アルゴリズム 30－16
(1992. 11. 21)

任意形状の非同期ネットワークにおける耐故障リーダ選挙について

松本 哲也　若林 真一　小出 哲士　吉田 典可

広島大学工学部
東広島市鏡山１丁目４－１

あらまし 分散アルゴリズムの基本的問題の１つにリーダ選挙問題がある．本研究では，リンク故障を含むと仮定した任意形状の完全非同期ネットワークにおけるリーダ選挙問題について考察を行う．まず，各プロセスが固有の識別子をもち，ネットワークサイズと故障リンク数の上限値を初期情報としてもつとしたとき，リーダ選挙問題に対するメッセージ数$O(m+n\log^2 n)$，メッセージサイズ$O(\log n)$ビットの終了検知を伴うアルゴリズムを提案する．ここでnはネットワーク中のプロセス数，mはリンク数である．次に，各プロセスがネットワークサイズを初期情報としてもつとしても固有の識別子をもたなければリーダ選挙問題は解けず，たとえ各プロセスが独立に確率的選択を行うランダム化アルゴリズムを用いても解けないことを示す．

和文キーワード 分散アルゴリズム，リーダ選挙，非同期システム，リンク故障，匿名ネットワーク，終了検知

On Leader Election for Faulty Asynchronous Networks with Arbitrary Topology

Tetsuya MATSUMOTO　Shin'ichi WAKABAYASHI　Tetsushi KOIDE　Noriyoshi YOSHIDA

Faculty of Engineering, Hiroshima University
4-1, Kagamiyama 1 chome, Higashi-Hiroshima, 724 JAPAN

Abstract Leader election is one of the basic problems on distributed networks. In this paper, leader election for fully asynchronous distributed networks with arbitrary topology with link faults is considered. First, when every process has a unique identifier and knows the size of network and the upper bound of number of link faults as initial information, we propose an algorithm for leader election with termination detection that uses $O(m+n\log^2 n)$ messages , each of size $O(\log n)$ bits, where n and m are the number of processes and links, respectively, in the network. Next, when every process knows the size of network but doesn't have unique identifiers, we show that even if a randomized algorithm that can execute random selection locally is used, leader election cannot be solved.

英文 key words distributed algorithm, leader election, asynchronous system, link fault, anonymous network, termination detection

1．まえがき

分散アルゴリズムの基本的問題の1つにリーダ選挙問題（ＬＥＰ）がある[10][11]．これは，トークンシステムにおいてトークン消失後にトークンを再生成させるプロセスを選択する場合など，分散システム中で特別な役割をもたせるプロセスをただ1つ選択することを目的とした問題であり，様々な応用がある．一方，分散システムでは耐故障性の考慮が重要な課題である．本稿では特に，故障を含む任意形状のネットワークにおけるＬＥＰについて考察する．従来，この問題は各プロセスが固有の識別子をもつと仮定して解かれており，故障を含み，かつ固有の識別子をもたないネットワークにおけるＬＥＰについての研究はほとんど行われていない．本稿では固有の識別子をもつ場合ともたない場合のＬＥＰについて考察を行う．

2．準備

本稿で考察する分散ネットワークは通信リンクを介して通信するプロセスの集合であると仮定する．各プロセスは独自に同じアルゴリズムを実行し，共有変数を持たず，プロセス間の通信は2プロセス間にリンクが存在するときのみ直接メッセージを送ることができる．ネットワーク全体は完全非同期である．ネットワークはn個のプロセスと，m本の双方向通信リンクの集合からなり，プロセスをノード，リンクを枝とする任意形状のグラフで表される．またアルゴリズム開始時，任意のプロセス部分集合が自発的に起動する始動プロセスに成り得るとする．各プロセスは初期情報として，実行するアルゴリズム，各自のポートをもち，ポートは区別できるとする．

本稿で対象とする故障はリンク故障である．故障リンクでは送信が行われても受信側プロセスにメッセージを伝えないものとする．すべての故障はアルゴリズム開始時に既に起こっているとし，プロセス故障は考えず，ネットワークは非故障リンクのみで連結とする．

本稿においてＬＥＰを解くとは，

(A) 1. ただ1つの特別なプロセスがリーダとして選ばれ，
2. 他のすべてのプロセスはリーダを知る，

ことであるとする．ただし，ＬＥＰを解くアルゴリズムはメッセージドリブンとする．

識別子と故障に関する条件を変えた種々のグラフとそれらのグラフに対するＬＥＰに関する過去の研究結果を表1にまとめる．ネットワークが固有の識別子をもたない場合，故障やネットワークサイズの知識にかかわらずＬＥＰを決定的に解くことができない場合が存在することが知られている[2]．従って本稿では，アルゴリズムが終了しても問題が正しく解けなかった場合でも，各プロセスが解けなかったことを検知できるなら解けたとみなす．これは，もし解けなかったことが検知できるならランダム化アルゴリズムを繰り返し実行すればいずれ必ず条件(A)でＬＥＰは解けるので許容できるといえる．結局，本稿においてＬＥＰを解くとは条件(A)を満たすか，次の条件(B)を満たせばよいとする．条件(B)で解ける場合は表1中では三角形で示している．

(B) アルゴリズム終了時に(A)の条件を満たさない場合，各プロセスはそのことを検知する．

表1　種々のネットワークに対するＬＥＰについての過去の研究結果

	識別子○		識別子×	
	n未知	n既知	n未知	n既知
故障無	G ○	G_n ○	G_a ×[2]	G_{an} △[5-8]
故障有	G_f ×[4]	G_{fn} ○[1][3]	G_{fa} ？	G_{fan} ？

識別子○：固有の識別子をもつネットワーク
識別子×：固有の識別子をもたないネットワーク
故障無　：故障を含まないと仮定したネットワーク
故障有　：故障を含むと仮定したネットワーク
n既知　：各プロセスはnを初期情報としてもつ
n未知　：各プロセスはnを初期情報としてもたない
○：必ず条件(A)を満たす
△：条件(A)または(B)を満たす
×：条件(A)も(B)も満たさない

ネットワークのクラスGの添え字のfは故障を含むこと(faulty)，aは固有の識別子をもたないこと(anonymous)，nはネットワークサイズ既知であること(number of nodes)を表す．例えば，固有の識別子をもたず，故障を含み，各プロセスがネットワークサイズnを既知であるようなネットワークのクラスはG_{fan}で表す．

本稿ではG_{fn}とG_{fan}について考察を行った．[1]ではG_{fn}においてメッセージ数$O(m+n\log^5 n)$でＬＥＰを解くアルゴリズムが提案されている．G_{fn}の各プロセスが故障リンク数の上限値f_l（定数）を初期情報としてもつとしたネットワークのクラスをG_{fn}'と定義する．3．ではG_{fn}'に対して条件(A)を必ず満たしてメッセージ数$O(m+n\log^2 n)$でＬＥＰを解くアルゴリズムALEPを提案する．また，4．ではG_{fan}に対しＬＥＰは解けないことを示す．

3．固有の識別子をもち，故障を含むネットワークについて

本節ではネットワークG_1（$\in G_{fn}'$）上でＬＥＰを解くアルゴリズムALEP(Algorithm for LEP)について示す．

3．1　アルゴリズムALEP

ALEPはスパニング森を成長させる手続きＧＳＴと，スパニング森の連結成分である各木のノード数を数える手続きＮＣからなり，これらは同時に実行される．ＧＳＴによって始動プロセスから根付きスパニング森を構築していく．ところがG_1は故障リンクを含むのでＧＳＴでは木のノード数は不明で終了検知ができない場合が存在する．よってＮＣにより木のノード数を数え，n/2以上のノードを含む木ができたとき，そのルートがリーダとなる．以下ではＧＳＴとＮＣを分けて説明する．

3.1.1　スパニング森生成手続きＧＳＴ

ＧＳＴは[1]の終了検知なしの耐故障リーダ選挙手法とほとんど同じものを用いる．ＧＳＴで構成するスパニング木のリンクをBranchと呼ぶ．Branchの集合は森を構成する．森の各連結成分をフラグメント，フラグメントのルートをコアと呼ぶ．最初Branchの集合は空で，各始動プロセスがそのフラグメントのコアになる（図1）．

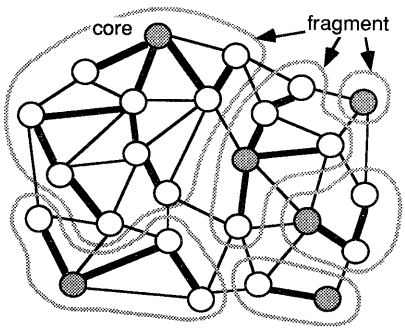


図1　コアとフラグメント

ＧＳＴにおいて，各フラグメントの動作はコアによって制御され，隣接する他のフラグメントと結合するためのリンクを１本選択する．選ばれたリンクはBranchになってフラグメント同士が結合し，どちらか一方のフラグメントのコアが拡張されたフラグメントのコアになる．いずれただ１つのフラグメントが残り，そのコアがリーダになる．

フラグメントは他のフラグメントと結合しながら成長するが，他のフラグメントと識別するためにラベルを用いる．各フラグメントは（レベル，コアの識別子）の順序対であるラベルをもつとし，レベルは非負の整数であるとする．アルゴリズム開始時，各フラグメントは１ノードからなり，レベルは０とする．

(A)　リンクの分類

ネットワークは非同期のため，故障リンクを遅延の大きなリンクと区別することは不可能であり，アルゴリズムは各リンク上でメッセージが必ず有限時間内で通信できることを仮定して動作することは許されない．それ故，各ノードｖは隣接リンクを，メッセージを受け取ったか否かによりOperational，Quietに分類する．

初期時，全リンクはQuietとする．ノードは自発的，もしくはメッセージ受信により起動すると，まずHelloメッセージを隣接リンクの各々に送る．Helloメッセージを受け取ったノードはそのリンクをOperationalと指定する．リンクがQuietであるかぎりは，さらなるメッセージは送らない．

Operationalリンクは以下のように３つに分類される（図2）．

（ｉ）Branch(v)　：フラグメントの木のリンクでノードｖに接するリンク集合
　ｖがコアでなければコアの方向にparent(v)，逆向きにchildren(v)と設定する．

（ii）Rejected(v)　：同じフラグメントの他のノードに接続し，Branchではないリンク集合
　いったんRejectedと分類されれば存在しないものとして扱われる．

（iii）Linkqueue(v)　：(i)，(ii)以外のリンク集合
　Operationalになるとキューに加えられる．キューの先頭のリンクをtestlink(v)とし，ノードはtestlink(v)をBranch(v)に加えようとする．

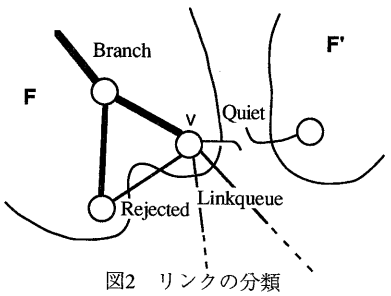


図2　リンクの分類

(B)　外向枝の選択

フラグメントのコアがそのラベル（レベル）を変えるときはいつでも，

(1) 新しいラベルをフラグメント内の各ノードに知らせ，

(2) 新しい外向枝選択フェーズを始める，

ために，Initiateメッセージをフラグメントの全ノードに放送する．ノードはInitiateメッセージを受け取ると

そのラベルを更新し，メッセージを受け取ったリンクをparent(v)とし，同じメッセージを子に放送する．このラベル更新手続きをＩＮＩＴ手続きと呼ぶ．それから，後に述べるテスト動作により，Linkqueue中から自フラグメントより大きいラベルのフラグメントを導く外向枝を探し始める．最初に見つかった枝をOutlink(v)とし，親にPossibleOutlinkメッセージを送る．ｖはLinkqueue中からどれか１つをOutlink(v)として見つけるか，もしくは子からPossibleOutlinkメッセージを受け取るかして外向枝が見つかったことを知る．ｖの子もまた同じことをする．いったんOutlink(v)を選ぶと，その後で報告されたものは無視される．

リンクの探索はTestメッセージをtestlink(v)に送ることで行う．ノードｖがノードｗにTestメッセージを送ったとする（図3(a)）．

(1) ｖとｗが同じラベルをもつとき　（図3(b)）
（すなわち同じフラグメントに属する）
ｗはｖにRejectメッセージを返し，ｖとｗはそのリンクをRejectedと指定する．そしてｖはLinkqueue(v)中の次のリンクを新しいtestlink(v)とする．

(2) ｗのラベルの方が大きいとき　（図3(c)）
ｗはｖにPossibleOutlinkメッセージを返す．

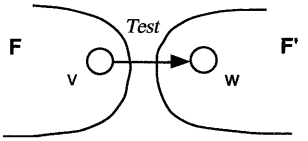
(3) ｗのラベルの方が小さいとき　（図3(d)）
とりあえず返答しない．いつかｗが大きくなったとき返答するか，もしくはｗがｖにTestメッセージを出すことになる．（ただし，ＮＣのために，ｗはTestを受け取り，ｗのラベルの方が小さかった事実をｖに伝えることだけは行う．）

ｖはPossibleOutlinkメッセージをｗから受け取ると，もし以前に受け取っていなければ，そのtestlink(v)をOutlink(v)とし，親へ伝え，いずれこのことはコアまで伝わることになる（図3(c)）．

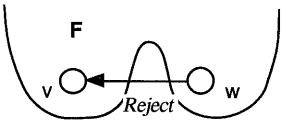
Initiateメッセージに関連して使われるすべてのメッセージはフラグメントのラベルでタグ付けされている．新しいInitiateメッセージを受け取ると現在のリンク選択手続きはアボートされる．すなわち，ノードはOutlinkをnullにリセットし，前のレベルのPossibleOutlinkメッセージは無視する．

(C) フラグメントの結合

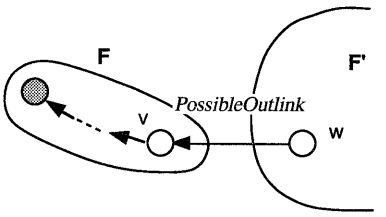
いったんコアがOutlinkを選ぶと，Outlinkに沿ってConnectメッセージが送られ，そのOutlinkに対応する外向枝を見つけたｖに伝えられる．ｖはそのOutlinkをBranchに変え，リンクのもう片方の端点であるｗにConnectメッセージを送る．フラグメントの結合には次の２通りがある．



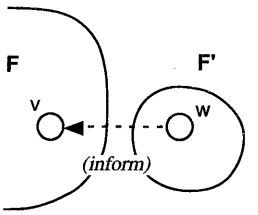
(a) Testメッセージ



(b) label(v)＝label(w)のとき



(c) label(v)＜label(w)のとき



(d) label(v)＞label(w)のとき

図３　リンクのテスト

(1) level(v)<level(w)のとき
[吸収手続き]　（図4(a)）

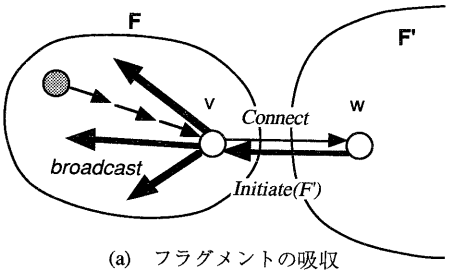
レベルの小さい方のフラグメントに属するノードのみがラベルを更新し，大きい方のラベルを受け継ぐ．具体的には，ｗはｖにInitiateメッセージを送る．すなわちｗのラベルがｖ側のフラグメントに放送されることになる．そしてｖ側のフラグメントはOutlinkを探すｗのフラグメントサーチに参加する．このとき，ｗ側のフラグメントではｖ側のフラグメントが追加されたことを知らせるメッセージは使われない．

(2) level(v)=level(w)=jのとき
[合併手続き]　（図4(b)）

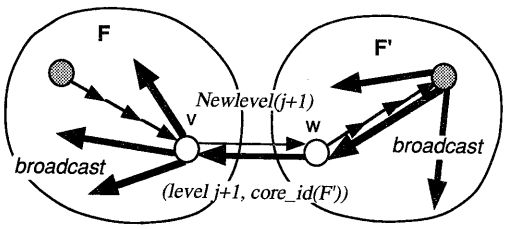
両方のフラグメントに属するノードがレベルをインクリメントし，どちらかのコアの識別子をもつようラベル更新する．具体的には，ｗはNewlevel(j+1)メッセージを親に送り，コアまで伝える．もし途中

のノードで他のNewlevelメッセージやInitiateメッセージが先に受け取られていれば，このNewlevelメッセージはアボートされ，これ以上何もしない．そうでなければメッセージはコアに到着する．もしコアのレベルが既に上がっていれば，もしくは既にConnectメッセージを送っていればこのNewlevelメッセージは無視される．そうでなければレベルをj+1にし，新しいフラグメントラベルでInitiateメッセージを全ノードに放送する．これはいずれwに到着し，いずれvのフラグメントのノードすべてに到達する．

Newlevelメッセージがアボートされるときは，wのフラグメントのレベルが他の理由で増加していることになり，それ故いずれInitiateメッセージが受け取られることになる．



(a) フラグメントの吸収



(b) フラグメントの合併

図4 フラグメントの結合

レベルjのフラグメントは少なくとも2^jのノードをもつので，可能な最大レベルはlognとなる．ノードのフラグメントレベルは単調増加であり，あるノードが同じレベルの間はそのフラグメントのコアの識別子は変わらないので，各ノードは高々logn個のフラグメントに属することになる．

3.1.2 ノード数計算手続きＮＣ

フラグメントのノード数がn/2を超えるとそのコアはリーダとなり，そのことを放送してアルゴリズムは終了する．しかし，ＧＳＴの動作のみではノード数が不明で終了検知ができない．よってＮＣではフラグメント内のノード数を数えることを目的とする．レベルjのフラグメントには2^j以上のノードが含まれている．ＮＣではこのフラグメントのノード数が2^{j+1}を超えたとき必ずレベルj+1のラベルに更新する．こうすると過半数のノードを含むフラグメントは必ずj≧logn-1なので，ここで１つでもノードが加わるとコアはリーダとなり，アルゴリズムが終了する．以下で具体的な動作について説明する．

あるフラグメントのレベルが上がり，ノードのラベルが更新されると，ＧＳＴの再開と同時にＮＣが開始される．各フラグメントに対し，$2f_i+1$個のトークンがコアから深さ優先順にノード数を数えながらBranchから構成される木を探索する（図5）．このメッセージにはフラグメントの前レベルまでの正確なノード数が含まれている．リンクには，Branch，Rejected，Linkqueue，Quietの高々４種類が存在する．あるノードにおいてトークンが次に探索するリンクの種類により，ＮＣは以下の動作を行う（図6）．

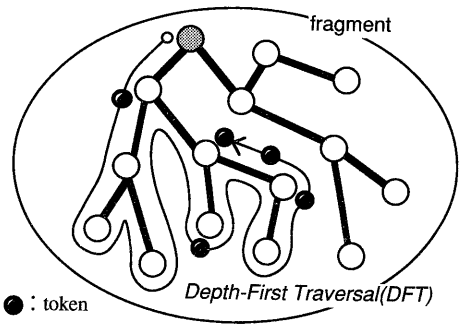


図5 トークンによる深さ優先探索

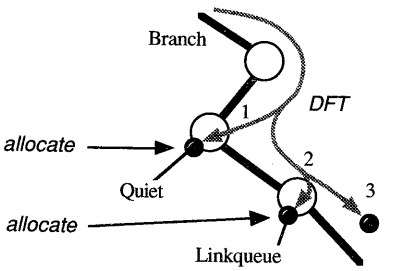


図6 トークンの探索と割り当て

Case 1. Branch

探索順どうりにトークンを進める．

Case 2. Rejected及び既にトークンの割り当てられたリンク

そのリンクを無視し，そのノードの次のリンクの探索に移る．

Case 3. Quiet

まだトークンが割り当てられていなけ

ればトークンを割り当てる．既に割り当てられていれば，トークンは次のリンク探索に向かう．この後，このリンクがOperationalに変わればCase 4.の動作を行う．

Case 4. Linkqueue

まず，そのリンクがRejected，外向枝のいずれかに分類されるまで待ち，Rejectedになれば Case 2.の動作に従う．外向枝と判断された場合は１トークンを割り当てる．後の動作は以下で詳しく説明する．

もしトークンがコアに戻って来たときのノード数が次レベルの必要値を超えていたら，ＧＳＴのラベル更新手続きＩＮＩＴを起動する．

Case 4.で探索リンクが外向枝と判断された場合について考える．あるフラグメントＦのノードＸにおいて１トークンがリンクＬに割り当てられ，Ｌに接するもう一方のフラグメントをＦ’とする．次の２つの場合を考える．

(1)level(F)<=level(F')のとき

ＧＳＴの動作より，レベルが等しければ合併手続き，異なれば吸収手続きにより，いずれにしてもＦのラベル更新が行われる．よってこのレベルのＮＣは自動的にアボートされる．

(2)level(F)>level(F')のとき

Ｌに接するＦ’側のノードをＹとする．ＧＳＴの動作より，ＸはＦ’のレベルがＦのレベルより小さいことを知る．するとＸはＬにトークンを割り当て，これ以降Ｘに到着したトークンはＬをRejectedと同様に扱う．Ｆ’のレベルはより小さいので，今後Ｆ’がＦに吸収されることがあり，この場合吸収したフラグメントのノード数も数える必要がある．吸収が起こった場合の動作は以下のようになる．

吸収の際，ＸはＦ’のノード数$n_{F'}$を知ることができる．よってＦ’のノードが加わることにより次レベルの必要ノード数を超える場合は，Ｆのコアに手続きＩＮＩＴを起動させるようメッセージを送る．超えていなくても他のリンクを通して別のフラグメントが吸収されている可能性があり，全体のノード数を$n_F+n_{F'}\times(2f_l+1)$と見積もる（図7）．

[1]見積もりが次レベルの必要ノード数を超える場合

この場合，放送によるフラグメント内ノード数計算手続き（ＣＮＢ）を起動する．すなわち次レベルに必要なノード数の残りの$1/(2f_l+1)$が加わればＣＮＢを起動することになる．ＣＮＢの動作は以下の通りで，Branchリンクのみを使って行われる．まずコアからメッセージCNodesを現在フラグメントに属している全ノードに放送する．そしてリーフノードは親ノードにノード数１を返し，内部ノードは子ノードすべてから返されたノード数の合計に１を足して親ノードに返す．コアは子のすべてからノード数を受け取ると，それを合計してフラグメントの実際のノード数を知ることができる（図8）．

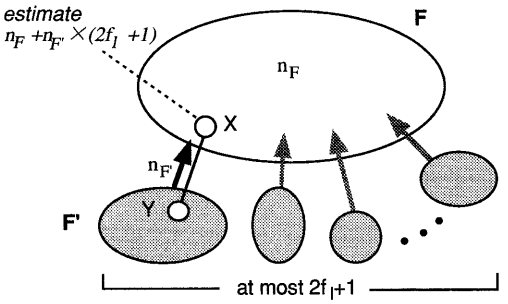


図7　吸収の見積もり

そして実際にフラグメントのラベル更新に必要なノード数を超えていれば手続きＩＮＩＴをＦのコアが起動する．超えていなければ現レベルのまま，実ノード数をもってＮＣを再起動する．このとき，Ｆ’はＦに吸収されており，新しく成長したＦで探索が行われる．

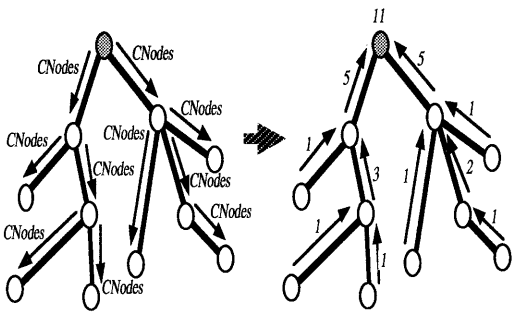


図8　手続きＣＮＢの動作

[2]見積もりが次レベルの必要ノード数を超えない場合

ＹをＦ’部分の仮のルートとしてＮＣを再帰的に起動する（図9）．すなわち元Ｆ’部分上で$2f_l+1$トークンにより深さ優先順で探索し，ＸがＣＮＢを起動するために必要なノード数をＹが集めなければならないノード数とする．元Ｆ’中のノードはその$1/(2f_l+1)$を集めればＦ’部でＣＮＢを起動し，これを繰り返すことによりＹの必要数に達すればＸに伝える．

Branch以外の全リンクがRejectedになると部分木の探索は終了する．Xは新しく加わったノード数をトークンに記録して，このトークンはFの次リンク探索へと進む．訪れたノードではノード数を更新していく．探索の途中で次レベルの必要ノード数に達したときも手続きＩＮＩＴを起動する．

また，トークンは深さ優先探索を行うので１パスで同じノードを２回訪れることになるが，２回目は何も行わず，Branchを使って移動のみを行う．

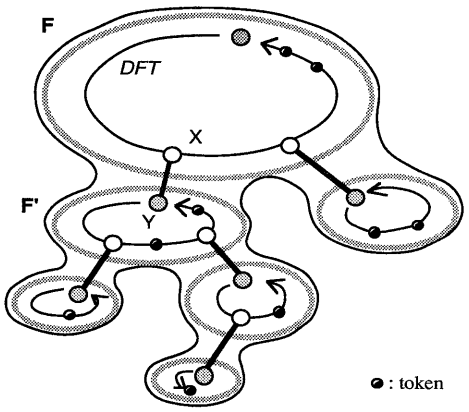


図9　ＮＣの再帰的起動

以上がALEPの動作である．プロセス故障を含むネットワークに対しても，故障したプロセスに接続している全リンクが故障リンクであると考えることにより，ALEPが適用可能である．

3．2　ALEPの正当性の証明

本アルゴリズムはノード数がn/2以上になったフラグメントのコアがリーダ宣言し，全ての処理を強制的に終了させる．従って証明すべき点は，

1．ＧＳＴはいずれ１つのスパニング木を構成する．
2．ＮＣはフラグメントのノード数をコアにおいて正しく計算する．

の２つである．1．に関しては[9]などから明らかなので省略する．

ＧＳＴのラベル更新手続きＩＮＩＴを行う際にルートは自フラグメントのノード数を数えることができるので，2．は以下のように置き換えることができる．

2'．フラグメントのノード数が次レベルに必要なノード数に達したときは，いずれ必ず手続きＩＮＩＴを起動する．

以下ではこれを証明する．

あるフラグメントＦを考え，その上でＮＣを実行することを考える．Ｆのコアをｒ，ノード数n_Fとする．

Ｆに属する枝の種類はBranch，Rejected，Linkqueue，Quietである．トークンｔはＦ上をｒから深さ優先順に探索する．次に探索するリンクがBranchなら深さ優先順に次にノードへ進む．Rejectedなら無視し，そのノードの次のリンクへ探索を進める．Quietで，まだトークンが割り当てられていないリンクなら，トークンを１つそのリンクに割り当てる．既に割り当てられていればトークンはそのノードの次のリンク探索に進む．従って，もしＦの全リンクがBranch，Rejected，Quietのいずれかで，Quietリンクはすべて故障リンクであるとしても，故障リンク数の上限はf_1であり，使用トークン数$2f_1+1$だから少なくとも１トークンはいずれ必ずｒに帰ってくる．よってｒはn_Fを知ることができ，もし次レベルに必要なノード数に達していれば手続きＩＮＩＴを起動する．

次に探索リンクがLinkqueueであるときを考える．これはトークンを割り当てられたQuietリンクが後にLinkqueueになった場合も含む．このリンクＬにはＧＳＴによりTestメッセージが送られ，Rejectedか外向枝に判断される．Rejectedであれば上記と同様，そのノードの次のリンク探索に進む．外向枝と判断された場合，以下の２つの場合が考えられる．

相手側のフラグメントをＦ’とすると，最初の場合はlebel(F)≦lebel(F')のときである（図3(c)）．このときはＬをPossibleOutlinkとしてｒに伝えようとする．もし伝わればＬをOutlinkとしてＦはＦ’に吸収されるか，もしくは合併が起こる（図4(a), 4(b)）．すなわちＦはラベル更新を行うので，その際に$O(n)$のメッセージ数でＦの正しいノード数を計算でき，このレベルにおけるトークンによる探索はアボートしてよい．またｒがＬをOutlinkと選ばなくても，別の外向枝をOutkinkとして吸収または合併が起こるので，同様にこのレベルのトークンによる探索はアボートしてよい．

２番目の場合はlevel(F)＞level(F')のときである．Ｌに接するＦ側のノードをＸ，Ｆ’側のノードをＹとする．ＧＳＴの動作より，ＸはＦ’のレベルがＦのレベルより小さいことを知る．するとＬにトークンを割り当て，これ以降Ｘに到着したトークンはＬを無視する．Ｆ’のレベルは Ｆより小さいのでＦ’がＦに吸収されることがあり，この場合を詳しく考える（図7）．Ｆ’はラベル更新を行うのでその際にＦ’のノード数$n_{F'}$をＸは知ることができる．Ｘは現在のn_Fに$n_{F'}$を加えて，もし次レベルに必要なノード数に達するならば手続きＩＮＩＴを起動するようｒに要請する．もし達していなくても他のリンクを通して別のフラグメントが吸収されている可能性があり，全体のノード数を見積もる必要がある．トークンの数は$2f_1+1$個なのでそうした吸

収は高々$2f_l+1$箇所でしか起こりえない．たとえ１トークンによって何回も吸収が行われていてもその合計はＣＮＢ起動のための必要数を超えていない．従って，全体のノード数を$n_F+n_{F'}\times(2f_l+1)$と見積もり，これが次レベルのノード数を超えていればフラグメント内ノード数計算手続きＣＮＢをｒに起動させる．そして得られた実際のノード数が次レベルの必要数を超えていれば手続きＩＮＩＴを起動する．見積もりでは超えたが実際には超えていなかった場合，得られたノード数を新しいn_Fとして新たにＮＣを起動する．見積もりは$2f_l+1$倍，すなわちトークンをもつ全ノードでノード数が増加していると考えて求めているので，必要数の残りの$1/(2f_l+1)$以上のノードが新しく加わったところが存在するなら必ずＣＮＢが起動される．

また，見積もりが超えなかった場合でもＹを仮のルートとして再帰的にＮＣを起動するので，後にＣＮＢ起動に必要なノード数が増加すればＸは知ることができ，しきい値に達すれば必ずＣＮＢが起動される．すなわち，Ｆのノード数が次レベルに必要なノード数に達した場合は必ず手続きＩＮＩＴが起動されることになる．

また，F'の全リンクがRejectedとなった場合，トークンｔは次のリンク探索に進むので全トークンが最大f_l本のリンクの両端に割り当てられたとしても少なくとも１トークンは動作状態にあり，ＮＣがデッドロックに陥ることはない．

よって2'が証明され，アルゴリズムALEPは正しくＬＥＰを解くことが示された．

3．3　ALEPのメッセージ複雑度の解析

ＧＳＴのメッセージ複雑度はO(m+nlogn)であることは[1]などから明らかなので，ＮＣについてのみ解析を行う．

各レベルで考える．まず，あるフラグメントＦにおいてＮＣが起動されると，$2f_l+1$個（定数）のトークンがＦの深さ優先探索を行うので$(2f_l+1)\times2\times O(n)$のメッセージが使われる．すなわち，トークンによる探索はO(n)メッセージが必要である．

トークンを割り当てられたリンクを通してF'を吸収すれば，F'においてもトークンによる探索が行われる．吸収した元のフラグメント部分のノード数をそれぞれn_iとすると，$\Sigma n_i<n$であり，各フラグメント探索は重なりがないので，見積もりによるトークン探索が行われないかぎりはO(n)メッセージしか必要としない．

従って，ワーストケースは吸収で必要ノード数に達さず，見積もりによりＣＮＢを起動しても最小ノード数しかノードが加わっていない場合が続くときである．あるフラグメントF_0がF_1を吸収し，その後F_1のリンクを通してF_iを吸収したとする（F_iは複数存在する）．F_1について考える．

F_1がF_iを吸収することで，F_iではそれぞれトークンが回されている．（F_0の見積もりのための）F_1の必要ノード数の$1/(2f_l+1)$のノードがF_iに加われば，F_1でＣＮＢが起動する．ＣＮＢ起動ごとに必要残り数は減っていくので，これをO(logn)回繰り返せばF_1は必ず必要数を得ることができ，F_0のＣＮＢを起動させることができる．よって，F_1は必要ノード数を得るために，見積もりによるＣＮＢ起動を高々O(logn)回行う．F_0のＣＮＢを起動させた後，F_1はF_0，F_1を含む新たな深さ優先探索木の一部となる．

ＮＣの動作は，吸収したフラグメントにおいて再帰的にＮＣを起動することがあるので，部分フラグメントが見積もりによるＣＮＢの起動をO(logn)回行って上のレベルの部分フラグメントの必要見積もりノード数を集めることが，鎖状になったフラグメントの数だけ起こる可能性がある．従って１つのフラグメントにおいて１レベルの間に最も多くのメッセージが使われるのは以下のような場合である．

(logn-1)レベルになると，あとはO(n)メッセージでリーダ宣言し終了するので，１レベルの間にレベル(logn-2)のフラグメントがレベル(logn-3)のフラグメントを吸収し，このレベル(logn-3)のフラグメントがレベル(logn-4)のフラグメントを吸収し，これが連続してレベル２，１，０と鎖上に連続吸収が行われた場合にメッセージ数最大となる（図10）．このフラグメントの各部分を$F_{(logn-2)}$，$F_{(logn-3)}$，・・・，F_2，F_1，F_0とする．$F_{(logn-2)}$において最多メッセージが使われるのは，上の議論より$F_{(logn-2)}$上でＣＮＢが(logn-2)回起動されるときで，$F_{(logn-2)}$のノード数$n_{F(logn-2)}$とすると$n_{F(logn-2)}\times$(logn-2)のメッセージが$F_{(logn-2)}$で使われる．次に，$F_{(logn-3)}$において最多メッセージが使われるのは，$F_{(logn-3)}$上でＣＮＢが(logn-3)回起動されることにより$F_{(logn-2)}$の見積もりが必要数を超えて$F_{(logn-2)}$のＣＮＢを起動するが，実際には必要数を満たさないときである．いったん$F_{(logn-2)}$のＣＮＢを起動すると$F_{(logn-3)}$は$F_{(logn-2)}$の探索木の一部となるので，$F_{(logn-2)}$と結合した後のＣＮＢ起動を考えて，$F_{(logn-3)}$で使われる最多メッセージ数は$n_{F(logn-3)}\times$((logn-3)+((logn-2)-1))となる．同様に，F_iにおいて使用される最大メッセージ数は，

$$n_{Fi}\times\{i+((i+1)-1)+((i+2)-1)+\cdots+((logn-3)-1)+((logn-2)-1)\}$$
$$=n_{Fi}\times\{(logn-3)+(logn-4)+(logn-5)+\cdots+(i+2)+(i+1)+i\times2\}$$

となる．部分フラグメントの鎖の長さの最大はlogn-1である．これが起こるのは$n_{F(\log n-2)}$，$n_{F(\log n-3}$，・・・，n_2，n_1，n_0がそれぞれn/2，n/4，・・・，8，4，2より小さいときである．もしいずれかがこれらの値より大きければ途中でラベル更新が起こるはずである．従って，1レベルにおける最大メッセージ数は，

n/2×(logn-2)

n/2 ×(logn-3)

n/8 ×(logn-3) + n/4 ×(logn-4)

n/16×(logn-3) + n/16×(logn-4) + n/8 ×(logn-5)

n/32×(logn-3) + n/32×(logn-4) + n/32×(logn-5) + n/16×(logn-6)

:

8 ×(logn-3) + 8 ×(logn-4) + 8 ×(logn-5) +

4 ×(logn-3) + 4 ×(logn-4) + 4 ×(logn-5) + ・・・

+)2 ×(logn-3) + 2 ×(logn-4) + 2 ×(logn-5) +

<n/2×(logn-2)+

n ×(logn-3) + n/2 ×(logn-4) + n/4 ×(logn-5) + n/8 × (logn-6)

+・・・

< nlogn×(1/2+1+1/2+1/4+1/8+...)

= 2.5×nlogn

= O(nlogn)

となる．

また，複数のフラグメントが存在する場合を考える．上の議論でフラグメントの最大レベルを1下げると，各部分フラグメントのノード数は半分以下，レベル数も1ずつ下がるので最大メッセージ数は先の半分以下である．こうしたフラグメントの数は高々2倍にしかならないので，いずれの場合も1レベルで使われるメッセージはO(nlogn)の上限を超えることはない．

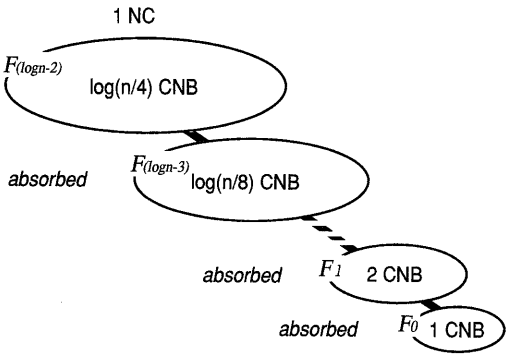


図10　最大メッセージ数を要するフラグメント吸収

ＧＳＴより，いずれのノード，リンクとも高々lognレベルにしか属さず，レベルの更新は単調増加なのでＮＣで使用される総メッセージ数はlogn×O(nlogn)=O(nlog^{2}n)となる．ＧＳＴで使われるメッセージ数はO(m+nlogn)だから，アルゴリズムALEPのメッセージ複雑度はO(m+nlog^{2}n)，メッセージサイズO(logn)ビットとなる．

4．固有の識別子をもたず，故障を含むネットワークについて

これまでに固有の識別子をもたず，かつ故障を含むネットワークにおけるＬＥＰについての結果は知られていない．本節ではG_{fa}とG_{fan}について考察を行う．まず，G_{fa}では，G_a及びG_fにおいての結果から解けないことは明らかである．次にG_{fan}においてもＬＥＰが解けないことを示す．

まず，G_{fan}においてＬＥＰが条件(A)で解けるための条件を考える．あるプロセスが他のプロセスをリーダとして承認を与えるということは，局所状態の差異による順序関係がつくことである．また，全体の過半数のプロセスからリーダと承認を受けたプロセスが1つでも存在するとき，残りのプロセスは決して過半数から承認を得ることはない．各プロセスはnを既知なので，結局G_{fan}においてＬＥＰが解けるための条件は全体の過半数のプロセス間で区別（存在の認識）がつくことである．

G_{fan}の非可解性を示す準備としてG_fにおいてＬＥＰが解けないことを示す．図11のリングG_2（$\in G_f$）を考える．リンクＬは故障リンクとする．G_2上でＬＥＰを正しく解くアルゴリズムＡが存在するとする．また，G_3はG_2をＬで切ったリング2つを対称的に配置し，一方は故障リンクで，もう一方は遅延の非常に大きなリンクで結合したものである．アルゴリズムＡをG_3に適用することを考える．ＡがG_2で条件(A)を満たすとすると，G_3の左右両部分でリーダを選んでしまうので矛盾する．また，ＡがG_2において(A)を満たさず(B)を満たすとする．Ａを適用すると条件(A)で解けるような別のリングを切ってG_3の右半分に置き換えると，左半分は解けなかったと判断してしまい，矛盾する．よってG_fにおいてＬＥＰを解くことはできない．

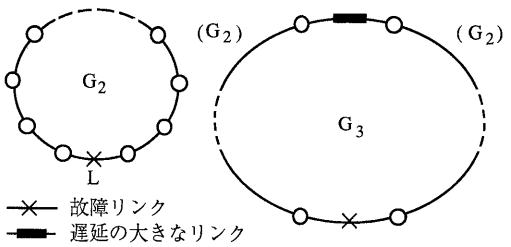


図11　G_fにおけるＬＥＰの非可解性

あるネットワークG_4（$\in G_{fan}$）を考える．各プロセスの識別子は固有ではないので，同じ識別子をもつ場合があり，そのときはポート数，ポートラベリングなどの局所的な情報が全く同じプロセスを区別することは不可能である．ＬＥＰを解くためのあるアルゴリズムA'が終了したとき，あるプロセスＰが識別できたプロセス集合からなる部分グラフをG_dとする．G_dに含まれるプロセス数が全体の過半数を超えるようなG_dが存在すれば条件(A)で解ける．しかし，そうしたG_dが存在しない場合は各G_dで条件(B)を満たす必要がある．

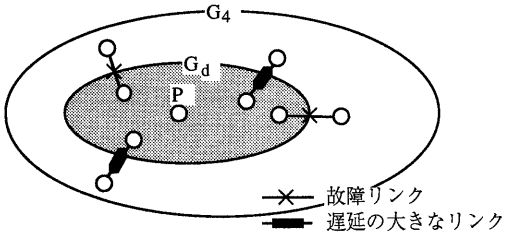


図12　識別可能な部分グラフG_d

ここで図12のように考えると，G_dにおいてＬＥＰを解くことはG_fにおいてＬＥＰを解くことと等価である．ところがG_fにおいてはＬＥＰを解くことは不可能であり，解けなかったことの検知はできない．結局G_4（$\in G_{fan}$）においては条件(A)を満足しない場合が存在し，かつその場合条件(B)を満足できないので，G_{fan}においてＬＥＰは解けない．

このことはランダム化アルゴリズムを用いても同じことが言える．ランダム化アルゴリズムとは各プロセスが独立に確率的選択を行えるアルゴリズムである．ここで独立な確率的選択とは識別子の変更のみに対して行われると仮定する．もし他の動作を確率的に行いたければ識別子に依存して決定的動作を行う手続きで代用できる．また識別子以外の値などを確率的に選択するとしても，識別子を元の識別子とその値の２項組にするなどすれば代用でき，この仮定は一般性を失わない．

ランダム化アルゴリズムA"を用いてG_{fan}においてＬＥＰを解くことを考える．しかし，識別子変更を行っても新たに区別可能となるプロセスが増えるとは限らない．よって条件(A)を満たさない場合は存在する．また，G_fにおいてはランダム化アルゴリズムを用いてもＬＥＰが解けないことは先の議論からも明らかである．従ってG_{fan}においてはランダム化アルゴリズムを用いてもＬＥＰは解けない．

もちろんアルゴリズムをメッセージドリブンではなく，自発的に識別子を変更して送信するようなものにすればG_{fan}においてもＬＥＰは解ける．しかしメッセージ遅延は無限大にしない限りいくらでも大きく考えることができ，送信間隔は遅延に比べいくらでも小さくなる．したがってメッセージ数の上限は存在しないことになる．

5．むすび

本稿では故障を含むと仮定した任意形状の非同期ネットワークにおけるリーダ選挙問題について，各プロセスが固有の識別子をもつ場合ともたない場合について考察を行った．固有の識別子をもつ場合については故障リンク数の上限値が与えられたとした場合に対してメッセージ数$O(m+n\log^2 n)$のアルゴリズムの提案，もたない場合については非可解性の証明を行った．

今後の課題としては提案アルゴリズムALEPのメッセージ複雑度の改善，また，今回はすべての故障がアルゴリズム開始時に起こっているとしたので形状が動的に変化するネットワークにおけるＬＥＰの考察などが挙げられる．

最後に貴重な御意見を頂いた本学の山下雅史先生に深謝致します．本研究の成果の一部は文部省科学研究費補助金一般研究（Ｂ）（課題番号04452195）による．

文献

[1]Afek, Y. and Saks, M.:"Detecting global termination conditions in the face of uncertainty," Proc. 6th Ann. ACM Symp. on Principles of Distributed Computing, pp.109-124 (1987).

[2]Angluin, D.:"Local and global properties in networks of processors," Proc. 12th Ann. ACM Symp. on Theory of Computing, pp.82-93 (1980).

[3]Bar-Yehuda, R., Kutten, S., Wolfstahl, Y. and Zaks, S.:"Making distributed spanning tree algorithms fault-resilient," Proc. 4th Symp. on Theoretical Aspects of Computer Science, pp.432-444, LNCS 247, Springer-Verlag (1987).

[4]Goldreich, O. and Shrira, L.:"Electing a leader in a ring with link failures," Acta Informatica, Vol. 24, pp.79-91 (1987).

[5]Itai, A. and Rodeh, M.:"Symmetry breaking in distributive networks," Proc. 22nd IEEE Symp. on Foundations of Computer Science, pp.150-158 (1981).

[6]Lavallee, I. and Lavault, C.:"Spanning tree construction for nameless networks," 4th International Workshop on Distributed Algorithms, pp.41-56, LNCS 486, Springer-Verlag (1990).

[7]Matias, Y. and Afek, Y.:"Simple and efficient election algorithms for anonymous networks," 3rd International Workshop on Distributed Algorithms, pp.183-194, LNCS 392, Springer-Verlag (1989).

[8]Schieber, B. and Snir, M.:"Calling names on nameless networks," Proc. 8th Ann. ACM Symp. on Principles of Distributed Computing, pp.319-328 (1989).

[9]Welch, J., Lamport, L. and Lynch, N.:"A lattice-structured proof technique applied to a minimum spanning tree algorithm," Proc. 7th Ann. ACM Symp. on Principles of Distibuted Computing, pp.28-43 (1988).

[10]萩原，増澤:"分散アルゴリズム," 情報処理学会誌，Vol.31, No.9, pp.1245-1256 (1990).

[11]亀田，山下:"分散アルゴリズム," 出版予定.