Graph Searching Games and Graph Decompositions (Course notes)

Nicolas Nisse

Un objectif de ce cours est la présentation des jeux de gendarmes et du voleur comme une introduction à l'étude algorithmique des décompositions de graphes.

Table des matières

1 Quelques références

- Hans L. Bodlaender, *A Partial k-Arboretum of Graphs with Bounded Treewidth*. Theor. Comput. Sci. 209(1-2) : 1-45 (1998)
  http ://people.cs.uu.nl/hansb/
- Fedor V. Fomin, Dimitrios M. Thilikos, *An annotated bibliography on guaranteed graph searching*. Theor. Comput. Sci. 399(3) : 236-245 (2008)
  http ://www.ii.uib.no/ fomin/fedor/papers.html
- N. Nisse, *Les jeux des gendarmes et du voleur dans les graphes. Mineurs de graphes, stratégies connexes et approche distribuée*, thèse de doctorat, Univ. Paris-Sud 11.
  http ://www-sop.inria.fr/members/Nicolas.Nisse/these.pdf

2 Cops and robber / Pursuit-Evasion / Graph Searching games

Beaucoup de noms pour beaucoup de modèles différents qui visent tous le même objectif : la capture (ou le sauvetage, pour être politiquement correct) d'un intrus dans un réseau.

2.1 Objectif commun

Les *jeux des gendarmes et du voleur* traitent tous du même problème : la capture d'un fugitif (ou voleur) dans un réseau (modélisé par un graphe), par une équipe de chercheurs (ou gendarmes ou agents). 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.

2.2 Motivations

intelligence artificielle, robotique, sécurité dans les réseaux informatiques ou la localisation de cibles mobiles par des robots, reconfiguration du routage dans les réseaux optiques, etc.

D'un point de vue plus fondamental : complexité des machines de Turing, décompositions de graphes, étude de l'algorithmique distribuée, etc.

2.3 Nombreuses variantes

facteurs discriminants :

- façon de capturer (lorsqu'un gendarme occupe la même position que le voleur, lorsqu'il se situe à une distance donnée...),
- visibilité des protagonistes (voleur visible/invisible/parfois visible,...),
- positions des protagonistes (sommets et/ou arêtes du graphe),
- façon de "jouer" (voleur et gendarmes se déplacent simultanément, tour-à-tour, etc.),
- Manière de se déplacer (à quelle vitesse ? etc.),
- etc.

Le tableau 1 représente une classification (pas du tout exhaustive ! !) de ces jeux. Les cases marquées de point d'interrogation sont des variantes qui, à ma connaissance, n'ont pas été étudiées.

déplacements	fugitif lent		fugitif rapide	
	visible	invisible	visible	invisible
tour-à-tour	Cops and robber games [Qui83, NW83]	[CN00]	Fast robber [FGK08, NS08]	?
simultanés	?	Helicopter search game [Fom98]	[ST93, FFN05]	Graph searching [Bre67, Par78]

TAB. 1 – Classification générale des jeux des gendarmes et du voleur

2.4 Problème d'optimisation

Un autre critère important : que doit on optimiser ? Dans ce cours, le paramètre que nous considérerons est **le nombre de gendarmes**. Cela correspond à minimiser les ressources nécessaires à la capture. Mais on peut penser au temps de capture, au temps d'occupation des sommets par les gendarmes, etc.

3 Modèle de Parson [Par78]

Nous commençons par présenter le jeu formalisé par Parson en 1976, qui est la première formalisation de ces jeux en termes de théorie des graphes.

3.1 Définition

Etant donné un graphe G (non orienté), les gendarmes (ou agents, ou chercheurs) sont placés sur les sommets et se déplacent d'un sommet à un de ces voisins en suivant les arêtes du graphe. Le voleur (ou fugitif) se situe sur les sommets ou les arêtes de G et se déplace en suivant les chemins dans G. Les protagonistes se déplacent simultanément.

Le voleur est arbitrairement rapide : il se déplace à distance quelconque tant qu'il ne rencontre pas de gendarmes sur son chemin. Enfin, le voleur est *invisible* : les gendarmes ne connaissent sa position que s'ils le rencontrent.

Le voleur est capturé si il croise un agent sur une arête ou un sommet, ou s'il occupe le même sommet qu'un agent (et ne peut pas s'échapper).

Une *stratégie-arête* dans G est une séquence d'opérations des agents, 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.

Initiallement, toutes les arêtes sont dites *contaminées*, i.e., elles hébergent potentiellement le fugitif. 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.

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 dans le pire cas. C'est-à-dire que l'on peut considérer le fugitif comme étant *omniscient* : il voit les agents, connaît leurs déplacements à l'avance et choisit toujours la meilleure solution qui s'offre à lui.

Définition 1 *Le* edge search number $es(G)$ *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.*

La proposition suivante prouve que l'edge search number est bien défini.

Proposition 1 *Pour tout graphe G avec n sommets, $es(G) \leq n$*

(preuve laissée en exercice)

Quel est le edge search number d'un chemin ? d'un cycle ?

Au cours d'une stratégie, une arête est *recontaminée* lors d'une étape s, si elle a été nettoyée à une étape s' antérieure à s, et qu'elle est contaminée (accessible au fugitif) à l'étape s.

Définition 2 *Une stratégie de capture est dite* monotone *si elle n'admet aucune étape de recontamination.*

On note $mes(G)$ le plus petit entier k tel qu'il existe une stratégie de capture-arête **monotone** pour G utilisant au plus k chercheurs. De manière évidente, pour tout graphe G, $es(G) \leq mes(G)$ (preuve ?). En fait, nous verrons qu'il y a égalité [LaP93, BS91].

Ce résultat est fondamental pour plusieurs raisons. Un intérêt des stratégies monotones est qu'elles sont beaucoup plus simples à concevoir et à analyser (cf. Proposition 2). Un second intérêt est que la propriété de monotonie permet de faire le lien entre stratégies de capture et décompositions de graphes (cf. sections suivantes).

Enfin, les stratégies de capture monotones assurent que la capture du fugitif est effectuée en un nombre d'étapes polynomial en la taille du graphe (preuve ?). En déduire le théorème suivant :

Théorème 1 *Le problème de décision suivant appartient à la classe NP :*
Entrée *: un graphe G et un entier $k > 0$,*
Question *: $es(G) \leq k$?*

3.2 Exemples et bref état de l'art

Exercice 1 *Montrer que dans une grille G $n \times m$, $es(G) \leq n$.*

Proposition 2 *Déterminer $es(K_n)$.*

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]. □

Exercice 2 *Soit $k \geq 1$. Montrer que la classe de graphes $\{G \mid es(G) \leq k\}$ est close par mineur. Que peut on en déduire sur la complexité du problème de décision suivant, à k paramètre fixé :*
Entrée : un graphe G
Question : $es(G) \leq k$?

Exercice 3 *[MHG+88] Soit T un arbre à n sommets. Montrer que $es(T) = O(\log n)$.*

Indication. Prouver que $es(T) \geq k+1$ si et seulement si il existe un sommet $v \in V(T)$ tel que il existe au moins 3 composantes T_1, T_2, T_3 de $T \setminus v$ avec $es(T_i) \geq k$, $i \in \{1, 2, 3\}$.

Megiddo *et al.* [MHG+88] prouvent que le problème de déterminer l'edge search number d'un graphe G est NP-difficile, en le réduisant au problème qui consiste à trouver une coupe minimale équilibrée. Donc d'après le Théorème 1 :

Théorème 2 *[MHG+88] Le problème de décision suivant est NP-complet :*
Entrée *: un graphe G et un entier $k > 0$,*
Question *: $es(G) \leq k$?*

Ce problème est cependant polynomial dans certaines classes de graphes. Citons par exemple :

Théorème 3 *[EST94] Il existe un algorithme linéaire pour résoudre le problème :*
Entrée *: un arbre T et un entier $k > 0$,*
Question *: $es(T) \leq k$?*

4 Node search number [KP85]

Dans cette section, nous étudions une variante du modèle de Parson définie par Kirousis et Papadimitriou. Cette variante est une interprétation algorithmique des décompositions linéaires (*path-decompositions*) des graphes.

4.1 Définition

Le modèle que nous présentons dans cette section diffère du précédent par le fait que le fugitif ne peut plus se réfugier "dans" les arêtes, mais est contraint de se positionner sur les sommets du graphe. Plus formellement, le fugitif se situe sur les sommets du graphe se déplace en suivant les chemins dans G. Le voleur est capturé s'il occupe le même sommet qu'un agent (et ne peut pas s'échapper). Une arête est donc nettoyée lorsque chacune de ses deux extrémités est occupée par un agent.

Les autres caractéristiques du modèle sont identiques à celles du modèle de Parson. Une *stratégie-sommet* dans G est une séquence d'opérations des agents choisies parmi les **deux** opérations suivantes :

- Placer un chercheur sur un sommet du graphe ;
- Retirer (ou supprimer) un chercheur d'un sommet du graphe.

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.

Définition 3 *Le* node search number $ns(G)$ *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.*

Montrer que ce nouveau paramètre est bien défini. Quel est le node search number d'un chemin ? d'un cycle ? de $K_{3,3}$? Comparer avec l'edge search number de ces graphes.

Proposition 3 *Soit G une grille $n \times m$, $n \leq m$. $ns(G) = n + 1$.*

Preuve. Pour montrer que $ns(G) \leq n + 1$, il suffit d'exhiber une stratégie de capture-sommet impliquant $n + 1$ gendarmes. Cette partie est laissée à titre d'exercice. Notons qu'exhiber une stratégie *monotone* sera plus naturel et plus facile.

Montrons que $ns(G) > n$. Pour cela, nous allons définir une stratégie du fugitif qui lui permettra de s'échapper face à n gendarmes quelle que soit la stratégie employée par les gendarmes. Introduisons tout d'abord quelques notations et remarques.

Nous notons $G_{p\times q}$ une grille avec p lignes et q colonnes. Sans perte de généralité, $G = G_{n\times m}$. G admet $G_{n\times n}$ comme sous-graphe ($n \leq m$), donc $G_{n\times m} \succeq G_{n\times n}$. On en déduit $ns(G) \geq ns(G_{n\times n})$ (cf. Exercice 5). Il suffit donc de démontrer que $ns(G_{n\times n}) > n$. Soit C l'ensemble des sommets de la "première" colonne de $G_{n\times n}$ (la colonne la "plus à gauche"). Soit L l'ensemble des sommets de la "dernière" ligne de $G_{n\times n}$ moins le sommet appartenant à C. Soit $G' = G_{n\times n} \setminus (L \cup C)$. G' est une grille carré de côté $n - 1$. Soit $X = \{X_1, \cdots, X_{(n-1)^2}\}$ l'ensemble de tous les sous-ensembles X_i de sommets constitués d'une ligne et d'une colonne de G'.

Posons $\mathcal{P} = X \cup \{L\} \cup \{C\}$. Les éléments de $\mathcal{P}$ sont des sous-ensembles de sommets qui induisent des sous-graphes connexes de $G_{n\times n}$. De plus, pour tous $X, Y \in \mathcal{P}$, $G[X]$ *touche* $G[Y]$, i.e., $X \cap Y \neq \emptyset$ ou il existe une arête de $G_{n\times n}$ avec une extrémité dans X et une autre dans Y. Enfin, pour tout

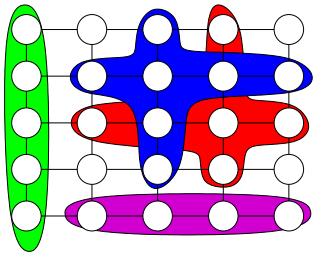

FIG. 1 – Stratégie du voleur dans une grille

ensemble Z de au plus n sommets de $G_{n\times n}$, il existe $\mathcal{P}(Z) \in \mathcal{P}$ tel que $\mathcal{P}(Z) \cap Z = \emptyset$ (ce dernier point est laissé en exercice).

Nous pouvons à présent définir la stratégie du fugitif. Soit $Z_1 \in V(G_{n\times n})$ l'ensemble des positions initiales des gendarmes. Le voleur choisit un sommet de $\mathcal{P}(Z_1)$. A l'étape $i \geq 1$, supposons que les gendarmes occupent les sommets de $Z_i \in V(G_{n\times n})$ et que le voleur occupe un sommet de $\mathcal{P}(Z_i)$. La première phase de la stratégie des gendarmes consiste à supprimer certains agents des sommets qu'ils occupent. Dans un second temps, des agents disponibles sont placés sur des sommets du graphe. Soit Z_{i+1} l'ensemble des sommets occupés par les agents à la suite de cette étape. La stratégie du fugitif consiste à déplacer le fugitif, après la première phase de la stratégie des gendarmes, du sommet qu'il occupe dans $\mathcal{P}(Z_i)$ vers un sommet quelconque de $\mathcal{P}(Z_{i+1})$.

Pour conclure, montrer la validité de la stratégie proposée. □

Exercice 4 *Trouver une stratégie de capture-sommet utilisant* 4 *agents dans le graphe représenté sur la figure 2.*

Exercice 5 *Reprendre la proposition 2 et l'exercice 2 en considérant le node search number.*

Exercice 6 *Soit k fixé. Donner un algorithme de cmoplexité $O(n^{O(k)})$ pour résoudre le problème :*
Entrée *: un graphe G de n sommets,*
Question *: $ns(G) \leq k$?*

Indication. On considérera le *graphe des configurations*. C'est-à-dire le graphe dont chaque sommet est un ensemble composé des positions des k agents et des sommets propres (non occupés par le fugitif). De plus, il y a une arête d'une configuration à une autre si une opération élémentaire d'une stratégie de capture-sommet permet de passer de cette configuration à la suivante. On montrera que $ns(G) \leq k$ si et seulement si il existe un chemin dans H entre la configuration "sale" (aucun agent n'est placé) et la configuration "propre" (tous les sommets sont propres).

4.2 Lien avec l'edge search number

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

Théorème 4 *Pour tout graphe G, $ns(G) - 1 \leq es(G) \leq ns(G) + 1$.*

Preuve. 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. Montrer que les stratégies ainsi définies sont valides. □

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.

Exercice 7 *[KP86] Pour tout graphe G, $ns(G) = es(G^{///}) - 1$, et $es(G) = ns(G^{++}) - 1$. En déduire la compléxité du problème de décision correspondant à ns.*

5 Path decompositions [RS83]

5.1 Définition et premières remarques

Définition 4 *Une décomposition linéaire (path decomposition) d'un graphe G est une paire $(P, \mathcal{X})$, telle que $P = (V(P), E(P))$ est un chemin, et $\mathcal{X} = (X_t)_{t \in V(P)}$ est une famille de sous-ensembles de sommets de G, appelés sacs, qui satisfait :*

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

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

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

De manière équivalente, on représentera une décomposition linéaire d'un graphe G par une séquence $\mathcal{P} = (X_1, \cdots, X_p)$ de sous ensemble de sommets de G.

Remarquons qu'un graphe G admet toujours une décomposition linéaire triviale réduite à un seul sac. La largeur d'une décomposition linéaire $(X_1, \cdots, X_p)$ est égale à $\max_{t \leq p} |X_t| - 1$. 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.

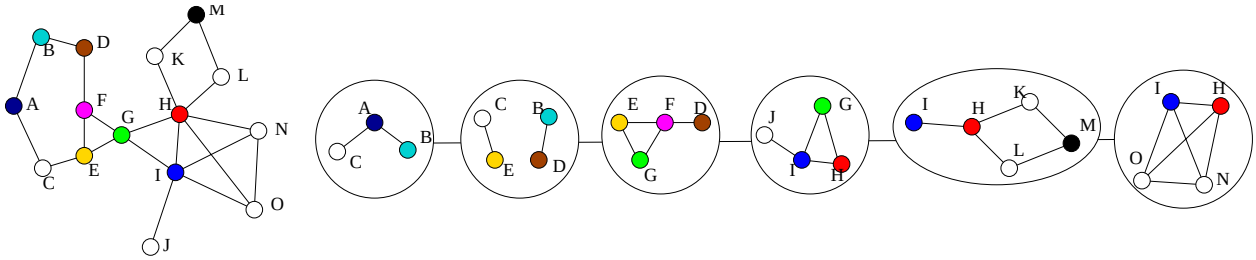

FIG. 2 – Exemple d'un graphe et d'une décomposition linéaire de ce graphe

Exercice 8 *A l'aide de l'exercice 12, déterminer la pathwidth du graphe représenté sur la Figure 2.*

Exercice 9 *Quelle est la pathwidth d'un chemin ? d'un cycle ?*

Exercice 10 *Montrer que la pathwidth est un paramètre clos par mineur.*

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.

Exercice 11 *Soit $(X_1, \cdots, X_p)$ une décomposition linéaire d'un graphe G. Montrer que, pour tout $i < p$, $X_i \cap X_{i+1}$ est un séparateur de G.*

Exercice 12 *Soit G un graphe contenant une clique K_h comme sous graphe. Soit $(X_1, \cdots, X_p)$ une décomposition linéaire de G. Montrer qu'il existe $i \leq p$ tel que $V(K_h) \subseteq X_i$. Donner une borne inférieure pour $pw(G)$. En déduire la pathwidth de K_n, $n \geq 1$.*

Un *graphe d'interval* est un graphe dont les sommets peuvent être repr{esentés par des intervalles réels tels que deux sommets sont adjacents si et seulement si les intervalles correspondant s'intersectent. La figure 3 représente un exemple de graphe d'intervale.

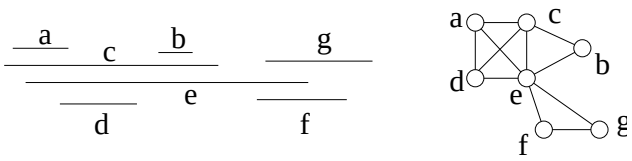

FIG. 3 – Exemple de graphe d'intervale

Exercice 13 *Prouver que le problème suivant est polynomial :*
Entrée Un graphe d'intervalle G et un entier k
Question : $pw(G) \leq k$?

Indication. Donner une preuve par récurrence sur $|V(G)|$. On remarquera qu'on peut toujours trouver une décomposition linéaire $(X_1, \cdots, X_p)$ telle que X_i induit une clique pour tout $i \leq p$.

Théorème 5 *[EST94] Pour tout graphe G, $ns(G) = pw(G) + 1$.*

Preuve. 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 (2-a) supprimer les chercheurs de $X_i \setminus X_{i-1}$ (il reste des chercheurs sur chaque sommet de $X_i \cap X_{i-1}$), (2-b) placer des chercheurs sur les sommets de X_i. Prouver que cette stratégie est une stratégie de capture-sommet monotone. En déduire que $ns(G) \leq pw(G) + 1$.

Réciproquement, 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. Soit X_i l'ensemble des sommets occupés par un gendarme lorsqu'un gendarme est placé sur v_i. Montrer que $(X_1, \cdots, X_n)$ est une décomposition linéaire de G.

Nous avons donc prouvé que $ns(G) \leq pw(G) + 1 \leq mns(G)$, avec $mns(G)$ le plus petit entier k tel qu'il existe une stratégie de capture-sommet **monotone** pour G utilisant au plus k chercheurs. Pour prouver le théorème, nous allons prouver que $ns(G) = mns(G)$ (cf. Théorème 6). □

5.2 Monotonie : "La recontamination n'aide pas"

Notons que le résultat de monotonie a d'abord été prouvé dans le cas de l'edge search number par Andrea LaPaugh [LaP93]. Nous présentons ici la preuve de Bienstock et Seymour plus générale et (beaucoup) plus élégante [BS91]. Cette section est dédiée à la preuve du théorème suivant.

Théorème 6 *[BS91] Pour tout graphe G, $ns(G) = mns(G)$.*

Pour prouver ce résultat, nous devons introduire un autre modèle de capture, dit *mixte-search* car il fait le lien entre node-search et edge-search. Une *stratégie-mixte* se définit comme une stratégie-arête (3 opérations possibles). C'est le nettoyage des arêtes qui est réalisé différemment. Dans ce modèle, une arête est nettoyée lorsque, soit elle est traversée par un agent, soit lorsque chacune de ses deux extrémités est occupée par un agent. En d'autres termes, le fugitif peut se trouver sur les sommets ou les arêtes, mais il est capturé s'il occupe la même position qu'un agent OU s'il occupe une arête dont les extrémités sont occupées.

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.

Définition 5 *Le* mixte search number $mixs(G)$ *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 2 arêtes en parallèle, et G^{+} le graphe obtenu à partir de G en remplaçant chaque arête par 2 arêtes en série.

Exercice 14 *Montrer que, pour tout graphe G, $mixs(G^{+}) = es(G)$, et $mixs(G^{//}) = ns(G)$. En particulier, montrer que ce résultat est vrai pour les variantes monotones de ces paramètres.*

Nous présentons maintenant la preuve de la monotonie du mixte-search number. Le résultat de l'exercice 14 permet de conclure facilement la preuve du Théorème 6.

On considère un graphe $G = (V, E)$. Une *croisade* est une séquence $(E_0, \cdots, E_p)$ de sous-ensemble de E tels que : (1) $E_0 = \emptyset$ et $E_p = E$, et (2) pour tout $i > 0$, $|E_i \setminus E_{i-1}| \leq 1$. Une croisade est *monotone* si $E_0 \subseteq E_1 \subseteq \cdots \subseteq E_p$ et si, pour tout $i > 0$, $|E_i \setminus E_{i-1}| = 1$.

Soit $F \subseteq E$, $\delta(F)$ est l'ensemble des sommets incidents à une arête dans F et une arête dans $E \setminus F$. Informellement, $\delta(F)$ représente la *frontière* de l'ensemble d'arêtes F. La fonction $|\delta|$ a l'avantage d'être symétrique et sous-modulaire, c'est-à-dire : $|\delta(F)| = |\delta(E \setminus F)$ (symétrie) et, pour tout $X, Y \subseteq E$, $|\delta(X \cap Y)| + |\delta(X \cup Y)| \leq |\delta(X)| + |\delta(Y)|$ (sous-modularité).

Exercice 15 *Vérifier que $|\delta|$ est symétrique et sous-modulaire.*

Une croisade $(E_0, \cdots, E_p)$ et de largeur $\leq k$ si, pour tout $i \geq 0$, $|\delta(E_i)| \leq k$.

Exercice 16 *Montrer que si $mixs(G) \leq k$, alors il existe une croisade de G de largeur $\leq k$.*

Exercice 17 *Montrer que si il existe une croisade monotone de G de largeur $\leq k$, alors il existe une stratégie monotone de capture-mixte pour G utilisant au plus k chercheurs.*

Indication. Pour simplifier, on ne considérera que des graphes de degré minimum 2.

Lemme 1 *Si il existe une croisade de G de largeur $\leq k$, alors il existe une croisade monotone de G de largeur $\leq k$.*

Preuve. Soit une croisade $(E_0, \cdots, E_p)$ de G de largeur $\leq k$ telle que :

1. $\sum_i (|\delta(E_i)| + 1)$ est minimum,
2. sous la condition (1), $\sum_i |E_i|$ est minimum.

Nous montrons qu'une telle croisade est monotone.

- $|E_i \setminus E_{i-1}| = 1$, sinon $(E_0, \cdots, E_{i-1}, E_{i+1}, \cdots, E_p)$ serait une croisade contredisant (1).
- $|\delta(E_{i-1} \cup E_i)| \geq |\delta(E_i)|$, sinon $(E_0, \cdots, E_{i-1}, E_{i-1} \cup E_i, E_{i+1}, \cdots, E_p)$ serait une croisade contredisant (1).
- Par sous-modularité, $|\delta(E_{i-1} \cap E_i)| \leq |\delta(E_i)|$, donc $(E_0, \cdots, E_{i-2}, E_{i-1} \cap E_i, E_i, E_{i+1}, \cdots, E_p)$ est une croisade de largeur $\leq k$.
- D'après (2), on doit avoir $|E_{i-1} \cap E_i| \geq |E_{i-1}|$, donc $E_{i-1} \subseteq E_i$.

□

Déduire des exercices 16, 17 et du lemme 1, qu'il existe toujours une stratégie de capture-mixte monotone utilisant $mixs(G)$ agents. Puis, en utilisant le résultat de l'exercice 14, en déduire le Théorème 6. Conclure la preuve du Théorème 5.

6 Tree decompositions [RS86]

6.1 Définition et premières remarques

Définition 6 *Une décomposition arborescente (tree decomposition) 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, $X_a \cap X_b \subseteq X_c$.*

Remarquons qu'un graphe G admet toujours une décomposition arborescente triviale réduite à un seul sac. La largeur d'une décomposition arborescente $(T, \mathcal{X})$ est égale à $\max_{t \in V(T)} |X_t| - 1$. La largeur arborescente, notée $tw(G)$ (pour *treewidth* en anglais), d'un graphe G est la largeur minimum d'une décomposition arborescente de G parmi toute les décompositions arborescentes du graphe G.

Exercice 18 *Montrer que pour tout graphe G, $tw(G) \leq pw(G)$.*

Exercice 19 *Reprendre l'exercice 12 en considérant une décomposition arborescente. En déduire, la treewidth du graphe représenté sur la Figure 4.*

Exercice 20 *Quelle est la treewidth d'un arbre ? d'un cycle ? d'une grille carrée de côté n ? Donner un graphe pour lequel l'inégalité $tw(G) \leq pw(G)$ est stricte.*

Exercice 21 *Montrer que la treewidth est un paramètre clos par mineur.*

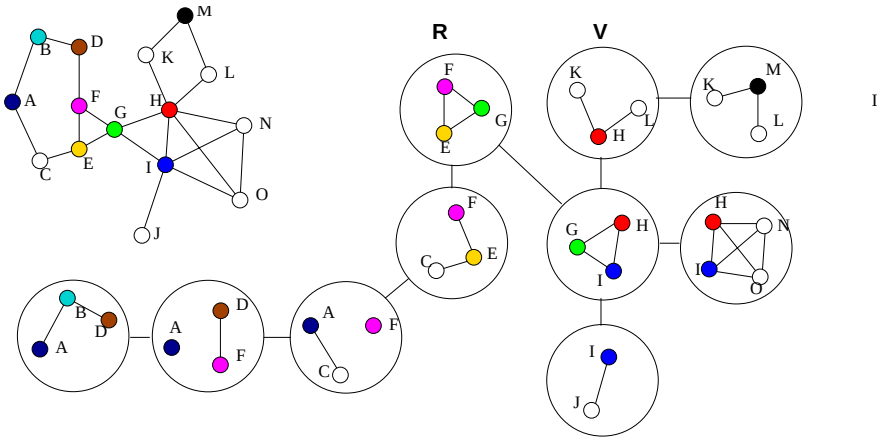

FIG. 4 – Exemple d'un graphe et d'une décomposition arborescente de ce graphe

Soit $(T, \mathcal{X})$ une décomposition arborescente d'un graphe G. Soit $r \in V(T)$ et supposons l'arbre enraciné en r. Pour tout $v \in V(T)$, on pose T_v le sous arbre de T contenant v, obtenu après suppression du père de v dans T. De plus, on note $G[T_v]$ le sous-graphe de G induit par $\cup_{w \in V(T_v)} X_w$. Par exemple, dans la Figure 4, l'arbre de décomposition est enraciné en $R \in V(T)$ et $G[T_V]$ est le sous-graphe induit par $\{H, K, L, M\} \subset V(G)$.

Exercice 22 *Soit $(T, \mathcal{X})$ une décomposition arborescente d'un graphe G, avec T enraciné. Soit $\{u, v\} \in E(T)$ avec u le père de v. Soit $S = X_u \cap X_v$ et soit $A = V(G[T_v])$. Montrer que S est un séparateur de $A \setminus S$ et $V(G) \setminus A$.*

Un graphe est un *k-arbre* si c'est une clique de $k+1$ sommets, ou s'il peut être obtenu d'un k-arbre H en ajoutant un nouveau sommet adjacent à ceux d'une clique de taille k induite dans H. Un graphe est un *k-arbre partiel* s'il est un sous-graphe d'un k-arbre.

Exercice 23 *Montrer que la treewidth d'un graphe G est égale au plus petit entier k tel que G est un k-arbre partiel.*

Lemme 2 *Soit G un graphe de treewidth au plus k. Alors, il existe une décomposition arborescente $(T, \mathcal{X})$ de G, de largeur k et telle que : $|V(T)| = O(|V(G)|)$ et T de degré maximum 3.*

Preuve. Laissée en exercice. Un lecteur paresseux pourra, par exemple, se référer à [Bod98]. □

Parlons maintenant un peu de complexité.

Théorème 7 *[ACP87] Le problème suivant est NP-complet :*
Entrée Un graphe G et un entier k
Question : $tw(G) \leq k$?

Par contre, lorsque k est un paramètre fixe, il existe un algorithme linéaire pour la même question. En d'autres termes, la complexité du problème de déterminer la treewidth d'un graphe n'est pas due à la taille de l'instance mais à sa treewidth.

Théorème 8 *[Bod96, BK96] Soit k fixé. Il existe un algorithme linéaire pour résoudre le problème :*
Entrée Un graphe G
Question : $tw(G) \leq k$?

Attention, la complexité de l'algorithme mensionné ci-dessus est de l'ordre de $O(2^{k^2}|V(G)|)$. Il n'est donc utilisable en pratique que pour de toutes petites valeurs de k :(

Un graphe est *cordal* si tout cycle de longueur au moins 4 contient une corde. C'est à dire que les seuls cycles induits sont ceux de longueur 3.

Exercice 24 *Prouver que le problème suivant est polynomial :*
Entrée Un graphe cordal G et un entier k
Question : $tw(G) \leq k$?

Indication. Montrer que tout graphe cordal admet un sommet simplicial (dont le voisinage induit une clique). Puis procéder par récurrence.

6.2 Lien avec le graph searching

Dans cette section, nous présentons brièvement une variante de jeu de capture qui est une interprétation algorithmique des décompositions arborescentes.

Cette variante est identique au modèle "node search", mais les agents ont connaissance de la position du voleur à tout moment. On dit que le voleur est *visible*.

Une *stratégie-sommet* pour capturer un fugitif visible dans G est définie de la même manière que dans le cas d'un fugitif invisible (node search). C'est une séquence d'opérations des agents choisies parmi les **deux** opérations suivantes :

- Placer un chercheur sur un sommet du graphe ;
- Retirer (ou supprimer) un chercheur d'un sommet du graphe.

Etant donné un graphe G, une *stratégie de capture-sommet* est une stratégie-sommet qui garantit la capture du fugitif visible quelle que soit sa stratégie.

Définition 7 *Le* visible node search number *$vns(G)$ d'un graphe G, est le plus petit entier k tel qu'il existe une stratégie de capture-sommet pour capturer un fugitif visible dans G en utilisant au plus k chercheurs.*

Calculer le visible node search number d'un arbre, d'une clique à n sommets.

Exercice 25 *Montrer que pour tout graphe G, $vns(G) \leq ns(G)$. Donner un graphe pour lequel l'inégalité est stricte.*

Théorème 9 *[ST93, MN08] Pour tout graphe G, $vns(G) = tw(G) + 1$.*

Preuve. Soit $(T, \mathcal{X})$ une décomposition arborescente de G, avec T enraciné. Considérons la stratégie définie par récurrence de la manière suivante : initialement, placer les agents sur les sommets de X_r. A l'étape $i \geq 1$, supposons que les agents occupent les sommets de X_v où $v \in V(T)$ et supposons que le fugitif occupe un sommet de $G[T_v]$. Notons que cela est vrai à l'étape initiale.

Si v est une feuille de T, $V(G[T_v]) = X_v$ et le fugitif est capturé (il occupe le même sommet qu'un agent). Donc, soit $w_1, \cdots, w_r$ les fils de v dans T. Sans perte de généralité, supposons que le

fugitif occupe un sommet de $G[T_{w_1}]$. L'étape $i+1$ consiste alors à (1) retirer les agents qui occupent des sommets dans $X_v \setminus X_{w_1}$, puis (2) placer les agents sur les sommets de X_{w_1}.

En utilisant l'exercice 22, prouver que cette stratégie est une stratégie de capture-sommet monotone d'un fugitif visible. En déduire que $vns(G) \leq tw(G) + 1$.

Pour prouver l'inégalité inverse, nous prouvons une assertion plus forte par récurrence sur $n = |V(G)|$. Soit S une stratégie de capture-sommet monotone pour un fugitif visible dans G utilisant k agents. Soit R_S l'ensemble des sommets occupés par des agents avant la première étape de suppression d'un agent dans S. Nous prouvons qu'il existe une décomposition arborescente de G, de largeur au plus $k-1$, dont l'un des sacs est exactement R_S.

Le résultat est vrai pour $n = 1$. Supposons que $n > 1$ et que le résultat est vrai pour tout $j < n$. Soit S une stratégie de capture-sommet monotone pour un fugitif visible dans G en utilisant au plus k agents. Soit v le premier sommet duquel est supprimé un agent au cours de la stratégie.

Soit S' la séquence d'opérations obtenue à partir de S en supprimant les opérations "placer un agent sur v" et "supprimer l'agent de v". Montrer que S' est une stratégie de capture-sommet monotone pour un fugitif visible dans $G \setminus \{v\}$ en utilisant au plus k agents. Montrer que $R_S \setminus \{v\} \subseteq R_{S'}$.

D'après l'hypothèse de récurrence, il existe une décomposition arborescente $(T', \mathcal{X}')$ de $G \setminus \{v\}$ de largeur au plus $k-1$. De plus, il existe $x \in V(T')$ tel que $X'_x = R_{S'}$.

Soit $(T, \mathcal{X})$ défini de la manière suivante : T est obtenu de T' en ajoutant un sommet y adjacent à x, et pour tout $z \in V(T)$, $X_z = X'_z$ si $z \neq y$ et $X_y = R_S$. Montrer que $(T, \mathcal{X})$ est une décomposition arborescente de G de largeur au plus $k-1$.

Nous avons donc prouvé que $vns(G) \leq tw(G)+1 \leq mvns(G)$, avec $mvns(G)$ le plus petit entier k tel qu'il existe une stratégie de capture-sommet **monotone** pour un fugitif visible dans G utilisant au plus k chercheurs. Pour prouver le théorème, il "suffit" de prouver que $vns(G) = mvns(G)$. Pour cela, voir [ST93, MN08]. □

Un graphe est *planaire extérieur* si il est planaire (on peut le dessiner dans le plan sans croisement d'arêtes) et que l'on peut le dessiner de manière à ce que tous les sommets soient sur la face extérieure.

Exercice 26 *Quelle est la complexité du problème :*
Entrée *: un graphe outer-planar G et un entier $k > 0$,*
Question *: $tw(G) \leq k$?*

Indication. Trouver une borne supérieure simple pour $vns(G)$, puis utiliser les théorèmes 8 et 9.

7 Intérêt algorithmique des décompositions

Le but de cette section est de donner un aperçu de la "puissance" des décompositions arborescentes des graphes.

7.1 Exemples d'applications

De nombreux graphes rencontrés en pratique sont de petites treewidth. Par exemple, les graphes associés aux programmes sont de treewidth $4, 5$ ou 6 selon le langage de programmation [Tho98].

Il en est de même pour les graphes issus des problèmes de satisfaction de contraintes. Par ailleurs, les bases de données peuvent êtres représentées grâce à des hypergraphes acycliques. etc.

Une raison intuitive d'utiliser des décompositions de graphes est liée au paradigme "diviser pour régner". Décomposer un problème en différents sous-problèmes plus faciles, associés aux différentes parties de la décomposition donnée du graphe, puis combiner les solutions entre elles. En général, le temps nécessaire pour recombiner les solutions entre elles dépend uniquement de la *largeur* de la décomposition. Ainsi, ces décompositions sont intéressantes entre autres car elles fournissent un cadre général pour résoudre des problèmes NP-complets en général en temps polynomial lorsque les instances sont restreintes à certaines classes particulières.

Un résultat particulièrement important relatif à la notion de décomposition arborescente, dû à Courcelle et Mosbah [CM93], est que tout problème exprimable en logique du second ordre monadique peut être résolu en temps linéaire dans les classes de graphes de largeur arborescente bornée.

Par ailleurs, le concept de programmation dynamique effectuée grâce à une décomposition arborescente de l'instance d'un problème, a permis la conception d'algorithmes résolvant en temps polynomial (voire linéaire) en la taille de l'instance, dans le cas de nombreux problèmes NP-complets en général [Bod88].

7.2 Programmation dynamique par l'exemple.

Dans cette section, nous donnons l'ébauche d'un algorithme de programmation dynamique utilisant les décompositions arborescentes.

Soit $k \geq 1$. Dans la suite nous considérons uniquement des graphes de treewidth au plus k. Rappelons que, d'après le résultat de Bodlaender [Bod96, BKK95] (Théorème 8 de cette note), il existe un algorithme linéaire pour calculer une décomposition arborescente optimale pour toute instance appartenant à cette classe de graphe.

Informellement, la technique de programmation dynamique sur une décomposition arborescente fonctionne de la manière suivante.

1. Calculer une décomposition arborescente $(T, \mathcal{X})$ de largeur k du graphe G. On imposera que T ait un degré borné. De plus, T est supposé enraciné en un sommet r arbitraire.
2. Pour chaque feuille $v \in V(T)$, résoudre le problème sur $G[X_v]$. Notons que pour que l'algorithme soit efficace, le problème doit pouvoir être résolu efficacement sur des instances de taille bornée.
3. Pour tout sommet $v \in V(T)$ avec $r \geq 1$ fils $w_1, \cdots, w_r$. A partir des solutions obtenues pour $G[T_{w_1}], \cdots, G[T_{w_r}]$, résoudre le problème pour $G[T_v]$. Ceci n'est bien évidemment pas toujours possible. Nous verrons par l'exemple que les propriétés des décompositions arborescentes jouent un rôle important ici. Informellement, nous verons qu'il s'agit de résoudre le problème pour $G[T_v]$ à partir de solutions de $G[T_{w_1}] \cap X_v, \cdots, G[T_{w_r}] \cap X_v$ et que cela est possible (dans certains cas) puisque les ensembles $G[T_{w_i}] \cap X_v$ sont des séparateurs.

Remarque. Il est important de noter ici que les problèmes considérés doivent être invariant par sous-graphe, sans quoi la solution obtenue n'a aucune garantie d'être optimale.

Rappelons que le problème de coloration consiste à déterminer la nombre minimum de couleurs nécessaires pour étiqueter les sommets d'un graphe G de manière à obtenir une coloration propre, c.-à-d., telle que deux sommets adjacents aient des couleurs différentes. Ce nombre est le *nombre chromatique* de G, $\chi(G)$.

Théorème 10 *Le problème suivant est NP-complet.*
Entrée Un graphe G et un entier k.
Question : $\chi(G) \leq k$?

Exercice 27 *Soit $k \geq 1$. Prouver que le problème suivant est dans P*
Entrée Un graphe G de treewidth au plus k et un entier c.
Question : $\chi(G) \leq c$?

Le problème du stable maximum d'un graphe G consiste à trouver un sous-ensemble de sommets indépendants (non adjacents) de taille maximum. La taille d'un tel ensemble est notée $\alpha(G)$.

Théorème 11 *Le problème suivant est NP-complet.*
Entrée Un graphe G et un entier k.
Question : $\alpha(G) \geq k$?

Exercice 28 *Soit $k \geq 1$. Prouver que le problème suivant est dans P*
Entrée Un graphe G de treewidth au plus k et un entier c.
Question : $\alpha(G) \geq c$?

Références

[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.

[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.

[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.

[Bod96] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6) :1305–1317, 1996.

[Bod98] Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theor. Comput. Sci.*, 209(1-2) :1–45, 1998.

[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.

[CM93] Bruno Courcelle and Mohamed Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theor. Comput. Sci.*, 109(1&2) :49–82, 1993.

[CN00] Nancy E. Clarke and Richard J. Nowakowski. Cops, robber, and photo radar. *Ars Comb.*, 56 :97–103, 2000.

[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.

[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.

[FGK08] Fedor V. Fomin, Petr A. Golovach, and Jan Kratochvíl. On tractability of cops and robbers game. In *Fifth IFIP International Conference On Theoretical Computer Science (IFIP TCS)*, volume 273 of *IFIP*, pages 171–185. Springer, 2008.

[Fom98] Fedor V. Fomin. Helicopter search problems, bandwidth and pathwidth. *Discrete Applied Mathematics*, 85(1) :59–70, 1998.

[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.

[LaP93] Andrea S. LaPaugh. Recontamination does not help to search a graph. *Journal of the ACM*, 40(2) :224–245, 1993.

[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.

[MN08] Frédéric Mazoit and Nicolas Nisse. Monotonicity of non-deterministic graph searching. *Theoretical Computer Science*, 399 :169–178, 2008.

[NS08] Nicolas Nisse and Karol Suchan. Fast robber in planar graphs. In *Graph-Theoretic Concepts in Computer Science, 34th International Workshop (WG)*, volume 5344 of *Lecture Notes in Computer Science*, pages 312–323, 2008.

[NW83] Richard J. Nowakowski and Peter Winkler. Vertex-to-vertex pursuit in a graph. *Discrete Mathematics*, 43 :235–239, 1983.

[Par78] 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.

[Qui83] Alain Quilliot. *Thèse de doctorat d'état*. PhD thesis, Université de Paris VI, France, 1983.

[RS83] Neil Robertson and Paul D. Seymour. Graph minors. i. excluding a forest. *J. Comb. Theory, Ser. B*, 35(1) :39–61, 1983.

[RS86] Neil Robertson and Paul D. Seymour. Graph minors. ii. algorithmic aspects of tree-width. *J. Algorithms*, 7(3) :309–322, 1986.

[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.

[Tho98] Mikkel Thorup. All structured programs have small tree-width and good register allocation. *Inf. Comput.*, 142(2) :159–181, 1998.

8 Annexes

8.1 Mineur de graphes

The *contraction* of the edge e with endpoints u, v is the replacement of u and v with a single vertex whose neighbors are the vertices that were neighbors of u or v. A graph H is a *minor* of a graph G if H is a subgraph of a graph obtained by a succession of edge contractions of G. En d'autres termes, H peut être obtenu à partir de G par une succession de contractions et de déletions d'árêtes et de délétion de sommets. Si H est un mineur de G, on note $H \prec G$ (la relation de minoration est une relation d'ordre partielle). **Exemple** : H est planaire ssi $K_5 \not\prec H$ et $K_{3,3} \not\prec H$ (Théorème de Kuratowski).

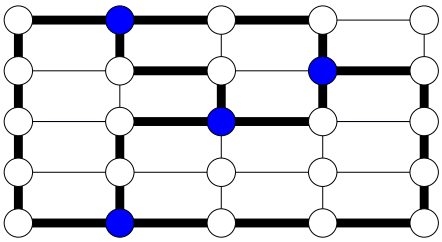

FIG. 5 – Exemple d'un mineur K_4 d'une grille 5×5

Une classe $\mathcal{C}$ de graphes est dite *close par mineurs*, si pour tout $G \in \mathcal{C}$, alors pour tout mineur H de G, $H \in \mathcal{C}$. Un invariant μ de graphe est dit *clos par mineurs* si, la classe de graphes $\{G \mid \mu(G) \leq k\}$ est close par mineurs. C'est-à-dire, μ est clos par mineurs si, pour tout graphe G et pour tout mineur H de G, $\mu(H) \leq \mu(G)$.

Nous citons un résultat fondamental dù à Robertson et Seymour, qu'il sera utile de connaître.

Théorème 12 *Pour toute classe de graphes close par mineurs, il existe un algorithme en temps polynomial pour décider si un graphe appartient à cette classe.*

8.2 Classes de complexité

Nnous ne donnons ici que des notions très basiques. Exemple de problèmes de décision :
Entrée : un graphe G, un entier k
Question : G est-il k colorable ?

On note P la classe des problèmes de décision pour lesquels il existe un algorithme déterministe résolvant ce problème en temps polynomial en la taille de l'instance.

On note NP la classe des problèmes de décision pour lesquels il existe un algorithme non déterministe résolvant ce problème en temps polynomial en la taille de l'instance. NP est aussi défini comme la classe des problèmes pour lesquels une solution (certificat) est vérifiable en temps polynomial. Bien sûr, $P \subseteq NP$. Challenge : $P = NP$?

Informellement, un problème est NP-difficile si "il est au moins aussi difficile que les problèmes les plus difficiles de NP". Plus précisément, un problème H est NP-difficile, s'il existe un problème NP-difficile qui peut être réduit à H en temps polynomial. Un problème est NP-complet s'il est NP-difficile et s'il appartient à la classe NP. Le problème 3-SAT appartient à la classe des problèmes NP-complets.