ECOLE POLYTECHNIQUE
PROMOTION 2007
HSU Henry

RAPPORT DE STAGE DE RECHERCHE

Pathwidth des graphes planaires extérieurs 2-connectés

NON CONFIDENTIEL

Henry Wei Cheng HSU, orienté par nicolas nisse et david coudert

Ecole Polytechnique - MASCOTTE, projet commun INRIA, I3S (CNRS, Univ. de Nice Sophia Antipolis)

5 Juillet - 2010

Option : Informatique
Champ de l'option : théorie des graphes
Directeur de stage : Nicolas Nisse et David Coudert
Dates du stage : 12 avril 2010 ~ 27 août 2010
Adresse de l'organisme :
Mascotte,
projet commun INRIA, I3S(CNRS, Univ. de Nice Sophia Antipolis)
INRIA Sophia Antipolis - Méditerranée - tél. : (+33) 4 92 38 77 77 - 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Résumé La pathwidth et la treewidth des graphes ont été beaucoup étudiées à cause de leurs importants aspects structurels et algorithmiques. Certains problèmes qui sont NP-complets en général peuvent être résolus de manière efficace dans la classe des graphes de treewidth et pathwidth bornées. La résolution est basée sur la décomposition en arbre et en chemin, qui sont les aspects constructives de la treewidth et de la pathwidth, respectivement.

La détermination de ces paramètres est NP-complet en général, pourtant elle peut devenir polynomiale pour certaines classes de graphes. En particulier, des algorithmes efficaces ont été proposés pour le calcul de la pathwidth dans la classe des arbres. Skodinis [11] (2000) a proposé un algorithme linéaire pour cette question. Pour la classe des graphes planaires extérieurs 2-connectés un algorithme en $O(n^{11})$ existe mais il est inefficace et impraticable.

La pathwidth a été aussi étudiée pour sa relation avec les jeux de recherche en graphes. Familièrement, ces jeux visent à déterminer le nombre minimum d'agents nécessaires pour capturer un fugitive dans un graphe. Dans le cas d'un fugitive invisible, la valeur ns(G) (node search number) est la quantité minimum d'agents requise pour la capture. Il est montré que la pathwidth d'un graphe G est égale à ns(G)-1.

Dans ce rapport on utilise l'interprétation des jeux de recherche pour l'élaboration d'un algorithme pour le calcul de la pathwidth des graphes planaires extérieurs 2-connectés. Dans ce stage, nous avons obtenu des résultats préliminaires pour cette étude. En particulier, deux théorèmes obtenus sont un pas important pour le développement d'un algorithme pratique et efficace pour cette classe de graphes.

Part I
Introduction

Les notions de pathwidth et de treewidth ont été introduites par Robertson et Seymour en 1986 comme éléments importants dans le projet des mineurs de graphes.[1] Depuis, ces notions attirent de plus en plus d'intérêt car ils sont à la base de divers résultats profonds en théorie de graphes et sont très utiles pour l'analyse de plusieurs problèmes pratiques.[5] Certains problèmes qui sont NP-complets deviennent polynomiaux pour des graphes de treewidth et pathwidth bornées.[5] En outre, ces notions sont l'ingrédient crucial pour divers applications en intelligence artificiel, base de donnés et conception de circuits logiques.

Intuitivement, la treewidth mesure jusqu'à quel point un graphe ressemble à un arbre. De la même façon, la pathwidth mesure la similarité d'un graphe avec un chemin. Par exemple, la treewidth d'un arbre est égale à 1 et la pathwidth d'un chemin vaut 1 aussi. Arbres et chemins sont des structures relativement simples et plusieurs problèmes difficiles peuvent être résolus efficacement dans ces classes de graphes.[5] La connaissance de la similarité d'un graphe à un arbre permet le développement d'algorithmes efficaces qui explorent la struc-

ture arborescente du graphe. Par exemple, des problèmes comme la coloration des graphes, le circuit hamiltonien et le "maximum independent set" peuvent être résolus en temps polynomial (en la taille de l'instance) par programmation dynamique dans la classe des graphes de treewidth borné, en utilisant une décomposition arborescente, c'est à dire la décomposition associée au paramètre treewidth.

Les deux paramètres, pathwidth et treewidth, possèdent des jolies interprétations en jeux de recherche [3]. La pathwidth peut être décrite comme un jeu où des agents, à la recherche d'un fugitif, sont successivement mis et enlevés des sommets d'un graphe.[12]. Le but est la capture du fugitif, qui est à la fois invisible et arbitrairement rapide sur les chemins du graphe. Le fugitif n'est pas autorisé à traverser les sommets actuellement occupés par des agents. Par ailleurs, le fugitif est capturé quand un agent est placé sur le sommet qu'il occupe, et en plus, les sommets voisins sont gardés par des agents, pour éviter sa fuite. La pathwidth du graphe G est égale au nombre minimum d'agents nécessaires pour la capture, moins un. De plus, une stratégie de capture est exactement équivalente à une décomposition en chemin[13]. De façon similaire, la treewidth peut être interprétée comme la capture d'un fugitif visible.

Dans le cas des graphes généraux, le calcul de la treewidth et la pathwidth est NP-Complet. Cependant, pour plusieurs classes de graphes, il existe des algorithmes polynomiaux pour ces calculs. Dans la classe des arbres, par exemple, la pathwidth peut être calculé en temps linéaire.

Dans ce rapport, nous étudions la classe des graphes planaires extérieurs 2-connexes. Ces graphes possèdent une représentation planaire de sorte que tous les sommets sont sur le bord de la face extérieure. De plus, le fait qu'ils sont 2-connexes indique l'existence de deux chemins disjoints entre chaque pair de sommets u et v du graphe, excluant u et v. Cette classe de graphes est une continuation naturel dans l'échelle de la treewidth, puisque ce sont justement les graphes de treewidth égales à 2.

Nous nous proposons à étudier la pathwidth des graphes planaires extérieurs 2-connexes. Bodlaender et Kloks ont proposé un algorithme en temps $O(n^{11})$ pour cette tâche. Pourtant, ce résultat possède une importance plutôt théorique. La complexité élevée et le fait qu'il est extrêmement difficile à implémenter rendent l'algorithme impraticable et inefficace. Récemment, des approximations ont été proposés [14][15][16], mais un algorithme exact et rapide n'est toujours pas connu.

Cet algorithme serait une étape importante pour le traitement de la pathwidth pour des graphes généraux. Après l'algorithme linéaire pour les arbres, un algorithme efficace pour les graphes de treewidth 2 est une approche naturel à suivre. Il nous permettrait de développer des nouveaux outils et de synthétiser une compréhension plus profonde pour pouvoir traiter la pathwidth en graphes à chaque fois plus géneraux.

L'algorithme linéaire pour la pathwidth des arbres est basé sur le théorème de Parsons [17] et utilise principe de la programmation dynamique. Nous proposons d'adapter cette méthode pour la recherche de l'algorithme pour les graphes planaires extérieurs 2-connexes. Ce rapport présente les résultats préliminaires

obtenus, en particulier, deux théorèmes qui sont un pas initial important vers un algorithme pratique et rapide pour la pathwidth des graphes planaires 2-connexes.

Part II
Préliminaires

Définition Treewidth

Une *décomposition arborescente* d'un graphe G = (V, E) est un pair ($\{X_i|$ i ϵ I$\}$, T=(I, F)) avec $\{X_i|$ i ϵ I$\}$ une famille de sous-ensembles de V, un pour chaque sommet de T, et T un arbre tel que

- $\bigcup_{i \epsilon I} X_i$ = V
- pour tous les arêtes (v, w) ϵ E, il existe un i ϵ I avec v ϵ X_i et w ϵ X_i.
- pour tous i, j, k ϵ I: si j est dans le chemin de i à k, alors $X_i \bigcap X_k \subseteq X_j$.

La largeur d'une décomposition arborescente ($\{X_i|$ i ϵ I$\}$, T = (I, F)) est $\max_{i \epsilon I} |X_i|$ - 1. La treewidth d'un graphe G est la largeur minimum parmi tous les décompositions arborescentes possibles de G.

On obtient une définition équivalente, quand la troisième condition dans la définition de décomposition arborescente est remplacé par:

- Pour tous v ϵ V, l'ensemble de sommets {i ϵ I | v ϵ X_i} forment un sous-graphe connexe (i.e., un sous-arbre) de T.

Remarque: Les arbres et forêts ont la treewidth égale à 1.

La figure ci-dessous donne l'illustration d'un graphe et ses décompositions en arbre et en chemin.

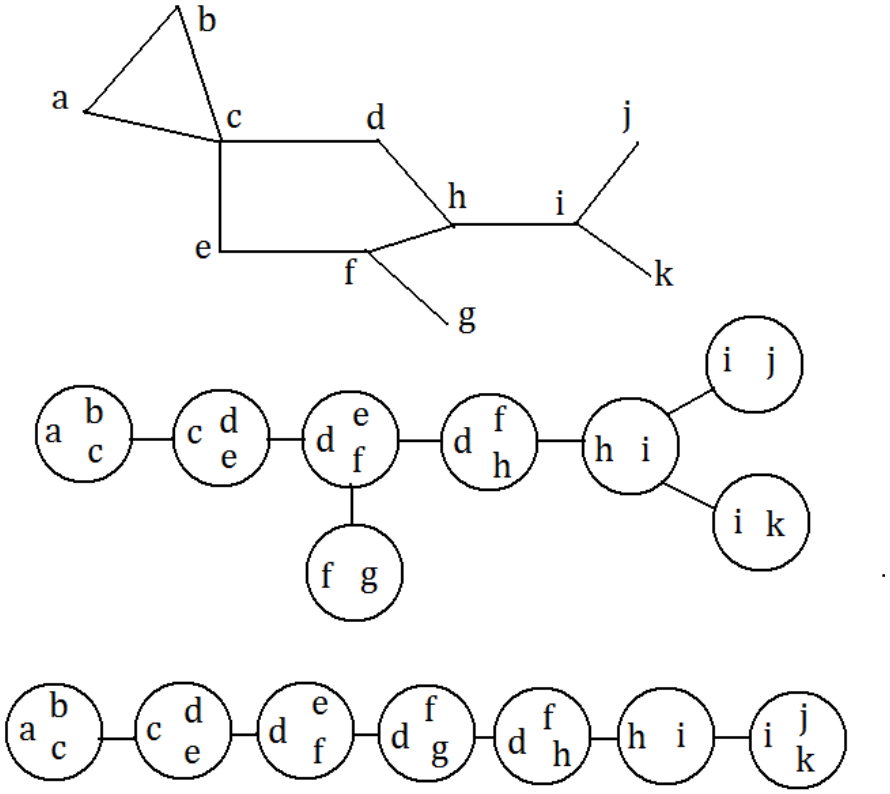


Figure: Un graphe et ses décompositions en arbre et en chemin

La notion de pathwidth est obtenue par la restriction des arbres dans la décomposition arborescente pour chemins.

Définition Pathwidth

Une *décomposition en chemin* d'un graphe G = (V, E) est une séquence de sous-ensembles de sommets (X_i, X_2, ..., X_r), tel que

- $\bigcup_{1 \leq i \leq r} X_i = V$.
- pour tous les arêtes (v, w) ϵ E, il existe un $1 \leq i \leq r$ avec v ϵ X_i et w ϵ X_i.
- pour tous i, j, k ϵ I: si i $\leq$j $\leq$k, alors $X_i \bigcap X_k \subseteq X_j$.

La largeur d'une décomposition en chemin (X_i, X_2, ..., X_r) est $\max_{1 \leq i \leq r} |X_i|$ - 1. La pathwidth d'un graphe G est la largeur minimum pour tous les décompositions en chemin possibles de G.

Notons que chaque décomposition en chemin (X_i, X_2, ..., X_r) peut être écrit comme une décomposition arborescente avec T un chemin.

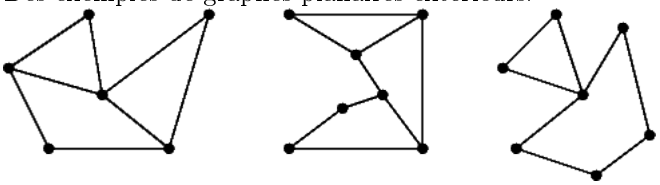
Plusiers investigations récents en théorie des graphes algorithmiques ont été dans le sujet de graphes avec treewidth bornée. La notion de treewidth a été introduit par Robertson et Seymour [1], et elle joue un rôle important dans leur travail fondamental en mineurs de graphes. La pathwidth a été aussi introduit par Robertson et Seymour [2]. Il y a beaucoup d'autres notions en théorie de graphes, qui peuvent être considérés comme équivalents à la treewidth ou à la pathwidth . Une exemple est l'équivalence entre ces deux paramètres et des jeux de recherche [3].

Une raison importante pour l'intérêt en la décomposition arborescente et la décomposition en chemin est le suivant: si l'on a un graphe G = (V, E) avec la treewidth bornée par une constante fixée k et on connaît sa décomposition arborescente, alors on peut résoudre plusieurs problèmes difficiles (NP complet en général) en temps polynomial et souvent en temps linéaire dans ce graphe G. Les problems qui peuvent être abordé de cette manière comportent divers problems NP-complets bien connus, par exemple, "Independent Set", "Circuit Hamiltonien", "Steiner Tree", etc. [5]. En général, nous pouvons citer le travail de Courcelle [21][22]: tout problème exprimable en MSOL (Monadic Second Order Logic) peut être résolu en temps polynomial dans la classe des graphes de treewidth bornée.

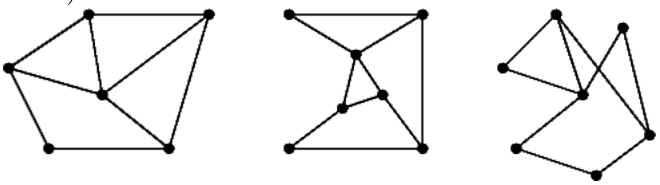
Les graphes planaires extérieurs 2-connexes

Les *graphes planaires extérieurs* sont une sous-classe de graphes planaires avec une simple restriction de plus: le graphe peut être dessiné sur le plan sans croisement d'arêtes de forme que tous les sommets touchent la face extérieure.

Des exemples de graphes planaires extérieurs:



Mais les graphes suivants ne sont pas planaires extérieurs (pourtant ils sont planaires):



Un graphe est 2-connexe si pour tout sommet u de G, le graphe obtenu en supprimant le sommet u et les arêtes qui lui sont incidentes est connexe. Cette propriété de 2-connexité indique que pour deux sommets v et w quelconques, il existe deux chemins de v à w disjoints sauf à v et à w.

Les graphes planaires extérieurs sont surtout étudiés pour ses applications en chimie et bioinfomatique. Ils attirent aussi l'intêrét dans les domaines de la

théorie des complexités et de dessins de graphes.

Définition (dual faible) Étant donné la représentation G d'un graphe planaire sans croisement d'arêtes, on construit le dual faible G* comme suit: chaque face en G, excluant la face extérieure, est un sommet en G*. Deux sommets en G* sont adjacents si, et seulement si les faces en G ont une arête en commun.

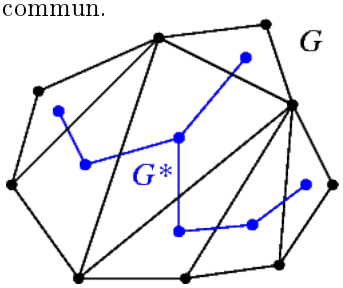


Figure: Illustration d'une graphe planaire G et son dual faible G*.

Observation 1: Le dual faible d'un graphe planaire extérieur est une forêt.

Observation 2: Le dual faible d'un graphe planaire extérieur est un arbre.

Observation 3: La treewidth des graphes planaires extérieurs est égale à 2.

Les 2-approximations pour le problème données en [15] et [16] sont basées sur la rélation entre la pathwidth d'un graphe planaire extérieur 2-connexe G et la pathwidth de son dual G*.

À cause de la structure des graphes planaires extérieurs 2-connexes qui ressemble à celle des arbres, nous essayons de calculer l'algorithme exact pour le problème, en adaptant l'algorithme linéaire déjà existant pour la pathwidth en arbres.

Complexité

Beaucoup de recherche a été fait sur la détermination des paramètres pathwidth et treewidth, ainsi que la décomposition optimale en chemin et en arbre. Le travail existant comprend le calcul de ces paramètres pour des classe spéciales de graphes. Le tableau ci-dessous montre quelques exemples de la complexité pour la pathwidth et la treewidht en différents classes de graphes.

classe	treewidth	pathwidth
degrée limité	NP-C [6]	NP-C [7]
bipartite	NP-C	NP-C
arbres/forêts	linéaire	linéaire [8]
planaire extérieur	linéaire	$O(n^{11})$ [9]
planar	ouvert	NP-C [7]
permutation	P [10]	P [10]
interval	P	P

La recherche en graphes (Graph Searching)

Les jeux de recherche sont des jeux entre un fugitif et des agents dans un graphe. Le fugitif et les agents occupent les sommets du graphe. Le but du fugitif est de s'échapper des agents. Il est capturé quand un agent est placé sur le sommet occupé par le fugitif, et le fugitif ne peut pas fuir. Le fugitif connaît toujours les positions des agents, et se déplace arbitrairement vite. Les agents ne connaissent pas la position du fugitif.

Plus formellement, un *programme de recherche* est un programme déterministe qui prend comme entrée un graphe G et un entier k≥1, et retourne une séquence ordonné d'étapes de recherche. Cette séquence est appelé la *stratégie de recherche* de G. Chaque étape est une opération qui consiste soit en "placer un agent sur v ϵ V" ou "enlever un agent de v ϵ V". Après le placement d'un agent s sur v, et avant qu'il soit enlevé de v, le sommet v est dit *occupé* par l'agent s. Quand un sommet a été occupé par un agent, il devient *nettoyé*. Les sommets qui ne sont pas encore nettoyés sont appelés *contaminés*. Un sommet qui a été nettoyé et qui par la suite est de nouveau accessible au fugitif est recontaminé. Le programme de recherche est correct s'il satisfait les contraintes suivants:

1. Il n'y a jamais plus de k agents simultanément sur les sommets de G;
2. l'étape "placer un agent sur v ϵ V" a lieu au plus une fois, pour tous les sommets v;
3. quand un agent est enlevé d'un sommet v, pour tout chemin P entre v et un sommet contaminé, il existe toujours un agent qui occupe un sommet de P. Nous verrons par la suite que la troisième contrainte n'est pas necessaire: en fait elle peut être omise dans la définition sans changer le problème (ce n'est pas trivial). Par contre, la considérer permet de simplifier la définition des stratégies sans en dégrader les performances.

Définition (node search number) Le node search number de G, noté ns(G), est le minimum nombre d'agents nécessaires par le programme de recherche pour capturer n'importe quel fugitif dans G.

La figure ci dessous ilustre la stratégie pour le node search number d'un chemin. Comme résultat, on a ns (chemin) = 2.

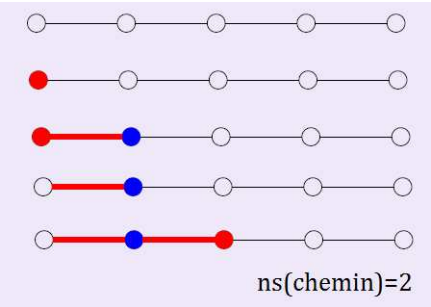


Figure: La stratégie pour nettoyer un chemin.
Les illustrations suivants montrent la stratégie pour nettoyer un cycle:

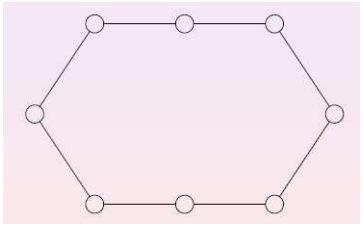


Figure: un cycle

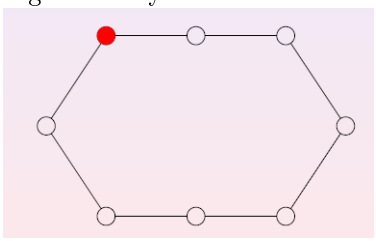


Figure: Le premier agent arrive

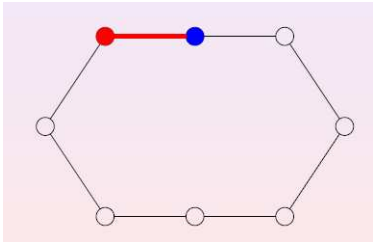


Figure: Le deuxième agent

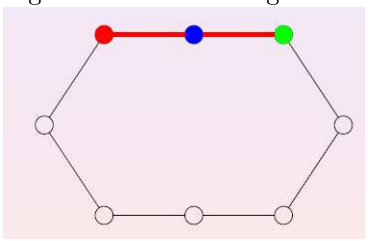


Figure: Le troisième agent

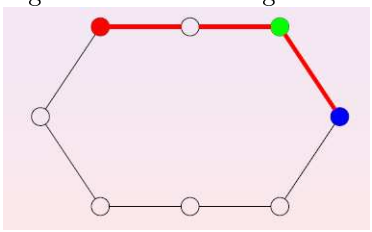


Figure: L'agent du milieu peut sortir parce qu'il n'existe pas de risque que le fugitif entre dans le sommet où l'agent était.

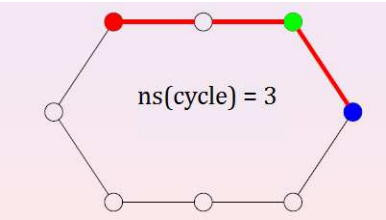


Figure: Si l'on continue, on a comme conclusion que ns(cycle) = 3

Relation Importante La rélation fondamental entre la pathwidth et le node search number a été montré par Ellis et al.[19] et est le suivant:

Pour tout graphe G, on a ns(G) = pathwidth(G) +1.

À cause de cette relation, nous pouvons calculer la pathwidth d'un graphe en utilisant les jeux de recherche.

Définition (p-stratégie monotone) Une p-stratégie est une stratégie qu'utilise p agents pour la capture du fugitif (quel que soit la stratégie du fugitif). Une p-stratégie monotone pour un graphe G est une p-stratégie pour G tel que: si un agent passe par un sommet u du graphe, le fugitif ne peut plus jamais occuper ce sommet. Alors, les agents sont toujours placés sur des sommets contaminés.

LaPaugh [20] a montré que si une p-stratégie existe pour un graphe G, alors il existe une p-stratégie monotone aussi. Dans ce rapport, nous alons travailler seulement avec strátégies monotones.

Définition (p-stratégie finissant en v) Étant donné un graphe G = (V, E) et un sommet v ϵ V, nous disons qu'une p-stratégie fini en v si, une fois qu'un agent arrive en v, il n'est pas enlevé jusqu'à que le fugitif soit capturé.

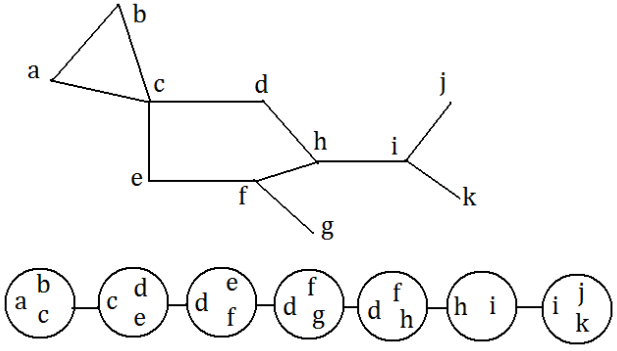
Définition (p-stratégie commençant en v) Étant donné un graphe G = (V, E) et un sommet v ϵ V, nous disons qu'une p-stratégie commence en v si v est le premier sommet couvert par un agent.

Lemme (réversibilité) Étant donné un graphe G = (V, E) et un sommet v ϵ V, si il existe une p-stratégie monotone commençant en v, alors il existe une p-stratégie finissant en v, et vice-versa. En outre, ces deux stratégies ont la propriété de symétrie suivant:

- Le sommet où le k-éme agent arrive dans la stratégie = dans la stratégie reverse, le sommet où se passe la (ns(G) - k +1)-éme sortie d'un agent.

En particulier, si la stratégie commence en v, alors la stratégie reverse fini en v.

Idée de la preuve du lemme de réversibilité Cette preuve est basée surtout sur la relation entre la décomposition en chemin et le node search number. Dans la décomposition arborescente, on peut établir un rapport entre chaque sac X_i et un état du jeu de recherche. Chaque état est caractérisé par quels sommets sont occupés par des agents. Par exemple, dans la figure ci-dessous les sept sacs de la décomposition en chemin représentent sept états différents.



Dans le premier état, trois agents sont placés sur les sommets a, b et c. Ensuite, dans le deuxième état, les agents sortent de a et b, et sont placés sur les sommets d et e, et ainsi de suite jusqu'au dernier état (i, j, k), quand le graphe est complètement nettoyé.

En fait, la décomposition en chemin répresente une stratégie pour nettoyer le graphe. Le plus grand sac de la décomposition indique ainsi le nombre total d'agents nécessaire pour nettoyer le graphe.

La lemme de réversibilité est basé simplement sur le fait qu'on peut suivre les sacs dans les deux sens du chemin. Dans l'exemple de la figure, ça veut dire qu'on a deux possibilités possibles.

On peut commencer en (a, b, c) et suivre jusqu'au état (i, j, k)

Ou sinon, on peut commencer la stratégie en plaçant des agents sur (i, j, k) et ensuite suivre les états en arrière et finir le nettoyage en (a, b, c).

□

Le cas spécial des arbres

Le problème de recherche en arbres est en P et l'on peut trouver le node search number d'un arbre en temps linéaire. [19]. L'idée de base pour l'algorithme est donnée par un théorème sur le node search number dans la classe des arbres. Étant donné un arbre T = (V, E) et un sommet v ϵ V, on appele un sous-arbre T' de T une branche en v si v a degré 1 en T' et T' est l'arbre maximal avec cette propriété. Parsons [17] a montré le théorème suivant.

Théorème de Parsons. *Pour n'importe quel arbre T et entier $k \geq 1$, $ns(T) \geq k+1$ si et seulement si T possède un sommet v sur lequel il y a trois ou plus branches avec node search number k ou plus.*

Cette théorème donne une construction qui oblige le node search number à croître en 1. En général, le théorème de parsons implique que pour tous les arbres T, ns(T) est plus petit que $\log_3(n)$, où n est la quantité de sommets de T. Ça peut être montré par induction sur le valeur du paramètre. En effet, la taille minimum d'une arbre de ns = p+1 est au moins trois fois la taille minimum d'une arbre avec ns = p.

Part III
Contribution

Le but du stage est chercher un algorithme efficace pour la pathwidth des graphes planaires extérieurs 2-connexes. Notre idée est de faire une adaptation de l'algorithme linéaire déjà existant pour la classe des arbres [19]. Dans cette section, nos résultats préliminaires seront présentés, en particulier, deux théorèmes qui sont un pas important pour le développement de l'algorithme.

Pour l'étude de certains propriétés des graphes planaires extérieurs 2-connexe, nous introduisons deux nouveaux paramètres.

Deux nouveaux paramètres

Définition (stratégie commençant en u et v) Soit G un graphe planaire extérieur 2-connexe, et u et v deux sommets de G. On veut nettoyer le graphe G, avec la contrainte que les deux premières agents soient placés sur les sommets u et v. On appelle *X (G, {u, v}, ϕ)* le nombre minimum d'agents nécessaires pour cette tâche.

Remarque: par le lemme de réversibilité, si l'on change la contrainte pour que la stratégie finisse en u et v, on a le même valeur *X (G, {u, v}, ϕ).*

Définition (stratégie commençant en u et finissant en v) Soit G un graphe planaire extérieur 2-connexe, et u et v deux sommets de G. On veut nettoyer le graphe G, avec la contrainte que le premier agent soit placé sur u et la stratégie finisse en v. On appelle *X(G, u, v)* le nombre minimum d'agents nécessaires pour cette tâche.

Remarque: par le lemme de réversibilité, commencer en u et finir en v exige la même quantité d'agents que commencer en v et finir en u.

La programmation dynamique

Définition (branche dans les graphes planaires extérieurs 2-connexes) La figure ci dessous illustre une face F avec deux branches

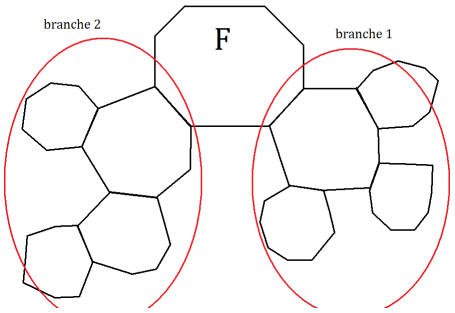


Figure: Une face F dans un graphe planaire extérieur 2-connexe, et ses deux branches.

Théorème 1: une petite introduction

Prenons un graphe planaire extérieur 2-connexe G. Soit F une face de G. On va étudier le cas où F possède au moins 2 branches. On va considérer aussi qu'il existe une arête {u, v} sur F tel que deg(u) = deg(v) = 2.

Pour simplifier les choses, on va partager les branches de F en deux structures, qu'on va appeler le fils gauche et le fils droite de F. Chaque fils consiste simplement d'une union de branches consécutives de F.

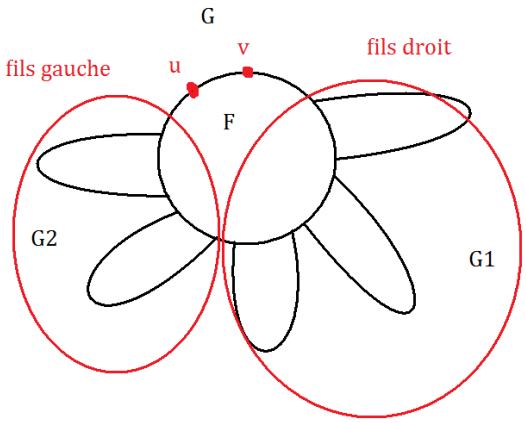
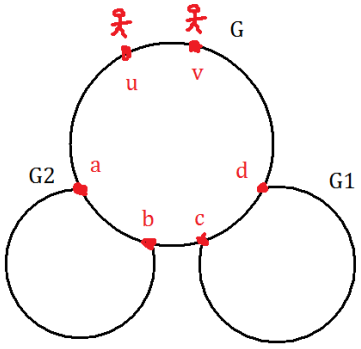


Figure: la partition des branches d'une face en deux structures, le fils gauche et le fils droit.

La division des branches peut être effectué en n'importe quel point, mais il doit y avoir au moins une branche dans chaque fils. On va appeler le fils droit de G_1et le fils gauche de G_2.



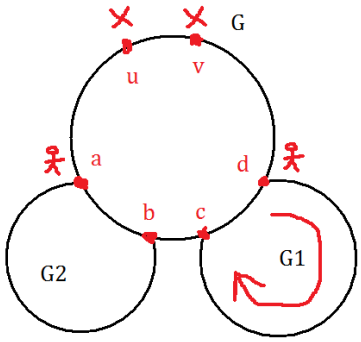
Le principe du théorème est le calcul de X(G, {u, v}, ϕ), le nombre d'agents pour nettoyer G en partant de u et v. Ce valeur sera calculé en fonction de X(G_1, {c, d}, ϕ), X(G_1, c, d), X(G_2, {a, b}, ϕ), X(G_2, a, b) , les paramètres de chaque fils.

Nous allons montrer qu'il est optimal de nettoyer complètement un des fils avant de commencer à nettoyer l'autre.

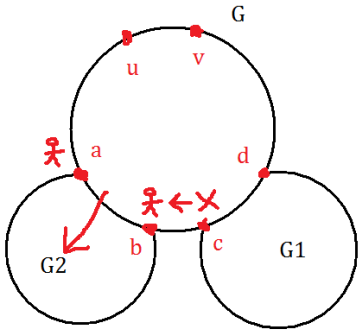
Donc, nous irons considérer deux stratégies possibles pour nettoyer G, en partant de u et v.

La première:

- Nettoyer complètement le fils droit, en utilisant X(G_1, d, c) agents. Comme il y a un agent de plus dans le fils gauche, le total d'agents dans cette étape est X(G_1, d, c) + 1.



- Après que le premier fils soit nettoyé, on aura besoin de X(G_2, {a, b}, ϕ) agents pour nettoyer le deuxième fils.



Nombre total d'agents = max{ X(G_1, d, c) + 1, X(G_2, {a, b}, ϕ) }

La deuxième stratégie est la symétrique de la première, et le nombre d'agents utilisés est max { X(G_2,a, b) +1, X(G_1, {c, d}, ϕ}.

On va montrer que pour toute stratégie optimale, une de ces deux stratégies est au moins aussi bien que la stratégie optimale.

Donc, on pourra affirmer le théorème 1:

Théorème 1: $X(G, \{u, v\}, \phi) = \min\left\{ \begin{matrix} max\{X(G_1,d,c)+1, X(G_2,\{a,b\},\phi\} \\ max\{X(G_2,a,b)+1, X(G_1,\{c,d\},\phi\} \end{matrix} \right\}$

PREUVE Prenons une stratégie optimale S. On suppose que le fils droit soit fini avant que le fils gauche (C'est ne pas nécessaire que le fils gauche commence après que le fils droit finisse, il faut juste que le fils gauche finisse après que le fils droit le fasse).

On va comparer S avec la stratégie max{ X(G_1, d, c) + 1, X(G_2, {a, b}, ϕ) }, dans laquelle le fils gauche commence après que le fils droit finisse.

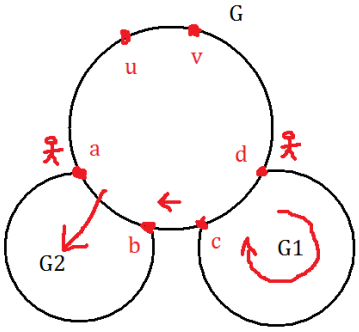


Figure: La stratégie max{ X(G_1, d, c) + 1, X(G_2, {a, b}, ϕ) }

Dans la stratégie S:

Il y aura un moment où un agent sort de la position c. On va appeler cette instant T.

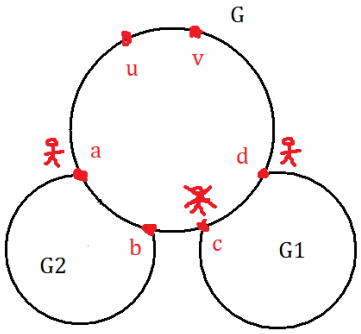
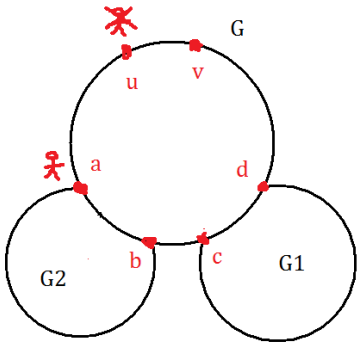


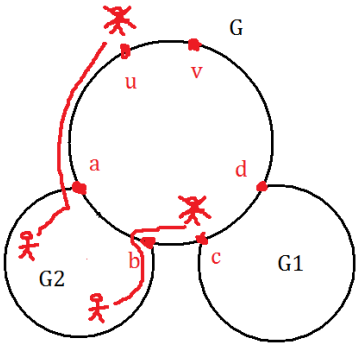
Figure: le moment T où un agent sort du sommet c.

Zomm sur le fils gauche:

Tant que le fils gauche n'est pas complètement nettoyé, il doit exister un agent pour protéger le sommet u déjà nettoyé de la région encore contaminé dans le fils gauche.



Après l'instant T, tant que le fils gauche n'est pas complètement nettoyé, il doit y avoir au moins deux agents dans le fils gauche. La raison est la protection des sommets u et c, déjà nettoyés, de la région contaminé dans le fils gauche.



Zoom dans le fils droit:

Prenons la stratégie S. Si l'on ne regarde que à ce qui arrive dans le fils droit, on a la stratégie S_R pour nettoyer le fils droit. Formellement, nous pouvons définir $S_R(t)$ comme le nombre d'agents dans le fils droit à l'instant t.

On définit aussi A(t) = $\{^{S_R(t), t<T}_{S_R(t)+1, t\geq T}$

A(t) est la quantité d'agents dans le fils droit, si l'on considère que l'agent reste sur le sommet c même après l'instant T.

Par la définition de X(G_1, d, c), on a que X(G_1, d, c)≤$\max_t$ A(t).

Globalement, nous avons au moins K(t) = $\{^{S_R(t)+1, t<T}_{S_R(t)+2, t\geq T}$ agents dans toute la structure.

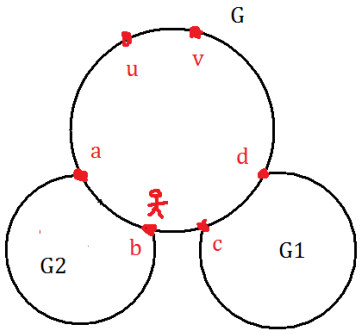
Alors,

X(G_1, d, c) + 1≤$\max_t$ A(t) + 1 = $\max_t$ K(t) ≤le nombre total d'agents

.

Maintenant, regardons un autre point de vue.

Il y aura un moment T' dans lequel un agent arrive sur le sommet b.



Il est facile de voir que le fils droit ne peut pas être complètement nettoyé avant qu'un agent arrive sur le sommet b.

Donc, pour t < T', il y a toujours au moins un agent dans le fils droit.

Maintenant, considérons S_L, la stratégie S limité au fils gauche. Si l'on additionne un agent sur le sommet b au tout début, on peut définir la stratégie B suivant:

B(t) = $\{^{S_L(t)+1, t<T'}_{S_L(t), t\geq T'}$

Par la définition de X(G_2, {a, b}, ϕ), nous avons que X(G_2, {a, b}, ϕ)≤$\max_t$B(t)

Il est facile de voir que le nombre global d'agents est au moins

Q(t) = $\{^{S_L(t)+1, t<T'}_{S_L(t), t\geq T'}$

Donc, nous avons X(G_2, {a, b}, ϕ)≤$\max_t$B(t) = $\max_t$Q(t) ≤le nombre total d'agents

Alors: nombre total d'agents ≥ max (X(G_1, d, c) + 1, X(G_2, {a, b}, ϕ))

.

Par analogie, si la stratégie S fini le fils gauche avant de finir le fils droit, on a: nombre total d'agents ≥ max(X(G_2, a, b) + 1, X(G_1, {c, d}, ϕ)

Par conséquent, on a montré que X(G, {u, v}, ϕ) = min$\{^{max(X(G_1,d,c)+1, X(G_2,\{a,b\},\phi)}_{max(X(G_2,a,b)+1, X(G_1,\{c,d\},\phi)}$.

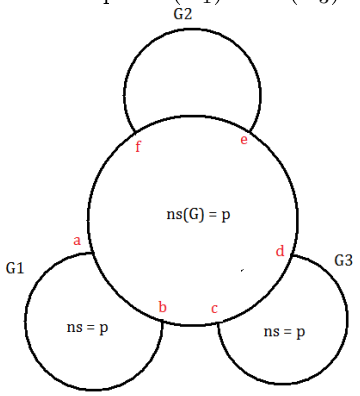
□

Avec le théorème 1, nous avons calculé le paramètre X(G, {u, v}, ϕ) de G à partir des paramètres de ses sous graphes, ce qui indique une programmation dynamique. En fait, si l'on arrive à trouver aussi une formule pour X(G, u, v), à partir des paramètres des sous graphes, alors on aurai tous les ingré-

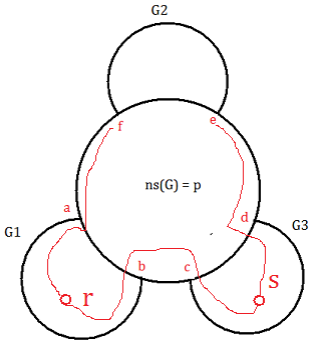
dients nécessaires pour calculer la pathwidth des graphes planaires extérieurs 2-connexes avec la programmation dynamique.

Théorème 2: petite introduction Dans cette théorème, nous allons faire une adaptation du théorème de parsons pour les graphes planaires extérieurs 2-connexes. Nous étudions ici le cas d'une face F avec exactement trois branches.

Théorème 2: Soit F une face avec exactement trois branches dans un graphe planaire extérieur 2-connexe G. Soit G_1, G_2, G_3 les trois branches.
Sachant que $ns(G_1) = ns(G_3) = p$, on a $ns(G) = p \Rightarrow X(G_2, e, f) < p$.



PREUVE
Supposons que G_1 arrive à p agents avant que G_3 le fasse. Soit T_1 le moment où il y a p agents sur G_1. Dans cet instant, il faut qu'il n'y aie aucun agent sur G_2 et G_3, puisque $ns(G) = p$.
Avant que le premier agent arrive sur G_2, il faut qu'un parmi les p agents de T_1 soit enlevé.
On appelle r le point où cet agent était.
Soit T_3 le moment où il y a p agents sur G_3. Dans cet instant, il faut qu'il n'y aie aucun agent sur G_1 et G_2, ça veut dire qu'ils sont complètement nettoyés. Le dernier agent arrive sur G_3 dans un instant $t \geq T_3$. On appelle s le point où cet agent arrive.



Maintenant, on définit les instants T_{2a} et T_{2b} qui sont les moments où le premier agent arrive sur G_2 et le dernier agent est enlevé de G_2, respectivement.

Pour $T_{2a} \leq t \leq T_{2b}$,

Avant qu'un agent arrive sur f, il existe toujours quelqu'un dans le chemin]r, f] (r inclusive, et f exclusive), pour ne pas avoir récontamination.

Après qu'un agent est enlevé de e, il existe toujours quelqu'un sur [e, s[.

Ces deux faits impliquent qu'on peut considérer la stratégie pour nettoyer G_2qui commence en f et qui finit en e.

En plus, on a que: Il existe toujours un agent sur]r, s[

Considérons la stratégie S_0pour nettoyer G_2:

- on place un agent sur f
- on suit $S_{|G_2}$(sauf le moment où on enlève l'agent de e)
- on enlève l'agent de e.

Nous avons, donc, que:

$S_0 \geq X(G_2, e, f)$ et $S \geq S_0 + 1$

Alors, $S \geq X(G_2, e, f) + 1$

□

Le théorème 2 permet une compréension plus profonde de la relation entre le node search number et la structure du graphe. Nous croyons qu'elle est une point clé pour le calcul de X(G, u, v) à partir des paramètres des sous-graphes.

Conclusion

Les théorèmes 1 et 2 sont un pas important vers le développement d'un algorithme rapide et pratique pour le node seach number en graphes planaires extérieurs 2-connexes. Cela semble être une tâche substantiellement plus difficile que l'algorithme pour les arbres. Beaucoup d'efforts ont été dépensés pour la recherche de cet algorithme, mais seulement des approximations ont été trouvés [14][15][16]. Ces théorèmes sont une base solide pour l'algorithme, qui au moins par l'instant, est hypothétique. Nos théorèmes donnent une relation entre les paramètres d'un graphe planaire extérieur 2-connexe et les paramètres de ses

sous-graphes, permenttant la pratique de la programmation dynamique. Une approche possible pour finir l'algorithme est le calcul du deuxième paramètre X(G, u, v) en fonction des paramètres des sous-graphes. Dans la suite du stage, nous nous concentrons sur cette tâche.

Ce travail a été possible en spécial grâce aux commentaire et la collaboration de David Coudert, Dorian Mazauric, Nathann Cohen et Nicolas Nisse.

Références

[1] N. Robertson and P. D. Seymour, *Graph minors. II. Algorithmic aspets of tree-width*, J. Algorithms, 7 (1986), pp. 309-322.

[2] N. Robertson and P. D. Seymour, *Graph minors. I. Excluding a forest,* J. Comb. Theroy Series B, 35 (1983), pp. 39-61.

[3] D. Bienstock, *Graph searching, path-width, tree-width and related problems (a survey)*, DIMACS Ser. in Dicrete Mathematics and Theoretical Computer Science, 5 (1991), pp. 33-49

[4] Hans L. Bodlaender: *A Partial k-Arboretum of Graphs with Bounded Treewidth.* Theor. Comput. Sci. 209(1-2): 1-45 (1998)

[5] Hans L. Bodlaender: *A Tourist Guide through Treewidth.* Acta Cybern. 11(1-2): 1-22 (1993)

[6] H. L. Bodlaender, T. Kloks, D. Kratsch and H. Muller, 1993. Unpublished results.

[7] Burkhard Monien, I.H. Sudborough: *Min Cut is NP-complete for Edge Weighted Trees.* Theoretical Computer Science 58, 209-229

[8] P Scheffler, *A linear algorithm for the pathwidth of trees (1990).* Topics in Combinatorics and Graph Theory

[9] Hans L. Bodlaender, Ton Kloks: *Efficient and Constructive Algorithms for the Pathwidth and Treewidth of Graphs.* J. Algorithms 21(2): 358-402 (1996)

[10] Hans L. Bodlaender, Ton Kloks, Dieter Kratsch: *Treewidth and Pathwidth of Permutation Graphs.* SIAM J. Discrete Math. 8(4): 606-616 (1995)

[11] Konstantin Skodinis: *Computing Optimal Linear Layouts of Trees in Linear Time.* ESA 2000: 403-414

[12] Lefteris M. Kirousis, Christos H. Papadimitriou: *Searching and Pebbling.* Theor. Comput. Sci. 47(3): 205-218 (1986)

[13] Nancy G. Kinnersley: *The Vertex Separation Number of a Graph equals its Path-Width.* Inf. Process. Lett. 42(6): 345-350 (1992)

[14] Rajeev Govindan, Michael A. Langston, Xudong Yan: *Approximation the Pathwidth of Outerplanar Graphs.* Inf. Process. Lett. 68(1): 17-23 (1998)

[15] Hans L. Bodlaender, Fedor V. Fomin: Approximation of pathwidth of outerplanar graphs. J. Algorithms 43(2): 190-200 (2002)

[16] D. Coudert, F. Huc, and J.-S. Sereni. Pathwidth of outerplanar graphs. Journal of Graph Theory, 55(1):27-41, May 2007.

[17] Parsons, T. D. (1976) *Pursuit-evasion in a graph.* In Theory and Applications of Graphs (Y. Alavi and D. Lick, eds), Lecture Notes in Mathematics,

Springer, pp. 426-441.

[18] Nimrod Megiddo, S. Louis Hakimi, M. R. Garey, David S. Johnson, Christos H. Papadimitriou: *The complexity of searching a graph.* J. ACM 35(1): 18-44 (1988)

[19] J. A. Ellis , I. H. Sudborough , J. S. Turner, *The vertex separation and search number of a graph*, Information and Computation, v.113 n.1, p.50-79, Aug. 15, 1994

[20] Andrea S. LaPaugh: *Recontamination Does Not Help to Search a Graph.* J. ACM 40(2): 224-245 (1993)

[21] Bruno Courcelle, Jens Lagergren: *Recognizable Sets of Graphs of Bounded Tree-Width.* Dagstuhl Seminar on Graph Transformations in Computer Science 1993: 138-152

[22] Bruno Courcelle, Jens Lagergren: *Equivalent Definitions of Recognizability for Sets of Graphs of Bounded Tree-Width.* Mathematical Structures in Computer Science 6(2): 141-165 (1996)