社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2008－AL－117 (1)
2008／3／7

極大クリーク分割に基づく自己安定クラスタリングアルゴリズム

西村 弘志 †， 泉 泰介 †， 片山 喜章 †， 和田 幸一 †

† 名古屋工業大学 大学院 工学研究科 情報工学専攻

本稿ではグラフを完全部分グラフ（クリーク）に分割する問題（クリーク分割問題）について考える．クリーク分割はアドホックネットワークのクラスタリングなどへの応用があり，分割数が小さい方がよいと考えられるが，分割数を最小とする分割を求めることはNP完全である．そこで，本稿では分割数を極小とする問題を考え，この問題を解く自己安定アルゴリズムを提案する．ここで提案するアルゴリズムは，マッチングとマージという単純な操作を行う自己安定アルゴリズムを公平な合成により多段階層化することでこの問題を解く．

A self-stabilizing clustering algorithm based on maximal clique partition

Hiroshi Nishimura † Taisuke Izumi † Yoshiaki Katakama † Koichi Wada †

† Department of Computer Science and Engineering, Graduate School of Engineering, Nagoya Institute of Technology

We consider a problem to partition a graph into complete subgraphs(cliques), witch is called clique partition problem. This problem is related to some clustering of ad-hoc networks. Since finding the partition such that the number of cliques is minimum is NP-complete, we consider a problem such that the number of cliques is minimal. We propose a self-stabilizing algorithm that solves this problem.Our proposed algorithm consists of multiple layers of two self-stabilizing algorithms which do two simple operations(matching and merging). They are done by "fair composition"

1 はじめに

アドホックネットワークとは，無線通信機能を持ち自律的に動作する端末（ノード）のみで構成されるネットワークである．一般的に，各ノードの計算能力や通信能力などは低いため，効率の良い通信方式が求められる．効率の良い通信を実現するためのアプローチとして，ネットワーク上にクラスタを構成する手法が知られている．これは，ネットワーク上のノードをいくつかの集合（クラスタ）に分割し，各クラスタ内でクラスタヘッド（以下，単にヘッド）と呼ばれるノードをひとつ選出し，そのヘッドがクラスタ内のノードの情報を管理する手法である．ヘッド以外のノードはクラスタメンバ（以下，単にメンバ）と呼ばれ，必ずいずれかのクラスタに属する．

ネットワークをクラスタに分割する問題（クラスタ分割問題）を扱った既存研究には，ヘッドの集合をグラフ（ネットワークをグラフとして捉える）の支配点集合として求める方法[1]や，各クラスタを完全部分グラフ（クリーク）となるよう分割しヘッドを決定する方法[2]などがある．ここで，支配点集合をヘッドとして選ぶ場合について考えてみる．ヘッドがネットワークから離脱すると，ヘッドの集合が支配点集合であるという条件を満たさなくなるので，ヘッドの再選択，つまり支配点集合の再計算が必要になる．このため，ネットワーク上のクラスタ構造が大きく変化する場合がある．

一方，クラスタがクリークとなるように分割し，ヘッドを各クラスタ内から選ぶようにした場合を考える．ヘッドが離脱してもヘッドの役割を同一クラスタ内の他のメンバに移すだけで済む．つまり，トポロジ変化に伴う再計算がクラスタ内で局所的に処理できるため，既存のクラスタに対する変更が小さい．このことから，クラスタとしてク

リークを採用する方法は，ノードの参加や離脱が頻発するアドホックネットワークとの親和性が高いと考える．

本稿ではネットワークをクリークに分割するアルゴリズムについて考える．分割数（＝ヘッドの数）が小さくなるように分割するとヘッド間のルーティングなどにかかる時間が小さくなる．分割数を最小にする最小クリーク分割問題はNP完全[3]である．したがって，本稿では分割数を極小とする問題を考え，その問題を解く自己安定アルゴリズムを提案する．提案アルゴリズムはマッチングとマージという単純な操作を行う自己安定アルゴリズムを公平な合成により多段階層化することで構成される．

2 システムモデル

ネットワークのトポロジがグラフ $G = (V, E)$ で与えられるとする．$V = \{v_1, v_2, ..., v_n\}$ はノードの集合であり，E はノード間の双方向リンクの集合である．つまり，ノード v_i と v_j が互いに通信が行えるとき，v_i と v_j は隣接しているといい，$(v_i, v_j) \in E$ であるとする．各ノード v_i は固有の識別子 (ID) と隣接ノードの集合 N_i を持つ．さらに，各ノードは局所変数を持つ．

2.1 システムの状況

各ノード v_i の局所変数の集合をそのノードの (局所) 状態と呼び，q_i で表す．システム全体の状況 c を $c = (q_1, q_2, ...q_n)$ で表す．v_i の取りうる全ての状態を Q_i，システム全体が取りうる状況を Γ とすると，$\Gamma = Q_1 \times Q_2 \times ... \times Q_n$ となる．

2.2 通信モデル

本稿では，通信モデルとして状態通信モデル (state communication model 又は，state reading model) を仮定する．これは隣接ノードの状態（局所変数）を遅延なしで直接読むことができるモデルである．書き込みは各ノードが持つ局所変数に対してのみ行うことができる．

2.3 実行

ノードの部分集合を $S \subseteq V$ とする．ある状況 $c_i \in \Gamma$ において，S に属するノードが同時にアルゴリズム A の1原子動作を実行することによって c_{i+1} になったとき，$c_{i+1} = \mathcal{C}(c_i, S, A)$ と表す．

ここでは，1原子動作で以下の3つの動作をすべて行うものとする．

1. 隣接ノードの局所変数を読み込む．
2. 隣接ノードの局所変数と自分の局所変数をもとに，内部計算を行う．
3. 内部計算の結果を自分の局所変数に書き込む．

空でないノードの集合の無限系列 $T = S(0), S(1), ...,$ をスケジュールと呼ぶ．状況の無限系列 $E = c_0, c_1, ...$ が $c_{i+1} = \mathcal{C}(c_i, S(i), A)$ を満たすとき，E を「初期状況 c_0，スケジュール T に対するアルゴリズム A の実行」と呼び，$E(A, T, c_0)$ と表す．また T に全ノードが無限回現れるとき，公平なスケジュールという．本稿では任意の $t \geq 0$ に対して，$|S(t)| = 1$ かつ公平であるようなスケジュール (C-daemon) を考える．

2.4 自己安定アルゴリズム

自己安定アルゴリズムとは耐故障性のある分散アルゴリズムの一種である．自己安定アルゴリズムは，任意の状況から開始しても要求される振舞いに収束する．つまり，ノードなどに一時的な故障やトポロジの変化が起きても，その状況を初期状況としていずれ正しい状況になる．

2.4.1 正当な状況

定義 1 （正当な状況）
Γ を全ての状況の集合とする．アルゴリズム A が以下の2つの条件を満たすときかつそのときに限り，A は $\Lambda \subseteq \Gamma$ に対して自己安定であるという．

1. 到達可能性：任意の状況 $c_0 \in \Gamma$ と任意のスケジュール T に対し，$E(A, T, c_0)$ には Λ に含まれる状況が現れる．
2. 閉包性：任意の状況 $c \in \Lambda$，ノードの任意の集合 S に対し，$c' = \mathcal{C}(c, S, A)$ とすると $c' \in \Lambda$ となる．

各 $\lambda \in \Lambda$ においてタスク（本稿の場合は極大クリーク分割問題）が解決されていることを示し，アルゴリズム A が Λ に対して自己安定であることを示せば，A がタスクを解決する自己安定アルゴリズムであることが示せる．

2.4.2 アルゴリズムの記述形式

本稿ではアルゴリズムをガード付きコマンド (Guarded Commands,GCs) 形式で記述する．各ノード v_i のアルゴリズムは GC の集合

$$*[g_1 \rightarrow c_1 \Box g_2 \rightarrow c_2 \Box g_3 \rightarrow c_3 \Box ...]$$

で与えられる．$g_j(j=1,2,...)$ はガード（guard）と呼ばれ，ノードとその隣接ノードの局所状態から成る述語である．c_j はコマンド（command）と呼ばれ，コマンドが実行されることでノードの局所状態が更新される．ノードは各ガードを評価し，真となるガードに対応するコマンドを実行する．真となるガードが複数あるときは，そのガードに対するコマンドの内一つを実行する．

2.5 非同期ラウンド

各ノードの計算速度や通信速度が異なるようなシステムでの時間複雑度の尺度として非同期ラウンドがある．直感的には，全てのノードが少なくとも1回，原子動作を行ったときを1ラウンドと呼ぶ．1ラウンドの間に原子動作を何回も行うノードがあってもよい．正確には，$S(0) \cup S(1) \cup ... \cup S(t_1) = V$ となるような最小の t_1 に対して，$R = S(0), S(1), ..., S(t_1)$ を第1ラウンドと呼ぶ．同様に，$S(t_1+1) \cup S(t_1+2) \cup ... \cup S(t_2) = V$ となる最小の t_2 に対して，$R = S(t_1+1), S(t_1+2), ..., S(t_2)$ を第2ラウンドとする．以下同様に定義する．本稿ではこの非同期ラウンドを用い，アルゴリズムの時間複雑度をラウンド数で評価する．

2.5.1 自己安定アルゴリズムの合成

自己安定アルゴリズム $A_1, A_2, ..., A_k$ を，各アルゴリズム A_i の出力が A_{i+1} の入力となるように組み合わせたアルゴリズム A を $A_1, A_2, ..., A_k$ の合成という [5]．また，各 A_i のステップが無限にしばしば実行されるとき，A を公平な合成という．公平な合成では，$A_1, ..A_{i-1}$ がそれぞれの正当な状況に到達したあとに，いずれ A_i は正当な状況に到達する．そして，いずれ A 全体に対して正当な状況となる．公平な合成を行うことにより，単純なアルゴリズムから複雑なアルゴリズムを構成することができる．

3 極大クリーク分割問題の定義

$G = (V, E)$ をグラフとする．本稿では簡単のため，「ノードの集合 $C \subseteq V$ で誘導される G の部分グラフが完全グラフである」ことを「C がクリークである」と言う．このとき，極大クリーク分割問題を以下のように定義する．

定義 2 （極大クリーク分割問題）
極大クリーク分割問題とは，以下の4つの条件を満たすノードの集合 $C_1, C_2, ..., C_k$ を求めることである．

1. $C_x, C_y (x \neq y)$ に対して，$C_x \cap C_y = \emptyset$.
2. $C_1 \cup C_2 \cup ..., \cup C_k = V$.
3. 各 C_x はクリークである．
4. $C_x, C_y (x \neq y)$ に対して，$C_x \cup C_y$ はクリークでない．

定義2を満たすような $C_1, C_2, ..., C_k$ を G の極大クリーク分割と呼ぶ．また，$C_1, C_2, ..., C_k$ が条件4以外を満たすときは，$C_1, C_2, ..., C_k$ を，単に G のクリーク分割と呼ぶ．G の極大クリーク分割における分割数は極小である．

ここでは，各ノード v_i に v_i の隣接ノードからなる集合 N_i が与えられたとき，各 v_i が v_i を含みかつ定義2を満たすクリーク C_x を計算するアルゴリズムを構成する．

4 極大クリーク分割を求める手法

グラフ G と G のクリーク分割 $C_1, C_2, ..., C_k$ に対して，各クリーク間の隣接関係を表したクリーク分割グラフ $CP(G; C_1, C_2, ..., C_k) = (V_{CP}, E_{CP})$ を次のように定義する．

定義 3 （クリーク分割グラフ CP）
グラフ G と G のクリーク分割 $C_1, C_2, ..., C_k$ に対して，クリーク分割グラフ $CP(G; C_1, C_2, ..., C_k)$ とは，次に示すような頂点集合 V_{CP} と辺集合 E_{CP} をもつグラフである．

$$V_{CP} = \{C_1, C_2, ..., C_k\}$$
$$E_{CP} = \{(C_x, C_y) | C_x, C_y \in V \wedge C_x \cup C_y \text{は} G \text{のクリーク}\}$$

$E_{CP} = \emptyset$ のとき，明らかに $C_1, C_2, ..., C_k$ は極大クリーク分割である．また，$(C_x, C_y) \in E_{CP}$ に対して，C_x と C_y を結合して得られる $C_1, ..., C_x \cup C_y, .., C_k$ もクリーク分割である．

あるクリーク分割 $C_1, C_2, ..., C_k$ に対する $CP(G; C_1, C_2, ..., C_k)$ から，隣接関係にあるクリーク同士を結合して得られたクリーク分割 $C'_1, C'_2, ..., C'_l$ に対する $CP(G; C'_1, C'_2, ..., C'_l)$ を計算することを「マージ」と呼ぶ．マージを CP が空グラフ (辺集合が空) となる分割を得るまでマー

ジを繰り返すことにより，極大クリーク分割を得ることができる．

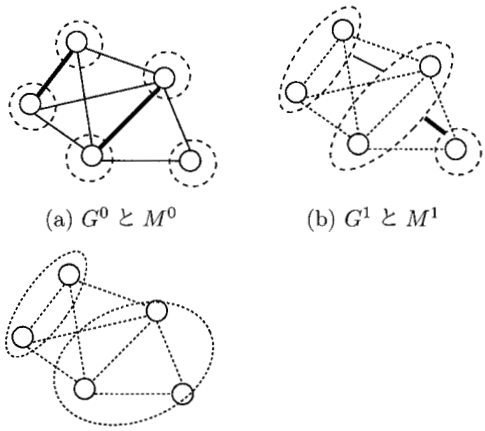
CP やマージの計算は各ノードが分散的に行うが，どの 2 つのクリーク同士をマージするかをシステム全体で一貫して決める必要がある．ここでは，マージするクリークを決めるために CP 上のマッチングを用いる．マッチングとは端点を共有しない辺集合 $M \subseteq E_{CP}$ である．また，マッチング M と任意の $e \in E_{CP} - M$ に対して $M \cup \{e\}$ がマッチングではないとき，M を極大マッチングという．CP の極大マッチング M を求め M に含まれる辺で結ばれる 2 つのクリークを結合することにより，システム全体で一貫したマージが可能となる．

クリーク分割グラフ CP と CP 上の極大マッチングを用いて，極大クリーク分割を求める手順を示す．G の各ノード v_i に対して，v_i のみから成るクリーク $C_i^0 = \{v_i\}$ を考える．$C_1^0, C_2^0, ..., C_n^0$ は G のクリーク分割であり，$G^0 = (V^0, E^0) = CP(G; C_1^0, C_2^0, ..., C_n^0)$ とする．G^0 上の極大マッチング $M^0 \subseteq E^0$ を求め，M^0 に含まれる辺で結ばれるクリーク対に対してマージを行う．この結果得られたグラフを $G^1 = (V^1, E^1) = CP(G; C_1^1, C_2^1, ..., C_{n^1}^1)$ とする．G^i 上の極大マッチング M^i を求めることを「マッチング」と呼ぶことにする．$E^k = \emptyset$ となるクリーク分割 $C_1^k, C_2^k, ..., C_{n^x}^k$ を得るまで，マッチングとマージを繰り返す．実行例を図 1 に示す．図 1 では，点線の円がクリーク (G^x の頂点) を表し，実線が G^x の辺を表す．実線のうち太いものがマッチングに含まれる辺を表す．

グラフ $G = (V, E)$ の最大次数を Δ とすると，次の定理が成り立つ．

定理 1 $E^k = \emptyset$ となる最小の k について $k \leq \lceil \lg(\Delta + 1) \rceil$ が成り立つ．

証明 G^0 から G^k を得るまでの k 回のマージについて調べる．k 回目のマージによって得られた $C \in V^k$ について注目する．C は G^0 で $|C|$ 個のクリークに分割されていると考えることができる．1 回目のマージではこの $|C|$ 個のクリーク同士で結合が行われる．全ての可能なペアを結合するので，マージの結果は C の $\lceil |C|/2 \rceil$ 分割となっている．同様に，2 回目のマージ後は $\lceil \lceil |C|/2 \rceil /2 \rceil = \lceil |C|/4 \rceil$ 分割となる．これを繰り返し k 回目のマージで 2 つのクリークを結合して C が得られるので，$\lceil |C|/2^{k-1} \rceil = 2$ となる．これより $|C|/2^{k-1} > 1$ となり，式を変形すると $k < \lg|C| + 1$ となる．また，k は整数なので $k \leq \lceil \lg|C| \rceil$ が成り立つ．ノードの最大次数は Δ なので，$|C| \leq \Delta + 1$ である．よって，$k \leq \lceil \lg(\Delta + 1) \rceil$ が成り立つ． □



(a) G^0 と M^0　　(b) G^1 と M^1

(c) G^2

図 1: マッチングとマージの実行例

定理 1 より，マッチングとマージを $\lceil \lg(\Delta + 1) \rceil$ 回行えば極大クリーク分割が得られる．

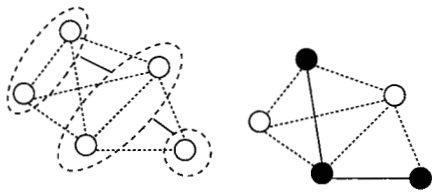
5 提案アルゴリズム *SSMM*

4 章で示した手法を各ノードが分散的に計算する方法を示す．それをもとに，極大クリーク分割を求める自己安定アルゴリズム *SSMM* を示す．このアルゴリズムは各ノード v_i に隣接ノードの集合 N_i が入力として与えられ，定義 2 を満たし v_i を含むクリーク C_i を出力する．

5.1 G^x の計算方法

$G^x = (V^x, E^x)$ の各クリーク C_i^x 内で ID が最大のものをヘッドとする．G^x のクリークでの隣接関係をヘッド間の隣接関係とすると，G^x を G の部分グラフ，M^x を部分グラフのマッチングで表すことができる (図 2)．図 2 の (a) は G^x を図 1 と同様に表しており，(b) はヘッドを黒丸，G^x の辺に対応する辺を実線で表している．

G^x において v_i が計算したクリークを C_i^x，ク



(a)G^x　　(b) G の部分グラフ

図 2: G^x と G^x を表す G の部分グラフ

リークのヘッドを $head_i^x$，G^x で隣接するクリークのヘッドの集合を N_i^x とする．マッチングアルゴリズムが，各 v_i に対して $(v_i, v_j) \in M^x$ のときは $m_i^x = v_j$，$(v_i, v_j) \in M^x$ となる v_j が存在しないときは $m_i^x = null$ となる m_i^x を出力するものとする．マッチングとマージの計算はヘッドのみが行い，ヘッド以外のノードはヘッドの計算結果をコピーする．C_i^{x+1} の計算は C_i^x と v_i マッチングの相手（m_i^x が $null$ でなければ）のクリークと結合するだけである．結合したクリークのヘッドの ID が大きいほうが結合後のクリークのヘッドとなる．よって C_i^{x+1} と $head_i^{x+1}$ は次のようになる．

- $m_i^x = v_j$のとき：
  $C_i^{x+1} = C_i^x \cup C_j^x$
  $head_i^{x+1} = v_i$と v_jで ID が大きい方のノード
- $m_i^x = null$ のとき：
  $C_i^{x+1} = C_i^x$
  $head_i^{x+1} = head_i^x$

N_i^{x+1} の計算には，$m_i^x = v_j$ である v_j の隣接ノードの集合 N_j^x と $v_k \in N_i^x \cap N_j^x$ のマッチングの出力 m_k^x について調べる必要がある．G^{x+1} 上で $C_i^x \cup C_j^x$ と隣接するクリークは，G^x 上で C_i^x と C_j^x に隣接しているクリーク同士で結合してできたクリークのみである．すなわち，$v_k \in N_i^x \cap N_j^x$ に対して，$m_k^x = null$ または $m_k^x \in N_i^x \cap N_j^x$ のときのみ，$v_k \in N_i^{x+1}$ となる．N_i^{x+1} は次のようにして計算することができる．

- $m_i^x = v_j$ のとき：

$$N_i^{x+1} = N_i^x \cap N_j^x - \{v_k \in N_i^x \cap N_j^x \mid m_k^x \notin (N_i^x \cap N_j^x) \cup \{null\}\}$$

- $m_i^x = null$ のとき：

$$N_i^{x+1} = N_i^x - \{v_k \in N_i^x | m_k^x \notin N_i^x \cup \{null\}\}$$

N_i^{x+1} は G^{x+1} 上でのヘッドのみを含むので，上記の計算の後にヘッド以外のノードを除く必要がある．

5.2 *SSMM* の構成

G^x 上での各ヘッド v_i が G^x の極大マッチングを求める自己安定アルゴリズム $Match_i^x$ と，G^x と $Match_i^x$ との結果から G^{x+1} の情報を出力する自己安定アルゴリズム $Merge_i^x$ を合成し，G^x から G^{x+1} を得る自己安定アルゴリズムを構成する．$Match_i^x$ は [4] のアルゴリズムを用いる．4 章で述べた手法を実現するために，G^0 の情報を出力するアルゴリズム $Init_i$ と G^x から G^{x+1} 出力するアルゴリズムを $\lceil \lg(\Delta+1) \rceil$ 段合成して，各ノードが実行するアルゴリズム $SSMM$ を構成する．つまり，$SSMM$ の出力 C_i は $C_i^{\lceil \lg(\Delta+1) \rceil}$ となる．ここでは，グラフ G の最大次数 Δ が前もって与えられていると仮定する．各アルゴリズムを図 3〜6 に示す．

6 正当性の証明と時間複雑度の解析

5 章で示したアルゴリズム $SSMM$ が極大クリーク分割を計算する自己安定アルゴリズムであることの証明と時間複雑度の解析を行う．

6.1 正当な状況の集合 Λ の定義

正当な状況の定義を行うために，全ての状況の集合 Γ に関する述語をいくつか定義する．全ノード v_i の $Init_i$ の局所変数 $C_i^0, N_i^0, head_i^0$ に関する述語 $INIT$ を次のように定義する．

定義 4（述語 $INIT$ の定義）

$INIT$ =全てのノード v_iに対して
　　$Init_i$のガードが偽

次に，$x(0 \leq x < \lceil \lg(\Delta+1) \rceil)$ 段目の $Match_i^x$ と $Merge_i^x$ の局所変数 $m_i^x, C_i^{x+1}, N_i^{x+1}, head_i^{x+1}$ に関する述語 $MATCH^x$，$MERGE^x$ を次のように定義する．

定義 5（述語 $MATCH^x$ の定義）

$MATCH^x$ =全てのノード v_iに対して
　　$Match_i^x$の全てののガードが偽

定義 6 （述語 $MERGE^x$ の定義）

$MATCH^x$ =全てのノード v_i に対して

$Merge_i^x$ の全てののガードが偽

さらに，$0 \leq x < \lceil \lg(\Delta+1) \rceil$ に対して，$MM^x = MATCH^x \wedge MERGE^x$ とする．

これらの定義を用いて正当な状況の集合 Λ を次のように定義する．

定義 7 （正当な状況の集合）
任意の状況 $\lambda \in \Lambda$ に対して次の述語が真となるとき，Λ を正当な状況の集合という．

$$INIT \wedge \bigwedge_{0 \leq x < \lceil \lg(\Delta+1) \rceil} MM^x$$

6.2 正当性の証明

まず，補題 1〜3 により，合成するアルゴリズムがそれぞれの問題を解く自己安定アルゴリズムであることを示す．

補題 1 $Init_i$ は G^0 を構成する自己安定アルゴリズムである．

証明 G^0 では各クリークが 1 ノードからなり，全ノードがヘッドとなる．よって，ヘッド間の隣接関係は G の隣接関係と等しくなる．したがって，全ノードが $Init_i$ のコマンドを実行すれば G^0 が構成され，その後 $Init_i$ の局所変数は変化しない． □

$G^0, ..., G^x$ が正しく構成されているとき，[4] より，次の補題が成り立つ．

補題 2 $Match_i^x$ は G^x 上でヘッドがマッチングを行う自己安定アルゴリズムである．

また，$G^0, ..., G^x$ が正しく構成されていて $Merge_i^x$ が安定しているとき，次の補題が成り立つ．

補題 3 $Merge_i^x$ は G^x 上でマージを行う自己安定アルゴリズムである．

証明 $G^0, ..., G^x$ が正しく構成されていて $Merge_i^x$ が安定しているとき，$merge_i^x$ が返す C と $head$ の値は G^{x+1} のクリークとそのヘッドとなっている．よって，全てのヘッド v_i が $Merge_i^x$ のコマンドを実行すると C_i^{x+1} と $head_i^{x+1}$ の値は正しくなる．この後，G^x 上でヘッドでないノード v_i が $Merge_i^x$ を実行すると，C_i^{x+1} と $head_i^{x+1}$ の値は正しくなる．全てのノード v_i に対して $head_i^{x+1}$ が正しい値となるので，ヘッド v_i が $Merge_i^x$ のコマンドを実行すると N_i^{x+1} の値は正しくなり，その後 $Merge_i^x$ の局所変数は変更されない． □

以下の補題において，アルゴリズム $SSMM$ は正当な状況の集合 Λ に対して，閉包性と到達可能性を満たすことを示す．

補題 4 アルゴリズム $SSMM$ は正当な状況の集合 Λ に対して，閉包性を満たす．

略証 正当な状況の定義より，正当な状況において全てのノードのアルゴリズムのガードは偽となる．よって，正当な状況では状態変化するノードは存在せず，閉包性は満たされている． □

補題 5 アルゴリズム $SSMM$ は正当な状況の集合 Λ に対して，到達可能性を満たす．

証明 任意の状況から正当な状況 λ に到達することを示す．スケジュールとして公平な実行および，公平な合成を仮定しているので，各アルゴリズムのガードが真となるノードはいずれガードに対応したコマンドを実行する．任意の状況から各ノード v_i が $Init_i$ のコマンドを実行すると $INIT$ が真となる状況 γ^0 に到達する．したがって，補題 1 より，いずれ $INIT$ が真となる状況 γ^0 に到達する．補題 2 より，γ^0 からいずれ $INIT \wedge MERGE^0$ が真となる状況となる．さらに，補題 3 より，この状況から $INIT \wedge MM^0$ が真となる状況 γ^1 に到達する．この議論を繰り返すことにより，いずれ正当な状況に到達することを示すことができる． □

補題 4 と 5 により次の定理が成り立つ．

定理 2 アルゴリズム $SSMM$ は正当な状況の集合 Λ に対して自己安定である．

次の定理によって，$SSMM$ の正当性を示す．

定理 3 正当な状況 $\lambda \in \Lambda$ において，$SSMM$ の出力は極大クリーク分割となっている．

証明 正当な状況 λ では $INIT$ が真であるので G^0 が正しく構成されている．G^0 が正しく構成されているので，補題 2 と 3 より G^1 も正しく構成されている．同様の議論により $G^2, ...G^{\lceil \lg(\Delta+1) \rceil}$ が正しく構成できていることを示すことができる．$G^{\lceil \lg(\Delta+1) \rceil}$ 上のヘッド v_i は v_i を含み定義 2 を満たすクリークを計算している．各 G^x 上のヘッド以外のノード v_i は，v_i を含むクリークのヘッドが計算した G^{x+1} のクリークとヘッドの値をコピーするので，$G^{\lceil \lg(\Delta+1) \rceil}$ でのヘッド以外のノード v_i も正しい C_i を出力する． □

定理 2，3 により，提案アルゴリズム $SSMM$ は極大クリーク分割問題を解く自己安定アルゴリズムである．

6.3 時間複雑度の解析

2.5 節で述べた非同期ラウンドを用いて $SSMM$ が正当な状況に到達するまでの時間複雑度を評価する．$|E| = m$ とする．また，各アルゴリズムのガードで真となるものが複数あるときは，以下の優先順位でコマンドを実行するスケジュールを仮定し，その下で評価する．

1. $Init_i$.
2. 各段の $Match_i^x$ と $Merge_i^x$ に対して，最小となる x.
3. 同一段の $Match_i^x$ と $Merge_i^x$ とでは，$Match_i^x$.

定理 4 $SSMM$ は任意の初期状況から $O(m\lg\Delta)$ ラウンドで正当な状況に到達する．

証明 1 ラウンドで各ノードは少なくとも 1 コマンドを実行する．よって，1 ラウンドで $INIT$ が真となる状況に到達する．0〜$x-1$ 段目が正しい値を出力してから，x 段目の出力が正しくなるまでの時間を考える．G^x 上での全てのヘッド v_i に対して，$Match_i^x$ の出力が正しくなる（$MATCH^x$ が真となる）までの時間は $O(m)$ ステップである [4]．よって，$O(m)$ ラウンドあれば $MATCH^x$ が真となる．その後，各ヘッドが $Merge_i^x$ を 1 回ずつ実行すれば $C_i^{x+1}, head_i^{x+1}$ は正しい値となる（1 ラウンド）．その後，ヘッド以外のノードが $Merge_i^x$ を 1 回ずつ実行し，ヘッドが $Merge_i^x$ を実行すると（2 ラウンド），$MERGE^x$ が真となる．

よって，$SSMM$ が正当に到達するまでの時間は $O(m\lg\Delta)$ ラウンドである． □

7 おわりに

本稿ではクラスタ構造をクリークをとし，分割数が極小となるような極大クリーク分割問題について考えた．極大クリーク分割を計算する自己安定アルゴリズムを示し，その正当性と安定するまでの時間複雑度を評価した．本稿で示したアルゴリズムは，既存の極大マッチングを求めるアルゴリズムを用いている．マッチングの結果をもとにマージという操作を行いクリーク間の関係を表すグラフ G^x を作るアルゴリズムを $\lceil \lg(\Delta+1) \rceil$ 段合成している．$\lceil \lg(\Delta+1) \rceil$ 分だけアルゴリズムが必要なため，アルゴリズムの入力であるグラフの最大次数 Δ(または Δ の上限) が予め与えられている必要がある．このような，入力の制限を必要としない方法を考える必要がある．

また，このアルゴリズムによって得られる分割の分割数などの評価が今後の課題である．

参考文献

[1] Bo Han and Weijia Jia. lustering wireless ad hoc networks with weakly connected dominating set. Journal of Parallel and Distributed Computing. Vol. 67,pp. 727-737, 2007.

[2] Kun Sun, Pai Peng, Peng Ning and Cliff Wang. Secure Distributed Cluster Formation in Wireless Sensor Networks. roceedings of the 22nd Annual Computer Security Applications Conference(ACSAC 22nd). pp. 131-140,2006.

[3] M.R. Cerioli, L. Faria, T.O. Ferreira and F. Protti. On minimum clique partition and maximum independent set on unit disc graphs and penny graphs: complexity and approximation. Electronic Notes in Discrete Mathematics. Vol. 18, pp.73-79, 2004.

[4] Stephen T. Hedetniemi, David P. Jacobs and Pradip K. Srimani. Maximal matching stabilizes in time $O(m)$. Information Processing Letters, Vol. 80, pp. 221-223, 2001.

[5] Shlomi Dolev. Self-Stabilization. MIT Press, 2000.

[6] Hiroko Ishii and Hirotsugu Kakugawa. A self-stabilizing algorithm for finding cliques in distributed systems. Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems(SRDS'02), pp. 390 - 395,2002.

//マクロ:
$isHead_i^x$:
  $head_i^x = v_i$
$Grd1_i^x$:
  $m_i^x = null \wedge \exists v_j \in N_i^x [m_j^x = v_i]$
$Grd2_i^x$:
  $m_i^x = null \wedge \exists v_j \in N_i^x [m_j^x = null]$
$Grd3_i^x$:
  $m_i^x \notin N_i^x \cup \{null\} \vee$
    $(m_i^x = v_j \wedge m_j^x \neq null \wedge m_j^x \neq v_i)$
//関数:
$merge_i^x$ {
if $(m_i^x = v_j)$**{**
  $C := C_i^x \cup C_j^x; N := N_i^x \cap N_j^x;$
  $head :=$ v_iと v_jで ID が大きい方
}else{
  $//m_i^x = null$ のとき
  $C := C_i^x$
  $N := N_i^x$
  $head = v_i$
}
$N := N - \{v_j \in N | \neg isHead_j^{x+1}$
  $\vee (m_j^x = u \wedge u \notin N \cup \{null\})\}$
return $(C, N, head)$
}

図 3: マクロおよび関数

//局所変数:
  $C_i^0, N_i^0, head_i^0$
//GC の集合
*[
$C_i^0 \neq \{v_i\} \vee N_i^0 \neq N_i \vee head_i^0 \neq v_i \rightarrow$
  $C_i^0 := \{v_i\}; N_i^0 := N_i; head_i^0 := v_i$
]

図 4: $Init_i$

//局所変数:
  m_i^x
//GC の集合
*[
$isHead_i^x \wedge Grd1_i^x \rightarrow$
  $m_i^x := v_j (v_j \in N_i^x \wedge m_j^x = v_i) \square$
$isHead_i^x \wedge \neg Grd1_i^x \wedge Grd2_i^x \rightarrow$
  $m_i^x := v_j (v_j \in N_i^x \wedge m_j^x = null) \square$
$isHead_i^x \wedge \neg Grd1_i^x \wedge \neg Grd2_i^x \wedge Grd3_i^x \rightarrow$
  $m_i^x := null$
]

図 5: マッチングアルゴリズム $Match_i^x$

//局所変数:
  $N_i^{x+1}, head_i^{x+1}, C_i^{x+1}$
//GC の集合
*[
$isHead_i^x \wedge$
$(C_i^{x+1}, N_i^{x+1}, head_i^{x+1}) \neq merge_i^x \rightarrow$
  $(C_i^{x+1}, N_i^{x+1}, head_i^{x+1}) := merge_i^x \square$
$\neg isHead_i^x \wedge head_i^x = v_j \wedge$
$(C_i^{x+1} \neq C_j^{x+1} \vee head_i^{x+1} \neq head_j^{x+1}) \rightarrow$
  $C_i^{x+1} := C_j^{x+1}; head_i^{x+1} := head_j^{x+1}$
]

図 6: マージアルゴリズム $Merge_i^x$