1. $CNet(G)$ 構築・維持アルゴリズム ー Move-In，Move-Out

クラスタ構造 $CNet(G)$ を構築・維持するために各ノードが実行する二つのアルゴリズム Move-In，Move-Out について解説する．新たなノードの発生時には Move-In アルゴリズムを，ノードの離脱時には Move-Out アルゴリズムを実行することで，$CNet(G)$ の構造を維持することが可能である．ただし，アルゴリズムの実行中に更にノードが発生または離脱することはないものとする．

1.1 ノードが保持する情報

提案アルゴリズムを実行するために，無線センサネットワークを構成する各ノード v は以下の情報を保持している．ただし，G 中でノード v が属する部分ネットワークを G_v とする．

・*v.status* ：ノードのステータス．「*head*」，「*gateway*」，「*member*」いずれかの値を持つ．*head* がヘッド，*gateway* がゲートウェイ，*member* がメンバにそれぞれ対応する．
・*v.prt* ：$CNet(G_v)$ 上での親ノードの ID.
・*v.chd* ：$CNet(G_v)$ 上での子ノードの ID の集合.
・*v.oneigh* ：$CNet(G_v)$ 上での親 (*v.prt*) および子 (*v.chd*) を除いた G_v 上での隣接ノードの ID の集合.
・*v.bneigh* ：$BTree(G_v)$ 上での隣接ノードの ID の集合.
・*v.root* ：$CNet(G_v)$ 上でのルートノードの ID(ルート ID).
・*v.nstatus* ：G_v 上で隣接するノードのステータスの集合.
・*v.minID(status)* ：v と G 上で隣接するノードのうちステータスが *status* の物で，一番小さな ID を持つノードの ID.

1.2 Move-In アルゴリズム

1.2.1 Move-In アルゴリズムで使用するメッセージ・述語

Move-In アルゴリズム中で各ノードが送信するメッセージや述語を説明する．ここで，新たに発生するノードを NEW とする．

・*AddMe* ：新規に発生するノードが送信するメッセージ
AddMe.ID := 自分の ID
・*Mystatus* ：AddMe メッセージを受信したノードがが送信するメッセージ
Mystatus.ID := 自分の ID
Mystatus.root := 自分のルート ID
・*ChangeMyStatus* ノードがステータスを変更した事を隣接ノードに連絡するメッセージ
ChangeMyStatus.ID := 自分の ID
ChangeMyStatus.status := 自分の新しいステータス

・*ImYourChild* ：送信元のノードが宛先のノードの子供になる際に用いるメッセージ
ImYourChild.status := 自分のステータス
ImYourChild.ID := 自分の ID
ImYourChild.To := 宛先の ID
・*ImYourParent* ：送信元のノードが宛先のノードの親になる際に用いるメッセージ
ImYourParent.status := 自分のステータス
ImYourParent.ID := 自分の ID
ImYourParent.To := 宛先の ID
ImYourParent.root := 自分のルート ID
MinID(status) ：隣接するノードのうちステータスが status の物で，一番小さな ID を持つノードの ID

1.2.2 Move-In アルゴリズム

Move-In アルゴリズムを図 1 に示す.

1.3 提案する Move-In アルゴリズムの正当性の証明

$CNet$ の性質を再掲する.

性質 1. $CNet(G)$ にはヘッド，ゲートウェイ，メンバが存在する．G 上でヘッド同士は隣接する事はなく，ゲートウェイは $CNet(G)$ 上で必ず一つ以上のヘッドとの間にリンクを持つ．また，$CNet(G)$ 上でメンバは一つのヘッドとの間にのみリンクを持つ.

性質 2. 各部分ネットワーク中，一つのヘッドをルートと呼ぶ.

性質 3. ヘッドとゲートウェイからなる木構造をバックボーンツリーと呼び，そのサイズはヘッドの個数を P としたとき高々 $3P-2$ 個である.

Move-In アルゴリズムの実行後もこれらの $CNet$ の性質が満たされている事を以下に示す.

補題 1. 提案アルゴリズムの実行により複数の部分クラスタネットワークが一つに統一されたとき，ヘッド同士は隣接しない

証明.
新たなヘッドが発生するのは Move-In アルゴリズム 8 行,21 行,28 行,46,53 行により NEW がヘッドになる場合だけであり，この時 NEW の隣接ノードにヘッドが存在しないことは，アルゴリズムより明らかである. □

補題 2. 提案アルゴリズムの実行により複数の部分クラスタネットワークが一つに統一されたとき，全てのゲートウェイの隣に少なくとも一つのヘッドが存在する

証明.
Move-In アルゴリズム 38 行より NEW がゲートウェイになる場合，その隣には必ずヘッドは存在する．また，NEW の親もしくは子供として選択されたがメンバだったとき，選択されたノードは Move-In アルゴリズム 90 行,103 行にてステータスがゲートウェイに変化する．この時メンバの定義より，その隣には必ずヘッドが存在する. □

補題 3. 提案アルゴリズムの実行により複数の部分クラスタネットワークが一つに統一されたとき，メンバは一つのヘッドとの間にのみ辺を持つ.

証明.
Move-In アルゴリズム 14 行より NEW がメンバになる時，この時 NEW は親であるヘッドとの間にのみ辺を持つ．また，部分クラスタネットワークの統合によりメンバが子供を持つ場合，Move-In アルゴリズム 90 行,103 行より新たに子供を持つメンバのステータスはゲートウェイに変化する．従って，メンバは一つのヘッドとの間にのみ辺を持つ. □

補題 4. 提案アルゴリズムの実行により複数の部分クラスタネットワークが一つに統一されたとき，全てのノードが保持するルート ID が等しい

Algorithm Move-In

• NEW が実行する動作

```
001   CNet に参加するノード NEW は隣接するノードに 1 から q まで
      番号を割り振り，1 から q まで順番に AddMe メッセージを送信；
002   WHILE(隣接する q 個のノード中に AddMe メッセージ未送信のノードが存在){
003     NEW は AddMe メッセージを未送信のノードへ AddMe メッセージを送信する；
004     MyStatus メッセージの受信を待つ；
005   }
006 IF( NEW がルート ID を一つも受信せず )
007   THEN
008     NEW.status := head；
009     NEW.root := NEW.ID；
010 ELSE IF( NEW がルート ID を一つのみ受信 )
011   THEN
012     IF(NEW と隣接するノードの中に head が存在)
013       THEN
014         NEW.status := member；
015         MinID(head) に ImYourChild メッセージを送信；
016         NEW.root := MinID(head) のルート ID；
017         NEW.prt := MinID(head)；
018         NEW.oneigh := MinID(head) 以外の隣接ノードの ID；
019     ELSE IF(NEW と隣接するノードの中に gateway が存在)
020       THEN
021         NEW.status := head；
022         MinID(gateway) に ImYourChild メッセージを送信；
023         NEW.root := MinID(gateway) のルート ID；
024         NEW.prt := MinID(gateway)；
025         NEW.bneigh := MinID(gateway)；
026         NEW.oneigh := MinID(gateway) 以外の隣接ノード；
027     ELSE
028         NEW.status := head；
029         MinID(member) に ImYourChild メッセージを送信；
030         NEW.root := MinID(member) のルート ID；
031         NEW.prt := MinID(member)；
032         NEW.bneigh := MinID(member)；
033         NEW.oneigh := MinID(member) 以外の隣接ノード；
034 ELSE //NEW が複数のルート ID を受信
035   THEN
036     IF(NEW と隣接するノードの中に head が存在)
037       THEN
038         NEW.status := gateway；
039         MinID(head) に ImYourChild メッセージを送信；
040         NEW.root := MinID(head) のルート ID；
041         NEW.prt := MinID(head)；
042         NEW.bneigh := MinID(head)；
043         NEW.oneigh := MinID(head) 以外の隣接ノードの ID；
044     ELSE IF(NEW と隣接するノードの中に gateway が存在)
045       THEN
046         NEW.status := head；
047         MinID(gateway) に ImYourChild メッセージを送信；
048         NEW.root := MinID(gateway) のルート ID；
049         NEW.prt := MinID(gateway)；
050         NEW.bneigh := MinID(gateway)；
051         NEW.oneigh := MinID(gateway) 以外の隣接ノード；
052     ELSE
053         NEW.status := head；
054         MinID(member) に ImYourChild メッセージを送信；
055         NEW.root := MinID(member) のルート ID；
056         NEW.prt := MinID(member)；
057         NEW.bneigh := MinID(member)；
058         NEW.oneigh := MinID(member) 以外の隣接ノード；
059 WHILE(NEW の親または子供が未選択の CNet が存在する){
060 IF(NEW の親または子供を未選択の CNet 中に NEW と隣接する head が存在)
061   THEN
062     親と子供を未選択の CNet 中での MinID(head) に
      ImYourParent メッセージを送信；
063     NEW.chd に親または子供を未選択の CNet 中での MinID(head) を追加；
064     NEW.bneigh に親または子供を未選択の CNet 中での MinID(head) を追加；
065     NEW.oneigh から MinID(head) を削除；
066 ELSE IF(NEW の親または子供を未選択な CNet 中に
      NEW と隣接する gateway が存在)
067 THEN
068   親と子供を未選択の CNet 中での MinID(gateway) に
      ImYourParent メッセージを送信；
069   NEW.chd に親または子供を未選択の CNet 中での MinID(gateway) を追加；
070   NEW.bneigh に親または子供を未選択の CNet 中での MinID(gateway) 追加；
071   NEW.oneigh から MinID(gateway) を削除；
072 ELSE
073   親と子供を未選択な CNet 中での MinID(member) に
      ImYourParent メッセージを送信；
074   NEW.chd に親または子供を未選択な CNet 中での MinID(member) を追加；
075   NEW.bneigh に親または子供を未選択な CNet 中での MinID(member) 追加；
076   NEW.oneigh から MinID(member) を削除；
077 } END WHILE
```

• NEW 以外のノード Pi が実行する動作

```
078 IF(AddMe メッセージを受信)
079   THEN
080     NEW へ MyStatus メッセージを送信；
081 IF(ImYourChild メッセージを受信)
082   THEN
083   IF(ImYourChild.To == Pi.ID)
084     THEN
085       Pi.chd := Pi.chd ∪ ImYourChild.ID；
086       IF( ImYourChild.status が head か gateway )
087         Pi.bneigh := Pi.chd ∪ ImYourChild.ID；
088       IF(Pi.status == member)
089         THEN
090           Pi.status := gateway；
091           Pi.bneigh := Pi.prt；
092           Pi は ChangeMyStatus メッセージを送信；
093   ELSE
094   Pi.oneigh := Pi.oneigh ∪ ImYourChild.ID；
095   ID が ImYourChild.ID のノードのステータスを ImYourChild.status に変更；
096 IF(ImYourParent メッセージを受信)
097   THEN
098     IF(ImYourParent.To == Pi.ID)
099       THEN
100         Pi.root := ImYourParent.root；
101         IF(Pi.status == member)
102           THEN
103             Pi.status := gateway；
104             Pi.bneigh に Pi.prt と ImYourParent.ID を追加
105             Pi は ChangeMyStatus メッセージを送信；
106             Pi.prt へ ImYourParent メッセージを送信；
107             Pi.chd := Pi.prt；
108             Pi.prt := ImYourParent.ID；
109         ELSE IF(Pi が CNet のルート)
110           THEN
111             Pi.chd に ImYourParent.ID があればこれを削除；
112             Pi.prt := ImYourParent.ID；
113             Pi.root を Broad を用いてブロードキャスト；
114         ELSE
115             Pi.chd に ImYourParent.ID があればこれを削除；
116             Pi.prt へ ImYourParent メッセージを送信；
117             Pi.chd := Pi.chd ∪ Pi.prt；
118             Pi.prt := ImYourParent.ID；
119 IF(ChangeMyStatus メッセージを受信した)
120 THEN
121   ID が ChangeMyStatus.ID のノードのステータスを
      ChangeMyStatus.status に変更；
122   IF(Pi.chd に ChangeMyStatus.ID が含まれる)
123     Pi.bneigh := Pi.bneigh ∪ ChangeMyStatus.ID；
```

図 1　Move-In アルゴリズム
Fig. 1　Move-In Algorithm

証明.
提案アルゴリズムによる部分クラスタネットワークの統合では，NEW の親として選択された部分クラスタネットワークのルートが統合後全体のルートとなる．このルート ID が NEW を介して統合される部分クラスタネットワークにブロードキャストされる事で，統合後の一つになった部分クラスタネットワークにおいてルート ID を一つだけにしている． □

補題 5. 提案アルゴリズムの実行により複数の部分クラスタネットワークが一つに統一された部分クラスタネットワークは，一つのヘッドを根とした木構造となる

証明.
NEW が子供を選択した時，子供となったノードは Move-In アルゴリズムより，所属していた部分クラスタネットワークのルートまで親子関係が入れ替わる．このように親子関係を逆転させることで，全ての統合される部分クラスタネットワークは NEW を根とする一つの木構造となる． □

定理 1. Move-In アルゴリズムにより複数の部分クラスタネットワークの統合が行われた結果，新しくできるネットワークも部分クラスタネットワークの性質を満たしている

証明.
補題 1，補題 2 より，ヘッド同士は隣接せず，またゲートウェイの隣には必ず一つ以上のヘッドが存在する．また，補題 3 より，メンバは一つのヘッドとの間にのみ辺を持つ．従って，$CNet$ の性質 1 は満たされる．

補題 4，補題 5 より，$CNet$ の性質 2 は満たされる．

補題 2 より，ゲートウェイの隣には必ず一つ以上のヘッドが存在する．従って，バックボーンツリーのサイズはヘッドの個数を P 個とした時高々$3P-2$ 個となり $CNet$ の性質 3 は満たされる．

従って，Move-In アルゴリズムの実行後，$CNet$ の性質は満たされる． □

定理 2. *Move-In* アルゴリズムは，無線ネットワーク中に新たにノードが発生したときに，$CNet(G)$ を正しく構築・維持するアルゴリズムであり，この時実行時間は $O(q + max\{$ 統合される部分クラスタネットワークの *BTree(G)* のサイズ *(*ノード数*)}*$)$ 時間となる

証明.
NEW は隣接ノードに対して 1 から q まで $O(q)$ ラウンドで番号を割り振る．その後，それら q 個の隣接ノードがステータス及びルート ID を NEW に送信するのに $O(q)$ ラウンド必要である．更に，NEW は親と子供を選択するメッセージを送信するのに $O(q)$ ラウンド必要である．加えて NEW の子供となった部分クラスタネットワークへ新しいルート ID をブロードキャストし，ルートまで親子関係を逆転するが，これはそれぞれの部分木に対して O(統合される部分クラスタネットワークの BTree(G) のサイズ) ラウンドで実行可能である．従って，全ての統合される部分クラスタネットワークに同時にブロードキャストを実行する事で，実際に必要なのは統合される部分クラスタネットワークのうち最大の物が完了するまでの実行時間である．以上より，Move-In アルゴリズムの実行に必要な実行時間は，NEW に隣接するノードの数を q とする時 O(q + max{ 統合される部分クラスタネットワークの BTree(G) のサイズ }) 時間となる． □

1.4 Move-Out アルゴリズム

1.4.1 Move-Out アルゴリズムで使用するメッセージ・関数

Move-Out アルゴリズム中で各ノードが送信するメッセージや述語を説明する．

・*Leave* ：$CNet$ から退出するノード DEL が送信メッセージ
Leave.ID := 自分の ID
Leave.status := 自分のステータス

・*ChangeMyStatus* ノードがステータスを変更した事を隣接ノードに連絡するメッセージ
ChangeMyStatus.ID := 自分の ID
ChangeMyStatus.status := 自分の新しいステータス

・*CheckYourChd* ：宛先のノードに子供が存在するかを確認させる時に用いるメッセージ
CheckYourChd.ID := 自分の ID
CheckYourChd.To := 宛先の ID

・*ChangeRootID* DEL.chd を根とする部分木のルート ID を変更する時に用いるメッセージ
ChangeRootID.ID := 自分の ID
ChangeRootID.To := 宛先の ID
ChangeRootID.root := 自分のルート ID

・*CheckYourNeighbors* ：DEL.chd を根とする部分木の隣接状況を調べるときに用いるメッセージ
ChechYourNeighbors.list := 隣接する部分木のルート ID を集めるリスト型の変数
ChechYourNeighbors.To := 宛先の ID

・*MergeToNeighbor.ID* ：送信元のノードが宛先のノードの親になる際に用いるメッセージ
MergeToNeighbor.ID := 統合先の部分木が持つルート ID
MergeToNeighbor.To := 宛先の ID
MergeToNeighbor.flag := 統合が実行済みかどうかを示すフラグ．実行済みの時 TRUE，未実行の時 FALSE

・*Independence* ：DEL.chd を根とする部分木を独立させる時に用いるメッセージ
Independence.To := 宛先の ID

・*ImYourChild* ：送信元のノードが宛先のノードの子供になる際に用いるメッセージ
ImYourChild.status := 自分のステータス
ImYourChild.ID := 自分の ID
ImYourChild.To := 宛先の ID

・*ImYourParent* ：送信元のノードが宛先のノードの親になる際に用いるメッセージ
ImYourParent.ID := 自分の ID

ImYourParent.To := 宛先の ID

・*MinID(status)* ：隣接するノードのうちステータスが status の物で，一番小さな ID を持つノードの ID
・*MinID(status,root)* ：隣接するノードのうちルート ID が root かつステータスが status の物で，一番小さな ID を持つノードの ID

Move-Out アルゴリズム

提案する Move-Out アルゴリズムを図 2 に示す.

1.5 提案する Move-Out アルゴリズムの正当性の証明

DEL が $CNet(G)$ より離脱する事により，次の条件を一つでも満たすノードは $CNet(G)$ を構成するために保持する情報の内容が変化する可能性がある.

条件 1　DEL.prt,(DEL.prt).prt,DEL.chd のいずれか

DEL が離脱する事により，隣接するヘッドが居なくなってしまうノードや次数が 1 となってしまうゲートウェイ等 $CNet$ の定義に反する事になるノードが発生する事がある．Move-Out アルゴリズムはそのようなノードが保持する情報を $CNet$ の性質を満たすような正しい内容へと変化させなければならない.

条件 2　DEL.chd を根とする部分木に属するノード (T1,T2,T3,...,Tq)

DEL.chd を根とする部分木は G 上において互いに非連結になる可能性があるため，これらの部分木に属するノードは保持する情報が変化する可能性がある．また，H へ統合されるときに部分木内部の親子関係が変化する可能性がある.

条件 3　H の中で条件 2 を満たすノードに隣接するノード

H に属するノードのうち，DEL を根とする $CNet(G)$ 上の部分木中に存在するノード (v とする) と隣接するノードは $CNet$ の統合時に v を子供とする可能性があるため，そのノードのステータスが変化する可能性がある.

従って，これらのノードがアルゴリズムの実行後も $CNet$ の定義を満たしている事を本節で示す.

補題 6. DEL がヘッドの時，提案アルゴリズムの実行後 DEL.prt,(DEL.prt).prt のステータスは $CNet(G)$ の性質 1 に反しない

証明.

$CNet$ を構成するためにノードが保持する情報の内容より，DEL.prt は以下の条件の組み合わせにより分類する事ができる．ただし，DEL がヘッドの時 DEL.prt は $CNet$ の定義よりゲートウェイである．これにより、ゲートウェイの親となる (DEL.prt).prt のステータスはヘッドかゲートウェイのいずれかである.

条件 1. *(DEL.prt).prt* のステータスがヘッドかゲートウェイか

(a) ヘッド　*(b)* ゲートウェイ

条件 2. *DEL.prt* は *DEL* 以外に子供を持つか持たないか

(a) 持つ　*(b)* 持たない

条件 3. *DEL.prt* は子供に *DEL* 以外のヘッドを持つか持たないか

(a) 持つ　*(b)* 持たない

条件 4. *DEL.prt* は G 上の隣接ノードに *DEL* 以外のヘッドを持つか持たないか

(a) 持つ　*(b)* 持たない

条件 1〜4 の組み合わせを表 1 に示す.

表 1　DEL がヘッドの時の DEL.prt

No	条件 1	条件 2	条件 3	条件 4
1	a	a	a	a
2	a	a	a	b
3	a	a	b	a
4	a	a	b	b
5	a	b	a	a
6	a	b	a	b
7	a	b	b	a
8	a	b	b	b
9	b	a	a	a
10	b	a	a	b
11	b	a	b	a
12	b	a	b	b
13	b	b	a	a
14	b	b	a	b
15	b	b	b	a
16	b	b	b	b

DEL がヘッドの時に提案する Move-Out アルゴリズムを実行すると，表 1 に示した 16 通りの DEL.prt は以下のように動作する.

No.1　**DEL.prt はアルゴリズムの動作後ゲートウェイのままとなる**

この DEL.prt は DEL の離脱後，親にヘッドを持ち，同時に子供も持つため，Move-Out アルゴリズムによる影響を受けない．この時 DEL.prt はゲートウェイのままで CNet(G) の性質 1 を満たす.

No.2　**DEL.prt はアルゴリズムの動作後ゲートウェイのままとなる**

No.1 と同様である.

No.3　**DEL.prt はアルゴリズムの動作後ゲートウェイのままとなる**

No.1 と同様である.

No.4　**DEL.prt はアルゴリズムの動作後ゲートウェイのままとなる**

No.1 と同様である.

No.5　**DEL.prt はアルゴリズムの動作後メンバになる**

この DEL.prt は DEL の離脱後，親にヘッドを持つが子供を持たないので，Move-Out アルゴリズムの 41 行によってステータスがメンバになり CNet(G) の性質 1 を満たす.

No.6　**DEL.prt はアルゴリズムの動作後メンバになる**

No.5 と同様である.

No.7　**DEL.prt はアルゴリズムの動作後メンバになる**

No.5 と同様である.

No.8　**DEL.prt はアルゴリズムの動作後メンバになる**

No.5 と同様である.

No.9　**DEL.prt はアルゴリズムの動作後ゲートウェイのままとなる**

DEL の離脱後，DEL.prt は親がゲートウェイだが子供にヘッドを持つため，Move-Out アルゴリズムによる影響を受けない.

この時 DEL.prt はゲートウェイのままで CNet(G) の性質 1 を満たす.

No.10　**DEL.prt はアルゴリズムの動作後ゲートウェイのまま**

No.9 と同様である.

No.11　**DEL.prt はアルゴリズムの動作後親が変更される**

DEL の離脱後，DEL.prt は親にも子供にもヘッドが存在しなくなるが DEL.oneigh の中にヘッドが存在するため，Move-Out アルゴリズム 35 行によって親を変更することで，DEL.prt はステータスをゲートウェイのまま CNet（G) の性質 1 を満たす．この時，ゲートウェイである（DEL.prt）.prt に子供がいなくなるとき，そのステータスも Move-Out アルゴリズム 144 行によってメンバへと変更されるため，この時 CNet(G) の性質 1 を満たす.

No.12　**DEL.prt はアルゴリズムの動作後ヘッドになる**

DEL の離脱後，DEL.prt の隣接ノードの中にヘッドは存在しなくなるため，DEL.prt が Move-Out アルゴリズム 37 行によってステータスがヘッドになり CNet（G) の性質 1 を満たす.

No.13　**DEL.prt はアルゴリズムの動作後メンバになり，親が変更される**

DEL の離脱後，DEL.prt は親がゲートウェイであり子供を持たないが DEL.oneigh の中にヘッドが存在するため，DEL.prt は Move-Out アルゴリズム 25 行,27 行によってステータスをメンバにし，親を変更することで CNet（G) の性質 1 を満たす．この時,（DEL.prt）.prt に子供がいなくなるとき，そのステータスも Move-Out アルゴリズム 144 行によって変更されるため，CNet(G) の性質 1 を満たす.

No.14　**DEL.prt はアルゴリズムの動作後ヘッドになる**

DEL の離脱後，DEL.prt の隣接ノードの中にヘッドは存在しなくなるため，Move-Out アルゴリズム 40 行によってステータスがヘッドになり CNet（G) の性質 1 を満たす.

No.15　**DEL.prt はアルゴリズムの動作後メンバになり，親が変更される**

No.13 と同様である.

No.16　**DEL.prt はアルゴリズムの動作後ヘッドになる**

No.14 と同様である.

従って，いずれの場合も Move-Out アルゴリズムの実行により，Move-Out アルゴリズムの実行後 DEL.prt,(DEL.prt).prt のステータスは CNet の性質 1 に反しない.　□

補題 7. *DEL* がヘッドの時，Move-Out アルゴリズムの実行後 *DEL*.chd のステータスは $CNet(G)$ の性質 1 に反しない

証明.

CNet を構成するためにノードが保持する情報の内容より，*DEL*.chd は以下の条件の組み合わせにより分類する事ができる．ただし *DEL* がヘッドの時，*DEL*.chd は *CNet* の定義よりゲートウェイもしくはメンバである.

条件 1. *DEL.chd* のステータスがゲートウェイかメンバか

(a) ゲートウェイ　*(b)* メンバ

条件 2. *DEL.chd* が持つ子供の数

(a)2 個以上持つ　*(b)1* 個しか持たない

条件 3. *DEL.chd* は隣接ノードに *DEL* 以外のヘッドを持つか持たないか

(a) 持つ　*(b)* 持たない

条件 1〜3 の組み合わせを表 2 に示す.

表 2　ヘッドが退出時の DEL.chd への影響

No	条件 1	条件 2	条件 3
1	a	a	a
2	a	a	b
3	a	b	a
4	a	b	b
5	b	a	a
6	b	a	b
7	b	b	a
8	b	b	b

DEL がヘッドの時に提案する Move-Out アルゴリズムを実行すると，表 2 に示した 8 通りの DEL.chd は以下のように動作する.

No.1　**DEL.chd はアルゴリズムの動作後ゲートウェイのままとなる**

DEL の離脱後，ゲートウェイである DEL.chd は子供を複数持っており，また隣接ノードの中にヘッドが存在する．従って，ステータスはゲートウェイのままで統合の実行の際そのヘッドの子供になることで CNet(G) の性質 1 を満たす.

No.2　**DEL.chd はアルゴリズムの動作後ヘッドとなる**

DEL の離脱後，ゲートウェイである DEL.chd は子供を複数持つが，隣接するヘッドが存在しないため，Move-Out アルゴリズム 53 行によってヘッドになる事で CNet(G) の性質 1 を満たす.

No.3　**DEL.chd はアルゴリズムの動作後メンバとなる**

DEL の離脱後，ゲートウェイである DEL.chd は子供を一つだけ持ち，隣接ノードにヘッドが存在するため，Move-Out アルゴリズム 60 行によってメンバになる事で CNet(G) の性質 1 を満たす．DEL.oneigh にヘッドが存在していた場合，後に Move-Out アルゴリズム 86 行でステータスをゲートウェイに変化する事で CNet(G) の性質 1 を満たす.

No.4　**DEL.chd はアルゴリズムの動作後ヘッドとなる**

DEL の離脱後，ゲートウェイである DEL.chd は子供を一つだけ持つが，隣接ノードにヘッドが存在しないため，Move-Out アルゴリズム 57 行によってヘッドになる事で CNet(G) の性質 1 を満たす.

No.5　**DEL.chd はアルゴリズムの動作後メンバのままとなる**

DEL の離脱後，メンバである DEL.chd は隣接ノードにヘッドが存在するためアルゴリズムの影響を受けず，メンバのままで CNet(G) の性質 1 を満たす.

No.6　**DEL.chd はアルゴリズムの動作後メンバのままとなる**

No.5 と同様である.

No.7　**DEL.chd はアルゴリズムの動作後ヘッドになる**

DEL の離脱後，メンバである DEL.chd は隣接ノードにヘッドが存在しないため，Move-Out アルゴリズム 64 行によってヘッドとなることで CNet(G) の性質 1 を満たす.

No.8　**DEL.chd はアルゴリズムの動作後ヘッドになる**
No.7 と同様である．
従って，いずれの場合も Move-Out アルゴリズムの実行により，DEL.chd のステータスは CNet(G) の性質 1 に反しない．

□

補題 8. *DEL* がゲートウェイの時，Move-Out アルゴリズムの実行後 *DEL*.prt,(*DEL*.prt).prt のステータスは *CNet* の性質 1 に反しない

証明.
CNet を構成するためにノードが保持する情報の内容より，*DEL*.prt は以下の条件の組み合わせにより分類する事ができる．ただし *DEL* がゲートウェイの時，*DEL*.prt は *CNet* の定義よりヘッドもしくはゲートウェイである．

条件 1. *DEL.prt* のステータスがヘッドかゲートウェイか

(a) ヘッド　*(b)* ゲートウェイ

条件 2. *DEL.prt* は *DEL* 以外に子供を持つか持たないか

(a) 持つ　*(b)* 持たない

条件 1〜2 の組み合わせと *DEL*.prt の変化を表 3 に示す．

表 3　ゲートウェイが退出時の DEL.prt への影響

No	条件 1	条件 2
1	a	a
2	a	b
3	b	a
4	b	b

DEL がゲートウェイの時に提案する Move-Out アルゴリズムを実行すると，表 2 に示した 8 通りの DEL.prt は以下のように動作する．
No.1　**DEL.prt はアルゴリズムの動作後ヘッドのままとなる**
DEL の離脱後，ヘッドである DEL.prt に子供の有無は関係無いので，DEL.prt はヘッドのままで CNet(G) の性質 1 を満たす．
No.2　**DEL.prt はアルゴリズムの動作後ヘッドのままとなる**
No.1 と同様である．
No.3　**DEL.prt はアルゴリズムの動作後ゲートウェイのままとなる**
DEL の離脱後，ゲートウェイである DEL.prt は DEL 以外に子供を持つ．この時，DEL.prt はゲートウェイのままで CNet(G) の性質 1 を満たす．
No.4　**DEL.prt はアルゴリズムの動作後メンバとなる**
DEL の離脱後，ゲートウェイである DEL.prt は子供を持たない．この時，DEL.prt がゲートウェイであることから (DEL.prt).prt はヘッドとなるため，DEL.prt は Move-Out アルゴリズム 46 行によってメンバになることで CNet(G) の性質 1 を満たす．
従って，いずれの場合も Move-Out アルゴリズムの実行により，DEL.prt,(DEL.prt).prt のステータスは CNet の性質 1 に反しない．

□

補題 9. *DEL* がゲートウェイの時，Move-Out アルゴリズムの実行後 *DEL*.chd のステータスは *CNet* の性質 1 に反しない

証明.
CNet を構成するためにノードが保持する情報の内容より，*DEL*.chd は以下の条件の組み合わせにより分類する事ができる．ただし *DEL* がゲートウェイの時，*CNet* の定義より *DEL*.chd はヘッドもしくはゲートウェイである．

条件 1. *DEL.chd* のステータスがヘッドかゲートウェイか

(a) ヘッド　*(b)* ゲートウェイ

条件 2. *DEL.chd* は子供をいくつ持つか

(a)2 個以上持つ　*(b)1* 個しか持たない

条件 1〜2 の組み合わせと *DEL*.chd のステータス変化を表 4 に示す．

表 4　ゲートウェイが退出時の DEL.chd への影響

No	条件 1	条件 2
1	a	a
2	a	b
3	b	a
4	b	b

DEL がゲートウェイの時に提案する Move-Out アルゴリズムを実行すると，表 2 に示した 8 通りの DEL.chd は以下のように動作する．
No.1　**DEL.chd はアルゴリズムの動作後ヘッドのままとなる**
DEL の離脱後，ヘッドである DEL.chd に子供の有無は関係無いので，DEL.chd はヘッドのままで CNet(G) の性質 1 を満たす．
No.2　**DEL.chd はアルゴリズムの動作後ヘッドのままとなる**
No.1 と同様である．
No.3　**DEL.chd はアルゴリズムの動作後ゲートウェイのままとなる**
DEL の離脱後，ゲートウェイである DEL.chd は DEL 以外に子供を持つ．この時，DEL.chd はゲートウェイのままで CNet(G) の性質 1 を満たす．
No.4　**DEL.chd はアルゴリズムの動作後メンバとなる**
DEL の離脱後，ゲートウェイである DEL.chd は子供を持たない．この時，DEL.chd は 69 行でメンバになることで CNet(G) の性質 1 を満たす．
従って，いずれの場合も Move-Out アルゴリズムの実行により，DEL.chd のステータスは CNet(G) の性質 1 に反しない．

□

補題 10. *DEL* がメンバの時，Move-Out アルゴリズムの実行後 *DEL*.prt,(*DEL*.prt).prt,*DEL*.chd のステータスは *CNet* の性質 1 に反しない

証明.
CNet の定義より，メンバが離脱してもその構造には変化は生じない．

□

補題 11. Move-Out アルゴリズムの実行により DEL が離脱した後，DEL.prt,(DEL.prt).prt,DEL.chd で $CNet(G)$ の定義を満たすステータスが維持される

証明.
補題 6，補題 7，補題 8，補題 9，補題 10 より，DEL.prt,(DEL.prt).prt,DEL.chd のステータスは $CNet$ の性質 1 に反しない. □

補題 12. Move-Out アルゴリズムの実行により DEL が離脱した後，他の部分木と隣接するノードで $CNet(G)$ の定義を満たすステータスが維持される

証明.
Move-Out アルゴリズムでは部分木同士の統合を実行する際にヘッドは発生しない．従って，部分木の統合を実行するノード付近でヘッド同士が隣接する事は無い．新たにゲートウェイが発生するのは Move-In アルゴリズム 121 行によりメンバがゲートウェイに変化する時だけである．従って，メンバの定義より，新たに発生するゲートウェイの隣には必ずヘッドが存在する．また，この時 Move-Out アルゴリズムの実行により，$CNet$ 内部の親子関係も統合地点を基点に変更されている．以上より，$CNet$ の性質は維持されている. □

補題 13. *Move-Out* アルゴリズムの実行により DEL が離脱した後に存在する $CNet$ にはそれぞれ唯一つのルートが存在する

証明.
Move-Out アルゴリズムにより，DEL.chd を根とする $CNet$ 上の部分木の統合が実行される．この時、統合が実行される部分木を構成するノードが持つルート ID は統合先の部分木が持つルート ID となる．この時孤立している部分木以外は全てが統合を実行される為，Move-Out アルゴリズムの実行により DEL が離脱した後に存在する $CNet$ はそれぞれ唯一つのルートを持つ. □

定理 3. Move-Out アルゴリズムの実行により $CNet$ からノードの離脱が行われた結果，新しくできるネットワークも $CNet$ の定義を満たしている

証明.
補題 11，補題 12 より，$CNet$ の性質 1 は維持される．補題 13 より，$CNet$ の性質 2 は維持される．性質 1 より，性質 3 はただちに導かれる.

従って，$CNet$ の性質が全て満たされる事から題意は成立する. □

定理 4. Move-Out アルゴリズムはノードの離脱に対して，効率よくクラスタ構造を維持する.

証明.
DEL が Leave メッセージを送信後，$DEL.chd$ を根とする全ての部分木に $DEL.chd$ の ID がルート ID として伝達されるまでに $O(|T|)$ 時間必要である．続いて，$T_1, T_2, ..., T_q$ それぞれの部分木の隣接情報を全て探索するのに $O(|T|)$ 時間必要である．最後に，それぞれの部分木へ最終的なルート ID の伝達及び統合の実行に再び $O(|T|)$ 時間必要となる．従って，Move-Out アルゴリズムの実行に必要な実行時間は $O(|T|)$ となる．また，離脱によってステータスが変化するノードは DEL の親，DEL の親の親，DEL の子，部分木の編入時に編入先の部分木の子となったノードとそのノードの親になったノードのみである．これは，既知のアルゴリズムに対して，クラスタ構造が変化するノードが少ないという意味で，効率が良い. □

Algorithm Move-Out

• DEL が実行する動作

```
001    CNet から離脱するノード DEL は Leave を送信；
002    WHILE(DEL は ChangeRootID を未送信の子供を持つ) {
003    DEL は ChangeRootID を未送信の子供に ChangeRootID を送信；
004    } END WHILE
005    WHILE(DEL は CheckYourNeighbors を未送信の子供を持つ) {
006    CheckYourNeighbors を未送信の子供に CheckYourNeighbors を送信する；
007    } END WHILE
008    DEL は DEL.chd を根とする部分木それぞれの隣接ルート ID リストより,
009    各部分木の再統合を実行する順番及び統合先の部分木を決定する；
010 WHILE(MergeToNeighbor 又は Indepentence を未送信の子供を持つ) {
011 IF( DEL に統合が実行可能で MergeToNeighbor を未送信の子供が存在する )
012    統合が可能で MergeToNeighbor を未送信の子供に MergeToNeighbor を送信する；
013 ELSE
014    MergeToNeighbor または Independence を未送信の子供に
015    Independence を送信する；
016 } END WHILE
```

• DEL 以外のノード Pi が実行する動作

```
017 IF(Leave を受信)
018    IF(Leave.ID が Pi.chd に含まれる)
019       Pi.chd から Leave.ID を削除；
020       IF( Leave.status == ヘッド )
021          IF( Pi.prt.status == ゲートウェイ )
022             IF( Pi は子供を持たない )
023                IF( Pi.oneigh の中にステータスがヘッドのノードが存在 )
024                   Pi は Pi.prt に CheckYourChd を送信；
025                   Pi.status := メンバ；
026                   Pi は MinID(ヘッド) に ImYourChd を送信；
027                   Pi.prt := MinID(ヘッド)；
028                ELSE
029                   Pi.status := ヘッド；
030                   Pi は ChangeMyStatus を送信；
031             ELSE
032                IF( Pi.chd の中にステータスがヘッドのノードが存在しない )
033                   IF( Pi.oneigh の中にステータスがヘッドのノードが存在 )
034                      Pi は Pi.prt に CheckYourChd を送信；
035                      Pi.prt := MinID(ヘッド)；
036                   ELSE
037                      Pi.status := ヘッド；
038                      Pi は ChangeMyStatus を送信；
039          ELSE
040             IF( Pi は子供を持たない )
041                Pi.status := メンバ；
042                Pi は ChangeMyStatus を送信；
043       ELSE  IF( Leave.status == ゲートウェイ )
044          IF( Pi.status == ゲートウェイ )
045             IF( Pi は子供を持たない )
046                Pi.status := メンバ；
047                Pi は MinID(ヘッド) に ImYourChd を送信；
048    ELSE IF( Leave.ID == Pi.prt )
049       IF( Leave.status == ヘッド )
050          IF( Pi.status == ゲートウェイ )
051             IF( Pi は子供を複数持つ )
052                IF( Pi.chd 及び Pi.oneigh の中にヘッドのノードが存在しない )
053                   Pi.status := ヘッド；
054                   Pi は ChangeMyStatus を送信；
055             ELSE
056                IF( Pi.chd 及び Pi.oneigh の中にヘッドのノードが存在しない )
057                   Pi.status := ヘッド；
058                   Pi は ChangeMyStatus を送信；
059                ELSE
060                   Pi.status := メンバ；
061                   Pi は ChangeMyStatus を送信；
062          IF( Pi.status == メンバ )
063             IF( Pi.oneigh の中にステータスがヘッドのノードが存在しない )
064                Pi.status := ヘッド；
065                Pi は ChangeMyStatus を送信；
066       ELSE IF( Leave.status == ゲートウェイ )
067          IF( Pi.status == ゲートウェイ )
068             IF( Pi は子供を一つだけ持つ )
069                Pi.status := メンバ；
070                Pi は ChangeMyStatus を送信；
071    ELSE IF( Leave.ID が Pi.oneigh に含まれる )
072       Pi.oneigh から Leave.ID を削除；
073 IF( CheckYourNeighbors を受信した )
074    IF( CheckYourNeighbors.To == Pi.ID )
075       IF( CheckYourNeighbors.list に存在しない (Pi.oneigh).root が存在 )
076          CheckYourNeighbors.list に存在しない (Pi.oneigh).root を追加；
077       WHILE(ChangeYourNeighbors を未送信の子供を持つ) {
078       CheckYourNeighbors を未送信の子供に CheckYourNeighbors を送信する；
079 IF( MergeToNeighbor を受信した )
080    Pi.root := MergeToNeighbor.ID
081    IF( MergeToNeighbor.ID が Pi.oneigh に含まれる )
082       IF( MergeToNeighbor.flag == FALSE )
083          MergeToNeighbor.flag := TRUE；
084          Pi.prt へ ImYourParent を送信；
085          IF( Pi.status == メンバ )
086             Pi.status := ゲートウェイ；
087             Pi.bneigh := Pi.prt；
088             Pi は ChangeMyStatus を送信；
089          IF(Pi.oneigh にルート ID が MergeToNeighbor.ID のヘッドが存在)
090             Pi は MinID(ヘッド,MergeToNeighbor.ID) へ ImYourChild を送信；
091             Pi.bneigh := Pi.bneigh ∪ MinID(ヘッド,MergeToNeighbor.ID)
092             Pi.oneigh から MinID(ヘッド,MergeToNeighbor.ID) を削除；
093             Pi.prt := MinID(ヘッド,MergeToNeighbor.ID)；
094          ELSE IF(Pi.oneigh に ID が MergeToNeighbor.ID の
                ゲートウェイが存在)
095             Pi は MinID(ゲートウェイ,MergeToNeighbor.ID) へ ImYourChild
096             を送信；
097             Pi.bneigh := Pi.bneigh ∪ MinID(ゲートウェイ,MergeToNeighbor.ID)；
098             Pi.oneigh から MinID(ゲートウェイ,MergeToNeighbor.ID) を削除；
099             Pi.prt := MinID(ゲートウェイ,MergeToNeighbor.ID)；
100          ELSE
101             Pi はルート ID が MergeToNeighbor.ID のノード中の MinID(メンバ)
102             へ ImYourChild を送信；
103             Pi.bneigh := Pi.bneigh ∪ MinID(メンバ,MergeToNeighbor.ID)；
104             Pi.oneigh から MinID(メンバ,MergeToNeighbor.ID) を削除；
105             Pi.prt := MinID(メンバ,MergeToNeighbor.ID)；
106    WHILE(MergeToNeighbor を未送信の子供が存在する) {
107       MergeToNeighbor を未送信の子供に MergeToNeighbor を送信する；
108    }
109 IF(Independence を受信)
110 THEN
111    IF(Independence.To == Pi.ID)
112       IF( Pi.status がゲートウェイもしくはメンバ )
113          Pi.chd の中での MinID(ヘッド) に ImYourChild を送信；
114          Pi.prt := Pi.chd の中での MinID(ヘッド)；
115          Pi.chd から Pi.chd の中での MinID(ヘッド) を削除；
116 IF(ImYourChild を受信)
117    IF(ImYourChild.To == Pi.ID)
118       Pi.chd := Pi.chd ∪ ImYourChild.ID；
119       Pi.bneigh := Pi.chd ∪ ImYourChild.ID；
120       IF(Pi.status == メンバ)
121          Pi.status := ゲートウェイ；
122          Pi.bneigh := Pi.prt；
123          Pi は ChangeMyStatus を送信；
124    ELSE
125       ID が ImYourChild.ID のノードのステータスを ImYourChild.status に変更；
126 IF(ImYourParent を受信)
127    IF(ImYourParent.To == Pi.ID)
128       Pi.prt へ ImYourParent を送信；
129       Pi.chd := Pi.chd ∪ Pi.prt；
130       Pi.prt := ImYourParent.ID；
131 IF(ChangeMyStatus を受信した)
132    ID が ChangeMyStatus.ID のノードのステータスを ChangeMyStatus.status に
       変更；
133    IF( ChangeMyStatus.status == メンバ )
134       IF(Pi.bneigh に ChangeMyStatus.ID が含まれる)
135          Pi.bneigh から ChangeMyStatus.ID を削除；
136    ELSE
137       IF(Pi.chd に ChangeMyStatus.ID が含まれる)
138          Pi.bneigh := Pi.bneigh ∪ ChangeMyStatus.ID；
139 IF( CheckYourChd を受信した )
140    IF(CheckYourChd.To == Pi.ID)
141       THEN
142          Pi.chd より CheckYourChd.ID を削除；
143          IF( Pi は子供を持たない )
144             Pi.status := メンバ；
145             Pi は ChangeMyStatus を送信；
146
```

図 2　Move-Out アルゴリズム

Fig. 2　Move-Out Algorithm