INFORMATIQUE THÉORIQUE ET APPLICATIONS

JEAN-MICHEL HELARY
AOMAR MADDI
MICHEL RAYNAL

Calcul réparti d'un extrémum et du routage associé dans un réseau quelconque

Informatique théorique et applications, tome 21, n° 3 (1987), p. 223-244

<http://www.numdam.org/item?id=ITA_1987__21_3_223_0>

Informatique théorique et Applications/Theoretical Informatics and Applications
(vol. 21, n° 3, 1987, p. 223 à 244)

CALCUL RÉPARTI D'UN EXTRÉMUM ET DU ROUTAGE ASSOCIÉ DANS UN RÉSEAU QUELCONQUE (*)

par Jean-Michel HELARY ([1]), Aomar MADDI ([1]) et Michel RAYNAL ([1])

Communiqué par J.-P. BANATRE

Résumé. – *Deux algorithmes distribués réalisant une élection dans un système distribué sont présentés. Leur originalité réside dans les hypothèses qu'ils nécessitent et les techniques qu'ils utilisent. Aucune hypothèse particulière n'est faite sur le graphe qui modélise le réseau si ce n'est sa connexité; de plus aucun des processus n'a besoin de connaître ni la structure globale du réseau ni même sa taille. Les algorithmes sont donc en ce sens entièrement distribués, et se distinguent fondamentalement des algorithmes qui s'appuient sur une topologie particulière connue de tous (tel qu'un anneau ou un maillage complet par exemple).*

Fondés sur un même principe : l'extinction sélective de messages, les algorithmes proposés généralisent au contexte distribué les stratégies de recherche séquentielle en profondeur et en largeur d'abord; des techniques et des outils spécialement adaptés sont introduits à cette fin. Leur emploi, en algorithmique distribuée, se révèle de portée générale.

Abstract. – *Two distributed algorithms for election in a distributed system are presented. Both are original with respect to hypothesis and techniques involved. No particular assumption on the network topology is made, but connectivity; moreover, no process needs to know or learn the network global structure or size. In that sense, both algorithms are fully distributed and fundamentally different from those assuming a particular topology, known by all processes (such as complete network or ring).*

Based upon the principle of "selective message extinction", the proposed algorithms extend to distributed contexts the sequential search strategies: "depth first" and "breadth first" respectively; to this end, specifically designed tools and techniques are introduced. They can be used in the general field of distributed algorithmic.

1. INTRODUCTION

La solution de certains problèmes de contrôle relatifs aux applications et aux systèmes distribués peut passer par la définition d'un rôle particulier que

(*) Reçu mars 1986, révisé octobre 1986.
Ce travail a été réalisé dans le cadre du G.R.E.C.O. et du P.R.C. C^3 supporté par le C.N.R.S. et le M.R.T.
([1]) I.R.I.S.A., Université de Rennes-I, Campus de Beaulieu, 35042 Rennes Cedex.

va jouer l'un (et un seul) des sites qui supportent l'application. C'est par exemple le cas dans les bases de données réparties où le problème du contrôle de la concurrence, dû à l'exécution parallèle de transactions accédant des données distribuées, peut être résolu en associant à chaque ensemble de données (dont chacune est dupliquée sur les divers sites du réseau) un site particulier; le rôle de ce site est de contrôler les accès qu'effectuent les transactions de façon à assurer la cohérence mutuelle de ces données (respect des contraintes d'intégrité) et la convergence de leurs copies vers une même valeur [2, 12]; un tel site est généralement appelé site primaire. La désignation d'un processus destiné à coordonner diverses activités dans un système distribué constitue un autre exemple du même problème [25].

Il est alors essentiel de savoir définir un tel site et de donner à chacun des autres sites les moyens de communiquer avec lui. Ce site est généralement choisi en fonction d'un critère de poids et il est d'usage de choisir celui dont le poids est un extrémum en considérant comme poids d'un site son identité représentée par un numéro. Une fois ce choix effectué il est nécessaire d'une part de communiquer l'identité de ce site à tous les autres sites et d'autre part d'établir des chemins de communication de ces sites vers le site choisi.

Une première solution consisterait à effectuer le choix d'un site, à établir les chemins correspondants de manière statique et à incorporer ces éléments dans les définitions des sites lors de la génération du système ou de l'application. L'absence de souplesse de cette solution est rédhibitoire et a conduit vers la recherche de solutions dynamiques.

Ces solutions réalisent ce qu'il est convenu d'appeler un algorithme distribué d'élection : tous les sites (nous confondrons dans ce qui suit les termes site et processus, un site se présentant en effet comme une seule activité du point de vue de l'élection) sont initialement candidats à jouer le rôle particulier; au terme de l'algorithme seul celui dont l'identité est la plus grande (ou la plus petite) peut effectivement le jouer, les autres pouvant alors solliciter ce dernier.

Les algorithmes proposés jusqu'alors ne s'intéressent, à notre connaissance, qu'au cas où le maillage du réseau d'une part est connu de tous (ce qui élimine le problème de l'établissement des chemins) et d'autre part est restreint à être soit un anneau (uni- ou bi-directionnel) [20, 5, 10, 8, 14, 24] soit un maillage complet [11, 19]. Les meilleurs de ces algorithmes ont, dans les deux cas, une complexité $O(n \log n)$ où n est le nombre de sites du réseau (c'est la complexité minimale possible dans le cas d'un anneau [23]).

Nous présentons dans cet article deux algorithmes distribués d'élection qui d'une part calculent un extrémum dans un réseau quelconque, et d'autre part définissent une arborescence couvrante de ce réseau (dont la racine est le

processus qui a pour identité l'extrémum) permettant à tous les processus de communiquer via des chemins uniques avec le processus associé à la racine.

La solution au problème de l'extrémum, proposée dans cet article, permet de développer des techniques algorithmiques réparties originales — notamment dans les définitions de messages et l'exploitation de leurs champs de valeurs; celles-ci peuvent être utilisées pour résoudre d'autres problèmes dans le contexte des systèmes et des applications réparties statiques et fiables.

L'article est divisé en six parties. Dans la seconde on précise les hypothèses et l'intérêt qu'elles présentent; dans la troisième les principes qui sous-tendent les algorithmes sont exposés. La quatrième partie présente un premier algorithme assimilable à une exploration distribuée en profondeur d'abord alors que l'algorithme présenté dans la cinquième partie s'appuie sur une exploration parallèle (en largeur). Enfin la conclusion synthétise les résultats obtenus et dégage l'originalité de la démarche proposée.

2. HYPOTHÈSES

2.1. Hypothèses sur le réseau

Les processus ne peuvent s'échanger de l'information qu'à l'aide de messages véhiculés sur un réseau; il n'y a donc aucune mémoire partagée entre les processus.

Le réseau de communication, modélisé par un graphe, est connexe mais quelconque. Les liaisons entre deux processus sont bi-directionnelles, en d'autres termes le graphe est non orienté. Elles sont dotées des propriétés de comportement suivantes : les messages ne se perdent pas et sont délivrés au bout d'un temps fini après leur émission, de plus ils ne sont pas altérés. Par contre, il n'est fait aucune hypothèse relative au non-dépassement de messages sur une ligne; en d'autres termes, que les messages se doublent ou non sur les lignes n'altère en rien le fonctionnement des algorithmes proposés. Ces derniers sont donc indépendants du caractère de la ligne : physique (dans ce cas les messages ne s'y doublent pas) ou logique (la mise en œuvre d'un contrôle réalisant le séquencement n'est pas nécessaire pour que l'algorithme fonctionne).

Ces hypothèses de comportement sont donc moins contraignantes que celles rencontrées dans la plupart des algorithmes se situant dans un contexte de système statique et fiable; dans un tel contexte, elles sont « minimales ».

2.2. Hypothèses sur les processus

Le réseau comporte n processus qui ont des identités distinctes. Un processus ne connaît initialement que ses voisins directs, de plus il les connaît par leurs identités. Au cours du calcul distribué un processus quelconque n'apprendra jamais la structure globale du réseau, ni même le nombre total n de processus participants : sa connaissance topologique se limitera toujours à ses voisins directs.

Insistons sur le fait que cette hypothèse est plus faible que toute autre impliquant, pour les processus, une connaissance *a priori* de la structure globale du réseau. Une telle connaissance induit en effet des propriétés globales que chaque processus peut exploiter localement : c'est le cas, par exemple, d'une topologie en anneau dans laquelle un message, émis par un processus vers l'un de ses deux voisins et lui revenant par l'autre, aura nécessairement visité tous les processus; ou encore le cas d'un réseau complet induisant la connaissance par chaque processus du nombre total de processus. *A contrario*, la seule connaissance, par un processus, de l'identité de ses voisins directs ne lui suffit pas pour établir son appartenance à une structure particulière (anneau, réseau complet, arbre, etc.); une telle configuration, si c'est celle dont rend compte la topologie du réseau, ne pourra être connue des processus qu'à l'issue d'un échange d'informations entre eux, c'est-à-dire après exécution d'un algorithme distribué.

2.3. Intérêt des hypothèses

Ces hypothèses sont particulièrement intéressantes car elles facilitent la génération de systèmes distribués. La définition d'un site n'inclut alors pas d'informations globales; en conséquence la configuration du système peut être modifiée sans que cela entraîne une nouvelle définition de tous les processus et sans que le système soit tributaire d'une topologie particulière; seuls les processus dont le voisinage a été modifié doivent être redéfinis et donc générés à nouveau (i. e. seul le texte de ces derniers doit être recompilé).

Dans le cas où un processus ne connaît son voisinage que par l'identité des liaisons (canaux) qui le connectent à ses voisins, l'identité de ceux-ci peut être acquise grâce à un protocole simple : deux messages par canal (un dans chaque sens) permettent l'échange des identités entre tout couple de processus voisins [28].

3. PRINCIPE DES SOLUTIONS

Dans un contexte séquentiel une réponse au problème consisterait à rechercher dans le graphe associé au réseau le sommet qui possède la plus grande identité puis à construire, avec ce processus pour racine, une arborescence recouvrante qui établirait les chemins entre un processus quelconque et la racine.

Dans un contexte distribué la solution dans laquelle tous les processus enverraient à l'un d'entre eux leur identité et celles de leurs voisins n'est pas possible car (outre le fait qu'elle ne respecte pas les hypothèses sur la localité des informations) elle nécessite la définition préalable d'un processus unique qui collecterait toutes les informations. Ce qui reviendrait à résoudre préalablement un problème... d'élection.

Une des caractéristiques essentielles du calcul distribué, qui fait une de ses originalités et de ses difficultés, réside dans le fait qu'un algorithme qui produit un unique résultat, peut être démarré « simultanément » par un, plusieurs, voire tous les processus. Le résultat doit *a priori* être indépendant du nombre de processus (au moins un) qui ont activé l'algorithme.

Le principe sur lequel reposent les deux algorithmes proposés est le suivant :

pour un processus $\mathscr{P}_i$, lancer une élection consiste à lancer une exploration du réseau dont le but est de visiter tous les processus en établissant des chemins de ceux-ci vers lui-même. Comme l'algorithme peut être démarré par un ou plusieurs processus, une ou plusieurs explorations peuvent être simultanément en cours. Au terme de l'algorithme il est nécessaire que l'exploration issue du processus doté de la plus grande identité d'une part se soit terminé correctement (i. e. ait visité tous les processus en établissant l'arborescence recouvrante) et d'autre part que celle-ci soit la dernière exploration prise en compte par les autres processus (i. e. que ceux-ci mémorisent les liaisons auxquelles ils participent dans l'arborescence construite). On peut identifier chaque exploration par son poids, c'est-à-dire l'identité du processus dont elle est issue; les deux objectifs ci-dessus seront atteints dès lors que : tout processus $\mathscr{P}_i$ visité par une exploration de poids j la stoppe sans la prendre en compte s'il a déjà été visité par une exploration de poids k avec $k>j$, ou bien, n'ayant encore jamais été visité et n'ayant pas lancé d'exploration il trouve $i>j$, auquel cas il lance une exploration; dans les cas contraires il prend en compte l'exploration de poids j et la fait progresser. Un tel principe d'« extinction sélective » se rencontre aussi bien en algorithmique séquentielle (par exemple le tri par bulles) que distribuée, sans être toujours mis en évidence. Il prend ici la forme d'un principe de développement

d'arborescences concurrentes. (Rappelons qu'une des premières utilisations de ce principe dans le domaine de l'algorithmique distribuée est dûe à Thomas [29] qui l'a utilisée pour le contrôle de la concurrence aux copies multiples d'un ensemble de données.)

Deux types d'explorations issues d'un processus $\mathscr{P}_i$ sont envisageables. Lorsqu'un processus est visité par une telle exploration et la prend en compte il peut la faire progresser soit vers l'un de ses voisins (le plus grand par exemple) soit vers tous : c'est ce qui distingue les deux algorithmes proposés. Comme on le voit ils étendent au contexte distribué les techniques séquentielles d'exploration en profondeur ou en largeur d'abord.

Nous allons présenter dans le détail le premier algorithme, en donner la preuve et une analyse de complexité. Pour le second nous nous limiterons à l'algorithme lui-même, la preuve étant analogue.

4. ALGORITHME 1 : EXPLORATION EN PROFONDEUR

4.1. L'algorithme

4.1.1. *Les étapes d'une exploration*

Une exploration E_k de poids k (issue de $\mathscr{P}_k$) peut être vue comme une activité séquentielle : une séquence de messages (de poids k) est engendrée; chaque message est traité par le processus qui le reçoit et celui-ci, à son tour, peut éventuellement émettre un nouveau message de poids k. A chaque instant de l'exploration, un seul message de poids k est en transit sur une ligne ou en traitement par un processus. On peut donc parler des étapes de l'exploration, matérialisées — par exemple — par la réception d'un message μ par un processus $\mathscr{P}_i$ (message et processus courants). Selon le contexte du processus $\mathscr{P}_i$ et l'information transportée par le message μ, on distingue quatre types d'étapes.

4.1.1.1. *Étape d'extinction*

— ou bien $\mathscr{P}_i$ a déjà été atteint par une exploration de poids supérieur à k (ou a déjà lancé sa propre exploration E_i si $i>k$),

— ou bien $\mathscr{P}_i$ n'a pas encore été atteint, mais $i>k$.

Dans un cas comme dans l'autre, l'exploration E_k est éteinte par $\mathscr{P}_i$; de plus, dans le deuxième cas, $\mathscr{P}_i$ lance l'exploration de poids i.

4.1.1.2. *Étape de conclusion*

Dans cette étape et les deux suivantes, on suppose que E_k est l'exploration de poids le plus fort ayant jamais atteint $\mathscr{P}_i$; *max* désignera la plus grande identité de l'ensemble des processus du réseau.

Cette étape intervient lorsque l'information transportée par le message reçu par $\mathscr{P}_i$ lui permet d'apprendre que tous les autres processus ont été atteints par l'exploration. Dans ce cas, E_k s'arrête et on a alors nécessairement : $k = max$; $\mathscr{P}_i$ peut donc conclure. L'exploration E_{max} a établi une arborescence recouvrant le réseau (modélisé par un graphe G non orienté connexe) et les autres processus sont informés de la fin de l'algorithme par un signal diffusé sur les branches de cette arborescence.

4.1.1.3. *Étape de génération*

$\mathscr{P}_i$ possède des voisins non encore atteints par E_k. Il met alors à jour les liaisons relatives à l'exploration E_k, et engendre un nouveau message d'exploration vers un de ses voisins non atteints. Le choix d'un tel voisin permet la mise en œuvre locale d'heuristiques de parcours; par exemple, le choix du voisin d'identité maximum — que nous retenons dans cet algorithme — définit une stratégie d'optimisation à court terme (dont, par ailleurs, rien ne prouve l'optimalité globale).

4.1.1.4. *Étape de renvoi*

Tous les voisins de $\mathscr{P}_i$ ont été atteints par E_k, mais l'information de contrôle amenée par le message indique que des processus du réseau n'ont pas encore été atteints par E_k. $\mathscr{P}_i$ engendre alors un message permettant de remonter vers un processus non encore atteint.

Dans ce qui suit nous dirons qu'un processus $\mathscr{P}_i$ est « visité » par l'exploration E_k lorsqu'il est atteint par cette exploration et ne l'éteint pas. De plus la fonction *maximum* appliquée à un ensemble z d'identités de processus délivre la plus grande d'entre elles.

4.1.2. *Les messages utilisés*

La mise en œuvre du principe énoncé précédemment nécessite trois types de messages distincts :

(i) les messages du type *explorer* sont de la forme suivante :

$$explorer\ (k, z, s)$$

émis lors d'une étape de génération, ils réalisent l'exploration des processus du réseau pour le compte du processus d'identité k. Le champ z contient les

identités des processus visités par ce message [on doit avoir $k = maximum\,(z)$], le champ s définit l'ensemble des processus voisins immédiats des processus de z qui n'ont pas encore été atteints par cette exploration (la condition $s \neq \varnothing$ indique donc que l'exploration lancée par $\mathscr{P}_k$ n'est pas terminée). s est donc un sous-ensemble des processus non atteints par E_k;

(ii) les messages du type *rebrousser*, émis lors d'une étape de renvoi, permettent à une exploration qui n'a pas atteint tous les processus (on a alors nécessairement $s \neq \varnothing$) mais qui ne peut progresser à partir du processus actuellement visité, de repartir à partir d'un processus visité qui possède un voisin appartenant à s. Les champs d'un tel message sont les mêmes que ceux du message de type *explorer*;

(iii) les messages du type *conclure* servent à signaler aux processus que l'exploration lancée par $\mathscr{P}_{\max}$ a atteint tous les processus; ils réalisent en conséquence la terminaison de l'algorithme.

4.1.3. *Contexte d'un processus*

Tous les processus ont les mêmes définitions de contextes et exécutent les mêmes textes d'algorithme. L'algorithme est symétrique en ce sens-là [26]. Dans tout ce qui suit on se place du point de vue d'un processus quelconque, que nous appelerons $\mathscr{P}_i$.

const $voisins_i$ **initialisée à** { identités des voisins de $\mathscr{P}_i$ }. Cet ensemble définit le voisinage de $\mathscr{P}_i$, qui est sa seule connaissance sur le réseau.

var $état_i$: (*initial*, *candidat*, *battu*, *élu*) **initialisé à** *initial*. Cette variable donne l'état courant de $\mathscr{P}_i$; à la terminaison de l'algorithme on doit avoir :

$$\exists!\, j : (état_j = élu \wedge \forall\, k \neq j : état_k = battu).$$

$pgvu_i$: **entier initialisé à** i. Donne l'identité de la plus grande exploration qu'ait jamais vu $\mathscr{P}_i$; à la fin on aura $pgvu_i = max$ (identité maximum).

$pred_i$: **entier initialisé à** *nil*.

$succ_i$: **ensemble d'entier initialisé à** $\varnothing$.

Ces deux variables serviront à placer $\mathscr{P}_i$ dans l'arborescence construite (*nil* désigne une valeur qui n'est l'identité d'aucun processus). (On a nécessairement $succ_i \cup \{ pred_i \} \subseteq voisins_i$.)

4.1.4. *Comportement de* $\mathscr{P}_i$

Le comportement auquel obéit un processus est donné (à la manière de [26]) sous la forme d'un automate associant un traitement à chaque événement qui peut survenir (réception d'un message d'un processus voisin ou décision

locale de lancer une élection). La définition suivante permet d'en condenser l'écriture.

```
procédure lancer-exploration;
début
  soit x = maximum (voisins);
  état_i : = candidat;
  pred_i : = nil;
  succ_i : = {x};
  envoyer explorer (i, {i}, voisins-{i}) à P_x
fin
```

Remarque : Vu l'heuristique choisie (sélection du voisin dont l'identité est la plus grande), toute exploration lancée par un processus dont un des voisins a une identité supérieure à la sienne, est vouée à l'extinction dès la première étape.

Un tel processus pourrait donc, en utilisant un nouveau type de message *activer*, demander à ce voisin un lancement d'élection. Cette modification est possible, puisque chaque processus connaît l'identité de ses voisins, et permet alors de remplacer certains messages *explorer* (k, z, s) par des messages *activer* sans paramètres, donc plus courts. Toutefois, cela ne change pas le nombre total de messages. Dans un souci de clarté, nous ne considérons pas, dans la suite, cette formulation.

```
lors de décision de lancer une élection
faire
  si état_i = initial alors lancer-exploration fsi
fait
lors de réception de explorer (k, z, s) depuis P_j
faire
  cas pgvu_i > k → si état_i = initial alors lancer-exploration fsi
      pgvu_i < k → état_i : = battu;
                   pgvu_i : = k;                                          (a1)
                   pred_i : = j;
                   soit y = voisins_i - z;
                   cas y = ∅ → succ_i : = ∅;
                     cas s = ∅ → envoyer conclure à P_j                   (a2)
                       s ≠ ∅ → envoyer rebrousser (k, z ∪ {i}, s) à P_j
                     fcas
                     y ≠ ∅ → soit x = maximum (y);
                       succ_i : = {x};
                       envoyer explorer (k, z ∪ {i}, s ∪ y-{x}) à P_x
                   fcas
  fcas
fait
```

lors de réception de *rebrousser* (k, z, s) **depuis** $\mathscr{P}_j$
faire
  si $pgvu_i = k$ **alors**
    soit $y = voisins_i \cap s$; $(a3)$
    cas $y = \varnothing \rightarrow$ **envoyer** *rebrousser* (k, z, s) **à** $\mathscr{P}_{pred_i}$

      $y \neq \varnothing \rightarrow$ **soit** $x = maximum\ (y)$;
        $succ_i := succ_i \cup \{x\}$;
        envoyer *explorer* $(k, z, s\text{-}\{x\})$ **à** $\mathscr{P}_x$ $(a4)$
    fcas
  fsi
fait
lors de réception de *conclure* **depuis** $\mathscr{P}_j$
faire
  si $pgvu_i = i$ **alors** $état_i := elu$ **fsi;**
  $\forall x \in (succ_i \cup pred_i)\text{-}\{j\}$: **envoyer** *conclure* **à** $\mathscr{P}_x$ $(a5)$
fait

4.2. Preuve

4.2.1. *Propriétés*

La compréhension et la preuve de l'algorithme sont facilitées si l'on met en évidence un certain nombre de propriétés ponctuelles ou invariantes, relatives à une exploration donnée; celles-ci concernent :

- la nature et le contenu des champs des messages,
- l'arborescence construite par une exploration sur le graphe G associé au réseau.

4.2.1.1. *Les messages de type* explorer *reçus par un processus*

(M 1) : Le premier message de poids k éventuellement reçu par un processus $\mathscr{P}_i$ avec $i \neq k$, est nécessairement du type *explorer*.

(M 2) : $\mathscr{P}_k$ ne peut recevoir aucun message *explorer* de poids k.

(M 3) : $\forall i \neq k$, $\mathscr{P}_i$ reçoit au plus un message *explorer* de poids k.

Il résulte en particulier de ces propriétés, que, lors de la réception par $\mathscr{P}_i$
– d'un message *explorer* (k, z, s) on a nécessairement

$$pgvu_i \neq k \quad \text{(d'après M 2)}$$

– d'un message *rebrousser* (k, z, s) on a nécessairement

$$pgvu_i \geq k \quad \text{(d'après M 1)}$$

4.2.1.2. *Arborescence d'exploration*

Une exploration E_k, de poids k, établit un sous-graphe partiel de G, désigné par $A_k=(X_k, \Gamma_k)$, avec :

$X_k=\{\mathscr{P}_i \mid \mathscr{P}_i$ a été visité par $E_k\}$

$\Gamma_k=\{(\mathscr{P}_i, \mathscr{P}_j)\in X_k^2 \mid \mathscr{P}_i$ a envoyé à $\mathscr{P}_j$ un message *explorer* de poids $k\}$

Autrement dit, lors d'une étape de E_k, le processus courant et la ligne par laquelle le message courant lui est parvenu (orientée par le sens de circulation de ce message) sont ajoutés à A_k si et seulement si, d'une part, le message courant est du type *explorer* et, d'autre part, le processus courant n'éteint pas l'exploration.

Au graphe A_k est associé l'invariant suivant :

(A 1) : A chaque étape de l'exploration de poids k, A_k définit une arborescence de racine $\mathscr{P}_k$ dans laquelle, pour tout $\mathscr{P}_i$ vérifiant $pgvu_i=k$:

- $pred_i$ désigne le père de $\mathscr{P}_i$,
- $succ_i$ désigne l'ensemble des fils de $\mathscr{P}_i$.

Remarque : Pour les processus $\mathscr{P}_i\in X_k$ qui ont ensuite été atteints par une exploration de poids plus élevé (on a alors $pgvu_i>k$) les liens dans l'arborescence ne sont plus mémorisés; seules les liens relatifs à l'arborescence de poids $pgvu_i$ sont mémorisés par $\mathscr{P}_i$.

Pour tout $\mathscr{P}_j\in X_k$, nous désignerons par γ_{kj} le chemin unique de A_k, reliant la racine $\mathscr{P}_k$ à $\mathscr{P}_j$.

4.2.1.3. *Invariants relatifs à l'information de contrôle*

Pour tout message de poids k (*explorer* (k, z, s) ou *rebrousser* (k, z, s)) émis de $\mathscr{P}_i$ vers $\mathscr{P}_j$, les champs z et s vérifient :

(P1) : z est l'ensemble des identités des processus ayant reçu un message *explorer* de poids k et n'ayant pas éteint l'exploration lorsque celle-ci les atteignit.

(P2) : $k=maximum\ (z)$.

(P3) : s est égal à l'ensemble des identités des processus non atteints par E_k, voisins des processus situés sur γ_{kj}.

(P4) : Si $l\in z$ et si $\mathscr{P}_l$ possède des voisins non atteints, alors $\mathscr{P}_l$ est situé sur γ_{kj}.

Remarque : Il en résulte de (P3) et (P4) que s a bien la propriété annoncée au 4.1.2 (i) : c'est l'ensemble des processus non atteints, voisins des processus déjà visités par E_k.

Pour ne pas alourdir l'exposé, la démonstration de ces propriétés n'est pas présentée ici; le lecteur intéressé se reportera à [16].

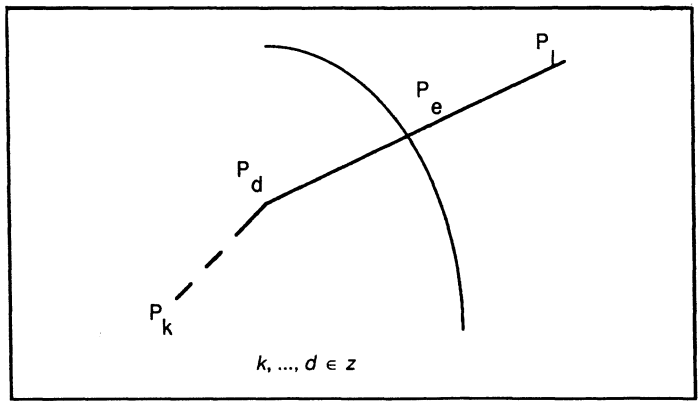
4.2.2. *Arrêt d'une exploration*

4.2.2.1. *Proposition* 1

Une exploration atteint l'étape de conclusion si et seulement si tous les processus du réseau sont visités par cette exploration.

Démonstration : supposons que E_k atteigne l'étape de conclusion lors de la réception du message μ par $\mathscr{P}_i$; les contextes de μ et $\mathscr{P}_i$ vérifient alors :

$voisins_i \subseteq z$ et $s = \varnothing$, et μ est du type *explorer* (ligne $a2$ de l'algorithme). Supposons qu'il existe un processus $\mathscr{P}_l$ tel que $l \notin z$. D'après la connexité du réseau, il existe dans G une chaîne de $\mathscr{P}_k$ à $\mathscr{P}_l$; comme $k \in z$, cette chaîne passe par une arête sortant de z, soit (d, e) cette arête avec $d \in z$ et $e \notin z$:



$\mathscr{P}_d$ possède donc un voisin non atteint $\mathscr{P}_e$. D'après (P3) : $\mathscr{P}_d$ n'est pas sur le chemin γ_{ki} puisque $s = \varnothing$ et $voisins_i \subseteq z$ (les voisins de $\mathscr{P}_i$ sont atteints). La propriété (P4) est donc contredite par $\mathscr{P}_d$ qui ne peut à la fois être et ne pas être sur γ_{ki}.

Q.E.D.

4.2.2.2. *Proposition* 2

Une exploration de poids k, si elle est lancée, est nécessairement stoppée au bout d'un nombre fini d'étapes :

– par une étape d'extinction si $k < max$,

– par l'étape de conclusion si $k = max$. A_{max} est alors une arborescence recouvrant le graphe G.

Démonstration : Lors d'une exploration E_k, l'étape courante (réception de μ par $\mathscr{P}_i$) peut être du type :

(i) extinction : si $k<max$, la proposition est démontrée. Si $k=max$, une telle étape ne sera pas rencontrée car, $\forall \mathscr{P}_i$ on a $pgvu_i \leqslant max$ ($pgvu_i$ est le maximum des poids des explorations ayant visité $\mathscr{P}_i$).

(ii) conclusion : Si $k<max$, une telle étape ne sera pas rencontrée, car d'après la proposition 1, cela signifierait que tous les processus sont visités par l'exploration. Or, il existe au moins un processus $\mathscr{P}_j$ avec $j>k$ qui, lorsqu'il est atteint, vérifie $pgvu_j \geqslant j>k$ et donc, éteint E_k. Si $k=max$, la proposition est démontrée.

(iii) génération : $\mathscr{P}_i$ envoie un message *explorer* à l'un de ses voisins, non encore atteint par E_k (propriété M2 et M3). Ce voisin :

– soit éteint E_k (ce qui implique $k<max$) et alors la proposition est démontrée,

– soit entre dans X_k, et donc le nombre de sommets visités par E_k augmente de un.

(iv) renvoi : tous les voisins de $\mathscr{P}_i$ ont été visités, mais $s \neq \varnothing$. E_k engendre alors une séquence de messages *rebrousser*, provoquant une remontée sur la branche γ_{ki}. Cette branche étant de longueur finie, la remontée est stoppée au bout d'un nombre fini d'étapes, soit :

– par la rencontre d'un processus $\mathscr{P}_j$ pour lequel $pgvu_j>k$, ce qui implique $k<max$ (depuis qu'il a été visité par l'exploration E_k, $\mathscr{P}_j$ a été visité par une exploration de poids supérieur à k), la propriété est alors démontrée.

– par la rencontre de $\mathscr{P}_l$ tel que $voisins_l \cap s \neq \varnothing$ (ligne a 3) : d'après (P3), ceci se produira nécessairement, au plus tard à la racine $\mathscr{P}_k$ de l'exploration E_k (sauf si E_k est éteinte). L'exploration est alors relancée par une étape de génération (ligne a 4).

Ainsi, tant que l'exploration E_k n'est pas stoppée, le nombre de sommets qu'elle visite croît strictement, et un nombre fini d'étapes séparent deux étapes de génération. Comme par ailleurs : $k=maximum\ (z)$ (propriété P2) il en résulte que :

• si $k<max$, un sommet $\mathscr{P}_j$ tel que $pgvu_j \geqslant j>k$ sera atteint au bout d'un nombre fini d'étapes (extinction),

• si $k = max$, tous les sommets seront visités au bout d'un nombre fini d'étapes (conclusion).

Q.E.D.

4.2.3. *Terminaison et correction partielle*

4.2.3.1. *Théorème* 1 : *Terminaison*

L'algorithme se termine au bout d'un temps fini.

Démonstration : (i) D'après les hypothèses sur le réseau, l'intervalle de temps séparant deux étapes d'une exploration donnée est fini (acheminement d'un message sur une ligne en un temps fini).

(ii) D'autre part, l'exploration E_{max} est lancée au bout d'un nombre fini d'étapes. En effet, par hypothèse, au moins un processus $\mathscr{P}_k$ lance une exploration de poids k. Si $k = max$, ce résultat est démontré. Sinon, d'après la proposition 2, l'exploration E_k sera éteinte au bout d'un nombre fini d'étapes, par un processus $\mathscr{P}_j$ tel que $pgvu_j = k_1 > k$. Si $k_1 \neq j$ l'exploration E_{k_1} a déjà été lancée, si $k_l = j$ elle est lancée à l'extinction de E_k. L'exploration de poids k_1 est donc lancée au plus tard à l'extinction de E_k. Réitérant ce raisonnement, on construit une suite strictement croissante $k < k_1 < k_2 < \ldots$ qui converge donc vers max en un nombre fini d'étapes.

(iii) D'après le point précédent (ii), l'exploration E_{max} est lancée après un nombre fini d'étapes et d'après la proposition 2 elle s'arrête en un nombre fini d'étapes.

(iv) Enfin, l'étape de conclusion est atteinte par E_{max} (propositions 1 et 2) : les messages *conclure* parcourent alors l'arborescence A_{max} (construite par E_{max}), chacune de ses arêtes étant visitée une et une seule fois (ligne *a* 5).

Les 4 points démontrent donc la terminaison de l'algorithme.

Q.E.D.

4.2.3.2. *Théorème* 2 : *Correction partielle*

A la fin de l'algorithme, tous les processus sont informés de la terminaison et connaissent :

– l'identité du processus élu, égale à max la plus grande identité; elle est mémorisée dans $pgvu_i$, $\forall \mathscr{P}_i$.

– le routage permettant d'y accéder, mémorisé dans $pred_i$, $\forall \mathscr{P}_i$.

Démonstration : Tous les processus ayant été visités par l'exploration de poids max, on a $\forall \mathscr{P}_i$: $pgvu_i = max$ (ligne *a* 1 et propositions 1 et 2).

D'autre part, cette exploration a établi l'arborescence A_{max} de racine $\mathscr{P}_{max}$ (proposition 2) telle que $pred_i$ est le père de $\mathscr{P}_i$ dans cette arborescence (invariant A 1); de plus la réception par $\mathscr{P}_i$ d'un message *conclure* lui signale la terminaison de l'algorithme (ligne a 5).

Q.E.D.

4.3. Analyse de complexité

Une exploration de poids k (issue de $\mathscr{P}_k$) est stoppée (proposition 2) :

— si $k < max$, au plus tard lorsqu'un processus $\mathscr{P}_l$, tel que $l > k$, est atteint par un message *explorer* de poids k,

— si $k = max$, lorsque tous les processus sont visités.

De plus, $\mathscr{P}_k$ envoie le premier message *explorer* vers son voisin de plus fort poids (*cf.* la procédure lancer-exploration). Il en résulte que :

— Si $k < max$ et $\mathscr{P}_k$ n'est pas un maximum local, alors E_k engendre un message *explorer* et aucun message *rebrousser*.

— Si $k < max$ et $\mathscr{P}_k$ est un maximum local alors E_k engendre au plus k' messages *explorer*, et donc au plus k' messages *rebrousser* (puisqu'un message *rebrousser* de poids k ne peut parcourir une ligne que si un message *explorer* a parcouru la même ligne en sens inverse), k' étant la position de $\mathscr{P}_k$ dans l'ensemble ordonné des identités ($k' \in 1 \ldots n$).

— Si $k = max$, alors E_{max} engendre exactement $n-1$ messages *explorer* et au plus $n-1$ messages *rebrousser*.

Chaque exploration est lancée au plus une fois (exactement une fois pour $\mathscr{P}_{max}$).

Enfin le nombre de messages *conclure* est égal exactement à $n-1$.

On obtient donc les résultats suivants.

4.3.1. *Cas le plus favorable*

Ce cas est obtenu lorsque $\mathscr{P}_{max}$ est seul à lancer une exploration; on a alors;

- $n-1$ messages *explorer*,
- 0 à $n-1$ messages *rebrousser* (selon la topologie du réseau; la valeur minimale 0 est obtenue lorsqu'il existe une chaîne dans le réseau incluant tous les processus dans l'ordre décroissant de leurs identités. C'est le cas par exemple d'un anneau, d'un réseau complet et d'un réseau linéaire lorsque

$\mathscr{P}_{max}$ est en bout de ligne; la valeur maximale $n-1$ est obtenue dans le cas d'un réseau en étoile autour de $\mathscr{P}_{max}$).

- $n-1$ messages *conclure.*

4.3.2. *Cas le plus défavorable*

Dans ce cas on a :

$$\text{au plus } \left(\sum_{k'=1}^{n-1} k'\right)+(n-1) \text{ messages du type } \textit{explorer},$$

autant de messages *rebrousser* et $n-1$ messages *conclure* soit :

$$2\left((\sum_{k'=1}^{n-1} k')+(n-1)\right)+n-1=(n-1)(n+3) \text{ messages, tous types confondus.}$$

La complexité est donc $O(n^2)$ dans le pire des cas.

5. ALGORITHME 2 : EXPLORATION PARALLÈLE

5.1. Arbre de contrôle

Dans cet algorithme l'exploration issue d'un processus $\mathscr{P}_k$ est lancée en parallèle vers tous ses voisins. Comme précédemment un processus ne laisse progresser vers ses voisins la composante de l'exploration qui l'atteint que si le poids k de celle-ci est supérieur au poids de la dernière exploration qu'il a pris en compte. Cette condition assure que seule l'exploration lancée par $\mathscr{P}_{max}$ pourra être prise en compte par tous les processus. L'arborescence recouvrante de racine $\mathscr{P}_{max}$ est construite par les messages relatifs à l'exploration correspondante.

Pour obtenir une arborescence recouvrante, il est nécessaire que tout processus autre que $\mathscr{P}_{max}$ ait un seul prédécesseur et que deux processus distincts aient des ensembles de successeurs disjoints. Ceci n'est pas implicitement réalisé par la circulation des messages relatifs à une exploration donnée. Si comme précédemment la prise en compte par $\mathscr{P}_i$ d'un message d'exploration peut permettre de définir son prédécesseur, il est nécessaire, pour établir ses successeurs, d'introduire un contrôle adéquat. Plusieurs protocoles permettent de résoudre ce problème [28]. Tout processus $\mathscr{P}_i$, lorsqu'il reçoit un message

relatif à une exploration qu'il prend en compte pour la première fois, mémorise l'émetteur $\mathscr{P}_j$ du message comme étant son prédécesseur, propage l'exploration vers ses voisins, et attend d'avoir reçu des acquittements de tous ceux-ci, avant d'émettre l'acquittement vers $\mathscr{P}_j$; dans le cas où il reçoit un message relatif à l'exploration déjà prise en compte, il l'acquitte immédiatement en indiquant à $\mathscr{P}_j$ de ne pas l'inclure dans ses successeurs.

Cette technique est abondamment utilisée dans de nombreux algorithmes où il s'avère nécessaire de construire une arborescence pour contrôler un calcul; citons à titre d'exemple : la terminaison distribuée [7, 9, 22], la détection de l'interblocage [4, 15], et le calcul distribué sur les graphes [3, 21].

5.2. L'algorithme

5.2.1. *Les messages utilisés*

Les message du type *rebrousser* sont maintenant inutiles à cause du parallélisme de l'exploration; il en est de même du champs s du message *explorer.* Les messages de ce type présentent la forme suivante :

$$explorer\ (k,\ z)$$

où k est le poids du processus qui a lancé l'exploration correspondante et où z désigne l'ensemble des processus dont on est sûr qu'ils ont été ou seront visités par des messages relatifs à cette exploration. Le fait de placer (selon la technique proposée dans [15]) dans z non seulement les processus visités mais aussi des informations sur le voisinage du processus émetteur va permettre de diminuer le nombre de messages échangés.

Les acquittements relatifs aux messages d'exploration sont véhiculés par les messages du type *acquitter* qui présentent deux champs :

$$acquitter\ (k,\ x)$$

Dans un tel message émis par $\mathscr{P}_i$: k désigne l'exploration considérée et x permet au récepteur de savoir s'il doit considérer $\mathscr{P}_i$ comme un de ses successeurs dans l'arborescence ($x = terminé$) ou non ($x = déjàvu$).

La réception par $\mathscr{P}_{max}$ de tous les acquittements en provenance de ses voisins indique la terminaison de l'algorithme.

5.2.2. *Contexte d'un processus*

La constante $voisins_i$ et les variables $état_i$, $pgvu_i$, $pred_i$ et $succ_i$ ont la même signification qu'au paragraphe 4.1.3.

La variable **var** $nbacq_i$ (**entier non négatif initialisé à** 0) permet à $\mathscr{P}_i$ de mémoriser le nombre d'acquittements attendus et de contrôler l'envoi de l'acquittement vers son prédécesseur dans l'arborescence construite.

5.2.3. *Comportement de* $\mathscr{P}_i$

procédure lancer-exploration;
début
  $état_i$: = *candidat;*
  $pred_i$: = *nil;*
  $succ_i$: = $voisins_i$;
  $nbacq_i$: = cardinal ($succ_i$);
  $\forall x \in succ_i$: **envoyer** *explorer* (*i*, $voisins_i$) **à** $\mathscr{P}_x$
fin

lors de décision de lancer une élection
faire
  si $état_i$ = *initial* **alors** lancer-exploration **fsi**
fait
lors de réception de *explorer* (*k*, *z*) **depuis** $\mathscr{P}_j$
faire
  cas
    $pgvu_i > k \rightarrow$ **si** $état_i$ = *initial* **alors** lancer-exploration **fsi**
    $pgvu_i = k \rightarrow$ **envoyer** *acquitter* (*k*, *déjàvu*) **à** $\mathscr{P}_j$
    $pgvu_i < k \rightarrow état_i$: = *battu;*
                $pgvu_i$: = *k;*
                $pred_i$: = *j;*
                $succ_i$: = $voisins_i - z$;
                cas
                  $succ_i = \varnothing \rightarrow$ **envoyer** *acquitter* (*k*, *terminé*) **à** $\mathscr{P}_j$
                  $succ_i \neq \varnothing \rightarrow nbacq_i$: = cardinal ($succ_i$);
                    $\forall x \in succ_i$: **envoyer** *explorer* (k, $z \cup succ_i$) **à** $\mathscr{P}_x$
                fcas
  fcas
fait
lors de réception de *acquitter* (*k*, *x*) **depuis** $\mathscr{P}_j$
faire
  si $pgvu_i = k$ **alors**
      si x = *déjàvu* **alors** $succ_i$: = $succ_i - \{j\}$ **fsi;**
      $nbacq_i$: = $nbacq_i - 1$;
    si $nbacq_i = 0$ **alors**
    cas $k = i \rightarrow état_i$: = *élu*
      $k \neq i \rightarrow$ **envoyer** *acquitter* (*k*, *terminé*) **à** $\mathscr{P}_{pred_i}$
    fcas
  fsi fsi
fait

6. CONCLUSION

Les deux algorithmes présentés s'appuient sur un principe général qu'il est important de préciser : l'extinction sélective de messages. Dans notre cas une exploration lancée par un processus sera ou non stoppée en fonction de son poids et de celui que mémorise le processus atteint. Ce principe, bien qu'il ne

soit pas toujours explicitement formulé, est souvent rencontré dans de nombreux algorithmes distribués de contrôle. Une de ses premières utilisations, en distribué, est dûe à Thomas [29] qui l'a introduit pour régler les problèmes posés par les accès concurrents aux copies multiples d'un même ensemble de données. Il est souvent utilisé dans la résolution de problèmes où l'arrivée d'un message doté d'un attribut tel que sa date d'émission (mis en œuvre par une estampille ou un numéro de séquence) rend caduques les messages qui, émis avant lui (et donc dotés d'une estampille inférieure), arriveront après lui (citons à titre d'exemples : [4] en ce qui concerne la détection de l'interblocage, [27] pour la diffusion fiable et [17] pour l'exclusion mutuelle dans un réseau quelconque).

Les deux algorithmes proposés constituent deux mises en œuvre de ce principe général, qui généralisent au contexte distribué les stratégies de recherche séquentielle que sont la « profondeur d'abord » et la « largeur d'abord ». D'autres algorithmes distribués réalisent de telles traversées de réseaux mais supposent que, seul un des processus, qui sera la racine de l'arborescence construite, a l'initiative du lancement [1, 6]. Les algorithmes qui sont ici proposés (dont la complexité en nombre de messages, dans le cas de l'exploration en profondeur, est identique à ceux-ci) présentent un double avantage. Tout d'abord les hypothèses faites sont plus faibles : aucun processus n'a besoin de connaître la structure du réseau, ni même sa taille. Ensuite l'un quelconque des processus (ou plusieurs d'entre eux) peut lancer l'algorithme; l'arborescence construite n'est pas quelconque : c'est le processus de poids maximum qui est particularisé et c'est l'arborescence correspondante qui est construite.

De plus si l'on considère (dans l'algorithme du paragraphe 4) une seule exploration en profondeur du réseau issue d'un processus prédéterminé $\mathscr{P}_{init}$, l'algorithme de parcours obtenu s'avère meilleur que le meilleur algorithme distribué connu réalisant le même type de parcours [1]; ce dernier présente en effet des complexités de $\mathcal{O}(e)$ en nombres de messages (où e est le nombre de lignes du réseau) et $\mathcal{O}(n\Delta)$ en temps (où Δ est le temps de transfert maximum d'un message sur une ligne), alors que celui déduit du paragraphe 4 ne requiert que $\mathcal{O}(n)$ messages et la même complexité temporelle.

Outre ces résultats, sur le plan de l'algorithmique distribuée les algorithmes proposés utilisent une technique de contrôle originale mise en œuvre par les champs z et s des messages de type *explorer*. Ces champs véhiculent une information de contrôle permettant de réduire le nombre de messages échangés d'une part en évitant d'effectuer des parcours totalement aveugles (utilisation de z et s dans le premier algorithme et de z dans le second) et d'autre

part en permettant de détecter la fin des parcours « au plus tôt » (utilisation de s dans le premier algorithme). (Il est en outre intéressant de noter que, en ce qui concerne la technique d'exploration en profondeur, l'information de contrôle s joue dans le contexte distribué le même rôle que la pile dans le contexte séquentiel).

Les informations de contrôle introduites sont d'un emploi général. Leur utilisation est illustrée ici, avec le calcul distribué d'un extrémum (en ce qui concerne le contrôle des systèmes, elles ont été appliquées dans [15] à la détection de l'interblocage).

Les techniques développées constituent un premier pas vers des modes de parcours « intelligents » de réseaux de processus et peuvent, à ce titre, être utilisées pour la résolution distribuée de certains problèmes d'optimisation combinatoire. ([13] présente une autre approche distribuée de ces problèmes.)

Signalons enfin que la conception de ces algorithmes a été facilitée par l'emploi d'un outil adéquat : le simulateur pour la validation de protocoles V.E.D.A. développé au C.N.E.T. de Lannion [18].

REMERCIEMENTS

Les auteurs remercient les lecteurs pour leurs critiques constructives et plus particulièrement l'un deux qui d'une part leur a permis de mieux préciser le contexte dans lequel évoluent les algorithmes proposés (notamment en ce qui concerne les hypothèses de connaissance initiale des processus) et d'autre part leur a signalé le caractère « novateur » des travaux de Thomas en ce qui concerne l'utilisation du principe d'extinction sélective en réparti.

BIBLIOGRAPHIE

1. B. Awerbuch, *A New Distributed Depth-First Search Algorithm*, Inf. Proc. Letters, vol. 20, avril 1985, p. 147-150.
2. P. A. Bernstein et M. Goodman, *Concurrency Control in Distributed Data Base Systems*, A.C.M., Computing Surveys, vol. 13, n° 2, juin 1981, p. 185-201.
3. K. M. Chandy et J. Misra, *Distributed Computing on Graphs: Shortest Paths Algorithms*, Comm. A.C.M., vol. 25, n° 11, novembre 1982, p. 833-837.
4. K. M. Chandy, J. Misra et J. Haas, *Distributed Deadlock Detection*, A.C.M. T.O.C.S., vol. 1, n° 2, mai 1983, p. 144-156.
5. E. J. Chang et R. Roberts, *An Improved Algorithm for Decentralized Extrema-Finding in Circular Configurations of Processors*, Comm. A.C.M. vol. 22, n° 5, mai 1979, p. 281-283.
6. T. Cheung, *Graph Traversal Techniques and the Maximum Flow Problem in Distributed Computation*, I.E.E.E. Trans. on soft. Eng., vol. SE9, n° 4, juillet 1983, p. 504-512.

7. E. W. DIJKSTRA et C. S. SHOLTEN, *Terminaison Detection for Diffusing Computations*, Inf. Proc. Letters, vol. 11, n° 1, août 1980, p. 1-4.
8. D. DOLEV, M. KLAWE et M. RODEH, *An O (n log n) Unidirectional Distributed Algorithm for Extrema Finding in a Circle*, Journal of Algorithms, vol. 3, 1982, p. 245-260.
9. N. FRANCEZ et M. RODEH, *Achieving Distributed Terminaison Without Freezing*, I.E.E.E. Trans. on Soft. Eng., vol. SE 8, n° 3, mai 1982, p. 287-292.
10. W. R. FRANKLIN, *On an Improved Algorithm for Decentralized Extrema-Finding in Circular Configurations of Processors*, Comm. A.C.M. vol. 25, n° 5, mai 1982, p. 336-337.
11. H. GARCIA-MOLINA, *Elections in a Distributed Computing System*, I.E.E.E. Trans. on Computers, vol. C31, n° 1, janvier 1981, p. 48-59.
12. J. N. GRAY, *Notes on Data Base Operating Systems*, L.N.C.S., n° 68, Springer-Verlag, 1978, p. 393-481.
13. T. HERMAN et K. M. CHANDY, *On Distributed Search*, Inf. Processing Letters, vol. 21, 1985, p. 129-133.
14. D. S. HIRSCHBERG et J. B. SINCLAIR, *Decentralized Extrema Finding in Circular Configurations of Processors*, Comm. A.C.M., vol. 23, n° 11, novembre 1980, p. 627-628.
15. J. M. HELARY, A. MADDI et M. RAYNAL, *Controlling Knowledge Transfers in Distributed Algorithms: Application to Deadlock Detection*, Rapport de recherche I.N.R.I.A., n° 493, mars 1986, 28 p.
16. J. M. HELARY, A. MADDI et M. RAYNAL, *Calcul distribué d'un extrêmum et du routage associé dans un réseau quelconque*, Rapport de recherche I.N.R.I.A., n° 516, avril 1986, 36 p. A paraître dans Computer journal 1988.
17. J. M. HELARY, N. PLOUZEAU et M. RAYNAL, *A Distributed Algorithm for Mutual Exclusion in an Arbitrary Network*, Rapport de recherche I.N.R.I.A. n° 496, mars 1986, 15 p.
18. C. JARD, J. F. MONIN et R. GROZ, *VEDA: a Software Simulator for the Validation of Protocol Specifications*, C.O.M.N.E.T., 1985, Hongrie, octobre 1985.
19. E. KORACH, S. MORAN et S. ZAKS, *Tight Lower and Upper Bounds for Some Distributed Algorithms for a Complete Network of Processors*, Proc. of the 3rd A.C.M. conf. on principles of distributed computing, août 1984, p. 199-207.
20. G. LE LANN, *Distributed Systems: Towards a Formal Approach*, I.F.I.P. Congres, Toronto, août 1977, p. 155-160.
21. J. MISRA et K. M. CHANDY, *A Distributed Graph Algorithm: Knot Detection*, A.C.M. T.O.P.L.A.S., vol. 4, n° 4, octobre 1982, p. 678-680.
22. J. MISRA et K. M. CHANDY, *Terminaison Detecting of Diffusing Computations in C.S.P.*, A.C.M. T.O.P.L.A.S., vol. 4, n° 1, janvier 1982, p. 37-43.
23. J. A. PACHL, E. KORACH et D. ROTEM, *Lower Bounds for Distributed Maximum Finding Algorithms*, Journal of the A.C.M., vol. 31, n° 4, octobre 1984, p. 905-918.
24. G. L. PETERSON, *An O (n log n) Unidirectional Algorithm for the Circular Extrema Problem*, A.C.M. T.O.P.L.A.S., vol. 4, n° 4, octobre 1982, p. 758-762.
25. J. PETERSON et A. SILBERSCHATZ, *Operating System Concepts*, Addison Wesley, 1983, 548 p.
26. M. RAYNAL, *Algorithmes distribués et protocoles*, Eyrolles, septembre 1985, 144 p.
27. F. D. SCHNEIDER, D. GRIES et R. SCHLICHTING, *Fault Tolerant Broadcasts*, Science of Programming, vol. 4, n° 1, 1984, p. 1-15.

28. A. SEGALL, *Distributed Network Protocols*, I.E.E.E. Trans. on Inf. Theory, vol. IT29, 1, janvier 1983, p. 23-35.

29. R. H. THOMAS *A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases*, A.C.M. Trans. on Database Systems, vol. 4, n° 2, juin 1979, p. 180-209.