Recherche de cycles dans les graphes

par Adrien Panhaleux

20 mai 2007

Introduction

La détection de cycles dans les graphes peut être intéressante pour plusieurs problèmes, que ce soit pour détecter des graphes caractérisés par sous-structure interdite, pour des études de complexité (certains problèmes étant dans P tandis que d'autre sont NP-complet), pour améliorer des algorithmes dans les réseaux,...

"Chercher un cycle dans un graphe" est en fait un problème vague. Le graphe est-il orienté ou non ? le cycle est-il élémentaire ? Cherche-t-on un cycle à taille k fixé ?

En précisant différents paramètres, on arrive à une multitude de problèmes aux complexités connues bien différentes. Un tableau récapitulatif se trouve en annexe à la page 10, construit à partir des différents articles présents dans la bibliographie. Ainsi, selon que le graphe soit orienté ou non, les problèmes suivants ont été relativement souvent étudiés :

- Existence d'un cycle
- Taille maximum
- Taille minimum
- Cycles de taille k (C_k), k étant fixé
- Cycles de taille impaire
- Cycles de taille paire

La suite est constituée d'abord de la présentation de résultats bien connus en partie 1, suivi de la présentation relativement détaillée de la recherche du cycle minimum (partie 2) et des cycles sans cordes (partie 3), suivi d'une présentation succinte de la recherche de cycles pairs ou impairs (partie 4).

1 Trivialités

Dans la recherche de cycles, certains résultats sont connus, avec une complexité certaine. Dans le tableau situé en page 10, ce sont les résultats ne faisant pas appel à des références d'articles cités en bibliographie.

Ces problèmes sont la recherche de l'existence d'un cycle quelconque et la recherche du cycle de taille maximale. Par contre, l'existence d'un cycle ne ga-

ranti pas l'existence d'un cycle sans corde, contrairement à ce que l'on pourrait croire, à cause de la définition de cycle sans corde (voir partie 3).

1.1 Existence d'un cycle

Que ce soit dans le cas orienté ou non-orienté, il est relativement aisé de chercher un cycle simple ou élémentaire. Il est en effet bien connu qu'un parcours du graphe, que ce soit en largeur ou en profondeur, permet si l'on a colorié le graphe au passage de détecter la présence d'un cycle.

Cet algorithme est optimal et à la base de plusieurs autres algorithmes : Recherche d'un arbre couvrant, garbage collector dans les langages de programmation tels Java ou Ocaml,...

Le parcours en profondeur comme en largeur explore une fois et une seule chaque sommet et chaque arête. Ainsi, sa complexité est en $O(n+m)$.

Dans le cas d'un graphe orienté, le parcours du graphe peut ne pas détecter les cycles. Il existe alors un autre algorithme en $O(n+m)$ rappelé ici (algorithme présenté en [2]) :

Algorithme 1. **Reconnaissance de graphes orientés sans circuits**

Pour tout $x \in X$, $d^-(x) = 0$;
Pour tout $x \in X$, faire
    Pour tout y successeur de x, $d^-(y)++$;
Fin pour tout
$\grave{a}\_traiter = \emptyset$;
$nb\_sommets = 0$;
Pour tout $x \in X$, faire
    Si $d^-(x) = 0$ alors
        $\grave{a}\_traiter = \{x\} \bigcup \grave{a}\_traiter$;
        $nb\_sommets++$;
Fin pour tout
Tant que $\grave{a}\_traiter \neq \emptyset$, faire
    $x = premier(\grave{a}\_traiter)$;
    $\grave{a}\_traiter = \grave{a}\_traiter - \{x\}$;
    Pour tout y successur de x, faire
        $d^-(y)--$;
        Si $d^-(y) = 0$ alors
            $\grave{a}\_traiter = \grave{a}\_traiter \bigcup \{y\}$;
            $nb\_sommets++$;
    Fin pour tout
Fin tant que
Si $nb\_sommets = n$ alors G n'a pas de circuit, sinon G a un circuit.

1.2 Longueur maximum, NP-complétude

Un autre problème grandement connu est la recherche d'un cycle de longueur maximum. Ce problème est en fait NP-complet, sauf peut être dans le cas de recherche de cycles sans cordes.

En effet, la recherche d'un cycle élémentaire de taille maximum se réduit au problème du cycle hamiltonien, que nous rappelons ici :

Cycle Hamiltonien : Étant donné un graphe $G = (V, E)$, existe-t-il un cycle passant une fois et une seule par chaque sommet de V ?

Ainsi, si l'on a un algorithme permettant de trouve un cycle élémentaire de taille maximum, on peut décider selon que la taille soit $|V|$ ou non si le graphe a un cycle hamiltonien. D'où la recherche de cycle élémentaire maximum est NP-complet, dans le cas orienté ou non orienté.

2 Cycles de longueur minimum

Un article de 1977 écrit par Alon Itai et Michael Rodeh [1] présente différents algorithmes pour chercher un cycle de longueur minimum, dans les cas orientés et non orientés.

Il est à remarquer que les cycles de taille minimum sont tous élémentaires, car si l'on a un cycle simple non-élémentaire, présentant donc au moins deux boucles, on peut obtenir un cycle plus court en enlevant une boucle surnuméraire. Encore une fois, la complexité n'est a priori pas la même pour un cycle sans corde à cause de la définition de celui-ci (voir partie 3).

2.1 Idée générale

L'idée de l'algorithme est que si l'on fait un parcours en largeur sur un des sommets du cycle de taille minimum, on obtient soit un cycle de taille minimum, soit un cycle plus grand d'un seul arc. De tels cycles sont appelés cycles presque minimaux ("almost minimum circuit" en anglais).

En particulier, si la taille du cycle minimum est impaire, il est possible que l'on trouve un circuit plus grand d'une arête, ce qui n'arrivera pas si le circuit minimum est de taille paire, comme le montre la figure 1 ci-dessous extrait de l'article. Ceci permet de comprendre vaguement pour les complexités sont différentes dans les cas pairs et impairs, ce qui est la base de la partie 4.

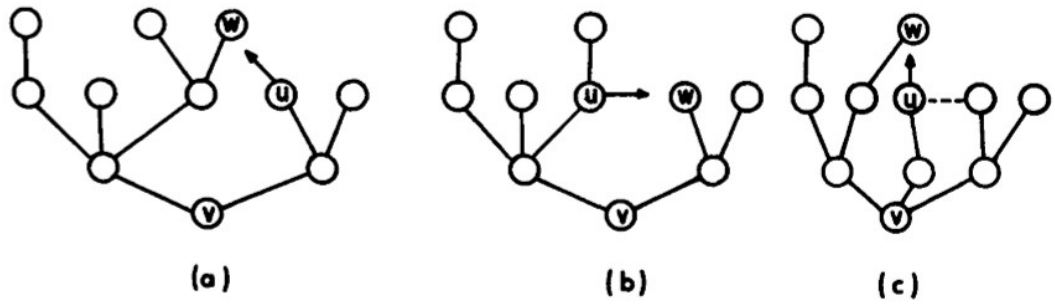

FIG. 1 – Différents cas lors de l'exploration en largeur

2.2 Algorithme - cas non-orienté

L'algorithme présenté se décompose en deux étapes :

1. Recherche d'un cycle presque minimal en $O(n^2)$
2. Si le cycle trouvé est de taille paire, chercher une arête dont les deux sommets sont du même niveau qu'un des sommets qui a arrêté l'algorithme précédent. (cf figure 1, dessin (c))

La première étape se fait par une exploration en largeur, v étant le sommet de départ.

Algorithme 2. **FRONT**(v)

Pour tout $u \in V$, $level(u) =$nil ;
$level(v) = 0$; $enqueue(v)$;
Tant que la queue n'est pas vide, faire
  $u = dequeue$;
  Pour tout $w \in A(u)$, faire
    Si $level(w) ==$nil,
      $level(w) = level(u) + 1$;
      $enqueue(w)$;
    Sinon, si $level(u) \leq level(w)$,
      //On a trouvé un cycle de longueur $\leq 2level(u) + 1$
      $return(level(u))$;
  Fin pour tout
Fin tant que
//Il n'y a pas de circuits dans cette composante connexe
$return(\infty)$;

Cet algorithme étant une exploration en largeur, sa complexité est de $O(n+m)$. Appliqué sur tous les sommets, on trouve la longueur d'un cycle presque minimal en temps $O(n(n+m))$. Une deuxième procédure va permettre de pouvoir passer au circuit de taille minimale. Cette procédure prend en entrée l'ensemble $F(v)$ définit comme suit à la fin de **FRONT** sur le sommet v, se terminant à $level(u)$:

$$F(v) = \{u\} \bigcup \{x \in V | x \in queue,\ level(x) = level(u)\}\,.$$

De plus, $F(v)$ est triée pour permettre une recherche rapide et pour permettre de dans les premiers et les derniers sommets. **EDGE** travaille sur une copie du graphe pour faire des recherches destructives (ou colorie le graphe, selon l'envie).

Algorithme 3. **EDGE**(S)

Pour i de 1 à $|S| - n^{1/3}$, faire
  $u = S(i)$;
  Tant qu'il y a des voisins w de u, faire
    Si $w \in S$ return (u, w) ;
    Supprimer w du graphe (ou le colorier) ;
  Fin tant que
Fin pour i
Pour i de $\max(1, |S| - n^{1/3} + 1)$ à $|S|$, faire
  $u = S(i)$;
  Pour j de $i + 1$ à $|S|$, faire
    $w = S(j)$;
    Si $(u, w) \in E$ alors $return(u, w)$
  Fin pour j
Fin pour i
$return(nil)$;

Ceci permet alors de faire la procédure permettant de trouver un circuit de taille minimum dans un graphe non-orienté :

Algorithme 4. **MIN_CIRCUIT**

Pour tout $v \in V$, faire $k_{min} = \min(k_{min}$,**FRONT**$(v))$;
Si $k_{min} = \infty$, alors $return(\infty)$;
Pour tout $v \in V$ avec $k(v) = k_{min}$,faire
  $a =$**EDGE**$(F(v))$;
  Si $a \neq nil$, alors $return(2k_{min} + 1)$;
Fin pour tout
$return(2k_{min} + 2)$;

La procédure **FRONT** étant en $O(n + m)$ et la procédure **EDGE** en $O(n)$, la procédure **MIN_CIRCUIT** est en $O(n(n + m))$.

2.3 Algorithme - cas orienté

L'article de Alon Itai et Michael Rodeh [1] présente deux algorithmes pour chercher un circuit dans un graphe orienté. Le premier, bien détaillé, a une complexité en moyenne de $O(n^2 \log(n))$ mais peut avoir en principe une complexité de $O(nm)$. Nous allons ici plutôt présenter l'algorithme en temps $n^\omega \log(n)$ se basant sur la multiplication de matrices[1]. Dans la suite, D_j représente une matrice telle que $(D_j)_{u,v} = 1$ si et seulement si il y a un chemin de taille $\leq j$ de u à v :

[1] n^ω représente la complexité de la multiplication de matrices de taille n.

Algorithme 5. **MIN_DICIRCUIT**

$D_1 = M$; //M est la matrice d'adjacence du graphe G
Pour i de 1 à $\lceil \log(n) \rceil + 1$, faire
    $D_{2^i} = D^2_{2^{i-1}} \vee M$ // $\vee$: ou appliqué élément par élément.
    Si $(diag(D_{2^i}) \neq 0)$, alors //Il y a un circuit de taille $2^{i-1} < j \leq 2^i$
        $return($**DICHOTOMIE**$(2^{i-1}, 2^i))$;
Fin pour i
$return(\infty)$; //Il n'y a pas de circuit dans G

Algorithme 6. **DICHOTOMIE**(min, max)

Si $min = max - 1$, $return(max)$;
Si $diag(D_{(min+max)/2} = 0)$,
    $return($**DICHOTOMIE**$(\frac{min+max}{2}, max))$;
Sinon, $return($**DICHOTOMIE**$(min, \frac{min+max}{2}))$;

3 Cycles sans cordes

Un cycle sans corde est défini comme un cycle C de taille ≥ 4 tel que :

$$\forall (u, v) \in C, (u, v) \in E \Rightarrow (u, v) \text{ adjacents dans } C$$

C'est parce que le cycle doit être de taille ≥ 4 que la complexité pour l'existence d'un cycle sans corde n'est pas la même que celle d'un cycle simple. En effet, si à partir d'un cycle simple on prend une corde pour faire un cycle plus petit ayant moins de cordes, on risque de finir par tomber sur un cycle de taille 3.

En anglais, les cycles sans cordes sont appelés *holes*, et *antiholes* définissent le complémentaire d'un *hole*.

3.1 Utilité

Chercher les cycles sans cordes peut avoir une certaine utilité. En effet, certaines classes de graphe connues peuvent être caractérisées par absence de structure interdite. Ainsi :

- Les graphes faiblement triangulés ne contiennent ni antiholes ni holes,
- Les graphes parfaits sont exactement les graphes ne contenant pas de holes et antiholes ayant un nombre impair de sommets.

3.2 Idée générale

L'article de Stavros D. Nikolopoulos et Leonidas Palios [3] présente un algorithme pour trouver des cycles sans corde en temps $O(n+m^2)$ et en espace $O(nm)$ dans les graphe snon orientés. Cependant, cet algorithme ne permet pas de décider les graphes parfaits. La meilleure borne pour décider un graphe parfait par la caractérisation précédemment citée est en $O(n^9)$.

L'algorithme de l'article utilise une caractérisation des cycles sans corde :

Lemme 1. *Un graphe G contient un cycle sans corde si et seulement si G contient un cycle $u_0u_1 \dots u_k$ avec $k \geq 4$ tel que pour tout $i = 0, 1, \dots, k-3$, $u_iu_{i+1}u_{i+2}u_{i+3}$ ainsi que $u_{k-2}u_{k-1}u_ku_0$ sont des P_4 dans G (chemins de taille 4).*

À partir de là, l'article présente un algorithme à base de recherche en profondeur appelée $P_4 - DFS$ (Deep First Search). À chaque étape, l'algorithme essaye à partir d'un P_3 abc de l'étendre à des P_4 $abcd$, pour ensuite repartir sur les P_3 bcd. La séquence des P_3 choisis forme une suite $abc, bcd, \dots wxy, xyz$, ce qui peut se résumer à la séquence $a, b, c, \dots, w, x, y, z$ appellée chemin courant de $P_4 - DFS$.

L'idée est qu'à partir du lemme précédent, on peut déduire trivialement le lemme suivant :

Lemme 2. *Si le chemin courant de $P_4 - DFS$, ρ, s'étend à un sommet de ρ, alors G contient un cycle obéissant aux conditions du lemme 1.*

Ce qui justifie la justesse d'une exploration en profondeur selon des P_4.

4 Cycles pairs et impairs

La partie 2 a déjà donné une idée de pourquoi chercher des cycles de taille paire ou impaire était différent. Dans l'article de Raphael Yuster et Uri Zwick [5], on voir à nouveau que selon la parité de k, chercher si G contient un C_k ne donne pas forcément la même complexité :

- Dans le cas général, chercher un cycle C_k donne une complexité en $O((k-1)!nm)$ selon cet article.
- Dans le cas où k est pair, chercher des C_{2k} peut être fait en $O((2k)!n^2)$.

L'idée principale de l'algorithme dans le cas C_{2k} est de faire une recherche en largeur à partir de chaque sommet s, comme lors de la recherche de cycle de taille minimum (partie 2). La recherche permet en fait de dire que soit s ne participe pas à un C_{2k}, soit trouver un C_{2k}, ne passant pas obligatoirement par s.

Pour une exploration à partir du sommet s, on note L_i l'ensemble des sommets au niveau i lors de l'exploration en largeur. Lors de l'exploration à partir du niveau i, on note L'_{i+1} l'ensemble déjà exploré du niveau $i+1$.

On arrête alors la recherche dans les trois cas suivants :

1. On a exploré $k-1$ niveau, ou l'exploration s'est arrêtée,
2. Il y a au moins $4k|L_i|$ arêtes reliant des sommets de L_i entre eux,
3. Il y a au moins $4k(|L_i| + |L'_{i+1}|)$ arêtes reliant des sommets de L_i à ceux de L'_{i+1}.

Dans le cas 1, on a pour le sous graphe G' définit par les $L_0, L_1, \dots, L_k$ (sans les arêtes entre sommets de L_k) un nombre d'arêtes proportionnel à n. Il suffit donc d'appliquer l'algorithme de recherche général de C_k sur G'.

Dans le cas 2, on prend un sous-graphe H de L_i tel que $|E_H| \leq 4k|V_H|$, et c le sommet le plus haut dans l'arbre de recherche tel que tout H soit descendant de c. c se trouve à une hauteur h dans l'arbre de recherche.
Selon si le sous-graphe H est biparti ou non, on fait une 3-coloration ou une

2-coloration des sommets, ces colorations n'obéissant à aucune règle précise, mais étant choisies judicieusement pour trouver des chemins de taille $2(k - (i - h))$, ce qui permet ensuite en passant par c, d'avoir des cycles C_{2k}.

Dans le cas 3, on prend un sous-graphe H comme dans le cas 2, sauf que l'on sait que H est biparti ($H = (U \bigcap L_i) \bigcup (U \bigcap L'_{i+1})$).

L'article introduit aussi des complexités pour chercher le plus court cycle de taille paire, en temps $O(n^2)$, et le plus court cycle de taille impaire, en temps $O(n^\omega \log(n))$ dans le cas non-orienté, et en temps $O(nm)$ dans le cas orienté.

Conclusion

On a pu voir le long des différents articles que "chercher un cycle dans un graphe", qui est un problème vague, peut avoir de nombreuses complexités différentes, selon les différents paramètres mis en place.

Ainsi, on peut retenir de manière relativement générale que l'existence est de complexité basse, ainsi que chercher le cycle de longueur minimum.
Par contre, chercher un cycle de longueur maximum, ou de longueur k, si k est donné en entrée, est NP-complet.

À k fixé, par contre, différents algorithmes ont pu être trouvés, de complexité plus ou moins grande (notons par exemple la recherche de cycles sans corde de taille paire k fixée qui a une complexité en $O(n^{15})$ selon [6], améliorant la borne précédemment connue de $O(n^{40})$).

De plus, de manière générale, le cas orienté rajoute souvent un élément de complexité, et la recherche de cycles sans corde est toujours bien plus difficile que la recherche de cycles élémentaires ou standards.

C'est pourquoi l'on peut dire que la recherche en graphes est loin d'être finie, car pour chaque problème possible, il y a de nombreux paramètres que l'on peut modifier ou fixer, changeant totalement les complexités envisageables.

Références

[1] Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. In *STOC '77 : Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 1–10, New York, NY, USA, 1977. ACM Press.

[2] Cours de Vincent Bouchitté rédigé par Brice Goglin. *Graphes et algorithmique des graphes*. École normale supérieure d'informatique, 1998.

[3] Stavros D. Nikolopoulos and Leonidas Palios. Hole and antihole detection in graphs. In *SODA '04 : Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 850–859, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics.

[4] Noga Alon, R. Yuster, and U. Zwick. Finding and Counting Given Length Cycles. In *Proc. 2nd Eur. Symp. Algorithms*, number 855 in Lecture Notes in Computer Science. Springer-Verlag, 1994.

[5] Raphael Yuster and Uri Zwick. Finding even cycles even faster. *SIAM J. Discret. Math.*, 10(2) :209–222, 1997.

[6] K. Kawarabayashi et P. Seymour M. Chudnovski. Detecting even holes. *Journal Graph Theory*, 48 :85–111, 2005.

Tableau de complexités[2]

	Graphes non orientés			Graphes orientés		
	cycle standard	cycle élémentaire	cycle sans corde	circuit standard	circuit élémentaire	circuit sans corde
Existence	$n+m$ [2]	$n+m$ [2]	$n+m$ [3], trouver le cycle : $n+m^2$ [3]	$n+m$ [2]	$n+m$ [2]	
Longueur maximum	NP-complet	NP-complet		NP-complet	NP-complet	
Longueur minimum	$n(n+m)$ [1]	$n(n+m)$ [1]		$n^\omega \log(n)$ ou nm [1]	$n^\omega \log(n)$ ou nm [1]	
Longueur pour k fixé	$n^\omega \log(n)$ [4] ou nm [5]		$n^{(k-3)}n^\omega$ [3]			
Longueur impaire (k fixé)	$m^{2-\frac{2}{k+1}}$ [4]	$n^\omega \log(n)$ [5]		$m^{2-\frac{2}{k+1}}$ [4]	nm [5]	
Longueur paire (k fixé)	$m^{2-\frac{2}{k}}$ [4] ou $2k!n^2$ [5]	n^2 [5]	n^{15} [6]	$m^{2-\frac{2}{k}}$ [4]		

[2]Les complexités présentées sont toujours asymptotiques.