DETERMINAREA SETULUI MAXIMAL INDEPENDENT ÎNTR-O REȚEA SINCRONĂ

Drd. Mihai Virgil Cherana

Universitatea Politehnica *Bucureşti*

Rezumat: În acest articol, voi prezenta un algoritm de determinare a unui set maximal independent, în cadrul unei rețele sincrone. Voi prezenta algoritmul formal, susținut de demonstrații matematice. La final, voi realiza o analiză de complexitate şi de timp a acestuia. Algoritmul a fost original dezvoltat în cadrul MIT, o contribuție importantă având-o doamna Nancy Lynch, profesoară în cadrul acestei instituții. Contribuția personală a autorului este legată de demonstrația teoremei 2, din cadrul algoritmului

Cuvinte cheie: Set maximal independent, Maximal Independent Set, algoritmi de optimizare.

1. Introducere

Problema găsirii unui set independent maximal poate fi motivată prin alocarea de resurse partajate proceselor din rețea. În cadrul unei aplicații distribuite, formată din funcții bloc, pot exista anumite funcții bloc, care nu pot efectua simultan activități legate de resurse partajate [1]. Din această cauză, trebuie identificat setul maximal independent de astfel de funcții bloc, care pot efectua simultan activități legate de resursele partajate. Pentru a evita conflictele, aceste noduri (funcții bloc) trebuie să conțină un set independent în cadrul unui graf G. Mai mult chiar, pentru performanță, este de nedorit să blocăm un proces dacă nici unul din vecinii săi nu este activ. Astfel, setul de procese ales trebuie să fie maximal [2].

2. Setul maximal independent

2.1 Enunțarea formală a problemei

Fie $G=(V,E)$ un graf nedirecționat. Un set $I \subseteq V$ de noduri este *independent* dacă pentru toate nodurile $i,j \in I$, $(i,j) \notin E$. Un set independent I este maximal dacă orice set I' care îl conține strict pe I nu este independent. Scopul este de a determina un set maximal independent al lui G. Mai exact, orice nod al cărui index este în I trebuie, eventual, să scoată la ieşire *winner (câştigător)*, adică trebuie să seteze valoarea unei variabile de stare *status* la valoarea *winner*, iar fiecare proces care nu este în I să seteze aceeaşi variabilă de stare *status* la valoarea *loser*.

În cele ce urmează, voi prezenta un algoritm care rezolvă această problemă. Acest algoritm poartă numele *LubyMIS* [3].

Algoritmul se bazează pe o schemă iterativă, în care un set independent arbitrar nevid este selectat din nodurile grafului G. Nodurile care aparțin setului sunt scoase din graf împreună cu toți vecinii lor. Procesul se repetă până când se identifică un set maximal independent. Dacă W este un subset al nodurilor grafului, voi folosi notația *nbrs(W)* pentru a denota vecinii nodurilor din W.

Fie *graph* o variabilă compusă (de tip record sau struct) care are componente câmpurile *nodes, edges, nbrs,* inițializate la valorile avute inițial în cadrul grafului original G. Pseudocodul algoritmului este:

while $graph.nodes \neq \Phi$ do

alege un set nevid $I' \subseteq graph.nodes$ care sunt independente în *graph*.

$I := I \cup I'$

graph: =subgraful indus al grafului G pe un subset W al acestuia (subgraful (W,E'), unde E' este setul de arce ce conectează nodurile din W)

end while

Nu este dificil de observat că aceasta produce întotdeauna un set maximal independent. Pentru a vedea de ce este independent, este de remarcat că la fiecare rundă a algoritmului, setul selectat I' este independent, iar eu am eliminat explicit toți vecinii ce au arce care intră în setul I. Pentru a vedea de ce este maximal, trebuie remarcat că singurele noduri ce sunt eliminate din graful original sunt vecinii nodurilor ce se adaugă la I.

Întrebarea cheie a acestui algoritm este cum se alege setul *I* la fiecare iterație, într-o aplicație distribuită. Aici intră în calcul randomizarea. La fiecare iterație, fiecare proces *i* alege un întreg val_i în intervalul $[1,\ldots,n^4]$, valoare randomizată, folosind distribuția uniformă. Motivul folosirii valorii n^4 este că această valoare este suficient de mare în așa fel încât, cu o mare probabilitate, toate procesele din graf vor alege valori distincte. Odată ce procesele au ales aceste valori randomizate, definesc *I'* constituit din nodurile *i a căror* valoare a variabilei de stare *status* este *winner,* adică acele noduri *i* pentru care $val_i > val_j$, pentru toți vecinii *j* a lui *i*. Astfel, se obține, în mod evident, un set independent deoarece doi vecini nu se pot simultan înfrânge unul pe celălalt.

În această implementare, este posibil, dacă valorile randomizate sunt alese nefavorabil, setul *I'* să fie vid pentru anumite iterații. Aceste iterații (stagii) vor fi inutile, nerealizând nimic. Presupunând că algoritmul nu ajunge în punctul în care realizează doar stagii inutile (este vid la acestea), putem ignora aceste stagii inutile și putem declara că acest algoritm este corect. Vor trebui, totuși, analizate aceste stagii inutile. Urmează definiția formală a algoritmului.

Algoritmul lucrează în stagii, fiecare stagiu având trei runde.

Runda 1: În prima rundă a stagiului, procesele își aleg valorile randomizate *val* și le trimit vecinilor. La sfârșitul rundei 1, când toate valorile *val* au fost recepționate, câștigătorii (procesele din *I),* știu cine sunt.

Runda 2: În runda doi, câștigătorii își înștiințează vecinii. La sfârșitul rundei doi, nodurile perdante, adică procesele având vecini în *I',* știu cine sunt.

Runda 3: În runda trei, fiecare proces perdant își înștiințează vecinii. Apoi, toate procesele implicate (câștigători, perdanți și vecinii perdanților) elimină nodurile și arcele respective din graf. Mai precis, aceasta înseamnă că perdanții și câștigătorii încheie participarea după acest stagiu, iar vecinii perdanților elimină toate arcele care sunt incidente la nodurile proaspăt eliminate.

Voi descrie în cele ce urmează algoritmul intr-un mod mai formal. Fiecare proces folosește o funcție specială numită $rand_i$, care se aplică la fiecare rundă înaintea aplicării funcțiilor $msgs_i$ și $trans_i$. În cele ce urmează, voi folosi *random* pentru a indica o alegere randomizată din intervalul $[1,\ldots,n^4]$, folosind distribuția uniformă.

Pseudocodul algoritmului:

$stări_i$:

round $\in \{1,\ldots,n^4\}$, inițial arbitrar

awake, variabila de tip Boolean, inițial *true*

rem-nbrs=G-I (vecinii nodurilor ce aparțin de I)

$rand_i$:

if awake and round=1 then *val: =random*

$msgs_i$:

if *awake* then

case

round=1;

send *val* către toate nodurile din *rem-nbrs*

round=2;

if *status=winner* then

send *winner* către toate nodurile din *rem-nbrs*

round=3;

if *status=loser* then

send *loser* către toate nodurile din *rem-nbrs*

endcase

$trans_i$:

if *awake* then

case

round=1;

if *val*>*v* pentru toate valorile *v* ce vin de la vecini then *status:=winner*

round=2;

if un mesaj de *winner* soseşte then *status:=loser*

round=3;

if *status* ∈{*winner,loser*} then *awake:=false*

rem-nbrs:=rem-nbrs - {*j:* un mesaj *loser* soseşte de la *j*}

endcase

round:=(round+1 mod 3)

2.2 Analiza algoritmului

După cum am amintit, presupun că algoritmul nu realizează numai ciclii inutili (în care setul *I'* este vid). Fac afirmaţia că numărul de arce eliminate la fiecare rundă a algoritmului este o constantă egală cu o fracţiune din numărul total de arce al grafului iniţial. Fie graful *G=(V,E)* şi, pentru un nod arbitrar *i* ∈*V,* se defineşte:

$$sum(i) = \sum_{j \in nbrs_i} \frac{1}{d(j)} \qquad (1)$$

unde d(j) este gradul lui *j în* graful *G* (numărul de arce ce pleacă de la j către vecinii săi). Următoarea lemă arată probabilitatea ca un nod *i* să facă parte din vecinii nodurilor din setul *I'*.

LEMA 1 Fie I' definit ca un subgraf independent al lui G la un anume stagiu al algoritmului. Pentru orice i din graf, înaintea stagiului respectiv, probabilitatea ca acesta să facă parte din vecinii nodurilor din setul I' este:

$$\Pr[i \in nbrs(I')] \geq \frac{1}{4} \min\left(\frac{sum(i)}{2}, 1\right) \qquad (2)$$

Folosind lema 2, se obţine numărul de arcuri eliminate la un stagiu din cadrul grafului:

TEOREMA 2 Numărul de arce eliminat din G la un singur stagiu este cel puţin E/8, unde E este numărul total de arce ce ies dintr-un nod.

Demonstraţie Algoritmul asigură faptul că fiecare arc cu cel puţin un capăt în *nbrs(I')* este eliminat. Urmează că numărul de arce eliminat este cel puţin:

$$\frac{1}{2} \sum_{i \in V} d(i) \cdot \Pr[i \in nbrs(I')] \qquad (3)$$

Această relaţie este adevărată deoarece fiecare nod *i* are probabilitatea indicată în lema 1 de a avea un vecin în *I'*. Dacă acesta este cazul, atunci *i* este eliminat ceea ce cauzează ştergerea tuturor arcurilor sale incidente *d(i)*. Factorul ½ este prezent pentru a compensa posibila supranumărare a arcurilor, deoarece fiecare arc are două capete care pot cauza ştergerea sa.

Prin înlocuirea lemei 2 în expresia 4, obţinem:

$$\frac{1}{8} \sum_{i \in V} d(i) \cdot \min\left(\frac{sum(i)}{2}, 1\right) \qquad (4)$$

Despărţind această expresie după termenul mai mic din expresia cu *min*, expresia (5) devine:

$$\frac{1}{8}\left(\frac{1}{2} \sum_{i:sum(i)<2} d(i) \cdot sum(i)\right) \qquad (5)$$

în caz că primul termen al expresiei *min* este mai mic sau

$$\frac{1}{8}\sum_{i:sum(i)\geq 2} d(i) \tag{6}$$

în caz că al doilea termen al expresiei *min* este mai mic. Înlocuind definiția pentru *sum(i)* din ecuația (1) rezultă:

$$\frac{1}{8}\left(\frac{1}{2}\sum_{i:sum(i)<2}\sum_{j\in nbrs_i}\frac{d(i)}{d(j)}\right) \tag{7}$$

în caz că primul termen al expresiei *min* din ecuația (5) este mai mic sau

$$\frac{1}{8}\sum_{i:sum(i)\geq 2}\sum_{j\in nbrs_i} 1 \tag{8}$$

în caz că al doilea termen al expresiei *min* din ecuația (4) este mai mic.

Fiecare arc nedirectat *(i,j)* contribuie de două ori la sumările anterioare, câte o dată pentru fiecare direcție. În fiecare caz, suma acestor termeni este mai mare decât 1, deci totalul este cel puțin $\frac{E}{8}$.

Teorema 2 poate fi folosită la următoarea lemă:

LEMA 3 Cu o probabilitate de cel puțin $\frac{1}{16}$, numărul de arce eliminate din G într-un singur stagiu este cel puțin $\frac{E}{16}$.

3. Concluzii

Tehnica randomizării este folosită frecvent în algoritmii distribuiți. Scopul lor principal este de a rupe simetria. De exemplu, algoritmul de alegere a liderului nu poate fi rezolvată în grafuri generale de procese deterministice, fără UID. Fără valoarea unică a UID, este imposibil să se rupă simetria şi, deci, algoritmul să funcționeze corect.

O problemă care o ridică algoritmii randomizați este că garantează corectitudinea cu o probabilitate mare, nu cu certitudine. În designul acestor algoritmi trebuie avut grijă ca proprietățile cheie ale acestora să fie definite cu certitudine, şi nu cu probabilitate, fie ea şi ridicată. În algoritmul randomizat, prezentat mai sus, este garantată corectitudinea acestuia atâta vreme cât mărimile *val* din intervalul $[1,\ldots,n^4]$ sunt unice. În caz contrar, algoritmul nu mai funcționează corect. Există chiar o posibilitate (cu probabilitate foarte aproape de zero) ca toate procesele să aleagă în mod repetat aceeaşi valoare, astfel blocând progresul algoritmului. Dacă aceste scăderi sunt serioase sau nu, depinde de aplicația la care este folosit. În cazul sistemelor distribuite de măsură şi control, se constată că aceşti algoritmi dau, în general, rezultate foarte bune.

Bibliografie

1. **LYNCH, N. A.**: Distributed Algorithms, Morgan Kaufmann Publishers,San Francisco, CA,1996.
2. **LYNCH, N. A., H. ATTIYA**: Time Bounds for Real-time Process Control în the Presence of Timing Uncertainty, Information and Computation, April 1994.
3. **ASPNES, J**: Fast Randomised Consensus Using Shared Memory. În: Journal of Algorithms, September 1990.