Un protocole distribué pour la 2-connexité

Jean-Claude Bermond, Jean-Claude König

Un protocole distribué pour la 2-connexité (*)

Jean-Claude BERMOND, Jean-Claude KÖNIG

PRÉSENTATION *Une classe importante de problèmes d'algorithmique distribuée consiste à trouver une information sur la topologie du graphe de connexion d'un réseau de processeurs. Les auteurs ont proposé pour ce type de problèmes un protocole général qui permet de calculer une fonction dont les arguments sont distribués sur les nœuds du réseau. Ils montrent dans ce texte comment appliquer ce protocole au problème de la détermination des points d'articulation et des composantes 2-connexes.*

Michel Cosnard

1. Introduction

Nous nous plaçons ici dans les hypothèses classiques de l'algorithmique distribuée (voir livres de Raynal [Raynal 88 et Raynal 85]). Les processeurs (ou nœuds) sont connectés selon un réseau quelconque supposé connexe. Ils communiquent entre eux par échange de messages sur des canaux ou liens supposés bidirectionnels. Les messages sont reçus sans erreur, ni duplication, ni perte (mais pas forcément dans l'ordre d'émission) au bout d'un temps fini. Le système est supposé asynchrone et il n'y a aucune ressource globale (pas d'horloge commune, pas de mémoire partagée, pas de contrôleur central, ...).

Le réseau est supposé non orienté et connexe et les nœuds ne connaissent que leur identité, les canaux (bidirectionnels) qui leur sont incidents et leurs données initiales. Ils n'ont aucune autre connaissance sur la topologie du réseau. Par contre on suppose dans toute la suite que les processeurs ont des identités toutes différentes.

Rappelons ici quelques définitions de théorie des graphes utiles pour la suite. La distance $d(x, y)$ entre 2 nœuds x et y est la longueur d'une plus courte chaîne entre x et y. L'excentricité d'un sommet x, $ecc(x)$, est le maximum des distances entre lui et tous les autres nœuds du réseau : $ecc(x) = \max_y d(x, y)$. Le diamètre D du réseau est le maximum des excentricités des nœuds c'est-à-dire le maximum des distances entre toutes paires de sommets :

$$D = \max_x ecc(x) = \max_{(x,y)} d(x, y) .$$

Un réseau est connexe s'il existe une chaîne entre toute paire de sommets. Dans toute la suite le réseau sera supposé connexe et dans ce cas $ecc(x)$ et D sont finis. Un réseau est dit 2-connexe s'il est connexe et s'il le reste après suppression de tout nœud. Un nœud est dit point d'articulation si sa suppression disconnecte le réseau. Un réseau connexe est donc 2-connexe s'il ne contient pas de point d'articulation. Une définition équivalente (théorème de Menger) est : un réseau est 2-connexe si entre toute paire de sommets x et y il existe 2 chaînes sommets-disjointes entre x et y. Une composante 2-connexe est un sous-réseau 2-connexe maximal.

Notons le fait que les composantes 2-connexes forment une partition des arêtes du réseau. En effet la relation « appartenir à une même composante 2-connexe » est une relation d'équivalence sur les arêtes. Rappelons que cette relation est équivalente à la relation « appartenir à un même cycle élémentaire ».

Pour l'analyse des complexités (mais ceci n'est pas nécessaire pour l'algorithme) nous noterons par :

- n le nombre de nœuds (processeurs ou sommets) du réseau ;
- m le nombre de canaux (liens ou arêtes) birectionnels ;
- D le diamètre du réseau ;

(*) Ce travail a été soutenu par le GRECO-PRC C^3.

• τ le temps maximum de transmission d'un message sur un lien.

Une importante classe d'algorithmes distribués consiste à trouver une information sur la topologie du réseau. Par exemple, chaque nœud veut connaître le diamètre du réseau ou simplement sa propre excentricité (dans ce cas le résultat est différent en chaque nœud). Chaque nœud peut aussi vouloir connaître une table des routages de plus courts chemins pour savoir quel canal utiliser pour envoyer un message à un nœud quelconque du réseau. Ces applications et d'autres sont développées dans [Bermond 87 et Bermond 89].

Elles y sont résolues par un protocole général dit « protocole de consensus ». Un tel protocole permet de calculer une fonction ou un prédicat dont les arguments sont distribués sur les nœuds d'un réseau. Essentiellement le protocole garantit que chaque nœud ait à la fin de l'exécution la trace de l'information initiale de chaque autre nœud nécessaire pour calculer le résultat. Le protocole présenté dans [Bermond 87] nécessite au plus $2(D+2)\,m$ messages et a une durée maximum de $2(D+1)\,\tau$.

Le problème de déterminer les points d'articulation et les composantes 2-connexes d'un réseau a reçu beaucoup d'attention récemment. Il a été explicitement proposé lors de la « problem session » du 2nd workshop sur les algorithmes distribués d'Amsterdam (1987). En effet la maintenance d'un système résistant aux pannes nécessite de faire attention qu'une simple panne ne mette pas en péril le réseau, c'est-à-dire ne le disconnecte pas. Pour cela il est important de savoir s'il existe des points d'articulation. Si l'on veut éviter de tels nœuds il faut joindre entre elles les composantes 2-connexes d'où la nécessité de déterminer ces composantes.

Dans l'article [Bermond 89] nous indiquions très succinctement comment appliquer notre protocole général à ce calcul. Le but de cet article est d'expliciter l'algorithme pour le problème particulier, de prouver effectivement sa validité et le comparer à divers algorithmes récemment parus.

Dans la section 2, nous rappelons les principes généraux du protocole et nous indiquons comment on peut l'adapter à toute application en gardant constant le nombre de messages et la durée. Mais une telle adaptation directe peut entraîner une redondance à la fois dans l'information transmise et dans le calcul local des nœuds.

Dans la section 3 nous donnons une version de l'algorithme adaptée au test de la 2-connexité qui permet de minimiser les paramètres ci-dessus (quantité d'information transmise et calcul local des nœuds). Essentiellement le protocole détermine localement si 2 liens ou arêtes appartiennent ou non à la même composante 2 connexe.

Dans la section 4 nous donnons la preuve de la validité de l'algorithme en utilisant des propriétés fines de 2-connexité. Enfin en section 5 nous redonnons les différentes complexités de notre algorithme et le comparons aux autres algorithmes récemment trouvés [Huang 88, Hohberg 90, Lin 90, Park 88]. Notre algorithme a des complexités voisines de ces algorithmes, meilleure pour la complexité en temps, analogue pour la qualité d'information et, suivant les cas, meilleure ou moins bonne pour la complexité en messages. Il présente par contre plusieurs avantages sur les autres algorithmes. D'abord il est plus simple. Ensuite il ne s'appuie pas sur la construction séquentielle qui utilise la construction d'un arbre en profondeur d'abord. Enfin il est réellement distribué car il ne suppose pas l'existence d'un nœud centralisateur que celui-ci soit donné au départ ou élu ou apparaisse comme la racine d'un arbre DFS. En effet, ce nœud permet la construction d'un arbre couvrant sur lequel une grande partie de l'information peut être transmise sans redondance (existence d'une chaîne unique entre chaque nœud et la racine) ; cette technique possède comme principal défaut, en plus du calcul préliminaire de ce nœud, une surcharge du nœud centralisateur qui gère à la fois les calculs de l'application et le contrôle associé.

2. Principes généraux

Nous donnons ici les idées générales du protocole décrit dans [Bermond 87 et Bermond 89]. La version détaillée est donnée en section 3 pour l'application à la 2-connexité. L'algorithme est divisé en 3 parties:

— initialisation (INIT) ;
— plusieurs phases (PHASE*) ;
— terminaison (TERM).

L'utilisation des phases induit une synchronisation logique. L'algorithme est donc virtuellement synchronisé par les événements réception de messages. En effet durant une phase donnée un nœud fait successivement trois actions :

— envoi d'un message sur tous les liens utiles ;
— réception d'un message sur tous les liens utiles ;
— filtrage de l'information reçue pour ne renvoyer à la phase suivante que ce qui est nouveau, et éventuellement calcul local.

L'algorithme est le même pour tous les nœuds. Il faut bien noter que l'on entame la phase $p+1$ que lorsque la phase p est finie.

Un message contient le numéro de la phase à laquelle il a été émis et la nouvelle information apprise par le processeur émetteur à la phase précédente (ou « fin » s'il n'a rien appris de nouveau). Pour garantir la terminaison la nouvelle information doit contenir au moins les identités des processeurs d'où vient cette nouvelle information. Par exemple dans une élection le maximum courant peut ne pas changer pendant plusieurs phases sans que l'exécution de l'algorithme soit finie.

Enfin, à la phase p, le processeur ne lit que les messages étiquetés avec p (les autres messages sont conservés dans une file d'attente associée au lien d'arrivée du message).

Avec ce protocole, à la fin de la phase p un nœud connaît l'information nécessaire (en particulier les identités) de tous les nœuds qui sont à distance au plus p de lui. Le protocole s'arrête lorsqu'un processeur n'apprend plus rien de nouveau, c'est-à-dire à la phase $ecc(P)+1$ où $ecc(P)$ dénote l'excentricité de P. Plus exactement, pour éviter qu'un processeur attende des messages d'un voisin qui en fait a fini, à la phase $ecc(P)+2$ le processeur doit signaler à ses voisins qu'il a fini à l'aide d'un message spécial de type « fin ». Quand un nœud reçoit un message de type « fin » d'un nœud voisin il supprime sa liaison avec ce voisin.

Dans [Bermond 87] et [Bermond 89], nous avons donné plusieurs applications de ce protocole : problème de choix de leader, table de routage... Pour certaines applications la stratégie à utiliser n'est pas évidente. Par exemple, si l'on désire calculer une information sur la topologie du réseau, on peut prendre une approche générale qui consiste à commencer par apprendre en chaque nœud toute la structure du réseau. Pour cela chaque nœud acquiert, lors d'une phase préliminaire, l'identité de ses voisins et transmet ensuite avec son identité les identités de ses voisins, et ceci se répète durant toutes les phases. Une fois la structure du réseau apprise en chaque nœud, il est aisé pour un nœud quelconque de déterminer la propriété ou le paramètre désiré. Mais la quantité d'information transmise est en $O(m^2)$ identificateurs soit $O(m^2 \log(n))$ bits. De plus chaque nœud effectue un calcul local important.

On peut dans certaines applications utiliser d'autres méthodes. Par exemple pour calculer le diamètre on peut utiliser 2 fois le protocole simple, une première fois pour calculer l'excentricité de chaque nœud (qui vaut $p-1$ si p est la phase où l'on n'apprend rien de nouveau) et une seconde fois en déterminant le maximum des excentricités. Dans ce cas la durée est le double mais la quantité d'information transmise est en $O(nm)$ identificateurs et les calculs locaux insignifiants par rapport à la première approche.

Dans le paragraphe suivant nous donnons un algorithme pour déterminer si un nœud est point d'articulation ou calculer les composantes 2-connexes de manière locale. Cet algorithme évite les inconvénients de l'approche générale, a la même complexité en temps et messages, mais la quantité d'information transmise est $O(nm \log(n))$ au lieu de $O(m^2 \log(n))$ bits et le calcul local est moindre et réellement local.

3. Algorithme pour la 2-connexité

Rappelons que le problème posé dans l'introduction est de déterminer s'il existe des points d'articulation et éventuellement de rendre le réseau 2-connexe en ajoutant le minimum d'arêtes. Pour ce faire il n'est pas nécessaire de déterminer complètement les composantes 2-connexes, mais il suffit de calculer en chaque nœud la projection de la relation d'équivalence « appartenir à la même composante 2-connexe » sur les arêtes incidentes à ce nœud. En effet s'il existe 2 arêtes incidentes au nœud P appartenant à 2 composantes connexes différentes ce nœud est un point d'articulation. Pour rendre le réseau 2-connexe il suffit en chaque point d'articulation de choisir une arête dans chaque classe d'équivalence et de connecter leurs extrémités par une chaîne ou un arbre.

L'idée de l'algorithme est la suivante. En un nœud P nous dirons que deux arêtes a et b, incidentes à ce nœud, sont en relation, aR b, si et seulement si il existe un nœud Q tel que P reçoit pour la première fois l'identité de Q via l'arête a lors de la phase p et P reçoit à nouveau l'identité de Q via l'arête b à la même phase p ou à la phase immédiatement suivante $p+1$. Nous verrons dans la section suivante que la fermeture transitive de la relation R est exactement la relation « appartenir à une même composante 2-connexe ».

Pour obtenir l'information nécessaire pour former la relation d'équivalence, il suffit en chaque nœud de disposer de deux tableaux, l'un $rout(Q)$ indiquant sur quel lien a été reçue pour la première fois l'identité de Q, et l'autre $dist(Q)$ indiquant à quelle phase a été reçue cette information. En fait $rout(Q)$ donne l'arête incidente à P sur une plus courte chaîne entre P et Q et $dist(Q)$ la distance entre P et Q.

L'algorithme est donné pour un nœud arbitraire P.

Variables :

— *Inf* est l'ensemble des identificateurs connus par P (au début de l'exécution de l'algorithme *Inf* contient l'identité de P).

— *Nouv* est l'ensemble des nouveaux identificateurs connus par P depuis le début de la phase courante.

— p est le numéro de la phase exécutée par P.

— $id(Q)$, $rout(Q)$, $dist(Q)$ sont respectivement l'identificateur du processeur Q, une arête d'une plus courte chaîne entre P et Q et la distance entre P et Q.

— ℓ indique un canal (ou voie de communication) incident à P.

— *Aux* est l'ensemble des voies de communication sur lesquelles P peut recevoir des messages (quand P reçoit un message « fin » sur une voie ℓ dans *Aux*, il enlève cette voie de *Aux*). *Aux* est initialisé à *voisins* qui contient l'ensemble des voies incidentes à P.

Rappelons qu'à la phase p le processeur P ne lit que les messages étiquetés avec p (les autres messages sont conservés dans une file d'attente associée au lien d'arrivée du message).

```
INIT
  Nouv=Inf=Id(P)
  p=0
  Aux=voisins
PHASES
  Tant que Nouv≠0/ faire
  début
    p=p+1
    Sur chaque voie ℓ de voisin envoyer <p, Nouv>
    Nouv=0/
```

```
    Pour chaque voie ℓ dans Aux faire
    début
      recevoir <p, I> sur le lien ℓ
      Si I="fin"
      alors Aux=Aux−{ℓ}
      sinon début
        Pour tout Id(Q) dans I faire
        début
        si id(Q)∉Inf
        alors début
          rout(Q)=ℓ
          dist(Q)=p
          fin
        sinon début
          si p≤(dist(Q)+1)
          alors "fusion des composantes de rout(Q) et ℓ"
          fin
        Nouv=Nouv∪(I−Inf)
        Inf=Inf∪I
        fin
    fin
  fin
TERM
  p=p+1
  Sur chaque voie de voisin envoyer <p," fin"'>
  Tant que Aux≠0' faire
  début
    Pour chaque voie ℓ dans Aux faire
    début
      recevoir <p, I> sur ℓ
      Si I="fin" alors Aux=Aux−{ℓ}
    fin
    p=p+1
  fin
```

4. Preuve de l'algorithme

Rappelons qu'en un nœud P deux arêtes a et b incidentes à ce nœud, sont en relation, $a\,\mathrm{R}\,b$, si et seulement s'il existe un nœud Q tel que P reçoit pour la première fois l'identité de Q via l'arête a lors de la phase p et P reçoit à nouveau l'identité de Q via l'arête b à la même phase p ou à la phase immédiatement suivante $p+1$.

Ceci peut encore s'exprimer en disant que $a\,\mathrm{R}\,b$ (en P) s'il existe un sommet Q tel qu'il existe une plus courte chaîne de P à Q utilisant a et une chaîne de P à Q utilisant b de longueur $d(P,Q)$ ou $d(P,Q)+1$.

Rappelons aussi que 2 arêtes a et b appartiennent à la même composante 2-connexe si et seulement s'il existe un cycle élémentaire passant par a et b.

PROPOSITION : *La fermeture transitive R^* de la relation R est exactement la relation « appartenir à la même composante 2-connexe du réseau ».*

Preuve : Supposons que $a'\,\mathrm{R}^*\,b'$ et que a' et b' soient dans deux composantes 2-connexes différentes. Il existe donc (à cause de la transitivité) deux arêtes a et b dans deux composantes 2-connexes différentes telles que $a\,\mathrm{R}\,b$. Si Q est un sommet qui a impliqué $a\,\mathrm{R}\,b$, on a donc une chaîne entre P et Q de longueur $d(P,Q)$ finissant par a et une autre chaîne de longueur $d(P,Q)$ ou $d(P,Q)+1$ finissant par b. Soit B l'autre extrémité de l'arête b. Si B est dans la même composante 2-connexe que Q alors il existe un chemin de B à Q ne passant pas par P. Ce chemin avec celui de P vers Q et l'arête a implique l'existence d'un cycle élémentaire contenant a et b ce qui est contraire à l'hypothèse. B est donc dans une composante 2-connexe différente de celle de Q, la seule manière d'atteindre B est de passer par P et donc toute chaîne de Q à P finissant par b est de longueur au moins $d(P,Q)+2$, contredisant les propriétés de Q.

Réciproquement, supposons qu'une composante 2-connexe contienne différentes classes de R*. Parmi toutes les paires d'arêtes appartenant à deux classes différentes, choisissons en une $\{a,b\}$ telle que le plus petit cycle élémentaire contenant a et b soit de longueur minimum. Soit c la longueur de ce cycle. Soit Q le nœud à distance $\left\lfloor \frac{c}{2} \right\rfloor$ de P sur le cycle en passant par a. Montrons que la distance $d(P,Q) = \left\lfloor \frac{c}{2} \right\rfloor$; en effet, sinon soit a' l'arête entrante en P sur la chaîne la plus courte, alors les deux paires $\{a,a'\}$, $\{b,a'\}$ appartiennent à un cycle de longueur strictement inférieure à c et comme a' ne peut être à la fois dans la classe de a et dans celle de b, l'une de ces paires contredit l'hypothèse de minimalité de $\{a,b\}$. Donc P reçoit l'information de Q par l'arête a à la phase $\left\lfloor \frac{c}{2} \right\rfloor$ et par b à la phase $\left\lceil \frac{c}{2} \right\rceil$ donc $a\,\mathrm{R}^*\,b$ d'où une contradiction.

5. Complexité et comparaisons avec d'autres algorithmes

Dans cette section, nous donnons un certain nombre de résultats dérivés de l'algorithme général. Ainsi les résultats de la sous-section 1 et 3 sont détaillés dans les articles [Bermond 87 et Bermond 89].

5.1. Nombre de messages

A chaque phase P envoie $d(P)$ messages, où $d(P)$ désigne le degré de P, c'est-à-dire le nombre de voisins de P.

Comme il y a $ecc(P)+2$ phases pour P, P envoie $(ecc(P)+2)\,d(P)$ messages en tout. Le nombre de messages total est donc $\sum_i (ecc(P)+2)\,d(P)$.

Cette valeur est bornée par $2(D+2)\,m$ où D est le diamètre et m le nombre d'arêtes. En effet, $\sum_i d(P_i) = 2m$.

5.2. Quantité d'information

Sur chaque arête il passe au plus 2 fois un même identificateur (une fois dans chaque sens). Donc, si on suppose que le codage d'un identificateur nécessite $\log_2(n)$ bits, on obtient une complexité en $O(nm \log(n))$. Cette valeur est à comparer au $O(m^2 \log(n))$ dans le cas de l'application directe de l'algorithme.

5.3. Durée de l'algorithme

Soit τ le temps maximum de transmission d'un message d'un nœud vers un nœud voisin. Tous les nœuds du réseau sont réveillés moins de τD après qu'un nœud ait commencé le calcul. Comme il y a au plus $D + 2$ phases qui ont chacune une durée bornée par τ, la durée totale est au plus $2(D + 1)\tau$. On remarque que cette valeur est optimale à une constante près dans un réseau asynchrone : en effet, si un seul processeur P se réveille spontanément, il faut au moins attendre D transmissions en séquentiel pour réveiller un processeur Q à distance D et il en faut encore D pour que l'information de Q revienne à P.

5.4. Comparaisons avec d'autres algorithmes

Notre algorithme est le meilleur pour la complexité en temps. Les autres algorithmes ont aussi une complexité en $O(D)$ [Hohberg 90, Huang 88, Lin 90] voire $O(n)$ [Park 88], mais la constante est moins bonne. Par exemple, dans [Hohberg 90], l'algorithme a une durée de $3\,D\tau$ plus le temps nécessaire à l'élection d'un leader, soit au moins $5\,D\tau$. Dans [Lin 90] la durée de l'algorithme est $(6\,D + 2\,\Delta)\,\tau$.

Pour la complexité en messages notre algorithme a une complexité au plus $2(D + 2)\,m = (D + 2)\,\Delta n$. Celui de [Huang 88] nécessite $O(\Delta n^2)$ messages, ceux de [Hohberg 90] et de [Park 88] $O(m + n \log(n))$ messages et celui de [Lin 90] $4\,m + 2\,\Delta n$ messages.

Mais tous ces algorithmes ont un défaut (outre d'être plus compliqués) à savoir de nécessiter l'existence d'un nœud centralisateur que celui-ci soit donné au départ ou élu ou apparaisse comme la racine d'un arbre DFS. Dans [Park 88] un arbre de recherche en profondeur est d'abord construit ; dans [Lin 90] ils supposent la donnée initiale d'un arbre couvrant et d'un initiateur. Enfin dans [Hohberg 90] une élection a d'abord lieu et c'est le sommet élu qui coordonne le reste de l'algorithme. Tous ces algorithmes ne sont donc pas vraiment distribués et de plus le nœud centralisateur doit effectuer un travail important gérant à la fois le contrôle associé à l'algorithme et des calculs plus importants.

En ce qui concerne la quantité d'information échangée nous conjecturons que pour calculer les composantes 2-connexes en un temps $O(D)$ il est nécessaire de transférer $O(nm \log n)$ bits. Il peut sembler que ce n'est pas le cas dans l'algorithme de [Hohberg 90 ou Lin 90] qui nécessite moins de transfert d'information une fois que l'on connaît un leader ou un arbre couvrant. Mais à notre connaissance il n'existe pas d'algorithme d'élection dans un réseau quelconque (ou de détermination d'arbre couvrant) en temps $O(D)$ qui utilise un transfert d'information moindre que $O(nm \log n)$. Par exemple l'algorithme de [Gallager 83] auquel se réfère [Hohberg 90] a une durée en temps en $O(n \log n)$. Ainsi il semble que supposer la connaissance d'un leader masque la partie significative du calcul.

Enfin notons que les divers algorithmes proposés dans la littérature ainsi que le notre la complexité en moyenne est du même ordre que la complexité dans le pire des cas. Dans l'étude de la complexité il faudrait remplacer pour le calcul du nombre de messages D par la distance moyenne.

BIBLIOGRAPHIE

[Bermond 87] J.-C. Bermond, J.-C. König, M. Raynal : *General and Efficient Decentralized Consensus Protocols,* in Distributed Algorithm, 2nd International Workshop on Distributed Algorithms, Lectures Notes in Computer Science 312, Springer Verlag 198, 41-56.

[Bermond 89] J.-C. Bermond and J.-C. König : *General and Efficient Decentralized Consensus Protocols II,* Proc. Workshop on Parallel and Distributed Algorithms, Bonas, 1988, North Holland, 1989, 199-210.

[Gallager 83] R. G. Gallager, P. A. Humblet, P. M. Spira : *A distributed algorithm for minimum weight spanning trees,* ACM Trans. Program. Lang. Syst. 5, 1983, 66-77.

[Hohberg 90] W. Hohberg : *How to Find Biconnected Components in Distributed Networks,* A paraître dans Journal of Parallel and Distributed Computing.

[Huang 88] S.-T. Huang : *A new distributed algorithm for biconnectivity problem,* Institute of Computer Sciences, National Tsing-Hua University, Hsinchu, Taiwan, R.O.C., Technical report TR-H7612, August 1988.

[Lin 90] J.-C. Lin and S.-N. Yang : *An improved distributed algorithm for biconnectivity problem,* manuscript.

[Park 88] J. Park, T. Masuzawa, K. Hagihara and N. Tokura : *On efficient distributed algorithms solving some problems about the connectivity of a network,* Comp. 88-61, pp. 105-113, 1988 (en japonais).

[Raynal 85] M. Raynal : *Algorithmes Distribués et Protocoles* Eyrolles, 1985, 141 p.

[Raynal 88] M. Raynal and J.-M. Helary : *Synchronisation et contrôle des systèmes et des programmes répartis,* Eyrolles, 1988, 194 p.