Quelques algorithmes linéaires de reconnaissance autour de Lex-BFS *

Christophe PAUL† Laurent VIENNOT‡

Résumé

Nous présentons dans cet article des algorithmes de reconnaissances pour différentes classes de graphes (co-triangulé, intervalles, convexes ...) tous basés sur Lex-BFS. Ces algorithmes sont optimaux, linéaires en la taille du graphe, et simples : ils évitent l'utilisation des PQ-arbres et de la décomposition modulaire.

1 Introduction

L'algorithme Lex-BFS a été le premier algorithme linéaire de reconnaissance des graphes triangulés [RTL76]. Ces dernières années, il a été utilisé dans plusieurs applications. Il permet par exemple de calculer une paire dominante d'un graphe sans triplet astéroïde [COS95]. Il est aussi à la base de la simplification [KM89] du premier algorithme linéaire de reconnaissance des graphes d'intervalles [BL76]. Ces deux algorithmes utilisent les PQ-arbres, une structure de données assez compliquée. Dans [HM91], un troisième algorithme algorithme pour ce problème est présenté. Il évite l'utilisation des PQ-arbres grâce à la décomposition modulaire. Nous montrerons comment Lex-BFS permet aussi d'éviter la décomposition modulaire.

Un graphe est communément décrit par des listes d'adjacence. On peut aussi le définir par sa matrice sommet-clique maximale, où une entrée prend la valeur 1 si le sommet correspondant appartient à la clique maximale correspondante. On explique comment Lex-BFS peut être modifié pour s'exécuter à partir de la matrice sommet-clique maximale d'un graphe. En utilisant, l'algorithme de reconnaissance des graphes des graphes d'intervalle, on peut avec cette nouvelle version de Lex-BFS reconnaître un graphe convexe et tester si une matrice possède la propriété des 1 consécutifs (sans utiliser les PQ-arbres [BL76, Hil93]).

Nous montrerons auparavant comment adapter Lex-BFS pour reconnaître les graphes co-triangulés. Commençons par quelques rappels.

2 Préliminaires

Un graphe est *triangulé* si tout cycle de longueur strictement supérieure à 3 possède une corde [Gol80]. Les graphes triangulés sont caractérisé par l'existence

*Une partie de ce travail a été soumis à CAAP'97

†LIRMM, Montpellier, France; email : paul@lirmm.fr

‡LITP, Paris, France; email : lavie@litp.ibp.fr

d'un *schéma d'élimination simplicial* [FG65]. On dit qu'un sommet est *simplicial* si son voisinage induit un sous-graphe complet, une *clique*. Les graphes triangulés peuvent donc être construit (ou réduit) en ajoutant (ou supprimant) un à un des sommets simpliciaux. Lex-BFS [RTL76] est le premier algortihme linéaire de reconnaissance des graphes triangulés. Il procède en deux étapes : construction d'un ordre d'élimination simplicial si et seulement si le graphe est triangulé puis vérification. Nous nous intéresserons au premier algorithme cité, en voici un schéma :

Algorithme 1 : Lex-BFS [RTL76]
Données : Un graphe $G = (V, \mathcal{E})$
Résultat : Un ordre d'élimination simplicial λ si et seulement si G est triangulé
début
  pour chaque *sommet* $x \in V$ **faire**
    étiquette$(x) = \emptyset$
  pour $i = n$ *jusqu'à 1* **faire**
    Choisir un sommet non numéroté $x \in V$ d'étiquette maximum
    $\lambda(i) \leftarrow x$
    pour chaque *voisin non numéroté* y *de* x **faire**
      Ajouter i à *étiquette*(y)
fin

En 1974, Gavril [Gav74] a proposé une autre caractérisation très utile du point de vue algorithmique. Un graphe est triangulé si et seulement si c'est le graphe d'intersection des sous-arbres d'un arbre. Cela signifie en fait que les cliques maximales peuvent être arrangées sous forme d'un arbre tel que les cliques contenant un sommet donné induisent un sous-arbre. Un tel arbre est appelé un *arbre de cliques*. Dans [GHP95], un algorithme linéaire, $O(m + n)$, simple basé sur Lex-BFS pour construire un arbre de cliques a été présenté. Le calcul d'un arbre de cliques simplifie grandement la procédure de vérification. Nous présentons ici cet algorithme, mais nous ne donnerons que les grandes lignes de sa preuve.

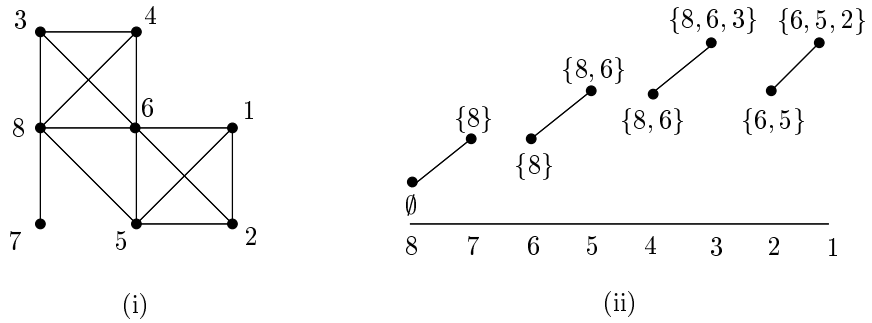

Figure 1: (i) Un graphe triangulé numéroté suivant un ordre Lex-BFS (ii) Le diagramme correspondant des étiquettes des sommets

Pour avoir une idée du fonctionnement de l'algorithme, considérons le diagramme composé des étiquettes des sommets lorsqu'ils sont numérotés. Lorsque le graphe est triangulé chaque séquence strictement croissante pour l'ordre

d'inclusion correspond à une clique maximale (cf. figure 1). On obtient donc facilement l'ensemble des cliques maximales, il reste à les connecter entre elles. Pour cela, on retient pour chaque sommet, la clique qui l'a découvert ainsi que le dernier sommet qui l'a marqué. Lorsqu'une nouvelle clique est visité, on la relie à la clique qui a découvert le dernier sommet ayant marqué l'ensemble de la séquence croissante. On assure ainsi que si une clique C contient un sommet x, alors toutes les cliques sur le chemin de l'arbre entre C et la clique $C(x)$ qui a découvert x, contiennent également x.

Algorithme 2 : Lex-BFS et arbre de cliques [GHP95]

Données : Un graphe $G = (V, \mathcal{E})$.

Résultat : Si G est triangulé : un ordre d'élimination simplicial et un arbre de cliques $T = (\mathcal{I}, \mathcal{F})$.

début
- **pour chaque** *sommet* $x \in V$ **faire**
  - *étiquette*$(x) = \emptyset$
- *étiquette précédente* $= \emptyset$
- $j = 0$
- **pour** $i = n$ *jusqu'à 1* **faire**
  - Choisir un sommet non numéroté $x \in V$ d'étiquette maximum
  - **si** étiquette précédente $\not\subset$ étiquette(x) **alors**
    - $j = j + 1$
    - Créer la clique maximale $C_j =$ *étiquette*$(x) \cup \{x\}$
    - $C($*dernier*$(x))$ est le père de C_j dans T
    - L'arc de l'arbre $C_j C($*dernier*$(x))$ est étiqueté par le séparateur minimal $S_j = C_j \cap C($*dernier*$(x)) =$ *étiquette*(x)
  - **sinon** $C_j = C_j \cup \{x\}$
  - **pour chaque** *voisin non numéroté* y *de* x **faire**
    - Ajouter i à *étiquette*(y)
    - *dernier*$(y) = x$
  - *étiquette précédente* $=$ *étiquette*(x)
  - $\lambda(i) \leftarrow x$
  - $C(x) = j$

fin

La procédure de vérification n'est pas décrite dans l'algorithme 2. Présentons la brièvement. Il y a deux types de tests à faire. Premièrement, il faut vérifier que dans une séquence croissante tous les sommets se sont marqués les uns les autres. Ensuite, il faut tester les liens entre les cliques voisines : on s'assure que tous les sommets qui ont marqués une séquence croissante sont contenu dans la clique "père".

3 Reconnaissance des graphes co-triangulés

Un graphe $G = (V, E)$ est dit *co-triangulé* si son complémentaire $\overline{G}$ est un graphe triangulé. A chaque étape, Lex-BFS choisit un sommet maximum dans l'ordre lexicographique. On définit l'étiquette complémentaire d'un sommet x comme l'ensemble ordonné lexicographiquement des sommets numérotés non voisins de x. Si x est le sommet d'étiquette minimum, alors son étiquette complémentaire

est maximum. Ainsi si à chaque étape on choisit un sommet d'étiquette minimum, cela revient à choisir un sommet d'étiquette maximum dans le graphe complémentaire. On obtient l'algorithme suivant :

Algorithme 3 : co-Lex-BFS

Données : Un graphe $G = (V, \mathcal{E})$

Résultat : Un ordre d'élimination simplicial λ de $\overline{G}$ si et seulement si $\overline{G}$ est triangulé

début

  pour chaque *sommet* $x \in V$ **faire**

    $\textit{étiquette}(x) = \emptyset$

  pour $i = n$ *jusqu'à 1* **faire**

    Choisir un sommet non numéroté $x \in V$ d'étiquette **minimum**

    $\lambda(i) \leftarrow x$

    pour chaque *voisin non numéroté* y *de* x **faire**

      Ajouter i à $\textit{étiquette}(y)$

fin

Nous sommes donc capables grâce à l'algorithme 3 de produire un ordre d'élimination simplicial de $\overline{G}$ en $O(n+m)$ lorsque G est un graphe co-triangulé. Cependant, il faut tester si on a produit un tel ordre. Cela pourrait être accompli avec l'algorithme 2 en construisant un arbre de cliques de $\overline{G}$. Mais la complexité serait alors $O(n + m')$ où m' est le nombre de non-arêtes de G. En fait, nous pouvons nous passer de l'arbre de cliques en calculant une structure arborescente isomorphe dont la taille est $O(n + m)$.

Dans le diagramme des étiquettes d'un ordre Lex-BFS d'un graphe triangulé, chaque séquence croissante correspond à une clique maximale. En effet les sommets d'une clique se marquent successivement. Or une clique maximale d'un graphe triangulé est un stable maximal de son complémentaire. Dans ce cas, les sommets ne se marquent pas puisqu'ils sont indépendants. L'algorithme 3 numérote systématiquement le sommet d'étiquette minimum. Lorsqu'un sommet x a été numéroté, les prochains sommets numérotés seront les non-voisins de x, ceux qui appartiennent à unstable contenant x. Ainsi le diagramme de étiquettes obtenu par l'algorithme 3 sur un graphe co-triangulé, peut être découpé en séquences constantes, chacune correspondant à un stable maximum.

Pour construire la structure arborescente, on procède de mainère complètementement similaire à la construction de l'arbre de clique (cf algorithme 2). La séquence constante courante S doit être connectée à la première séquence constante contenant le dernier non-voisin numéroté commun à aux sommets de S. Cette opération peut être faite en parcourant la liste des étiquettes d'un sommet de S (cf ligne 1). Ainsi l'ensemble des connections coûte $O(m)$. L'arbre construit est en fait un arbre de cliques de $\overline{G}$ dans lequel les sommets n'apparaissent que dans la première clique les contenant.

Il faut ensuite vérifier si l'ordre obtenu est un ordre d'élimination simplicial de $\overline{G}$. Pour cela il suffit de tester si l'étiquette d'une séquence constante (celle de chacun de ses sommets) est contenu dans chacune des étiquettes de ses fils (cf ligne 2). Si ce test est fait simultanément sur l'ensemble des fils d'une séquence constante, alors on ne parcourt qu'une fois l'ensemble des étiquettes. Ce qui donne une complexité en $O(n + m)$.

Algorithme 4 : Reconnaissance des graphes co-triangulé

Données : Un graphe $G = (V, \mathcal{E})$.
Résultat : Retourne oui si et seulement si G est un graphe co-triangulé
début
  co ordre d'élimination simplicial et arbre fco
  pour chaque *sommet* $x \in V$ **faire**
    étiquette$(x) \leftarrow \emptyset$
  étiquette précédente $= \emptyset$
  $j \leftarrow 0$
  pour $i = n$ *jusqu'à 1* **faire**
    Choisir un sommet non-numéroté x d'étiquette minimum
    si étiquette précédente $\neq$ étiquette(x) **alors**
      $j \leftarrow j + 1$
      Créer le noeud $Node_j \leftarrow \{x\}$
1   Connecter $Node_j$ au noeud $Node_k$ contenant le dernier non-voisin numéroté de x
    sinon $Node_j \leftarrow Node_j \cup \{x\}$
    pour chaque *voisin* y *de* x **faire**
      étiquette$(y) \leftarrow$ *étiquette*$(y) \cup \{x\}$
    étiquette précédente $\leftarrow$ *étiquette*(x)
    $\alpha(i) \leftarrow x$
    $Node(x) \leftarrow j$

  co Test si α *est un ordre d'élimination simplicial de* $\overline{G}$ *fco*
  pour chaque $Node$ *de* T **faire**
    pour chaque *fils* $Node_j$ *de* $Node$ **faire**
2   Tester si *étiquette*$(Node) \subset$ *étiquette*$(Node_j)$
  retourner *oui si et seulement si tous les tests sont positifs*
fin

Théorème 1 *On peut tester en* $O(n+min(m+m'))$ *si un graphe* G *est triangulé ou/et co-triangulé où* m' *est le nombre d'arêtes de* $\overline{G}$.

La reconnaissance d'un graphe triangulé en $O(n + m)$ est un résultat bien connu. En utilisant l'algorithme 4, on sait maintenant reconnaître en $O(n + m)$ si un graphe est co-triangulé. Pour se faire, l'algorithme construit un ordre d'élimination du graphe complémentaire et teste s'il est simplicial. Si le graphe est suffisamment dense (ie. il a de nombreuses arêtes), il peut être intéressant de travailler sur son complémentaire pour obtenir une complexité en fonction de m' et non de m.

4 Reconnaissance des graphes d'intervalle

Un *graphe d'intervalle* est le graphe d'intersection d'une famille de segments sur la droite réelle. C'est un graphe triangulé admettant un arbre de cliques qui est une chaîne. Autrement dit, il existe un ordonnancement des cliques maximales tel que les cliques contenant un sommet soient consécutives. On appelle un tel arrangement, *une chaîne de cliques*. Un graphe d'intervalle peut posséder plusieurs chaînes de cliques. Trouver une chaîne de cliques permet de reconnaître un graphe d'intervalle.

Le premier algorithme linéaire de reconnaissance des graphes d'intervalles, dû à Booth et Leuker [BL76] est incrémental mais utilise une structure de données très compliquée, les PQ-arbres. Le rôle des PQ-arbres est de mémoriser à chaque étape toutes les chaînes de cliques possibles pour prendre en compte le prochain sommet. Korte et Möhring [KM89] ont modifié une première fois l'aglorithme en simplifiant la structure de données.

Un graphe d'intervalle est aussi un graphe de co-comparabilité. Et trouver un chaîne de cliques revient à calculer une orientation transitive du graphe complémentaire. Le problème de l'orientation transitive d'un graphe est relié au problème de la décomposition modulaire. Dans un graphe, un *module* est un ensemble de sommets tel que tout sommet extérieur est soit voisin de tous les sommets le module soit d'aucun. Calculer la décomposition modulaire d'un graphe revient à substituer un sommet à chaque module jusqu'à obtenir un graphe *premier* (ie. sans module). Il est connu qu'un graphe de comparabilité premier ne possède qu'un orientation transitive (voir [MS94] par exemple). Ainsi un graphe d'intervalle premier ne possède qu'une chaîne de cliques. Hsu et Ma ont proposé [HM91] un algorithme de reconnaissance qui n'utilise pas les PQ-arbres : il calcule d'abord la décomposition modulaire du graphe. L'algorithme que nous présentons ici se passe de ce pré-traitement. Nous ne décrirons que les idées directrices de la preuve, la figure 2 illustre son exécution.

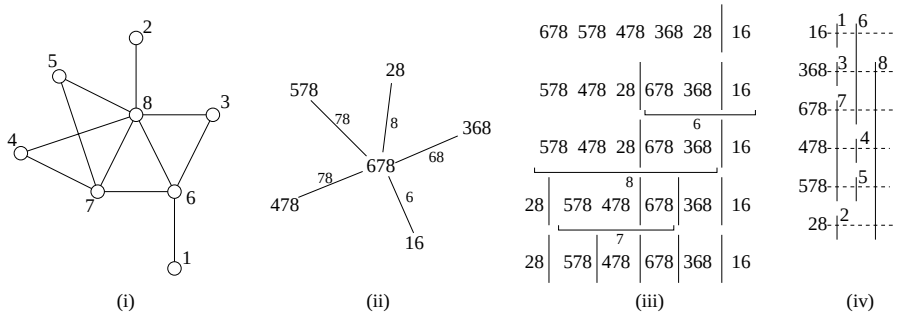

Figure 2: (i) Un graphe d'intervalle dont les sommets sont numérotés suivant un ordre Lex-BFS . (ii) Un arbre de cliques associé à l'ordre lex-BFS. (iii) Un partitionnement de l'ensemble des cliques. Noter que $\{4, 5\}$ est un module. (iv) Une représentation en intervalle associée à la chaîne de cliques calculée.

Tout d'abord, Korte et Möhring ont montré [KM89] qu'il existe toujours une chaîne de cliques ayant comme extrémité la dernière clique visitée par une exécution de Lex-BFS. Lex-BFS a aussi une propriété d'hérédité par rapport aux modules. Le sous-ordre induit par les sommets d'un module est aussi un ordre calculable par Lex-BFS sur le module. Donc la dernière clique visitée du module est aussi une clique extrémale de la chaîne de cliques induite. Cette propriété nous permet d'éviter la décomposition modulaire.

L'algorithme de reconnaissance partionne l'ensemble des cliques en s'appuyant sur un arbre de cliques. La partition de départ est constituée de la dernière clique visitée par Lex-BFS et des autres cliques. Dans une chaîne de cliques, les cliques contenant un sommet sont consécutives : chaque rafinemment du partionnement sépare la clique courante des cliques ne contenant pas un sommet *pivot* par les cliques qui le contiennent. Il ne faut utiliser un pivot que lorsqu'il peut séparer des cliques (ie. il appartient à certaines cliques d'une classe mais

pas à toutes). A partir des arêtes de l'arbre de cliques, on connaît pour une clique donnée, l'ensemble de ses sommets appartenant à d'autres cliques. On peut ainsi en parcourant l'arbre de cliques obtenir efficacement les pivots et surtout de ne pivoter qu'une fois par sommet.

Algorithme 5 : Partionnement des cliques maximales

Données : Un graphe $G = (V, \mathcal{E})$.

Résultat : Si G est un graphe d'intervalle : une chaîne de cliques L.

début

Calculer les cliques maximales et un arbre de cliques $T = (\mathcal{I}, \mathcal{F})$ en utilisant l'algorithme 2

si *G n'est pas triangulé* **alors retourner** *"G n'est pas un graphe d'intervalle"*

Soit$\mathcal{I}$ l'ensemble des cliques maximales $\mathcal{I} = \{C_1, \ldots, C_k\}$

Soit L la liste ordonnée $(\mathcal{I})$

pivots $= \emptyset$ est un pile vide

tant que *il existe une classe $\mathcal{I}_c$ qui n'est pas un singleton dans $L = (\mathcal{I}_1, \ldots, \mathcal{I}_l)$*

faire

  si pivots $= \emptyset$ **alors**

    Soit C_l la dernière clique de $\mathcal{I}_c$ découverte par Lex-BFS

    Remplacer $\mathcal{I}_c$ par $\mathcal{I}_c \setminus \{C_l\}, \{C_l\}$ dans L

    $\mathcal{C} = \{C_l\}$

  sinon Choisir un sommet x non utilisé de *pivots* (supprimer ceux qui ont été utilisé)

  Soit $\mathcal{C}$ l'ensemble de toutes les cliques contenant x

2 **si** *toutes les cliques de $\mathcal{C}$ apparaissent dans des classes consécutives* **alors**

3 Soit $\mathcal{I}_a$ la première classe contenant une telle clique

4 Soit $\mathcal{I}_b$ la dernière classe contenant une telle clique

  sinon retourner *"G n'est pas un graphe d'intervalle"*

5 **si** *Une classe strictement entre $\mathcal{I}_a$ et $\mathcal{I}_b$ contient une clique n'appartenant pas à $\mathcal{C}$* **alors retourner** *"G n'est pas un graphe d'intervalle"*

  Remplacer $\mathcal{I}_a$ par $\mathcal{I}_a \setminus \mathcal{C}, \mathcal{I}_a \cap \mathcal{C}$ et $\mathcal{I}_b$ par $\mathcal{I}_b \cap \mathcal{C}, \mathcal{I}_b \setminus \mathcal{C}$

  pour chaque *arc C_iC_j de l'arbre de cliques connectant une clique $C_i \in \mathcal{C}$ à une clique $C_j \notin \mathcal{C}$* **faire**

    pivots $=$ *pivots*$, C_i \cap C_j$

    Supprimer C_iC_j de l'arbre de cliques

fin

Théorème 2 *L'algorithme 5 calcule une chaîne de cliques si et seulement si le graphe est un graphe d'intervalle. Sa complexité est en $O(n + m)$.*

5 Applications aux matrices et aux graphes bipartis

Un graphe G est entièrement défini par l'ensemble des ses sommets V et l'ensemble de ses cliques maximales $\mathcal{C}$. Cette donnée peut prendre la forme d'une matrice dont les lignes sont les sommets et les colonnes les cliques maximales. L'algorithme 6 calcule un ordre lexicographique des sommets d'un graphe lorsqu'il est donné par sa matrice sommet-clique maximale. Il peut être implémenté en $O(l + c + m)$ où l est le nombre de lignes (de sommets), c le nombre de colonnes

(de cliques maximales) et m le nombre d'entrées positives, en ne fournissant que la liste des entrées positives.

Algorithme 6 : Lex-BFS sur les matrices

Données : M une matrice $0-1$ avec l lignes, c colonnes et m entrées positives
Résultat : Deux ordonnancements lexicographiques : σ de lignes et α des colonnes
début
  pour chaque *colonne* C **faire**
    $label(C) = \emptyset$
  $i \leftarrow n$
  pour $j = 1$ *jusqu'à* c **faire**
    Choisir une colonne C non-numérotée ayant une étiquette maximum
    $\alpha(j) \leftarrow C$
    tant que C *possède des entrées positives non-numérotées* **faire**
      Choisir une entrée (ie. ligne) non-numérotée k de C
      $\sigma(i) \leftarrow k$
      pour chaque *colonne* C' *ayant une entrée positive sur la ligne* k **faire**
        Ajouter i à $label(C')$
      $i \leftarrow i - 1$
fin

Comme on l'a fait pour Lex-BFS, l'algorithme 6 peut être adapté pour calculer aussi un arbre de cliques. Notons qu'une matrice, on peut associer un graphe biparti $B = (X, Y, E)$ où les sommets de X sont les lignes, les sommets de Y les colonnes et il y a une arête entre $x \in X$ et $y \in Y$ si l'entrée correspondante de la matrice est positive. Dès lors, en utilisant les algorithmes présentés dans les sections précédentes, on peut reconnaître si un graphe biparti est le biparti d'incidence sommet-clique maximale d'un graphe triangulé ou d'un graphe d'intervalle. Dans le cas des graphes triangulés, cette classe de graphe est connue sous le nom de *Y-semi-triangulés*.

Théorème 3 *On peut reconnaître en $O(l + c + m)$ si un graphe biparti est Y-semi-triangulé.*

Pour ce qui est des graphes d'intervalles, la classe des bipartis d'incidence sommet-clique maximale est une sous-classe des *graphes convexes*. Un graphe biparti $B = (X, Y, E)$ est convexe (par rapport à Y) si il existe une permutation σ des sommets de Y tel que pour tout sommet $x \in X$, son voisinage est un facteur de σ. La reconnaissance des graphes convexes se faisant déjà linéairement en utilisant les PQ-arbres [Hil93].

Un graphe convexe n'est pas le graphe biparti d'indicence sommet-clique maximale d'un graphe d'intervalle lorsqu'il existe un sommet $y \in Y$ tel que son voisinage n'est pas maximal pour l'inclusion. Cela ne correspondrait pas à une clique maximale du graphe. Mais tout graphe convexe peut être transformé en temps linéaire en un biparti d'incidence sommet-clique maximale d'un graphe d'intervalle. Il suffit de rajouter pour chaque sommet y de Y un voisin propre.

Théorème 4 *On peut reconnaître en $O(l + c + m)$ si un graphe biparti est convexe.*

Une matrice 0-1 possède la *propriété de consécutivité des 1* s'il existe une permutation de ses colonnes telle que les entrées positives de chacune des lignes sont consécutives. Il faut noter que reconnaître un graphe convexe à partir de sa matrice revient à tester si la matrice possède la propriété de consécutivité pour ses entrées positives. Ce problème n'était aussi résolu qu'en utilisant les PQ-arbres.

Théorème 5 *On peut tester en $O(l+c+m)$ si une matrice vérifie la propriété de consécutivité des 1 avec les algorithmes 6 et 5.*

References

[BL76] K.S. Booth and G.S. Leuker. Testing for the consecutive ones property, interval graphs and graph planarity using pq-tree algorithm. *J. Comput. Syst. Sci.*, 13:335–379, 1976.

[COS95] D.G. Corneil, S. Olariu, and L. Stewart. Linear time algorithms for dominating pairs in asteroidal triple-free graphs. Technical Report 294–95, University of Toronto, January 1995.

[FG65] D.R. Fulkerson and O.A. Gross. Incidence matrices and interval graphs. *Pacific Journal Math.*, 15:835–855, 1965.

[Gav74] F. Gavril. The intersection graphs of a path in a tree are exactly the chordal graphs. *Journ. Comb. Theory*, 16:47–56, 1974.

[GHP95] P. Galinier, M. Habib, and C. Paul. Chordal graphs and their clique graph. In *Graph-Theoretic Concepts in Computer Science, WG'95*, volume 1017 of *LNCS*, pages 358–371, 1995.

[Gol80] M.C. Golumbic. *Algorithms Graph Theory and Perfect Graphs.* Academic Press, New York University, 1980.

[Hil93] A. Hilali. *Ordres de contiguité.* PhD thesis, Université Claude Bernard, Lyon, Octobre 1993.

[HM91] Hsu and Ma. Substitution decomposition on chordal graphs and applications. In *Proceedings of the 2nd ACM-SIGSAM Internationnal Symposium on Symbolic and Algebraic Computation*, number 557 in LNCS, 1991.

[KM89] N. Kortee and R. Möhring. An incremental linear-time algorithm for recognizing interval graphs. *SIAM J. of Comput.*, 18:68–81, 1989.

[MS94] R. M. McConnell and J. P. Spinrad. Linear-time modular decomposition and efficient transitive orientation of comparability graphs. In *Proc. of SODA*, pages 536–545, 1994.

[RTL76] Donald J. Rose, R. Endre Tarjan, and George S. Leuker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. of Comput.*, 5(2):266–283, 1976.