RAIRO. RECHERCHE OPÉRATIONNELLE

I. LAVALLÉE

Un algorithme parallèle efficace pour construire un arbre de poids minimal dans un graphe

RAIRO. Recherche opérationnelle, tome 19, nº 1 (1985), p. 57-69

<http://www.numdam.org/item?id=RO_1985__19_1_57_0>

R.A.I.R.O. Recherche opérationnelle/Operations Research
(vol. 19, n° 1, février 1985, p. 57 à 69)

UN ALGORITHME PARALLÈLE EFFICACE POUR CONSTRUIRE UN ARBRE DE POIDS MINIMAL DANS UN GRAPHE (*)

par I. LAVALLÉE (1)

Résumé. — *A partir d'une propriété locale des arbres de poids minimal, due notamment à Sollin, nous donnons un algorithme parallèle qui calcule un tel arbre sur un graphe (ou un multigraphe) valué. On admet ici que deux arêtes différentes puissent avoir même poids. Cet algorithme peut être implémenté sur un ordinateur M.I.M.D. (Multi-Instructions, Multi-Données) et il utilise un nombre décroissant de processus au cours du calcul.*

La complexité (exprimée en « temps ») est comprise entre $n-1$ *et* $\log_2 n$. *La borne* $n-1$ *n'est atteinte que dans un cas très particulier. L'algorithme est d'autant plus efficace que le nombre de sommets du graphe est important. La complexité (en temps) tend vers* $\log_2 n$ *dans les meilleurs cas.*

Nous mettons en évidence et discutons des cas de conflits qui peuvent apparaître au cours du calcul.

Mots clés : Arbre couvrant minimal, Algorithmique parallèle, exclusion mutuelle.

Abstract. — *From a local property of a minimum spanning tree due to Sollin, we give an algorithm wich computes a minimum spanning tree in a valued undirected graph (or multigraph) with possible ties of weights on edges. This algorithm can be implemented on a M.I.M.D. computer with a decreasing number of process during calculation.*

The time complexity of this algorithm is less than $n-1$ *and greater or equal to* $\log_2 n$.

The bound $n-1$ *is reached only in a very particular pessimistic case. The "speed rate" of the algorithm increases with n in general, and time complexity approaches* $\log_2 n$ *in best cases. Finally, we discuss the possibility of conflicts wich may appear during computation.*

Keywords: Minimum Spanning Tree, parallel Algorithms, mutual exclusion.

(*) Reçu en mai 1983.

(1) Laboratoire de Recherche en Informatique, Bât. n° 490, Université Paris-Sud, 91405 Orsay Cedex, U.A. n° 410 du C.N.R.S. « Al Khowarizmi » et C.N.A.M., Chaire de Recherche opérationnelle, 292, rue Saint-Martin, 75003 Paris.

R.A.I.R.O. Recherche opérationnelle/Operations Research, 0399-0559/85/01 57 13/$ 3.30

INTRODUCTION

Nous donnons ici un algorithme parallèle utilisant des processeurs asynchrones disposant d'une mémoire commune, pour construire un arbre de poids minimal dans un graphe. (La machine supposée est donc de type M.I.M.D. [8].)

Contrairement à [12] et [14], nous admettons que deux arêtes distinctes puissent avoir le même poids.

Par conséquent, il peut exister plusieurs arbres de poids minimal; le problème est d'en exhiber au moins un.

Étant donné un graphe valué $G=(X, U)$, nous associons un processus [2] à chaque sommet (soit au total n processus) au début de l'exécution, et nous libérons *au moins* un processus (qui peut alors être restitué au système) par étape de l'algorithme grâce à une opération de réduction qui permet de continuer l'exécution sur le graphe associé à la partie de l'arbre (forêt) déjà construite.

Dans le cas général, la complexité de l'algorithme est telle que :

$$O([\log_2 n]) \leqq \alpha \leqq O(n-1) \quad ([\ ] \text{ signifiant « partie entière par excès »}).$$

Dans le cas du parallélisme, nous tentons quelques comparaisons des performances des différents algorithmes existants.

1. NOTATIONS ET DÉFINITIONS

Considérons un graphe connexe non orienté, valué défini comme suit :

$$G=(X, U, P), \tag{1.1}$$

où :

X est l'ensemble des sommets du graphe;

U est l'ensemble des arêtes du graphe;

P est une fonction de valuation de U dans $\mathbb{N}^*$.

L'application P induit en préordre sur U.

$$\{\forall u_1, u_2 \in U;\ P(u_1) \geqq P(u_2)\} \quad \Rightarrow \quad \{u_1 \geqq u_2\}. \tag{1.2}$$

([2]) On utilise ici le concept, logique, de processus plutôt que celui, physique, de processeur afin de ne pas avoir à entrer dans le détail de la gestion des ressources systèmes qui est propre à chaque machine.

A. Arbre couvrant G.

On dira qu'un arbre couvre G si tous les sommets de G sont sommets de l'arbre. On rappelle qu'un arbre couvrant $G=(X, U)$ est une composante connexe de $n-1$ arêtes ($|X|=n$), et qu'un arbre est sans cycle. Notons $A=(X, W)$ un arbre couvrant G; ($W \subset U$).

B. Poids d'un arbre.

Étant donné un arbre $A=(X, W)$, on appelle $\mathbb{P}(A)$, le « poids » de l'arbre A :

$$\mathbb{P}(A)=\sum_{(x,y)\in W} p(x, y).$$

2. L'ALGORITHME DE SOLLIN ET PRIM [2, 7, 11]

Nous rappelons l'algorithme dit de Sollin et de Prim 1 tel qu'il est exposé dans [7], p. 93, c'est-à-dire dans le cas séquentiel.

(*a*) Distinguer les arêtes de même poids initial.

(*b*) A chaque étape de l'algorithme, choisir arbitrairement un sommet en dehors de ceux qui ont déjà été retenus dans la construction, et relier par l'arête de poids le plus faible, ce sommet à l'un des sommets auquel il est adjacent, l'arête ainsi sélectionnée et ses extrémités sont dits alors « retenus ».

(*c*) Lorsque l'ensemble des sommets a été utilisé entièrement (i. e., tous les sommets ont été retenus) :

— ou bien le résultat obtenu est un arbre, le problème est alors résolu;

— ou bien on n'a encore que plusieurs sous-arbres qu'on considérera chacun comme l'un des sommets d'un multigraphe [3], les arêtes de ce multigraphe étant toutes les arêtes susceptibles de connecter deux à deux ces sous-arbres; on réitère l'algorithme à partir de l'étape (*b*) sur ce multigraphe.

3. UNE PROPRIÉTÉ LOCALE

Pour un graphe $G=(X, U, P)$, connexe, on peut énoncer le théorème suivant :

THÉORÈME : *Pour tout sommet x, soit $\tilde{x}$ tel que :*

$$p(x, \tilde{x})=\operatorname{Min}_{y\in\Gamma(x)}[p[x, y]],$$

(3) On appelle multigraphe, un graphe dont deux sommets au moins sont reliés par plus d'une arête (ou arc).

il existe un arbre de poids minimal, couvrant le graphe G, et contenant l'arête $[x, \tilde{x}]$.

Supposons qu'il existe un arbre couvrant G_1 : $A_1 = (X, W_1)$ avec $[x, \tilde{x}] \notin W_1$; A_1 étant connexe, il existe un sommet t $(t \neq \tilde{x})$ tel que $[x, t] \in W_1$; de plus :

$$[x, t] \in W_1 \quad \Rightarrow \quad p[x, t] \geqq p[x, \tilde{x}].$$

Dans A_1 il existe une chaîne et une seule entre x et $\tilde{x}$, soit :

$$[x, \gamma], [\gamma, \ldots], \ldots, [\ldots, \alpha], [\alpha, \tilde{x}]$$

cette chaîne. Or $p[x, \gamma] \geqq p[x, \tilde{x}]$; si on supprime l'arête $[x, \gamma]$ de A_1, on disconnecte A_1 en deux composantes connexes sans cycle : l'une C_1 contenant x, l'autre C_2 contenant $\tilde{x}$.

Par ajoût de $[x, \tilde{x}]$, on reconnecte C_1 et C_2, formant ainsi un arbre $A_2 = (X, W_2)$.

Puisqu'on passe de W_1 à W_2 en substituant à l'arête $[x, \gamma]$, l'arête $[x, \tilde{x}]$, il vient :

$$P(A_1) \geqq P(A_2).$$

N.B. : Ce théorème s'étend à tout cocycle $w(Y)$ $(Y \subset X)$ où $[x, \tilde{x}]$ désigne l'arête de plus faible valeur du cocycle $w(Y)$.

4. PRINCIPES D'UN ALGORITHME PARALLÈLE

1. En tout sommet $x \in X$ (et ceci de façon asynchrone et parallèle pour tous les sommets) associer :

$$\tilde{x} \in X \qquad \text{tel que} \quad p(x, \tilde{x}) = \operatorname*{Min}_{y \in \Gamma(x)} [[x, y]],$$

sans former de cycle (*voir* § 5).

2. Construire le multigraphe $\bar{G} = (\bar{X}, \bar{U})$ associé aux sous-arbres (comme au § II).

3. Si $|\bar{X}| = 1$ Fin. (L'arbre minimal est constitué de l'ensemble des $n-1$ arcs retenus en 1.)

Sinon — recommencer avec $G \leftarrow \bar{G}$

Convergence de l'algorithme.

A chaque itération, $\bar{G}$ est constructible car G est connexe, et le nombre de sommets de $\bar{G}$ diminue d'au moins une unité à chaque itération.

On peut alors montrer à l'aide de la propriété locale, que l'arbre ainsi trouvé est de poids minimal.

Donc, en *au plus* $n-1$ itérations, $\bar{G}$ est un graphe réduit à 1 sommet.

5. LES PROBLÈMES INFORMATIQUES A RÉSOUDRE, LES MOYENS A METTRE EN ŒUVRE

5.1. Les problèmes

Le principal problème à résoudre est celui des conflits entre processus dans le cas où il y a égalité de poids sur certaines arêtes. Par exemple, considérons le graphe de la figure 1.

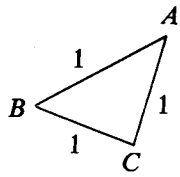

Figure 1.

En appliquant la règle ci-dessus, le processus associé au sommet A sélectionnera par exemple l'arête $[A, B]$, celui associé à B l'arête $[B, C]$, et celui associé à C, l'arête $[C, A]$. Ainsi on constituerait un cycle, ce qui est interdit.

Pour éviter ceci, il faut que lorsque deux processus A et B sont entrés en communication (i. e., l'arête $[A, B]$ est introduite dans l'arbre), ni A, ni B, ni les ressources système nécessaires à leur communication, ne soient accessibles par un autre processus.

On peut schématiser comme suit le fonctionnement logique d'un processus :

1. *Identification*

C'est une séquence du processus qui identifie le *sommet* du graphe initial auquel il faut connecter le processus (c'est-à-dire la composante connexe sans cycle — éventuellement réduite à un sommet — à laquelle est associé le processus en question).

2. *Sélection-Connexion*

C'est une séquence qui sélectionne le processus qui gère le sommet précédemment identifié et qui connecte les processus entre eux.

(C'est dans cette phase que se pose le problème du cyclage susmentionné, on utilisera alors la notion de « séquence exclusive » [5], [6], [14], explicitée plus loin.)

3. *« Phagocytage », mise à jour*

Lorsque le processus A entre en communication avec le processus B, A s'approprie toute l'information liée à B, relie les composantes connexes sans cycle associées respectivement à A et B, par l'arête qui a été sélectionnée en phase 1 et qui a servi à identifier le sommet géré par B.

Le processus A remet à jour les informations et variables globales affectées par ces modifications.

4. *Arrêt du processus, remise a disposition du système*

Le processus B n'ayant plus aucun rôle à jouer peut s'arrêter.

Les ressources système affectées au processus B (par ex., un processeur) peuvent alors être remises à la disposition du système pour d'autres tâches.

On pourrait craindre un autre type de problème lié à la simultanéité d'actions des processus. Ainsi A voulant entrer en communication avec B, B avec A et ceci simultanément. En fait, comme on s'est placé dans le cas asynchrone, on peut considérer que ce problème ne se pose pas. Si toutefois on souhaite envisager la solution de tels conflits, il suffit alors de créer une relation d'ordre strict (une numérotation par exemple) sur les processuss, et en cas de conflit, priorité est donnée au processus d'ordre supérieur.

5.2. Les moyens à mettre en œuvre

Dans le cas présent, on suppose qu'il y a au départ de l'algorithme un processus attaché à chaque sommet du graphe soit, pour un graphe d'ordre n, n processus.

De plus, la gestion des variables globales et les communications entre processus se font par l'intermédiaire d'une mémoire centrale partageable. Les cellules de cette mémoire sont accessibles en lecture simultanément sans conflit.

La mémoire centrale est accessible en écriture, mais de façon univoque.

Comme évoqué dans la phase 2, il y a des conflits à régler entre les processus pour éviter le cyclage, ceci s'obtient en interdisant pour certaines séquences de programme, le partage des ressources systèmes nécessaires à l'exécution de la dite séquence.

On considère donc ici qu'on dispose du point de vue logiciel, de la possibilité de rendre une séquence d'instructions exclusive (*voir* [4, 5, 6, 15]).

On entend par séquence exclusive, une séquence d'instructions appartenant à un processus telle que lorsque cette séquence est commencée, toutes les ressources (que ce soient des variables, des tableaux, ou des processeurs)

susceptibles d'être utilisées par le processus actif au sein de cette séquence, sont attachées, durant toute l'exécution de la séquence (mais seulement de la séquence) au processus concerné de façon exclusive.

Afin de distinguer les variables locales propres à chaque processus, on conviendra de représenter tout identificateur de variable par un couple $< N, C >$ où N est le numéro du processus et C le nom de la variable (C pouvant être un tableau).

Par ailleurs, les variables globales (i. e. stockées en mémoire centrale partageable), partageables par plusieurs processus, seront désignées par un couple de la forme $< O, C >$.

La notation $< N, . >$ sera équivalente à $< N, N >$, c'est le *numéro* (l'identificateur) du processus.

Signification des différentes variables locales :

$< N,$ CØMPT $>$ est un compteur qui contient le nombre *d'arêtes* du sous-arbre géré par le processus N.

$< N,$ DISP $>$ est une variable d'état du processus N.

DISP=0 signifie que le processus N est entièrement *disponible.*

DISP=1 signifie que le processus N est en *communication* avec un autre processus.

DISP=2 signifie que le processus N *n'est plus disponible* et ne le sera plus jamais, à sa prochaine activation le processus N se « suicidera » pour le problème en cours et se remettra à la disposition du système pour *un autre problème.*

$< 0,$ PØINT $>$ est un tableau situé en zone globale de mémoire. Le tableau PØINT permet de savoir quel processus gère quel sommet $\{$ PØINT $T(I)=P\} \Leftrightarrow \{$ le processus P gère le sommet $I\}$ PØINT est accessible en lecture simultanément par plusieurs processus; mais il n'est accessible en écriture que de façon exclusive.

$\not\equiv$ est le symbole de début ou de fin d'une séquence exclusive.

$< N,$ ARBR $>$ est le tableau du sous-arbre géré par le processeur N.

$\wedge$ $\vee$ sont les opérateurs logiques « et » et « ou ».

$|$ est l'opérateur de concaténation.

$<$ N, LIST $>$ est un tableau propre au processus N, il contient pour LIST(T) le numéro du sommet géré par N tel que $< N,$ MAT$(T) >$ donne la valuation de l'arête d'extrémités LIST(T) (dans N) et T [ailleurs, donné par $< 0,$ PØINT$(T) >$].

6. LA SPÉCIFICATION DE L'ALGORITHME

(1) WHILE $< N, C\emptyset MPT > \neq M-1$

/* On vérifie qu'on n'a pas déjà trouvé un arbre */

do
=

(2) $< N, T > \leftarrow (< N, T > : (< N, MAT(T) > = MIN(< N, MAT >))$

/* T est le nom du sommet relié par une arête de poids minimal au sous-arbre géré par N */

(3) $\not=$ WHEN $(< N, DISP > = 0 \wedge < P\emptyset INT(T), DISP > = 0)$
$\vee (< N, DISP > = 2)$

/* on n'exécute la séquence exclusive que si cette expression logique est vraie, c'est une commande gardée */

do :

If $< N, DISP > \neq 2$ then $< N, DISP > \leftarrow 1$; $< P\emptyset INT(T), DISP > \leftarrow 1$
$< N, Q > \leftarrow < 0, P\emptyset INT(T) >$;
$< 0, P\emptyset INT(T) > \leftarrow < N, . >$

/* Le processus dont le numéro est dans $P\emptyset INT(T)$ devient alors « esclave » du processus N lequel a alors accès à ses variables locales */

f I

/* On donne la valeur 1 aux variables d'états des processus qui sont en communication et on remet à jour le tableau PØINT dans lequel on écrit, et ce uniquement à travers cette séquence exclusive, on a auparavant sauvegardé le numéro du processus avec lequel on communique dans la variable locale $< N, Q >$ */

od

$\not=$

(4) If $< N, DISP > = 2$ then STOP *fi*

(5) $< N. COMPT > \leftarrow < N, COMPT > + < Q, COMPT > + 1$;

(6) $< N, ARBR > \leftarrow < N, ARBR > | < Q, ARBR > | \{ < N, LIST(T) >, < N, T > \}$;—

/* On remet à jour le compteur d'arêtes du processus, puis le tableau qui contient les dites arêtes (tableau ARBR). Les deux processus étant en communication, puisque dans la séquence exclusive leurs variables d'état ont été fixées à 1, on peut donc considérer que chacun d'entre eux a accès aux variables locales de l'autre. */

(7) PROCÉDURE A;

/* *voir* ci-après */

(8) $< Q, \text{DISP} > \leftarrow 2$;

(9) $< N, \text{DISP} > \leftarrow 0$;

od

(10) STOP.

La procédure A permet de transférer toute l'information liée au processus Q, phagocyté par N, et de recalculer les poids des arêtes dont une extrémité (et une seule) est hors du sous-arbre nouvellement formé, si deux telles arêtes ont même extrémité externe, on garde celle dont la valuation est minimale.

```
PROCÉDURE A
   DÉBUT
      I ← 1
      TANT QUE I≦M
      FAIRE :
      SI (< 0, PØINT(I) > =N ou < 0, PØINT(I) > =Q)
            ALORS
         < N, MAT(I) > ← ∞,
         < 0, PØINT(I) > ← N
            SINON :
         SI (< N, MAT(I) > .GT. < Q, MAT(I) >)
      FAIRE :
         < N, MAT(I) > ← < Q, MAT(I) >
         < N, LIST(I) > ← < Q, LIST(I) >
         fsi
      fsi
   FIN PROC.
```

Remarque :

La variable d'état $< N, \text{DISP} >$ se justifie pour les deux raisons suivantes :

(*a*) elle permet de gérer de façon efficace la commande gardée de la séquence exclusive, ce qui nécessite au moins deux valeurs pour DISP (disponible et non disponible);

(*b*) elle permet de résoudre un problème important, celui de l'arrêt d'un processus, pour celà il est nécessaire de distinguer deux cas dans l'état non disponible :

(i) le processus est en communication et il peut redevenir disponible,

(ii) le processus est en train d'être phagocyté et il ne redeviendra jamais disponible et il convient alors de pouvoir lui fournir un test d'arrêt (on pourrait bien sûr déléguer ce test au système, mais cette solution ne nous a pas paru satisfaisante).

On peut résumer par un graphe les transitions d'états d'un processus (*fig.* 2) :

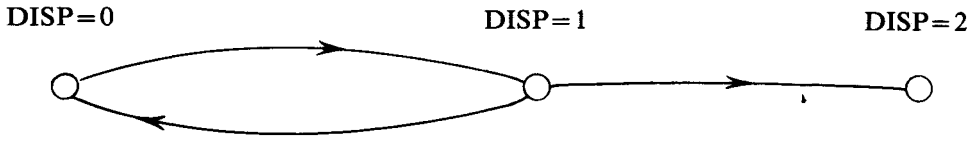

Figure 2

7. ÉVALUATION DES PERFORMANCES

Il est très difficile de comparer les performances des différents algorithmes (ainsi, ceux de [12] et [14]) car les hypothèses, tant sur le problème lui-même que sur le modèle de calcul, sont différentes. Par exemple [12] et [14] supposent que toutes les arêtes du graphe sont de poids distincts, [12] suppose un fonctionnement sur une machine vectorielle type array processeur avec n^2 processeurs (*voir* [8]); [14] suppose un réseau de processeurs complètement distribués.

Le fait de supposer qu'il n'y a pas deux arêtes de même poids évacue, comme le fait remarquer [12], p. 51, le problème de conflit lié à la possibilité de cycles d'arêtes équipondérées, problème qui est résolu ici par la structure même de la séquence exclusive.

De plus [14] identifie un autre cas de conflit susceptible de dégrader considérablement les performances, c'est le cas où la structure du graphe infère qu'un *seul processeur* travaille à chaque étape.

On en a un exemple avec le graphe de la figure 3 où pourtant il n'y a pas deux arêtes de même poids. Ainsi, dans le cas le plus défavorable (diamètre du graphe égal à 2) il se peut qu'il n'y ait qu'un seul processeur qui travaille à chaque étape, par conséquent il faut $n-1$ étapes, la complexité de l'algorithme est alors en $O(n-1)$. Par contre, dans le cas général, à chaque étape, chaque sommet se connecte à un autre, et, faisant abstraction du temps de calcul de chaque processeur pris individuellement, (comme [13]) (4) la complexité s'exprime alors en $O(\lceil \log n \rceil)$ ($\lceil\ \rceil$ signifiant « partie entière par

(4) Cette hypothèse faite par [13] se justifie dans le cas de machines M.I.M.D., mais pas dans le cas de machines S.I.M.D.

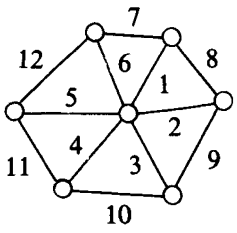

Figure 3.

excès »). Si on note α la complexité de l'algorithme ci-dessus, on a donc :

$$O([\log n]) \leqq \alpha \leqq O(n-1). \tag{6.1}$$

Efficacité d'un algorithme parallèle

Il serait intéressant de pouvoir comparer entre eux, d'une part différents algorithmes parallèles, d'autre part des algorithmes parallèles et des algorithmes séquentiels.

Le modèle théorique de la machine de Türing utilisé par la théorie de la complexité est difficilement applicable au parallélisme.

Nous prendrons donc ici comme mesure d'« efficacité » d'un algorithme le produit de la complexité intrinsèque d'un algorithme et du nombre de processus mis en œuvre. Ici, en appliquant ce critère on obtient en [11] un processus de complexité $O([\log n]^2)$ avec n^2 processus [5], soit donc une efficacité qui s'exprime en :

$$e_1 = O(n^2 [\log n]^2). \tag{6.2}$$

En appliquant la même mesure à l'algorithme de base présenté par [13], avec n processus on obtient une complexité qui s'exprime en $O(n \log n)$, soit donc une efficacité qui devient :

$$e_2 = O(n^2 \log n). \tag{6.3}$$

L'algorithme que nous avons présenté ici est de complexité α comme exprimé en (6.1) pour n processus soit donc une efficacité :

$$O(n[\log n]) \leqq e_3 \leqq O(n^2) \tag{6.4}$$

(5) Toutefois, ces n^2 processeurs sont des processeurs élémentaires de machine S.I.M.D. et de coût très inférieur à ceux des machines M.I.M.D.

Ces résultats sont à rapprocher de la complexité du meilleur algorithme séquentiel connu qui s'exprime en $O(n^2)$ avec *un seul* processus soit donc une efficacité :

$$e_4 = O(n^2). \tag{6.5}$$

CONCLUSION

Nous avons donné ici un algorithme réellement implémentable sur une machine M.I.M.D. sans structure particulièrement adaptée au problème contrairement à [1].

Nous n'avons d'autre part, fait aucune hypothèse simplificatrice sur la structure du graphe, malgré celà, les comparaisons d'efficacité sont en faveur de cet algorithme. De plus, nous avons donné une façon simple de résoudre les conflits.

BIBLIOGRAPHIE

1. J. L. BENTLEY, *A Parallel Algorithm for Constructing Minimum Spanning Trees*, Journal of algorithms, vol. 1, 1980, p. 51-59.
2. C. BERGE et A. GHOUILA HOURI, *Programmes, jeux et réseaux de transport*, Dunod, Paris, 1962.
3. C. BERGE, *Graphes & hypergraphes*, Dunod, Paris, 1973, 2e édition.
4. Per. BRINCH HANSEN, *Distribued Processes = A Concurrent Programming Concept*, CACM, vol. 21, n° 11, 1978, p. 934-941.
5. Q. S. F. CARVALHO et G. ROUCAIROL, *Une amélioration de l'algorithme d'exclusion mutuelle de Ricart et Agrawala*, L.I.T.P. Internal Report n° 81-58, novembre 1981.
6. Q. S. F. CARVALHO et G. ROUCAIROL, *On Mutual Exclusion in Computer Networks*, Technical correspondance, CACM, vol. 26, n° 2, février 1983.
7. R. FAURE, *Précis de Recherche Opérationnelle*, Dunod, Paris, 1979.
8. I. LAVALLÉE, *Notes sur le parallélisme*, C.N.R.S.-G.R. Claude François Picard, Tour 45, 4, place Jussieu, 75230 Paris Cedex 05, 1983.
9. I. LAVALLÉE, "*An efficient parallel algorithm for computing a minimal spanning tree*", Parallel Computing 83, Elsevier Science Publishers, B.V. (North-Holland) pp. 259-262, 1984.
10. F. MAFFIOLI, *Complexity of Optimum Undirected Tree Problems*, Analysis and Design of Algorithms in Combinatorial Optimization, G. AUSIELLO et M. LUCERTINI éd., Springer-Verlag, 1981.
11. R. C. PRIM, *Shortest Connections Networks and Some Generalizations*, Bell System Tech. J., vol. 36, 1957, p. 1389-1401.
12. C. SAVAGE et J. J JA'JA, *Fast, Efficient Parallel Algorithms for Some Graph Problems*, S.I.A.M. J. on Computing, vol. 10, n° 4, novembre 1981, p. 682-691.

13. M. SOLLIN, Exposé du Séminaire de C. Berge, I.H.P., 1961; repris *in extenso* dans *Méthodes et modèles de la R.O.* », t. 2, p. 33-45, A. KAUFMANN éd., Dunod, Paris, 1968.

14. D. STOTT PERKER et B. SAMADI, *Distributed Minimal Spanning Tree Algorithms*, Performances of data communication system and their applications, G. PUJOLLE éd., North Holland Publishing Company, 1981, p. 46-53.

15. G. RICART et A. AGRAWALA, *An Optimal Algorithm for Mutual Exclusion in Computer Networks*, Comm. A.C.M. 24. 1, janvier 1981, p. 9-17.

On trouvera une bibliographie très complète dans [14].