INFORMATIQUE THÉORIQUE ET APPLICATIONS

B. COURCELLE
A. PARIÈS

Mineurs d'arbres avec racines

Informatique théorique et applications, tome 29, n° 5 (1995), p. 401-422

<http://www.numdam.org/item?id=ITA_1995__29_5_401_0>

Informatique théorique et Applications/Theoretical Informatics and Applications
(vol. 29, n° 5, 1995, p. 401 à 422)

MINEURS D'ARBRES AVEC RACINES (*)

par B. COURCELLE (1) et A. PARIÈS (1)

Communiqué par A. ARNOLD

Résumé. – *Le problème consistant à tester l'inclusion au sens des mineurs de deux arbres avec racines est NP-complet. Nous proposons un algorithme que nous avons utilisé avec succès pour des arbres de taille 100 ou plus.*

Abstract. – *The problem to decide whether H is a minor of G, where H and G are two rooted trees, is NP-complete. We introduce here an algorithm which has been used successfully on trees with 100 or more nodes.*

1. INTRODUCTION

L'inclusion au sens des mineurs, ou encore **minoration**, est un bel ordre sur l'ensemble des graphes finis considérés à isomorphisme près (Robertson et Seymour [13]). Il en résulte que tout idéal au sens de cet ordre est caractérisé par l'ensemble fini des graphes minimaux de son complémentaire, appelés aussi ses **obstructions**. Ainsi l'idéal des graphes planaires est caractérisé par les deux graphes K_5 et $K_{3,3}$ du théorème de Kuratowski.

On ne connaît les obstructions que de très peu d'idéaux. Ainsi, les obstructions de l'idéal des graphes de largeur arborescente (*treewidth*) au plus k ne sont connues que pour $k \leq 3$ (*voir* [3]). De même, on ne connaît pas toutes les obstructions de l'idéal des graphes *presque planaires* (c'est-à-dire des graphes qui sont planaires si l'on retire au plus un sommet et les arcs incidents).

(*) This work has been supported by the 'Programme de recherches coordonnées « Mathématiques et Informatique » ' and the 'projet ESPRIT Recherche de base COMPUGRAPH 2'.

Reçu en janvier 1994; accepté en novembre 1994.

(1) Laboratoire Bordelais de Recherche en Informatique (CNRS), Université Bordeaux-I, 351, Cours de la Libération, 33405 Talence, France.
courcell@labri.u-bordeaux.fr
paries@labri.u-bordeaux.fr

Informatique théorique et Applications/Theoretical Informatics and Applications
0988-3754/95/05/$ 4.00/

Des méthodes algorithmiques de construction des obstructions ont été proposées (Fellows et Langston [5]; Proskurowski [9]). Elles nécessitent toutes de tester, un certain nombre de fois, si un graphe donné est mineur d'un autre. Le test de minoration est donc un point de passage obligé pour ces méthodes.

Le problème correspondant est NP-complet (car si G est un cycle à n sommets et H un graphe à n sommets, G est mineur de H si et seulement si H est hamiltonien) et « contient » le problème de l'isomorphisme des graphes, car si deux graphes G et H ont même nombre d'arcs et même nombre de sommets, G **minore** H (ce que nous noterons $G \trianglelefteq H$) si et seulement si G et H sont isomorphes. Par ailleurs, un algorithme pour tester si $G \trianglelefteq H$ a été donné par Robertson et Seymour (*voir* [12]), qui est cubique par rapport à la taille de H (pour G fixé) mais cet algorithme est trop complexe pour être implanté.

Notre objectif est de construire un algorithme utilisable pour des graphes de taille raisonnable même s'il n'est pas polynomial par rapport à H.

Dans cet article, nous nous limitons au cas des **arbres avec racine**, c'est-à-dire des arbres orientés de telle sorte que tout sommet est accessible par un chemin à partir d'un sommet particulier appelé racine.

Pour ces arbres, le problème de la minoration est NP-complet (*voir* [7]).

Les idées principales de l'algorithme proposé pour tester si $G \trianglelefteq H$ lorsque G et H sont des arbres sont les suivantes :

- transformer G en un graphe dirigé acyclique (*dag*) en identifiant les sommets d'où sont issus des sous-arbres isomorphes,
- déterminer par un calcul ascendant (des feuilles vers la racine) dans H, pour tout sous-arbre de H, l'ensemble des sous-arbres de G qui le minorent.

2. PRÉLIMINAIRES : MULTI-ENSEMBLES, ARBRES ET GRAPHES

2.1. Ensembles et multi-ensembles

Soient $\mathcal{A}$ un ensemble et a un élément de cet ensemble, on note $\mathcal{A} - \{a\}$ l'ensemble $\mathcal{A}$ privé de l'élément a. On note $\mathcal{P}(\mathcal{A})$ **l'ensemble des parties** de $\mathcal{A}$ et $\mathcal{M}(\mathcal{A})$ **l'ensemble des multi-ensembles finis** d'éléments de $\mathcal{A}$.

On note $e = [\![a_1, a_2, \ldots, a_k]\!]$ un multi-ensemble constitué de k éléments $a_1, a_2, \ldots, a_k$ de $\mathcal{A}$. On peut avoir $a_i = a_j$ pour $i \neq j$. On appelle k la cardinalité de e, notée $\mathrm{Card}(e)$ et on note $\mathrm{Occ}(a, e)$ la cardinalité de

l'ensemble $\{ i/1 \leq i \leq k,\ a_i = a \}$, c'est-à-dire le nombre d'occurrences de a dans e. On note $a \in e$ si $\mathrm{Occ}(a, e) \neq 0$.

L'union de deux multi-ensembles $e = [\![a_1, a_2, \ldots, a_k]\!]$ et $e' = [\![b_1, b_2, \ldots, b_l]\!]$ est le multi-ensemble $e \sqcup e' = [\![a_1, a_2, \ldots, a_k, b_1, b_2, \ldots, b_l]\!]$. Pour tout $a \in \mathcal{A}$, on a $\mathrm{Occ}(a, e \sqcup e') = \mathrm{Occ}(a, e) + \mathrm{Occ}(a, e')$. On définit **l'inclusion** de deux multi-ensembles e et e' de $\mathcal{M}(\mathcal{A})$ par :

$$e \sqsubseteq e' \quad \Leftrightarrow \quad \mathrm{Occ}(a, e) \leq \mathrm{Occ}(a, e') \quad \text{pour tout } a \in e$$

et **l'égalité** par :

$$e = e' \quad \Leftrightarrow \quad e \sqsubseteq e' \qquad \text{et} \qquad e' \sqsubseteq e.$$

Si $\mathcal{A}$ est totalement ordonné par $\leq$, on peut représenter un multi-ensemble e comme la suite (unique) $(a_1, a_2, \ldots, a_k)$ telle que $a_1 \leq a_2 \leq \ldots \leq a_k$ et $e = [\![a_1, a_2, \ldots, a_k]\!]$. On écrit alors $e = (a_1, a_2, \ldots, a_k)$. Il est clair, alors, que deux multi-ensembles e, $e' \in \mathcal{M}(\mathcal{A})$ sont égaux si et seulement si ils sont égaux comme suites. On en déduit un ordre total sur les multi-ensembles, noté $\leq_{\mathrm{lex}}$ qui n'est autre que l'ordre lexicographique associé à $\leq$ sur les suites d'éléments de $\mathcal{A}$.

2.2. Graphes, arbres et dags

Les graphes considérés dans cet article seront, sauf indications contraires, finis, orientés, sans boucles, avec éventuellement des arcs multiples.

Si on a, dans un graphe G, un arc de x vers y, on dit que y est un **successeur** de x. On note par $\mathrm{Succ}_G(x)$ l'ensemble des successeurs dans G d'un sommet x et par $\mathrm{MSucc}_G(x)$ le multi-ensemble des successeurs de x, où le nombre d'occurrences de y dans $\mathrm{MSucc}_G(x)$ est égal au nombre d'arcs de x vers y. Si G ne contient pas d'arcs multiples, les fonctions Succ_G et MSucc_G coïncident. Un graphe G est donc bien défini par l'ensemble de ses sommets V_G et la fonction $\mathrm{MSucc}_G : V_G \to \mathcal{M}(V_G)$. On définira alors un graphe G comme un couple $\langle V, M \rangle$ constitué d'un ensemble fini V (les sommets) et d'une fonction $M : V \to \mathcal{M}(V)$ (la fonction MSucc_G) telle que pour tout $x \in V$, on a $x \notin M(x)$ (on n'autorise pas la présence de boucles). Le **degré sortant** d'un sommet $x \in V$ est défini par $\deg\_\mathrm{s}(x) = \mathrm{Card}(M(x))$. On appelle **feuille** tout sommet de degré sortant égal à 0.

On appelle **arbre**, tout graphe T tel que :

• il existe un unique sommet qui n'est l'extrémité d'aucun arc; on l'appellera la **racine**, que l'on notera racine_T;

- tout sommet est accessible à partir de la racine par un chemin unique;
- il n'y a pas de cycles pour le graphe non orienté sous-jacent.

Soient x, y deux sommets d'un arbre T, on note $x \rightarrow_T y$ si x est **père** de y ($y \in \mathrm{MSucc}_T(x)$) et $x \rightsquigarrow_T y$ si x est **ancêtre** de y (il existe un chemin de x à y dans T avec $x \neq y$). x et y sont **comparables** si $x = y$ ou $x \rightsquigarrow_T y$ ou $y \rightsquigarrow_T x$.

On appelle **préarbre** le graphe non orienté connexe sans cycle obtenu à partir d'un arbre en oubliant la racine et l'orientation des arcs. Si P est un préarbre et r un sommet quelconque de P, il existe une unique orientation des arcs de P qui en font un arbre de racine r (voir *fig.* 1).

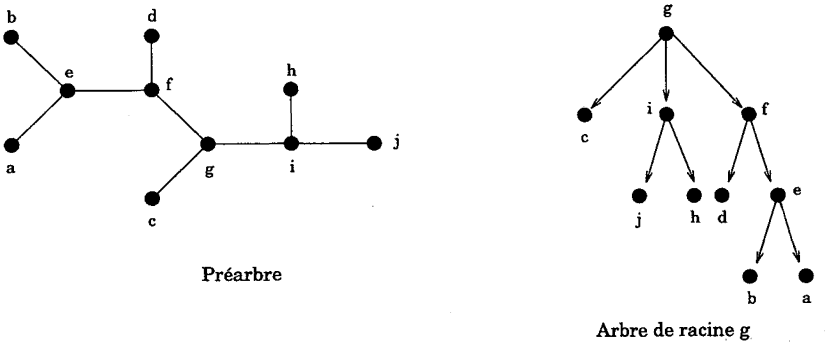


Figure 1. – Préarbre et arbre.

On notera $\mathrm{Succ}^*_G(x)$ l'ensemble des sommets y d'un graphe G tel qu'il existe dans G un chemin éventuellement vide de x à y (donc $x \in \mathrm{Succ}^*_G(x)$). Soit T un arbre, on appelle **sous-arbre issu de** x le sous-graphe induit de T dont l'ensemble des sommets est $\mathrm{Succ}^*_T(x)$. C'est un arbre de racine x noté T/x.

Un **dag** est un graphe orienté sans circuit; en particulier, tout arbre est un dag. Si G est un dag et v un sommet de G, on note G/v le sous-graphe induit dont l'ensemble des sommets est $\mathrm{Succ}^*_G(v)$ (voir *fig.* 2).

Soient G un dag et v un de ses sommets. On note $\mathrm{Arbre}(G, v)$ **l'arbre des chemins finis** issus de v (voir *fig.* 3).

Plus précisément, on définit $\mathrm{Chem}(G, v)$ comme l'ensemble des suites finies d'arcs de G qui forment un chemin d'origine v (c'est-à-dire l'ensemble des suites $(e_1, \ldots, e_k)$ telles que l'extrémité de e_i est l'origine de e_{i+1} et v est l'origine de e_1). On considère la suite vide comme un chemin d'origine et d'extrémité v. On a donc :

$$\mathrm{Arbre}(G, v) = \langle\, \mathrm{Chem}(G, v),\ S \,\rangle$$

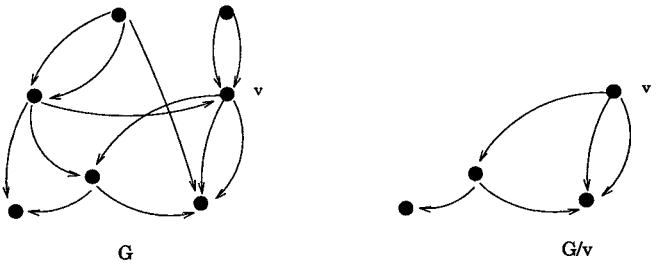


Figure 2. – Dags.

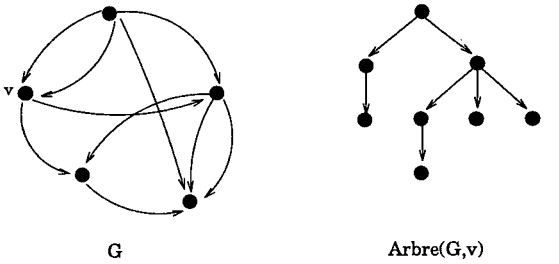


Figure 3. – Dag et arbre des chemins finis.

où $(p, p') \in S$ si et seulement si p et p' sont deux chemins issus de v tels que p' prolonge p au moyen d'un arc et d'un seul.

Remarque 1 : Un arc double de x vers y définit deux chemins issus de x et d'extrémité y.

LEMME 1 : *Si G est un dag et p un chemin dans G de v à x alors les arbres Arbre (G, x) et Arbre $(G, v)/p$ sont isomorphes.*

Preuve : La fonction qui à tout chemin w dans G issu de x associe le chemin $p.w$ dans G (qui est issu de v) définit l'isomorphisme souhaité. □

Remarque 2 : Avec ces notations, on a aussi

$$\text{Arbre}\,(G, x) = \text{Arbre}\,(G/x, x).$$

Noter que l'on a ici égalité et non seulement isomorphisme.

3. FACTORISATION D'ARBRES EN DAGS

3.1. Dag des sous-arbres non isomorphes

Soit $T = \langle V, \text{Succ}_T \rangle$ un arbre (pour les arbres, les fonctions Succ_T et MSucc_T coïncident). Soit $\approx$ la relation d'équivalence sur V telle que $x \approx y$

si et seulement si T/x et T/y sont isomorphes. Soit $G = \langle V', M \rangle$ le graphe tel que :

- $V' = V/\approx$;
- $M([x]) = [\![[y_1], \ldots, [y_k]]\!]$ où $[z]$ désigne la classe d'équivalence d'un sommet z et $\{ y_1, \ldots, y_k \}$ est l'ensemble des successeurs de x dans T.

On notera ce graphe $\mathrm{Dag}(T)$ et on l'appellera le **dag des sous-arbres non-isomorphes de T**. On trouvera sur la figure 4, un exemple d'arbre et de son dag. Noter que la fonction M est bien définie car $M([x])$ ne dépend pas de l'élément x (qui représente la classe $[x]$) et que $M([x])$ est un multi-ensemble.

LEMME 2 : *Le graphe $Dag(T)$ est un dag. Il a un nombre minimal de sommets parmi tous les dags D tels que $Arbre(D, r)$ est isomorphe à T pour quelque sommet r. C'est l'unique dag (unique à isomorphisme près) ayant ces propriétés. Le sommet r est l'unique sommet de D qui n'a pas de prédécesseur.*

Remarque 3 : Le dag des sous-arbres non isomorphes d'un arbre T vérifie les propriétés suivantes :

- $\mathrm{Dag}(T)$ a un unique sommet de degré sortant 0 que l'on appellera **la feuille**.
- $\mathrm{Dag}(T)$ a un unique sommet n'ayant pas de prédécesseur que l'on appellera **la racine**.

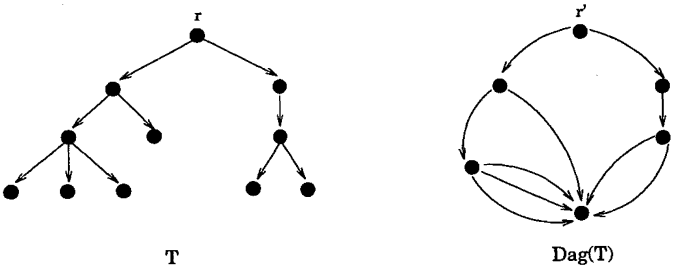


Figure 4. – Arbre et Dag des sous-arbres non isomorphes.

- Pour tout sommet de v de T, les arbres T/v et $\mathrm{Arbre}(\mathrm{Dag}(T), [v])$ sont isomorphes.
- Si v et v' sont deux sommets distincts de $\mathrm{Dag}(T)$, les arbres $\mathrm{Arbre}(\mathrm{Dag}(T), v)$ et $\mathrm{Arbre}(\mathrm{Dag}(T), v')$ ne sont pas isomorphes.

3.2. Un algorithme linéaire de construction de $\mathrm{Dag}(T)$

PROPOSITION 1 : *Soit* $T = \langle V_T, S_T \rangle$ *un arbre, on peut calculer* $Dag(T)$ *en temps* $\mathcal{O}(n)$ *où* $n = Card(V_T)$.

Preuve : L'algorithme qui suit est une simple variante de l'algorithme de Aho *et al.* (*voir* [1]) qui teste en temps $\mathcal{O}(n)$ si deux arbres avec racines à n sommets sont isomorphes.

Soit $T = \langle V_T, S_T \rangle$, on va construire $D = \langle V_D, S_D \rangle$ avec $V_D = \{1, 2, \ldots, n\}$, et une fonction $p : V_T \to V_D$ telle que $p(v)$ est l'élément de V_D qui représente la classe d'équivalence de v pour l'équivalence $\approx$ de la section 3.1. La fonction S_D satisfait alors :

$$S_D(p(v)) = [\![p(v_1), p(v_2), \ldots, p(v_k)]\!]$$

où $S_T(v) = \{v_1, v_2, \ldots, v_k\}$.

Définissons la hauteur $h(v)$ de $v \in V_T$ comme la longueur d'un plus long chemin dans T de v à une feuille. Il est clair que $v \approx v' \Rightarrow h(v) = h(v')$. Posons pour $h \in \mathbb{N}$, $B_h = \{v / v \in V_T, h(v) = h\}$. Remarquons que si $v \in B_h$, ses successeurs appartiennent tous à $\cup\{B_{h'} / h' < h\}$ et que, pour $v, v' \in B_h$: $v \approx v' \Leftrightarrow S_D(p(v)) = S_D(p(v'))$. Donc le calcul de p sur B_h peut se faire au moyen des restrictions des fonctions p sur les ensembles $B_{h'}$ pour $h' < h$. Remarquons que B_0 est l'ensemble des feuilles de T et $B_{\max}$ où $\max = h(\mathrm{racine}_T)$ est le singleton $\{\mathrm{racine}_T\}$.

À tout sommet w de T, on associera la suite $s(w)$ classée par ordre non décroissant, des entiers de $\{1, 2, \ldots, n\}$ représentant $S_D(p(w))$.

Sur chaque ensemble B_h, définissons l'ordre total

$$v \prec v' \quad \Leftrightarrow \quad p(S_T(v)) \leq_{\mathrm{lex}} p(S_T(v'))$$

où $\leq_{\mathrm{lex}}$ est l'ordre lexicographique sur les suites de sommets de T associé à $\leq$, on aura alors $v \approx v' \Leftrightarrow v \prec v'$ et $v' \prec v$.

L'algorithme est le suivant :

- 1) Calculer la suite $B_0, B_1, \ldots, B_{\max}$.
- 2) Initialiser à « vide » toutes les suites $s(w)$ pour $w \in V_T$.
- 3) Pour tout élément v de B_0 (*i.e.* toute feuille) définir $p(v) = 1$ et ajouter 1 à la suite $s(w)$ où w est le père de v. [Ainsi, on définit complètement p sur B_0, s sur $B_0 \cup B_1$ et partiellement s sur $B_2 \cup \ldots \cup B_{\max}$].
- 4) Pour tout $i = 1, \ldots, (\max - 1)$ faire les opérations suivantes :

– Trier B_i par ordre lexicographique pour s (*i.e.* mettre w avant w' si et seulement si $s(w) \leq_{\mathrm{lex}} s(w')$).

– Soit $B'_i = (b_1, b_2, \ldots, b_{m_i})$ le résultat de ce tri; soit r le premier entier non dans $p(B_0 \cup \ldots \cup B_{i-1})$. On définit alors :

* $p(b_1) = r$
* $p(b_j) = p(b_{j-1})$ si $s(b_j) = s(b_{j-1})$
* $p(b_j) = p(b_{j-1}) + 1$ si $s(b_j) >_{\text{lex}} s(b_{j-1})$

ceci jusqu'à $j = m_i$.

– Simultanément, à chaque fois que l'on définit $p(b_j)$, on place cet entier comme élément suivant de la suite $s(w)$ où w est le père de b_j (noter que $w \in B_{i'}$ pour quelque $i' > i$). On définit aussi $S_D(p(b_j)) = s(b_j)$.

• 5) Définir $p(\text{racine}_T) = r = \text{racine}_D$ où r est le premier entier non dans $p(B_0 \cup B_1 \cup \ldots \cup B_{\max -1})$ et $S_D(r) = s(\text{racine}_T)$.

Un exemple est donné sur la figure 5. À gauche de chaque sommet w figure $p(w)$ et à droite $s(w)$.

La figure 6 représente le dag obtenu par notre algorithme appliqué à l'arbre de la figure 4.

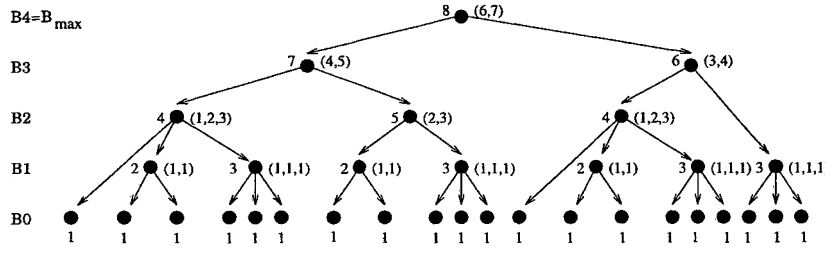


Figure 5. – Calcul de $p(w)$ et de $s(w)$ sur T.

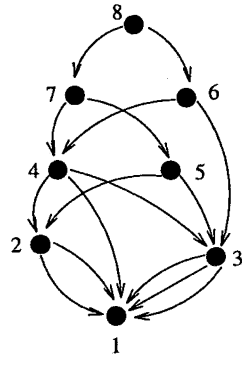


Figure 6. – Dag (T).

3.3. Complexité de l'algorithme

On suppose T donné avec, pour chaque sommet, le père et la liste des successeurs (dans un ordre quelconque). On pose $n = \mathrm{Card}(V_T)$.

- 1) Calcul de la hauteur de chaque sommet et constitution des listes $B_0, \ldots, B_{\max}$: temps $\mathcal{O}(n)$.
- 2) Initialisation des listes $s(w)$: temps $\mathcal{O}(n)$.
- 3) Traitement des feuilles : temps $\mathcal{O}(n)$.
- 4) Pour chaque i, le temps nécessaire pour le tri de B_i est de $\mathcal{O}(\Sigma_{w \in B_i} |s(w)| + \mathrm{Card}(\{ q/q \in s(w),\ w \in B_i \}))$ (d'après [1], théorème 3.2]), les autres opérations prennent un temps $\mathcal{O}(\mathrm{Card}(B_i))$.
- 5) En temps $\mathcal{O}(1)$.

Analysons la complexité de l'étape 4. En sommant sur tous les $i = 1, \ldots, (\max - 1)$, on trouve : $\Sigma_i \Sigma_{w \in B_i} |s(w)| = (n-1)$. Par ailleurs, $\mathrm{Card}(\{ q/q \in s(w),\ w \in B_i \})$ est le nombre de successeurs dans D des sommets de $p(B_i)$. Il est clair que $\Sigma_i \mathrm{Card}(\{ q/q \in s(w),\ w \in B_i \}) \leq n$. Au total, on a donc une complexité $\mathcal{O}(n)$. On omet la preuve de cet algorithme, très semblable à celui de Aho *et al.* (*voir* [1], théorème 3.3 [1]).

4. LE TEST DU MINEUR

On dit qu'un graphe H (orienté ou non) est un **mineur d'un graphe** G (de même nature) si H est isomorphe à un graphe obtenu à partir d'un sous-graphe de G par des contractions d'arcs. On dit aussi que H **minore** G et l'on note cette relation par $H \trianglelefteq G$.

Dans le cas de deux arbres U et T, on obtient la caractérisation équivalente suivante : $U \trianglelefteq T$ si et seulement si il existe une injection $h : V_U \to V_T$ telle que :

- pour tous $x, y \in V_U$ tels que $x \to_U y$ on a $h(x) \rightsquigarrow_T h(y)$, et
- pour tous $x, y, z \in V_U$ tels que $x \to_U y$ et $x \to_U z$ et $y \neq z$, $h(y)$ et $h(z)$ sont incomparables.

LEMME 3 : *Pour tous* $x, y \in V_U$, *on a* $x \rightsquigarrow_U y$ si et seulement si $h(x) \rightsquigarrow_T h(y)$.

Preuve : $\Rightarrow$ Il est clair que $x \rightsquigarrow_U y \Rightarrow h(x) \rightsquigarrow_T h(y)$.

$\Leftarrow$ Soient $x, y \in V_U$ avec $h(x) \rightsquigarrow_T h(y)$. S'il n'est pas vrai que $x \rightsquigarrow_U y$ alors il existe z, x' et y' tels que $z \to_U x' \rightsquigarrow_U x$, $z \to_U y' \rightsquigarrow_U y$ et $x' \neq y'$. Donc $h(z) \rightsquigarrow_T h(x') \rightsquigarrow_T h(x)$ et $h(z) \rightsquigarrow_T h(y') \rightsquigarrow_T h(y)$. De l'hypothèse

que $h(x) \rightsquigarrow_T h(y)$ et puisque T est un arbre, il en résulte que $h(x')$ et $h(y')$ sont comparables dans T. Cela contredit la seconde condition sur h. □

On s'intéresse au problème de tester si $U \trianglelefteq T$

NOTATIONS : On écrit $U = \bullet$ si U est réduit à une racine. On écrit $U = \bullet(U_1, \ldots, U_k)$ si la racine de U a k successeurs $u_1, \ldots u_k$ et $U_i = U/u_i$ pour tout $i = 1, \ldots, k$.

La clef de notre algorithme est le lemme suivant :

LEMME 4 : *Soient U et T deux arbres $U = \bullet(U_1, \ldots, U_k)$, alors $U \trianglelefteq T$ si et seulement si il existe dans T, k sommets $v_1, \ldots, v_k$ différents de la racine et deux à deux incomparables tels que $U_i \trianglelefteq T/v_i$ pour tout $i = 1, \ldots, k$.*

Preuve : Si : Soient $h_i : V_{U_i} \to V_{T/v_i}$ des plongements qui témoignent de ce que $U_i \trianglelefteq T/v_i$. On prend

$$h(r) = \text{racine}_T \qquad \text{si} \quad r = \text{racine}_U$$
$$h(x) = h_i(x) \qquad \text{si} \quad x \in U_i$$

Comme les $v_1, \ldots, v_k$ sont deux à deux incomparables, les images des plongements h_i sont deux à deux disjointes et h est bien injectif, le reste est évident.

Seulement si : Soit $U \trianglelefteq T$ et $h : V_U \to V_T$ un plongement qui en témoigne. On prend $u_1, \ldots, u_k$ les successeurs de la racine r de U et $v_i = h(u_i)$. Il reste à voir que $v_i \neq \text{racine}_T$ et que les v_i sont deux à deux incomparables. L'incomparabilité résulte de la seconde condition de la définition d'un plongement h. Si $v_i = \text{racine}_T$, on aurait $v_i \rightsquigarrow h(r)$ et donc $u_i \rightsquigarrow r$ (lemme 3), ce qui n'est pas non plus possible. □

4.2. Quelques fonctions auxiliaires

Notre objectif est de tester si $U \trianglelefteq T$. On suppose $U = \text{Arbre}(D, r)$ où $D = \langle V_D, \text{Succ}_D \rangle$ est un dag, non nécessairement minimal mais doté d'une unique feuille f_D et $r = \text{racine}_D$. Pour tout sommet v de T, on pose :

$$S(v) = \{ s \in V_D / \text{Arbre}(D, s) \trianglelefteq T/v \}.$$

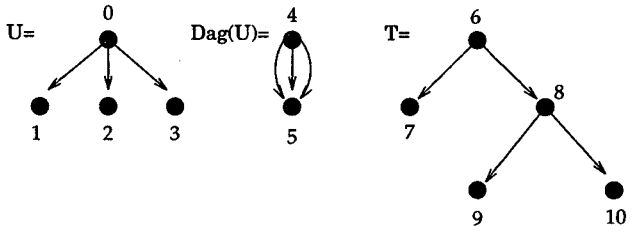
Si l'on décrit succintement la relation $\text{Arbre}(D, s) \trianglelefteq T/v$ en disant que s « se plonge » dans v, alors $S(v)$ n'est autre que l'ensemble des sommets de D qui se plongent dans v. Il en résulte que $U \trianglelefteq T$ si et seulement si $r \in S(v)$ pour quelque sommet v de T. Il suffit donc de calculer la fonction S sur V_T pour décider si $U \trianglelefteq T$. (En fait, il n'est pas nécessaire de calculer

S complètement : on peut arrêter les calculs dès que l'on a trouvé $v \in V_T$ tel que $r \in S(v)$.)

Nous souhaitons faire le calcul de S en montant dans T et donc pouvoir déterminer $S(v)$ en fonction de

$$S(v_1), \ldots, S(v_k) \qquad \text{où} \quad \{v_1, \ldots, v_k\} = \mathrm{Succ}_T(v)$$

Mais ce n'est pas possible directement comme le montre l'exemple suivant :



En effet :

$S(8) = S(7) = S(9) = S(10) = \{5\}$.

$S(6) = \{4, 5\}$.

Donc $S(6) \neq S(8)$ alors que $\mathrm{Succ}_T(6) = \{7, 8\}$, $\mathrm{Succ}_T(8) = \{9, 10\}$, $S(7) = S(9)$ et $S(8) = S(10)$.

Pour pallier cette difficulté, nous intoduisons une fonction auxiliaire R, telle que, si $\{v_1, \ldots, v_k\} = \mathrm{Succ}_T(v)$ alors $S(v)$ et $R(v)$ se calculent en fonction de $S(v_1), \ldots, S(v_k)$ et $R(v_1), \ldots, R(v_k)$. Nous exprimerons cela en disant que R et S se calculent **inductivement** sur T.

Soit d le degré sortant maximal d'un sommet de U (ou de D). Pour tout sommet $v \in V_T$, on définit $R(v)$ comme l'ensemble des couples $(s, [\![w_1, \ldots, w_n]\!])$ tels que

$$s \in V_D, \quad s \notin S(v), \qquad [\![w_1, \ldots, w_n]\!]) \sqsubseteq \mathrm{MSucc}_D(s), \qquad n \geq 1$$

et qu'il existe des sommets deux à deux incomparables $v_1, \ldots, v_n \in V_T$ tels que $v \rightsquigarrow_T v_i$ et $w_i \in S(v_i)$ pour tout $i = 1, \ldots, n$. Il en résulte que $n \leq (d-1)$ et que $s \neq f_D$ si $(s, M) \in R(v)$.

Un couple $(s, [\![w_1, w_2, \ldots, w_n]\!])$ tel que $[\![w_1, w_2, \ldots, w_n]\!]) \sqsubseteq \mathrm{MSucc}_D(s)$ et $v \rightsquigarrow_T v_i$ où $v_1, v_2, \ldots, v_n$ sont incomparables, avec $w_i \in S(v_i)$ peut être vu comme définissant un plongement partiel de $\mathrm{Arbre}(D, s)$ dans T/v. Il en résulte un plongement si $[\![w_1, w_2, \ldots, w_n]\!] = \mathrm{MSucc}_D(s)$ d'après le lemme 4. Autrement dit, $R(v)$ décrit les plongements partiels dans T/v des arbres $\mathrm{Arbre}(D, s)$ qui n'ont pas de plongement dans T/v. Le couple $(S(v), R(v))$ décrit donc en un sens « tout ce que l'on peut plonger » de

D dans T/v, soit totalement, soit partiellement. Nous allons montrer que $(S(v), R(v))$ peut se calculer inductivement (en montrant). Si l'on remarque que $S(v)$ est fermé par la relation de descendance et que $R(v)$ est fermé par la relation d'inclusion sur les secondes composantes de ses éléments, on s'aperçoit que $S(v)$ et $R(v)$ sont totalement déterminés par leurs éléments maximaux. Précisons ces définitions.

On note $m \ll m'$ si et seulement si $m = m'$ ou m est un descendant de m'. Pour tout ensemble, $M \subseteq V_D$, on note

$$\max(M) = \{ m \in M / \text{si } m \ll m' \text{ et } m' \in M \text{ alors } m = m' \}$$

et

$$S_{\max}(v) = \max(S(v))$$

On pose aussi, si $L \subseteq V_D \times \mathcal{M}(V_D)$: $\mathrm{MAX}(L) = \{ (s, e) \in L / \text{il n'existe pas } (s, e') \in L \text{ avec } e \sqsubseteq e' \text{ et } e \neq e' \}$. On définit $R_{\mathrm{MAX}}(v) = \mathrm{MAX}(R(v))$.

THÉORÈME 1 : *Les fonctions* R_{MAX} et $S_{\max}$ *se calculent inductivement sur* T. *Plus précisément,*

• *si* v *est une feuille de* T :

$$S_{\max}(v) = \{ f_D \} \qquad et \qquad R_{\mathrm{MAX}}(v) = \emptyset$$

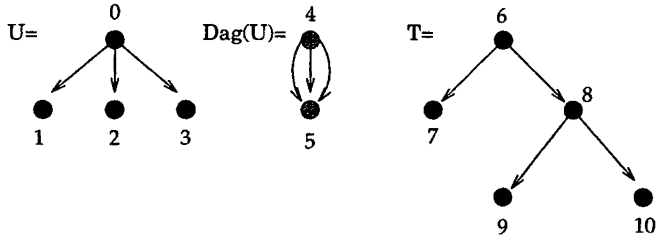
où f_D *est l'unique feuille de* D;

• *si* v *n'est pas une feuille et* $Succ_T(v) = \{ v_1, \ldots, v_k \}$ *on a :*

1) $S_{\max}(v) = \max(S_{\max}(v_1) \cup \ldots \cup S_{\max}(v_k)$
$\cup \{ s \in V_D / \mathrm{MSucc}_D(s) \sqsubseteq M_1 \sqcup M_2 \sqcup \ldots \sqcup M_k,$
où, pour tout $i = 1, \ldots, k$
ou bien $M_i = [\![s']\!]$ *et* $s' \in S_{\max}(v_i)$
ou bien $(s, M_i) \in R_{\mathrm{MAX}}(v_i)$
ou bien $M_i = \emptyset \})$

2) $R_{\mathrm{MAX}}(v) = \mathrm{MAX}(\{ (s, M) \in V_D, M \sqsubseteq \mathrm{MSucc}_D(s),$
$M \sqsubseteq M_1 \sqcup M_2 \sqcup \ldots \sqcup M_k$ *où* $M_1, \ldots M_k$ *ont comme dans* 1),
et il n'existe pas $s'' \in S_{\max}(v)$ *tel que* $s \ll s'' \})$

Exemple 1 : Calcul des ensembles $S_{\max}$ et R_{MAX}.



- $R_{\mathrm{MAX}}(7) = R_{\mathrm{MAX}}(9) = R_{\mathrm{MAX}}(10) = \emptyset$
  $S_{\max}(7) = S_{\max}(9) = S_{\max}(10) = \{5\}$
- $R_{\mathrm{MAX}}(8) = \{(4, [\![5, 5]\!])\}$
  $S_{\max}(8) = \{5\}$
- $R_{\mathrm{MAX}}(6) = \emptyset$
  $S_{\max}(6) = \{4\}$

Preuve du théorème 1 : Le résultat est clair si v est une feuille de T.

Autrement, soit v qui n'est pas une feuille et $\mathrm{Succ}_T(v) = \{v_1, \ldots, v_k\}$. Montrons 1). Tout d'abord, $S_{\max}(v_i) \subseteq S(v)$ pour tout i. Soit $s \in V_D$, $s \neq f_D$ tel que $\mathrm{MSucc}_D(s) \sqsubseteq M_1 \sqcup M_2 \sqcup \ldots \sqcup M_k$ avec pour chaque i, $M_i = [\![s']\!]$ pour quelque $s' \in S_{\max}(v_i)$ ou $(s, M_i) \in R_{\mathrm{MAX}}(v_i)$ ou $M_i = \emptyset$. Il est facile de voir que par le lemme 4, on a $\mathrm{Arbre}(D, s) \trianglelefteq T/v$ et donc $s \in S(v)$.

Réciproquement, soit $s \in S_{\max}(v)$. Ou bien

$$s \in S_{\max}(v_1) \cup \ldots \cup S_{\max}(v_k)$$

(et c'est le cas si $s = f_D$) ou bien

$$s \in \{ s' \in V_D / \mathrm{MSucc}_D(s') \sqsubseteq M_1 \sqcup M_2 \sqcup \ldots \sqcup M_k \text{ avec } M_1, \ldots M_k \text{ comme dans le premier cas} \}.$$

Montrons ce dernier point. Soit

$$s \neq f_D, \qquad s \in S_{\max}(v) - (S(v_1) \cup \ldots \cup S(v_k))$$

et $\mathrm{MSucc}_D(s) = \{s_1, \ldots, s_k\}$.

On a, par le lemme 4, $\mathrm{Arbre}(D, s_i) \trianglelefteq T/w_i$ pour tout $i = 1, \ldots, n$ où $w_1, \ldots, w_n$ sont des descendants deux à deux incomparables de v. Soit $i \in \{1, \ldots, k\}$: si un seul des w_j est dans T/v_i alors on pose, $M_i = [\![s_j]\!]$ et l'on a bien $s_j \in S(v_i)$ de plus $s_j \in S_{\max}(v_i)$ car autrement son père s appartient à $S(v_i)$ contrairement à l'hypothèse. Autrement, si $w_{j_1}, \ldots, w_{j_m}$ sont dans T/v_i, alors on pose $N_i = [\![s_{j_1}, \ldots, s_{j_m}]\!]$ et l'on a bien $(s, N_i) \in R(v_i)$. On prend alors M_i tel que $N_i \sqsubseteq M_i$ et $(s, M_i) \in R_{\mathrm{MAX}}(v_i)$. Il résulte de ces remarques que 1) est valide. L'égalité 2) se prouve facilement, essentiellement comme 1). □

Le calcul des fonctions $S_{\max}$ et R_{MAX} sur T permet de décider si $\mathrm{Arbre}(D) \trianglelefteq T$. En effet, on a déjà observé que $\mathrm{Arbre}(D) \trianglelefteq T$ si et seulement si $\mathrm{racine}_D \in S_{\max}(\mathrm{racine}_T)$. Tout revient donc à évaluer la complexité du calcul de $S_{\max}$ et R_{MAX}. Il faut préciser les structures de données employées et les algorithmes.

Paramètres

Les paramètres de l'évaluation de la complexité sont les suivants :

- n = Card (V_D)
- h = hauteur de la racine dans D
- d = degré sortant maximum des sommets de D
- a = taille maximale d'une antichaîne de D, c'est-à-dire d'un ensemble de sommets de D deux à deux incomparables.
- p = nombre maximum d'éléments différents dans un multi-ensemble $\mathrm{MSucc}_D(s)$ [donc $p \leq a$].
- m = Card (V_T).
- l = degré sortant maximum des sommets de T.

THÉORÈME 2 : *On peut décider si* $Arbre(D) \trianglelefteq T$ *en temps*

$$\mathcal{O}\left(n.m.\left(a + \frac{1.4}{\sqrt{d}}\,2^d\right)^l (al + d)\right).$$

Structure de $S_{\max}(v)$

$S_{\max}(v)$ est une liste sans répétition de sommets de D deux à deux incomparables pour $\ll$ que l'on peut classer en ordre croissant en fonction d'un tri topologique fixé $\leq_D$ sur V_D, par exemple l'ordre dans lequel les sommets sont numérotés dans l'algorithme de la section 3 lorsque D est construit à partir d'un arbre (*cf.* 3.2). (L'ordre $\leq_D$ est tel que $s \leq_D s'$ si s est un successeur de s'.) La taille de $S_{\max}(v)$ est au plus a.

Structure de $R_{\mathrm{MAX}}(v)$

$R_{\mathrm{MAX}}(v)$ est une liste de paires (s, M) où $s \in V_D$ et M est un multi-ensemble contenu strictement dans $\mathrm{MSucc}_D(s)$ donc de taille au plus $(d-1)$. On pourra le représenter comme une liste classée lexicographiquement d'abord sur les premières composantes puis sur les secondes, qui sont elles-mêmes des listes comparables lexicographiquement.

LEMME 5 : *Soit* $v \in V_T$ *avec* $Succ_T(v) = \{v_1, \ldots, v_k\}$, $k \neq 0$, *et donc* $k \leq l$.

- *Si* $s \in S_{\max}(v) - \max(S_{\max}(v_1) \cup \ldots \cup S_{\max}(v_k))$ *alors tous les successeurs de* s *appartiennent à* $\max(S_{\max}(v_1) \cup \ldots \cup S(v_k))$.
- *si* $(s, M) \in R_{\mathrm{MAX}}(v)$ *alors* s *a au moins un successeur dans* $\max(S_{\max}(v_1) \cup \ldots \cup S_{\max}(v_k))$.

Preuve : 1) Soit $s \in S_{\max}(v) - \max(S_{\max}(v_1) \cup \ldots \cup S_{\max}(v_k))$. Il résulte du lemme 4 que ses successeurs s' sont tous dans $S(v_1) \cup \ldots \cup S(v_k)$. Si un

tel successeur $s' \notin \max(S_{\max}(v_1) \cup \ldots \cup S_{\max}(v_k))$ il a un ancêtre strict s'' dans $\max(S(v_1) \cup \ldots \cup S(v_k)) = \max(S_{\max}(v_1) \cup \ldots \cup S_{\max}(v_k))$

Si s'' est égal au père de s' alors $s \in (S_{\max}(v_1) \cup \ldots \cup S_{\max}(v_k))$ contrairement à l'hypothèse. Si s'' est un ancêtre strict de s, alors $s'' \in S(v_i)$ pour quelque $i = 1, \ldots, k$ et donc $s'' \in S(v)$, car $S(v_i) \subseteq S(v)$ et $s \notin S_{\max}(v)$ contrairement à l'hypothèse.

2) Soit $(s, M) \in R_{\mathrm{MAX}}(v)$ et $s' \in M$; donc s' est un successeur de s et l'on a deux cas possibles :

cas 1 : $s' \in S_{\max}(v_i)$, $i \in \{1, \ldots, k\}$.

cas 2 : $s' \in M_i$ avec $(s, M_i) \in R_{\mathrm{MAX}}(v_i)$.

Dans les deux cas $s' \in S(v_i)$. Si s' n'est pas maximum dans $S(v_1) \cup \ldots \cup S(v_k)$ alors son père s appartient à $S(v_1) \cup \ldots \cup S(v_k)$ et donc $s \in S(v)$. Mais c'est contraire à l'hypothèse que $(s, M) \in R_{\mathrm{MAX}}(v)$ laquelle implique que $s' \notin S(v)$. □

Le cœur de l'algorithme est le calcul de $S_{\max}(v)$ et $R_{\mathrm{MAX}}(v)$ en fonction de $S_{\max}(v_i)$ et $R_{\mathrm{MAX}}(v_i)$, $i = 1, \ldots, k$, où $\mathrm{Succ}_T(v) = \{v_1, \ldots, v_k\}$, qui peut se faire de la façon suivante :

Procédure $Q(v)$ [avec $\mathrm{Succ}_T(v) = \{v_1, \ldots, v_k\} \neq \emptyset$]

- 0) Initialiser $R_{\mathrm{MAX}}(v) := \emptyset$.
- 1) Calculer $L := \max(S_{\max}(v_1) \cup \ldots \cup S_{\max}(v_k))$.
- 2) Calculer L', l'ensemble des sommets de D qui ont au moins un successeur dans L.
- 3) Pour tout $s \in L'$:

3.1) Calculer l'ensemble $P(s)$ des multi-ensembles M de sommets de D de la forme $M = M_1 \sqcup M_2 \sqcup \ldots \sqcup M_k$ avec pour tout $i = 1, 2, \ldots, k$, ou bien $M_i = [\![s']\!]$ avec $s' \in S_{\max}(v_i)$, ou bien M_i est tel que $(s, M_i) \in R_{\mathrm{MAX}}(v_i)$ ou bien (et seulement si aucune des deux possibilités n'est réalisable), $M_i = \emptyset$.

3.2) Si il existe $M \in P(s)$ tel que $\mathrm{MSucc}_D(s) \sqsubseteq M$ alors faire $L := L \cup \{s\}$.

3.3) Sinon faire

$$R_{\mathrm{MAX}}(v) := R_{\mathrm{MAX}}(v) \cup \mathrm{MAX}(\{(s, M)/M \in P(s)\}).$$

- 4) Définir $S_{\max}(v) := \max(L)$.

Compte tenu du lemme 5, ces instructions traduisent le théorème 1, la validité est donc immédiate. Il reste à analyser la complexité.

LEMME 6 : *Soit $L \subseteq V_D$ un ensemble donné comme une liste strictement croissante pour $\leq_D$ d'éléments de V_D. On peut calculer* $\max(L)$ *en temps* $\mathcal{O}(n.Card(L))$.

Preuve : Remarquons tout d'abord que l'on peut décider si $x \ll y$ pour $x, y \in V_D$ en temps $\mathcal{O}(\text{Card}(\text{Succ}_D^*(y)))$. Il suffit de parcourir en largeur le dag D/y à partir de y : $x \ll y$ si et seulement si x est rencontré pendant ce parcours.

Soit m le dernier élément de la suite L. Il est certainement dans $\max(L)$ puisque $\leq_D$ est tel que $s \rightsquigarrow_D s'$ s'implique $s' \leq_D s$. On parcourt D/m en largeur, on enlève de L les sommets rencontrés (plus précisément, on les marque) et on répète à partir du dernier élément restant dans L. Plus précisément, l'algorithme est le suivant :

Initialisation :

- les sommets de D sont tous non marqués;
- $L' := \emptyset$;

Tant que $(L \neq \emptyset)$ *faire :*

- $m :=$ le dernier élément de L;
- $L' := L' \cup \{m\}$;
- parcourir en largeur à partir de m le sous-dag de D/m constitué des sommets non marqués; marquer au passage chaque sommet rencontré (y compris m); pour chaque sommet rencontré s autre que m pendant ce parcours, faire :

– *si* $(s \in L)$ *alors* $L := L - \{s\}$ et $L' := L' \cup \{s\}$.

Fin de *Tant que.*

Résultat : L'

Il est clair que le résultat L' de ce calcul n'est autre que $\max(L)$. Le temps de calcul est $\mathcal{O}(n.\text{Card}(L))$. En effet, chaque itération de la boucle visite un ensemble de sommets de D et pour chaque sommet, on parcourt une sous-liste de la liste d'origine L. À cause du marquage, chaque sommet n'est visité qu'une fois. D'où la complexité $\mathcal{O}(n.\text{Card}(L))$. □

On note $[0, d_i]$ l'ensemble ordonné

$$\{0, 1, 2, \ldots, d_i\} \qquad \text{avec} \quad 0 < 1 < 2 < \ldots < d_i.$$

LEMME 7 : *Pour tous p et $d \in \mathbb{N}_+$, la taille maximum $f(p, d)$ d'une antichaîne de l'ensemble ordonné $[0, d_1] \times [0, d_2] \times \ldots \times [0, d_p]$ avec $0 \leq d_i$ et $d_1 + d_2 + \ldots + d_p \leq d$ vérifie :*

- $f(p, d) = 1$ *si* $p = 1$

• $f(p, d) \leq \dfrac{1.4}{\sqrt{d}} 2^d$ *si* $p > 1$.

Preuve : Il est clair que $f(p, d) = 1$ car $[0, d_1]$ est totalement ordonné.

On sait grâce à De Bruijn *et al.* [4] que la taille maximum d'une antichaîne de $[0, d_1] \times \ldots \times [0, d_p]$ est égale à la cardinalité de

$$\{(x_1, \ldots, x_p)/0 \leq x_i \leq d_i \text{ et } x_1 + \ldots + x_p = (d_1 + \ldots + d_p)/2\}.$$

Pour $d_1, \ldots, d_p$ non nuls, Anderson montre ([2], page 68) que ce nombre est majoré par :

$$g(d_1, \ldots, d_p) = \sqrt{\frac{6}{\pi}} \frac{(d_1 + 1)(d_2 + 1) \ldots (d_p + 1)}{\sqrt{d_1(d_1 + 2) + \ldots + d_p(d_p + 2)}}.$$

Il en résulte que

$$f(p, d) \leq \text{Max}\{g(d_1, \ldots, d_p)/p \leq d, 1 \leq d_i, d_1 + \ldots + d_p \leq d\}.$$

En écrivant dans l'expression de $g(d_1, \ldots, d_p)$

$$\frac{d_1 + 1}{\sqrt{d_1(d_1 + 2) + A}}$$

sous la forme

$$\frac{1}{\sqrt{1 + \dfrac{A - 1}{(d_1 + 1)^2}}}$$

on voit que $g(d_1, \ldots, d_p)$ est croissante par rapport à chacune de ses variables (par symétrie). Il suffit donc de considérer son maximum pour $d_1 + \ldots + d_p = d$. En écrivant :

$$\frac{(d_1 + 1)(d_2 + 1)}{\sqrt{d_1(d_1 + 2) + d_2(d_2 + 2) + A}}$$

sous la forme

$$\frac{d_1 d_2 + (d_1 + d_2) + 1}{\sqrt{(d_1 + d_2)^2 - 2 d_1 d_2 + 2(d_1 + d_2) + A - 1}}$$

on voit que $g(d_1, d_2, d_3, \ldots, d_p) \leq g(d_1 + d_2)/2, (d_1 + d_2)/2, d_3, \ldots, d_p)$ avec égalité si et seulement si $d_1 = d_2$. Il en résulte que le maximum de

$g(d_1, \ldots, d_p)$ pour $d_1 + \ldots + d_p = d$ est atteint pour $d_1 = \ldots = d_p = \frac{d}{p}$. Cela donne :

$$g(d_1, \ldots, d_p) \leq \sqrt{\frac{6}{\pi}} \frac{\left(1+\frac{d}{p}\right)^p}{\sqrt{p\left(\frac{d}{p}\left(\frac{d}{p}+2\right)\right)}} = \sqrt{\frac{6}{\pi}} \frac{\sqrt{p}}{d} \frac{\left(1+\frac{d}{p}\right)^p}{\sqrt{1+\frac{2p}{d}}}$$

avec $p \leq d$ d'où

$$g(d_1, \ldots, d_p) \leq \sqrt{\frac{6}{\pi}} \frac{1}{\sqrt{d}} 2^d$$

car on a bien $(p+d)^p \leq 2^d p^p$ pour $p \leq d$. D'où finalement $f(p, d) \leq \frac{1.4}{\sqrt{d}} 2^d$ en prenant $\sqrt{\frac{6}{\pi}} \approx 1.4$. □

Preuve du théorème 2 : Évaluons la complexité du calcul de $Q(v)$.

Étape 0 : $\mathcal{O}(1)$.

Étape 1 : la fusion des listes $S_{\max}(v_1), \ldots, S_{\max}(v_k)$ avec élimination des répétitions se fait en temps $\mathcal{O}(a.k)$ (car $|S_{\max}(v_i)| \leq a$). Prendre le maximum se fait en temps $\mathcal{O}(n.a.k)$ d'après le lemme 6.

Étape 2 : l'ensemble L' a au plus a éléments. Cette étape se réalise en temps $\mathcal{O}(a)$.

Étape 4 : l'ensemble L dont on doit extraire le maximum est de taille au plus $2a$. La construction de $\max(L)$ se fait donc en temps $\mathcal{O}(n.a)$ d'après le lemme 6.

Étape 3 : examinons chaque itération de la boucle « *pour tout* $s \in L'$ » .

Il y a au plus $f(\bar{p}, \bar{d})$ multi-ensembles M tels que $(s, M) \in R_{\mathrm{MAX}}(v_i)$ où $\bar{p}$ est le nombre d'éléments distincts de $\mathrm{MSucc}_D(s)$ et $\bar{d}$ est le degré sortant de s. En effet, soit d_i la multiplicité de s_i dans $\mathrm{MSucc}_D(s)$. Un multi-ensemble $M \sqsubseteq \mathrm{MSucc}_D(s)$ n'est autre qu'un vecteur $(x_1, \ldots, x_{\bar{p}})$ avec $0 \leq x_i \leq d_i$ pour tout $i = 1, \ldots, \bar{p}$. On a donc $\bar{d} = d_1 + \ldots + d_{\bar{p}}$. Deux multi-ensembles M et M' sont incomparables pour $\sqsubseteq$ si et seulement si les vecteurs associés le sont pour l'ordre produit sur $[0, d_1] \times \ldots \times [0, d_{\bar{p}}]$. Or la taille d'une antichaîne dans $[0, d_1] \times \ldots \times [0, d_{\bar{p}}]$ est bornée par $\frac{1.4}{\sqrt{d}} 2^d$ par le lemme 7. Il en résulte que $\mathrm{Card}\{ M/(s, M) \in R_{\mathrm{MAX}}(v_i) \}$ est borné de la même façon.

Le calcul $P(s)$ consiste à prendre au plus $(a+f(p,\ d))^k$ multi-ensembles. Le calcul de chacun d'eux comme une liste triée prend un temps $\mathcal{O}(a.k)$. Tester si $L \sqsubseteq M$ où L et M sont deux multi-ensembles représentés par des listes triées prend un temps $\mathcal{O}(|L| + |M|)$. Les étapes 3.1 et 3.2 prennent donc un temps $\mathcal{O}((a + f(p,\ d))^k (a.k + d))$ et l'étape 3.3 un temps $\mathcal{O}((a + f(p,\ d))^k$. Donc pour s fixé on trouve un temps $\mathcal{O}((a + f(p,\ d))^k (a.k + d))$ avec k égal au degré sortant de v.

Au total, l'étape 3 utilise un temps $\mathcal{O}(n(a + f(p,\ d))^k (a.k + d))$. L'algorithme complet utilise un temps $\mathcal{O}\left(n.m\left(a+\frac{1.4}{\sqrt{d}}2^d\right)^l (al+d)\right)$. □

Rappelons que n, a, d dépendent du petit arbre et m et l du grand. On a aussi $a \leq n$. Si l'on fixe d degré maximum du petit arbre et l, degré du grand, on obtient un algorithme polynomial en $\mathcal{O}(m.n^{l+2})$ pour le test du mineur où m est le nombre de sommets du grand arbre et n le nombre de sommets d'un dag représentant le petit.

4.4. Cas particulier

On examine ici un cas particulier. Pour tout arbre U et tout entier $d > 0$ posons $d \bullet U = \bullet(U,\ U,\ \ldots,\ U)$ avec d occurrences de U. On note 0 l'arbre réduit à une racine. Ainsi, l'arbre binaire complet de hauteur n se note $2 \bullet 2 \bullet 2 \bullet \ldots \bullet 2 \bullet 0$ avec n occurrences de 2.

THÉORÈME 3 : *Pour tout arbre T et tous entiers strictement positifs d_1, d_2, ..., d_n, on peut décider en temps*

$$\mathcal{O}(n.Card(V_T)) \qquad si \quad d_n \bullet d_{n-1} \bullet \ldots \bullet d_1 \bullet 0 \trianglelefteq T.$$

Preuve : Soient T et $d_1, \ldots, d_n$ donnés. Pour tout $v \in V_T$, on définit un vecteur $SR(v) \in \mathbb{N}^n$ de la façon suivante, $SR(v) = (\alpha_1,\ \alpha_2,\ \ldots,\ \alpha_n)$ où pour tout $i = 1, \ldots, n$, α_i est le plus grand entier α tel que $\alpha \bullet d_{i-1} \bullet d_{i-2} \bullet \ldots \bullet d_1 \bullet 0 \trianglelefteq T/v$; si un tel α n'existe pas, on prend $\alpha_i = 0$. Le vecteur $SR(v)$ combine les fonctions $S_{\max}$ et R_{MAX} de l'algorithme général. On notera $SR(v)(n)$ la n-ième composante du vecteur $SR(v)$.

Fait 1 : si $v \in V_T$ alors $d_n \bullet d_{n-1} \bullet \ldots \bullet d_1 \bullet 0 \trianglelefteq T/v$ si et seulement si $SR(v)(n) \geq d_n$.

Fait 2 : si v est une feuille de T alors $SR(v) = (0,\ 0, \ldots,\ 0)$.

Fait 3 : si v n'est pas une feuille et $\mathrm{Succ}_T(v) = \{v_1, \ldots, v_k\}$ alors $SR(v) = (\alpha_1,\ \alpha_2,\ \ldots,\ \alpha_n)$ avec :

• $\alpha_1 = \gamma_1 + \gamma_2 + \ldots + \gamma_k$ avec $\gamma_j = \mathrm{Max}\{1,\ SR(v_j)(1)\}$ pour $j = 1, \ldots, k$,

• et pour $i = 2, \ldots, n$, $\alpha_i = \gamma_1 + \gamma_2 + \ldots + \gamma_k$ avec, pour $j = 1, \ldots, k$,

– $\gamma_j = SR(v_j)(i)$ si $SR(v_j)(i) > 0$,

– $\gamma_j = 1$ si $SR(v_j)(i) = 0$ et $SR(v_j)(i-1) \geq d_{i-1}$,

– $\gamma_j = 0$ sinon.

Preuve du fait 3 : Il résulte du lemme 4 que, pour tous arbres U, $T_1, \ldots, T_k$: $d \bullet U \trianglelefteq \bullet(T_1, \ldots, T_k)$ si et seulement $d = d_1 + d_2 + \ldots + d_k$ avec, pour tout $j = 1, \ldots, k$, $d_j \geq 0$ et :

• si $d_j > 1$ alors $d_j \bullet U \trianglelefteq T_j$,

• si $d_j = 1$ alors $U \trianglelefteq T_j$.

Remarquons que $1 \bullet U \trianglelefteq T_j$ implique $U \trianglelefteq T_j$. Donc la dernière de ces conditions pourrait s'écrire aussi : si $d_j = 1$ alors $d_j \bullet U \trianglelefteq T_j$ ou $U \trianglelefteq T_j$. Le fait 3 en résulte, avec l'aide du fait 1, qui implique que $SR(v_j)(i-1) \geq d_{i-1}$ équivaut à $d_{i-1} \bullet d_{i-2} \bullet \ldots \bullet d_1 \bullet 0 \trianglelefteq T/v_j$. □

Au moyen d'un parcours montant dans l'arbre T, on peut calculer $SR(v)$ pour tout $v \in V_T$ grâce aux faits 2 et 3 et décider si $d_n \bullet d_{n-1} \bullet \ldots \bullet d_1 \bullet 0 \trianglelefteq T$ grâce au fait 1. Le calcul de $SR(v)$ pour les feuilles prend un temps $n.(nombre\ de\ feuilles)$. Le calcul de $SR(v)$, si v est de degré sortant k, prend (fait 3) un temps $\mathcal{O}(n.k)$, soit pour tous les sommets qui ne sont pas des feuilles $\mathcal{O}(n.\Sigma\{\deg_-\mathrm{s}(v)/v \in V_T\}) = \mathcal{O}(n.\mathrm{Card}(V_T))$. □

Le cas particulier est intéressant comme test préliminaire. Avant de lancer l'algorithme principal testant si $U \trianglelefteq T$, on teste si $U' \trianglelefteq T$ où $U' = d_n \bullet d_{n-1} \bullet \ldots \bullet d_1 \bullet 0$ avec $d_n, \ldots, d_1$ tel que $U \trianglelefteq U'$ choisis de la façon suivante :

• n = hauteur de U,

• d_n = degré sortant de la racine de U,

• d_{n-i} = degré sortant maximum d'un sommet à distance i de la racine pour $i = 1, \ldots, n-1$.

Si l'on s'aperçoit que $U' \trianglelefteq T$, on peut conclure que $U \trianglelefteq T$. Sinon, on lance l'algorithme principal testant $U \trianglelefteq T$.

4.5. Extension aux préarbres

Nous avons supposé T donné comme un arbre. Si T est donné comme préarbre P augmenté d'une racine r, alors on peut construire un tri topologique de $T = (P, r)$ au moyen d'un parcours en profondeur. On peut donc, au cours d'un même parcours :

• construire la fonction Succ_T (c'est-à-dire bâtir la structure de données représentant T à partir de celle représentant P (donnée, par exemple, au moyen de listes d'incidences)),

• construire le tri topologique de T,

• calculer les fonctions $S_{\max}$ et R_{MAX}.

La complexité globale restant la même.

5. CONCLUSION

L'algorithme que nous avons défini, est utilisable sur des arbres de taille raisonnable. D'un point de vue expérimental, la taille de l'arbre U a un rôle primordial. Par exemple, il est possible de comparer un arbre U de taille < 45 et un arbre T de taille > 300 en quelques secondes, le même test appliqué sur un arbre U de taille 100 demande, par contre, plusieurs heures de calcul. Au-delà de 100 sommets, les temps de calcul deviennent donc très importants. Nous avons généré aléatoirement 3 ensembles A, B et C d'arbres. L'ensemble A composé de 20 arbres de taille 50, l'ensemble B de 100 arbres de taille 60 et l'ensemble C comprend 100 arbres de taille 110. Le but du test était de comparer chaque arbre de A avec chaque arbre des ensembles A, B ou C. La première série de tests compare deux à deux les arbres de A, donc réalise 400 appels à l'algorithme et se termine en 6 minutes. La seconde série de tests compare chaque arbre de A avec chaque arbre de B et nécessite 50 minutes de calcul. La dernière série qui compare des arbres de A et de C, s'effectue en 15 h.

REMERCIEMENTS

Nous remercions D. Loeb qui nous a indiqué les références [2] et [4].

RÉFÉRENCES

1. A. Aho, J. Hopcroft et J. Ullman, *The design and analysis of computer algorithms*, Addison-Wesley, 1974.
2. I. Anderson, *Combinatorics of finite sets*, Clarendon Press, Oxford, 1987.
3. S. Arnborg, A. Proskurowski et D. Corneil, Forbidden minors characterization of partial 3-trees, *Discrete Mathematics*, 1990, *80*, p. 1-19.
4. N. De Bruin, C. A. Van Ebbenhorst et D. Kruyswijk, On the set of divisors of a number, *Nieuw Arch. Wisk*, 1952, *23*, p. 191-193.
5. M. Fellows et M. Langston, An analogue of the Myhill-Nerode theorem and its use in computing finite basis characterization, *Proceedings of "Foundations of Computer Science"*, 1989, p. 520-525.

6. M. FELLOWS, The Roberston-Seymour theorems: A survey of applications, A.M.S., *Contemporary Mathematics*, 1989, *89*, p. 1-17.
7. G. A. IKORONG, Arbres et largeur linéaires des graphes, *Thèse*, Université Joseph Fourier - Grenoble-I, 1992.
8. J. MATOUŠEK et R. THOMAS, On the complexity of finding iso- and other morphisms for partial k-trees, *Discrete Mathematics*, 1992, *108*, p. 343-364.
9. A. PROSKUROWSKI, Graph reductions and techniques for finding minimal forbidden minors, dans Graph structure theory, N. ROBERTSON et P. SEYMOUR Eds., A.M.S., *Contemporary Mathematics*, 1993, *147*, p. 591-600.
10. N. ROBERSTON et P. SEYMOUR, Graph minors *I*: Excluding a forest, *Journal of Combinatorial Theory*, Series B, 1983, *35*, p. 39-61.
11. N. ROBERTSON et P. SEYMOUR, Graph minors *II*: Algorithmic aspects of tree-width, *Journal of algorithms*, 1986, *7*, p. 309-322.
12. N. ROBERTSON et P. SEYMOUR, *Graph minors XIII: The disjoint paths problem*, septembre 1986.
13. N. ROBERTSON et P. SEYMOUR, *Graph minors XX: Wagner's conjecture*, septembre 1988.