Algorithme générique pour les jeux de capture dans les arbres

David Coudert, Florian Huc, Dorian Mazauric

Algorithme générique pour les jeux de capture dans les arbres[†]

David Coudert and Florian Huc and Dorian Mazauric

MASCOTTE, I3S(CNRS/UNSA)-INRIA, 2004 route des Lucioles – B.P. 93, 06902 Sophia Antipolis, France.
E-mail: firstname.lastname@sophia.inria.fr

Nous présentons un algorithme distribué simple calculant le process number des arbres en n étapes, avec un nombre total d'opérations en $O(n\log(n))$ et un total de $O(n\log(n))$ bits échangés. De plus cet algorithme est facilement adaptable pour calculer d'autre paramètres sur l'arbre, dont le node search number.

Keywords: process number, pathwidth, search number, arbre, algorithme

Nous commençons par introduire deux invariants de graphes : le *process number* et le *node search number*. Le point commun entre ces deux invariants est qu'ils peuvent être calculés à partir d'actions élémentaires sur les sommets.

Process number Le process number d'un graphe orienté (digraphe) est un invariant qui a été introduit dans [1, 2] dans le cadre de l'étude de la reconfiguration du routage à l'intérieur d'un réseau WDM. Les modifications à apporter sont modélisées par un digraphe, qui peut être vu comme un digraphe de conflits. Passer d'un routage R_1 à un routage R_2 revient à supprimer tous les sommets du digraphe des conflits en respectant les règles suivantes :

– On peut à tout moment mettre un sommet dans une mémoire temporaire s'il n'y est pas déjà. Ce faisant, on supprime tous les arcs entrant dans ce sommet.
– On peut supprimer un sommet qui n'a pas d'arc sortant. Si de plus le sommet était dans une mémoire temporaire, il la libère.

Une suite d'actions permettant de supprimer tous les sommets d'un digraphe D est appelée stratégie. Le nombre maximun de mémoires temporaires simultanément utilisées par une stratégie est appelé coût de la stratégie. Le minimum des coûts des stratégies permettant de supprimer tous les sommets est le process number du digraphe, noté $\mathrm{pn}(D)$.

Le process number d'un graphe (non orienté) G est le process number du graphe orienté symétrique associé D.

Node search number Le node search number est un autre invariant important qui a été introduit par Parson [6] dans le cadre des jeux de capture. Ces jeux tour à tour opposent un fugitif à des agents. Le but de ces jeux est de déterminer le nombre minimun d'agents nécessaire pour capturer le fugitif dans un graphe sous certaines conditions de déplacement des agents et du fugitif. Dans le cadre du node search number, le fugitif a le droit à chaque tour de se déplacer sur n'importe quel sommet du graphe tant qu'il ne passe pas par un sommet occupé par un agent. Ensuite, à chaque tour, un des agents peut faire une des deux actions suivantes :

– Se placer sur un sommet (quelconque) du graphe s'il est en dehors du graphe.
– Sortir du graphe s'il est sur un sommet du graphe.

La seule information dont les agents disposent est la position des autres agents. Le fugitif est capturé s'il se trouve sur un sommet occupé par un agent. Etant donné un graphe G, le nombre minimun d'agents nécessaire pour capturer le fugitif est appelé le node search number et est noté $\mathrm{sn}(G)$.

Il a été prouvé par Ellis *et al.* [3] que $\mathrm{sn}(G) = \mathrm{pw}(G)+1$, où $\mathrm{pw}(G)$ est la *pathwidth* de G, et par Kinnersley [5] que $\mathrm{pw}(G) = \mathrm{vs}(G)$, où $\mathrm{vs}(G)$ est la *vertex separation* de G. Ces résultats montrent que la

[†]Recherche soutenue par les projets européens IST FET AEOLUS et COST 293 GRAAL, l'ANR JC OSERA et la région PACA.

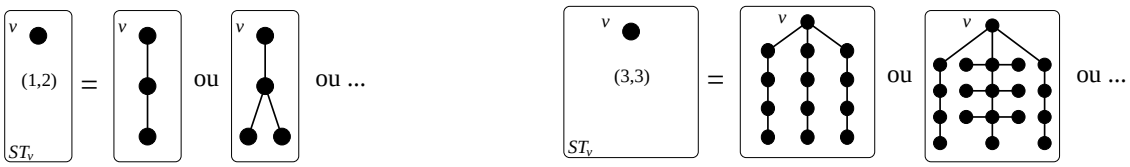


FIG. 1: Exemple d'arbres dont les vecteurs associés sont $vect(w) = (1,2)$ et $vect(w) = (3,3)$.

vertex separation, le node search number et la pathwidth sont équivalents. Le lecteur est invité à lire l'état-de-l'art établit par Fomin et Thilikos [4] présentant de nombreuses variantes des problèmes de capture.

Lien entre le process number et le node search number La proposition 1, prouvée dans [1], établit le lien étroit qui existe entre le process number et la vertex separation d'un digraphe D, notée $\mathrm{vs}(D)$. A ce jour il n'est pas connu si le process number et la vertex separation sont équivalents.

Proposition 1 *Pour tout digraphe D, on a* $\mathrm{vs}(D) \leq \mathrm{pn}(D) \leq \mathrm{vs}(D)+1$.

Nos résultats Nous présentons ici un algorithme distribué permettant de calculer le process number d'un arbre et qui peut être aisément adapté au calcul de son node search number ou à celui d'invariants correspondant à différentes variantes des jeux de capture. Notre algorithme en n étapes effectue un total de $O(n\log(n))$ opérations et transmet $O(n\log(n))$ bits. Ainsi, la version de cet algorithme adaptée pour calculer le node search number ne donnera pas l'algorithme le plus performant qui soit connu puisqu'il existe un algorithme linéaire [7]. Cependant notre algorithme est le premier permettant de calculer le process number d'un arbre et donne une caractérisation des arbres T pour lesquels on a $\mathrm{pn}(T) = \mathrm{vs}(T)$ $(= \mathrm{pw}(T))$, l'algorithme présenté dans [1] étant inexact.

Notre algorithme est basé sur la décomposition d'un arbre $T = (V,E)$ en sous-arbres de telle manière que ces sous-arbres forment une structure hiérarchique. Son principe d'exécution est le suivant :

(1) Initialisation de l'algorithme au niveau des feuilles. Les feuilles envoient un message à leurs pères (unique voisin). Les feuilles deviennent traitées.
(2) Un sommet v, ayant reçu un message de tous ses voisins sauf un, traite les informations reçues et envoie un message à son dernier voisin (son père). Le sommet v devient traité.
(3) Le dernier sommet reçoit un message de tous ses voisins puis calcule le process number de l'arbre.

Nous allons décrire plus en détails la nature des messages envoyés et ce qu'ils décrivent. Avant cela on note que l'algorithme est bien distribué, qu'il peut s'éxécuter dans un environnement asynchrone et qu'il y a au total autant d'étapes que de sommets dans l'arbre.

Le but d'un message transmis lors du point (2) par un sommet v au voisin v_0 dont il n'a pas reçu de message, est de décrire de manière synthétique la structure de l'arbre enraciné en v. L'arbre enraciné en v étant la partie connexe de l'arbre T privé de l'arête vv_0 ($T \setminus vv_0$) contenant v. On note ce sous-arbre T_v.

En fait nous décrivons une décomposition hiérarchique du sous-arbre T_v. Nous décomposons un tel arbre en sous-arbres : T_v^w. Les sous-arbres T_v^w forment une partition de T_v et on note R_v l'ensemble des racines de ces sous-arbres. Un sous-arbre peut être de deux types : stable ou instable. Un sous-arbre stable a un process number qui n'est pas influencé par l'ajout de structures de même taille alors qu'un sous-arbre instable va voir son process number augmenter.

Un sous-arbre T_v^w de T_v enraciné en w, qu'il soit stable ou instable, est décrit par le couple de paramètres $(\mathrm{pn}(w), \mathrm{pn}^+(w))$, où $\mathrm{pn}(w)$ est le process number de T_v^w et $\mathrm{pn}^+(w)$ est le process number de T_v^w avec la contrainte supplémentaire que le traitement termine avec w en mémoire. De manière générale, on représente un sous-arbre par ce couple de paramètres, comme illustré par la Fig. 1. Les valeurs $\mathrm{pn}(w)$ et $\mathrm{pn}^+(w)$ sont associées à w de manière unique et indépendante de v au cours de l'algorithme. Un tel couple est associé à chaque sommet.

Le sous-arbre T_v^w est dit *stable* si $\mathrm{pn}(w) = \mathrm{pn}^+(w)$, *instable* sinon. Par extension, il en est de même pour w.

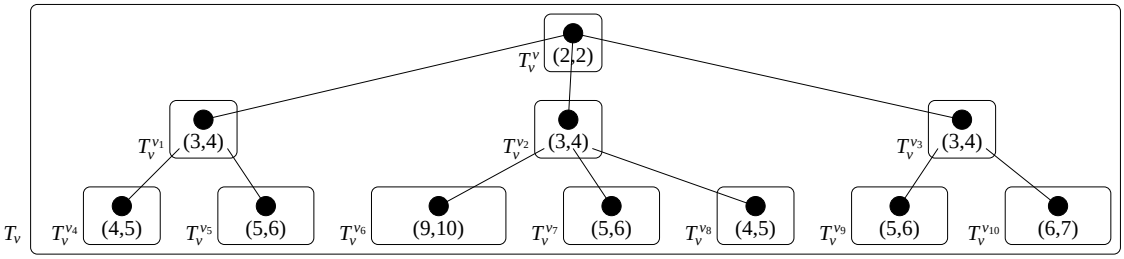


FIG. 2: Exemple de décomposition hiérarchique d'un arbre T_v de process number 9.

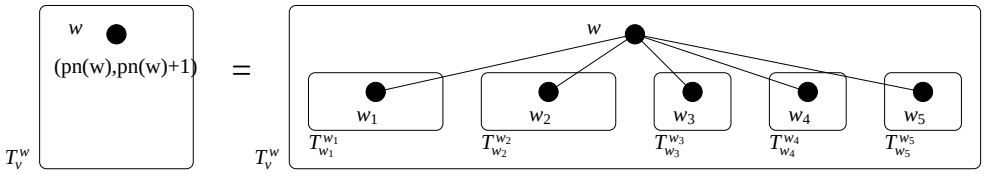
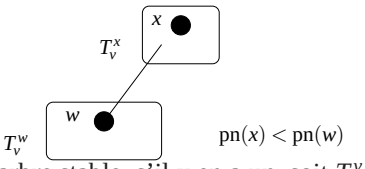


FIG. 3: Structure d'un sous-arbre instable S_v. $vect(v_1) = vect(v_2) = (\mathrm{pn}(v), \mathrm{pn}(v))$ et $\forall i \in [3,5], \mathrm{pn}(T_i) < \mathrm{pn}(v)$.

Dans la décomposition hiérarchique de l'arbre T_v, on impose qu'un sous-arbre T_v^w ait un process number plus grand que le sous-arbre T_v^x de T_v contenant le père de w, comme l'illustre la figure ci-contre.



On impose de plus à une décomposition hiérarchique que le seul sous-arbre stable, s'il y en a un, soit T_v^v, le sous-arbre de T_v enraciné en v. La Fig. 2 montre une décomposition hiérarchique d'un arbre T_v avec un tel sous-arbre stable.

La Proposition 2 décrit la structure d'un sous-arbre instable. Connaissant cette structure, il est possible à partir de la décomposition hiérarchique de calculer le process number de l'arbre T_v.

Proposition 2 (c.f. fig. 3) *Soit w un sommet et T_v^w son sous-arbre associé. On note $\Gamma(w) \cap T_w = \{w_1, \ldots, w_k\}$, où T_w est le sous-arbre enraciné en w. Si w est instable, alors w a deux voisins $w_1, w_2 \in \Gamma(w) \cap T_w$ qui sont tous les deux stables et vérifient* $\mathrm{pn}(w_1) = \mathrm{pn}(w_2) = \mathrm{pn}(w)$. *De plus T_v^w est formé de sa racine w, et de $l \leq k-2$ autres sous-arbres $T_{w_3}^{w_3}, \ldots, T_{w_{l+2}}^{w_{l+2}}$ dont les racines sont des voisins traités. Le process number des autres sous-arbres $T_3, \ldots, T_{l+2}$ est au plus* $\mathrm{pn}(w) - 1$.

De la décomposition hiérarchique d'un arbre, on en déduit un parcours. Dans l'exemple de la Fig.2, la stratégie consiste à traiter une des branches de $T_v^{v_6}$ de process number 9, donc en utilisant 9 mémoires, puis de laisser le sommet v_6 en mémoire. Il reste alors 8 mémoires disponibles. Ensuite, on traite la totalité de $T_v^{v_{10}}$ en utilisant 7 mémoires et en terminant avec v_{10} en mémoire. On traite de même $T_v^{v_9}$ en terminant avec v_9 en mémoire. Il reste alors 6 mémoires disponibles, ce qui est suffisant pour traiter $T_v^{v_3}$ en terminant avec v_3 en mémoire. On peut alors traiter v_9 et v_{10}, libérant ainsi 2 mémoires. On vient de finir le traitement de T_{v_3}. Maintenant, en partant de v_3, on traite une partie de T_v^v en remontant vers v. Si on rencontre un parent u de v_1 ou de v_2, on laisse le sommet u en mémoire, et avec les 7 mémoires disponibles, on traite $T_u - \{u\} - T_v^{v_6}$ comme on a traité T_{v_3}. Ensuite, on continue à traiter T_v^v en direction de v. Lorsque l'on a finalement traité v, 8 mémoires sont disponibles et on les utilise pour finir le traitement de $T_v^{v_6}$, la Proposition 2 nous assurant que cela est bien possible.

En définitive, pour connaître le process number de T_v, il suffit de connaître les paramètres des sous-arbres formant sa décomposition hiérarchique. Afin de faciliter l'accès à ces informations, on stocke dans chaque sommet u de l'arbre, les paramètres des sous-arbres de l'arbre T_u dans un tableau. Ainsi, dans l'exemple de la Fig.2, v stocke le tableau suivant, ainsi que le couple de valeurs $(\mathrm{pn}(v), \mathrm{pn}^+(v))$ de T_v^v :

$$t_v = \boxed{0}\boxed{0}\boxed{2}\boxed{2}\boxed{2}\boxed{1}\boxed{0}\boxed{0}\boxed{1} \text{ et } (\mathrm{pn}(v), \mathrm{pn}^+(v)) = (2,2)$$

Le tableau t est indicé de 1 à $L(t)$, où $L(t)$ est le plus grand indice d'une case non nulle. La case d'indice i contient 2 s'il y a au moins 2 sous-arbres dont le couple est $(i, i+1)$, 1 s'il y en a qu'un seul, 0 sinon.

C'est donc un tableau d'au plus $2L(t)$ bits.

Etant donné un tableau correspondant à une structure hiérarchique d'un arbre T, le process number de T est donné par le Théorème suivant :

Théorème 1 *Etant donné un arbre T, un sommet $v \in V$, une structure hiérarchique de racine v et son tableau associé t,* $\mathrm{pn}(T_v) = L(t)$ *ssi* $\exists i \in [2, L(t)], t_i = 0$ *et* $\forall j \in [i+1, L(t)], t_j = 1$, *ou* $\forall i \in [2, L(t)], t_i = 1$.

Nous pouvons maintenant décrire plus précisement les étapes de l'algorithme.

(1) Chaque feuille de l'arbre initialise son tableau à 0 et son couple à $(0,0)$. Elle transmet ensuite ces informations à son père (son unique voisin).

(2) Un sommet v reçoit ces informations de tous ses voisins sauf un. Il effectue un traitement en temps linéaire en la somme des tailles des tableaux reçus pour calculer son tableau et son couple qu'il transmet ensuite à son père (unique voisin n'ayant pas transmis d'information). Les détails de cette étape se trouvent sur la page internet `http://www-sop.inria.fr/mascotte/David.Coudert/Capture/`.

(3) Le dernier sommet effectue le même traitement qu'un sommet à l'étape (2), en ayant reçu les informations de tous ses voisins. Le Théorème 1 permet de conclure quant au process number de l'arbre.

Complexité L'analyse de cet algorithme permet de déduire :

- Nombre d'étapes de l'algorithme : n.
- Coût des calculs au sommet v : $O(\sum_{w \in N(v)} L(t_w))$, où $N(v)$ est le voisinage de v qui a envoyé un message à v, et t_w le tableau transmis par w.
- Coût total des calculs dans l'arbre : $O(n\log(n))$. En effet, on a $\sum_{v \in V} \sum_{w \in N(v)} L(t_w) \leq n\mathrm{pn}(T) \leq n\log(n)$.
- Nombre de messages transmis : $n-1$.
- Taille du message transmis par le sommet v à son père : les $2L(t_v)$ bits du tableau et le couple $(\mathrm{pn}(v), \mathrm{pn}^+(v))$. Comme $\mathrm{pn}^+(v) \leq \mathrm{pn}(T)+1 \leq \log(n)$, la transmission du couple de valeurs requiert $O(\log\log(n))$ bits. Le sommet v transmet donc $O(\log(n))$ bits.
- Quantité totale d'information transmise au cours de l'algorithme : $O(n\log(n))$ bits.

De manière générale, on ne sait pas caractériser en temps polynomial les graphes G tels que $\mathrm{pn}(G) = \mathrm{pw}(G)$. Dans le cas de arbres, on peut établir le lemme suivant :

Lemme 1 *A la fin de l'exécution de l'algorithme on obtient que le sommet v terminant l'algorithme est instable si et seuleument si* $\mathrm{pn}(T) = \mathrm{pw}(T) = \mathrm{sn}(T) - 1$.

Conclusion Il reste encore de nombreux problèmes à explorer dont le premier est de déterminer si le process number est équivalent ou non au search number. Nous cherchons également à étendre notre algorithme aux graphes planaires extérieurs pour lesquels le meilleur algorithme connu a une complexité supérieure à $O(n^{11})$.

Références

[1] D. Coudert, S. Perennes, Q.-C. Pham, and J.-S. Sereni. Rerouting requests in wdm networks. In *AlgoTel'05*, pages 17–20, Presqu'île de Giens, France, mai 2005.

[2] D. Coudert and J-S. Sereni. Characterization of graphs and digraphs with small process number. Research Report 6285, INRIA, September 2007.

[3] J.A. Ellis, I.H. Sudborough, and J.S. Turner. The vertex separation and search number of a graph. *Information and Computation*, 113(1) :50–79, 1994.

[4] F. V. Fomin and D. Thilikos. An annotated bibliography on guaranteed graph searching. *Theoretical Computer Science, Special Issue on Graph Searching*, 2008, to appear.

[5] N. G. Kinnersley. The vertex separation number of a graph equals its pathwidth. *Inform. Process. Lett.*, 42(6) :345–350, 1992.

[6] T. D. Parsons. Pursuit-evasion in a graph. In *Theory and applications of graphs*, pages 426–441. Lecture Notes in Math., Vol. 642. Springer, Berlin, 1978.

[7] K. Skodinis. Construction of linear tree-layouts which are optimal with respect to vertex separation in linear time. *J. Algorithms*, 47(1) :40–59, 2003.