theory, Ser. B*, 48(2) :255–288, 1990.

[RS91] Neil Robertson and Paul D. Seymour. Graph minors. x. obstructions to tree-decomposition. *J. Comb. Theory, Ser. B*, 52(2) :153–190, 1991.

[RS95] Neil Robertson and Paul D. Seymour. Graph minors .xiii. the disjoint paths problem. *J. Comb. Theory, Ser. B*, 63(1) :65–110, 1995.

[RS03a] Neil Robertson and Paul D. Seymour. Graph minors. xvi. excluding a non-planar graph. *J. Comb. Theory, Ser. B*, 89(1) :43–76, 2003.

[RS03b] Neil Robertson and Paul D. Seymour. Graph minors. xviii. tree-decompositions and well-quasi-ordering. *J. Comb. Theory, Ser. B*, 89(1) :77–108, 2003.

[RS04] Neil Robertson and Paul D. Seymour. Graph minors. xx. wagner's conjecture. *J. Comb. Theory, Ser. B*, 92(2) :325–357, 2004.

[RT75] Donald J. Rose and Robert Endre Tarjan. Algorithmic aspects of vertex elimination. In *Proceedings of the 7th Annual ACM Symposium on Theory of computing (STOC)*, pages 245–254, 1975.

[Sch01] Bernd S. W. Schröder. The copnumber of a graph is bounded by $\lfloor \frac{3}{2} genus(g) \rfloor + 3$. *Trends Math.*, pages 243–263, 2001.

[Set73] Ravi Sethi. Complete register allocation problems. In *Proceedings of the 5th annual ACM symposium on Theory of computing (STOC)*, pages 182–195, 1973.

[Sko03] Konstantin Skodinis. Computing optimal linear layouts of trees in linear time. *J. Algorithms*, 47(1) :40–59, 2003.

[SS79] John E. Savage and Sowmitri Swamy. Space-time tradeoffs for oblivious interger multiplications. In *Proceedings of the 6th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 498–504, 1979.

[ST93] Paul D. Seymour and Robin Thomas. Graph searching and a min-max theorem for tree-width. *J. Comb. Theory, Ser. B*, 58(1) :22–33, 1993.

[ST94] Paul D. Seymour and Robin Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2) :217–241, 1994.

[Thi00] Dimitrios M. Thilikos. Algorithms and obstructions for linear-width and related search parameters. *Discrete Applied Mathematics*, 105(1-3) :239–271, 2000.

[Tom78] Martin Tompa. Time-space tradeoffs for computing functions, using connectivity properties of their circuits. In *Proceedings of the 10th annual ACM symposium on Theory of computing (STOC)*, pages 196–204, 1978.

[TUK91] Atsushi Takahashi, Shuichi Ueno, and Yoji Kajitani. Mixed-searching and proper-path-width. In *Proceeedings of the 2nd International Symposium on Algorithms (ISA)*, pages 61–71, 1991.

[TUK95] Atsushi Takahashi, Shuichi Ueno, and Yoji Kajitani. Mixed searching and proper-path-width. *Theor. Comput. Sci.*, 137(2) :253–268, 1995.

[Vil06] Yngve Villanger. Improved exponential-time algorithms for treewidth and minimum fill-in. In *Proceedings of the 7th Latin American Symposium (LATIN)*, pages 800–811, 2006.

[Yan07] Boting Yang. Strong-mixed searching and pathwidth. *J. Comb. Optim.*, 13(1) :47–59, 2007.

[YDA04] Boting Yang, Danny Dyer, and Brian Alspach. Sweeping graphs with large clique number. In *Proceedings of the 15th International Symposium on Algorithms and Computation (ISAAC)*, pages 908–920, 2004.

Jeux des gendarmes et du voleur dans les graphes.

Mineurs de graphes, stratégies connexes, et approche distribuée.

Résumé : Les jeux des gendarmes et du voleur dans les graphes traitent de la capture d'un voleur qui se déplace dans un réseau par une équipe de gendarmes. Ces jeux trouvent leurs motivations en informatique fondamentale, notamment dans le cadre de la théorie de la complexité et dans celui de la théorie des mineurs de graphes. Ces jeux ont également des applications en intelligence artificielle et en robotique. Quel que soit le contexte, le nombre de gendarmes utilisés a un coût et doit être minimisé. Dans cette thèse, nous étudions diverses contraintes auxquelles les stratégies de capture sont soumises, ainsi que le coût de ces contraintes en terme de nombre de gendarmes. Nous distinguons principalement trois cadres d'étude.

Dans la première partie de cette thèse, nous définissons une variante de stratégie de capture qui établit un pont entre la largeur arborescente et la largeur linéaire des graphes. En particulier, nous prouvons la monotonie de cette variante générale et donnons un algorithme exponentiel exact pour calculer de telles stratégies.

Dans la seconde partie de cette thèse, nous nous intéressons aux stratégies dites connexes qui doivent assurer que la partie propre du réseau est constamment connexe. Nous prouvons plusieurs bornes supérieures et inférieures du coût de cette contrainte en terme de nombre de gendarmes. Nous étudions également la propriété de monotonie des stratégies de capture connexe.

Dans la troisième partie de cette thèse, nous étudions les stratégies de capture dans un contexte décentralisé. Nous proposons plusieurs algorithmes décentralisés qui permettent aux gendarmes de calculer eux-mêmes la stratégie qu'ils doivent réaliser.
Mots-clés : stratégie de capture, graphe, monotonie, décompositions arborescente et linéaires.

Graph searching and related problems.

Graph Minors, connected search strategies, and distributed approach.

Abstract: In graph searching problems, a team of searchers is aiming at catching a fugitive running in a network. These problems are motivated by several aspects of theoretical computer science, including computational complexity and minor graph theory. They have also numerous practical applications in artificial intelligence and robotics. In all these contexts, the number of searchers implied in the capture of the fugitive has a cost and must be minimized. In this thesis, we study several constraints of search strategie and their cost in terms of number of searchers. The thesis is divided in three parts.

In the first part of the thesis, we introduce a new variant of search strategy that establishs a link between the treewidth and the pathwidth of a graph. In particular, we prove that this kind of strategy satisfies the monotonicity property, and we propose a exponential exact algorithm computing such a strategy.

In the second part of the thesis, we focus on the so-called connected search strategy that must insure that the clear part of the network remains permanently connected. We prove several upper and lower bounds related to the cost of this constraint in terms of number of searchers. We also study the monotonicity property of the connected search strategies.

In the third part of the thesis, we study the search strategy in a decentralized context. We propose several decentralized algorithms allowing searchers to compute by themselves the strategy that must be performed.
Keywords: graph searching, graphs, monotonicity, tree-decomposition