G-14

単方向リングにおいて部分集合問題を解決する匿名エージェント乱択アルゴリズム

Randomized Partial Gathering of Mobile Agents in Anonymous Unidirectional Ring

河田 倫和 †　柴田 将拡 ‡　首藤 裕一 †
Norikazu Kawata　Masahiro Shibata　Yuichi Sudo

大下 福仁 §　角川 裕次 †　増澤 利光 †
Fukuhito Ooshita　Hirotsugu Kakugawa　Toshimitsu Masuzawa

1. はじめに

分散システムは，複数の計算機(ノード) とそれらを結ぶ通信リンクで構成されたものである．分散システムは大規模化・複雑化が進んでおり，分散システムの設計は困難となっている．そのため，大規模化・複雑化する分散システムを設計するための有効な手法として，モバイルエージェントを利用した設計方法が注目を集めている[1,2,3,4,5,6,7,8,9]．モバイルエージェント(以下，エージェント) とは，ネットワーク中に分散していて，ノード間を自律的に移動できるソフトウェアを指す．複数のエージェントが通信などを行い協調的に動作することで，ユーザが達成しようとするタスクを代行し，分散システムの設計を容易にすることが期待される．

エージェントに関する問題に集合問題がある[2,3,4,5,6,8,9]．集合問題は，全てのエージェントが同一のノードに集まることを要求する問題である．集合問題が解決されれば，エージェント間での情報の共有や，動作の同期を行うことができる．これに対し，g -部分集合問題とは，入力値 $g(\leq k)$が与えられたときに，全てのエージェントが g 体以上ずつのグループに分かれて複数のノードに集合することを要求する問題である[1]．g -部分集合問題が解決されれば，g 体以上ずつのエージェントが集合している各グループ間で情報を交換・共有することができる．本稿では，リングネットワークにおいて，ノード数 n，エージェント数 k が分かっていない状況での g-部分集合問題を考察する．表 1 に既存研究との比較を示す．文献[1]は，エージェント数が分かっている状況で，g-部分集合問題を確率 1 で解決する乱択アルゴリズムを提案している．文献[2] は，エージェント数が分かっていない状況で，集合問題を確率 p ($0<p<1$) で解決する乱択アルゴリズムが提案されている．本稿では，条件を g-部分集合に緩めることにより，エージェント数が分かっていない状況でも，確率 1 で g-部分集合問題を解決する乱択アルゴリズムを提案する．

本稿の構成は以下のとおりである．2 節では，本稿で用いる用語の定義を行う．3 節では，本モデルにおいて，g -部分集合問題を確率 1 で解く乱択アルゴリズムを提案する．4 節では，本稿の結果をまとめる．

表 1：単方向リングでの集合／部分集合問題

	既存研究[1]	既存研究[2]	提案手法
集合/部分集合	部分集合	集合	部分集合
同期/非同期	非同期	非同期	非同期
白板	使用可能	使用可能	使用可能
前提知識	k	なし	なし
確率	1	$p(0<p<1)$	1

2. 諸定義

2.1 ネットワークモデル

単方向リングネットワークを $R = (V,L)$ と表す．ここで，$V=\{v_0, v_1, \ldots, v_{n-1}\}$ はネットワークのノード集合，$L=\{(v_i, v_{(i+1) \bmod n}) \mid 0 \leq i \leq n\text{-}1\}$は 2 つのノードを接続する一方向通信リンクの集合である．以降では，$v_{(i+1) \bmod n}$ を単に v_{i+1} と表す．ネットワーク中の各ノードには白板と呼ばれるノードのメモリが存在し，エージェントは自身が存在するノードの白板を読み書きすることができる．ノード$v_0, v_1, \ldots, v_{n-1}$は識別子を持たない．したがって，本エージェントにおけるエージェントは，白板の内容以外で 2 つのノードを区別する手段を持たない．

2.2 モバイルエージェント

ネットワーク上には k 体のエージェント$a_1, a_2, \ldots, a_k$が存在する．エージェントの集合を $A = \{a_1, a_2, \ldots, a_k\}$とする．各エージェントは確率的有限オートマトン $(S, \delta, s_{initial})$ で定義される．状態集合 S はエージェントが取りうる状態の集合である．全てのエージェントは同一の初期状態$s_{initial} \in S$ から実行を開始する．状態遷移関数 $\delta: S \times W \times RN \rightarrow S \times W \times M$ はエージェントの状態，白板の内容，乱数値から，次のエージェントの状態，次の白板の内容，移動するか否か有無を決める関数である．W は白板の状態集合を表す．RN は乱数の取り得る値の集合を表す．M ={*move*,*stay*}はエージェントの移動を表

† 大阪大学大学院情報科学研究科，Graduate School of Information Science and Technology, Osaka University

‡九州工業大学 大学院情報工学研究院電子情報工学研究系 Department of Computer Science and Electronics, Kyushu Institute of Technology

§ 奈良先端科学技術大学院大学, Nara Institute of Science and Technology

す．*move* はエージェントが移動することを表し，*stay* はエージェントが移動しないことを表す．全てのエージェントは同一の状態遷移関数に従って動作する．

本モデルはエージェント間の通信を含まないが，エージェントは白板の内容更新により他のエージェントに情報の伝達を行うことができる．エージェントはノード数 n やエージェント数 k の前提知識を持たない．エージェントの状態はエージェントが持つすべての変数の割り当てによって定義される．同様に，ノードの白板の状態は白板が持つすべての変数の割り当てによって定義される．

2.3 システムの状況

エージェントシステムの状況 c は，全エージェントa_iの状態$s_i \in S$，各ノードv_j の状態$w_j \in W$，各エージェントの位置の三つの要素で構成される．エージェントシステムの取りうる状況全ての集合を C とする．初期状況$c_0 \in C$ で各エージェントa_iが存在するノードは任意である．ただし，複数のエージェントが同じノードには存在していないものとする．各ノードv_jには$initial_j$という特別な変数があり，初期状況c_0でエージェントが存在するノードは *true*，エージェントが存在しないノードでは *false* が実行開始時に割り当てられている．白板に記憶されるその他の変数は，状況c_0において，（すべてのノードの白板に共通の）初期値を保持する．初期状況c_0においてすべてのエージェントの状態は$s_{initial}$であるから，初期状況は実行開始時における全エージェントの位置によってのみ決定される．

$A_i \subseteq A$ を空でない任意のエージェントの集合とする．状況 c において，A_i に属する全てのエージェントが状態遷移関数に従って動作した結果，エージェントシステムの状況がc'に遷移することを$c \xrightarrow{A_i} c'$ と表記する．あるエージェントが，自身の状態を変化させる，あるいはノード間を移動するとき，エージェントがアクションを行ったと表現する．同一ノード上の複数のエージェントがA_iに属していた場合，任意の順でアクションを行う．

どのエージェントもアクションできない状況を終了状況と呼ぶ．終了状況以降どのエージェントもアクションを行わない．初期状況から i ステップ後の状況をc_i と呼ぶ．実行 E は下記条件を満たす状況の列である．

- $E = c_0, c_1, \ldots$
- $c_i \xrightarrow{A_i} c_{i+1}$
- E は無限列，あるいは E=$c_0,\ldots,c_t$ でありc_tが終了状況である．

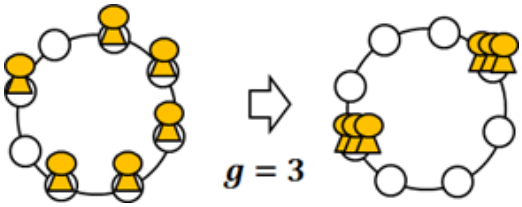
ここで任意の i に対して$A_i = A$ が成り立つとき，全てのエージェントが同一のタイミングで動作を行い続けるという意味で，実行 E を同期的であるという．本稿ではエージェントシステムの実行が同期的であることを仮定しない．

2.4 g- 部分集合問題

g -部分集合問題は，図 1 のようにネットワークにおいて任意のノードに分散している複数のエージェントが，g 体以上のエージェントで構成されるグループに分かれて，有限時間内にグループごとに同一のノードに集合することを目的とする．部分集合問題を解決するための入力値を g とする．ここで，入力を g とする g -部分集合問題を以下のように定義する．

[定義 2.1] 以下の条件を満たすとき，実行 E は g-部分集合問題を解決するという．
・ 実行 E は有限系列 E=$c_0,\ldots,c_t$ である．
・ 状況c_tにおいて，エージェントが存在するすべてのノードには g 体以上のエージェントが存在している．

図 1：g -部分集合問題の例



3. 提案アルゴリズム

本節では，エージェントがネットワークに関する前提知識を持たない場合に，g-部分集合問題を確率 1 で解決する乱択アルゴリズムを提案する．まず 3.1 節で提案アルゴリズムの概要を説明し，次に 3.2 節で詳細について述べる．その後，3.3 節で提案アルゴリズムの正当性を示す．

3.1 提案アルゴリズムの概要

提案アルゴリズムにおいて，エージェントはステップと呼ぶひとまとまりの動作を繰り返す．最初のステップ（ステップ 1 と呼ぶ）では，初期状況でエージェントが存在している k 個のノード各々に一様ランダムにブール値を割り当て(すなわち，確率 1/2 で 0，確率 1/2 で 1 を割り当てる．この値をステップ 1 における各ノードの ID と呼ぶ)，そうして得られる長さ k のビット列$b_0, b_1, \ldots, b_k$の周期k'_1を計算する．たとえば，k=12 のときビット列が 100010001000 であれば，このビット列は長さ 4 のビット列 1000 を 3 回繰り返すことで構成されているため，周期k'_1は 4 と計算する．その後，k/k'_1個のノード各々にk'_1個のエージェントが後述する処理によって移動する．この k/k'_1個のノードをステップ 1 における集合ノードと呼ぶ．このとき$k'_1 \geq g$ であればエージェントは動作を休止する．$k'_1 < g$ であるとき，エージェントは次のステップ，すなわ

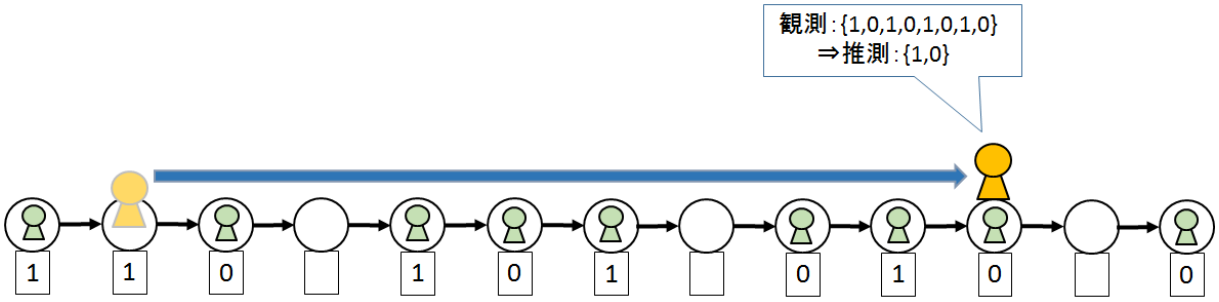
ちステップ 2 に移行する．ステップ 2 では，ステップ 1 と同様に，ステップ 1 における集合ノードすべてに 1 ビットの ID をランダムに割り当て，そうして得られる長さ k/k'_1 ビットのビット列の周期k'_2を計算し，$k/(k'_1\ k'_2)$個のノード各々に $k'_1\ k'_2$個のエージェントが集合する．$k'_1\ k'_2$ が g 以上であればエージェントは休止状態に移行し，そうでなければステップ 3 に移行する．各エージェントは，この処理を，$c_i \stackrel{\text{def}}{=} \prod_{l=1}^{i} k'_l \geq g$となるステップ i まで繰り返し，休止状態に移行する．上記処理が実現できたとき，明らかに k 個のエージェントは g-集合問題を解くことができる．以降では，ステップ i においてエージェントがk'_iを計算する方法，および，k'_iの計算後に各c_i個のエージェントが同一のノードに移動する方法を説明する．

まず，ステップ i におけるk'_iの計算方法について説明する．各エージェントは，リング上のノードを移動しながらステップ i の ID 列を記憶し，k'_iの推定を試みる．具体的には，ステップ i 開始時から観測した ID 列の履歴が$B = b_0, b_1, \ldots, b_l$であるとき，$\forall x, 0 \leq x \leq l-d: b_x = b_{x+d}$を満たす最小の d $(1 \leq d \leq \lfloor l/2 \rfloor)$を$k'_i$の推定値とする．この推定は，同じ ID 列を 4 回連続で観測するまで行う．すなわち，ステップ i 開始時から観測した ID 列の履歴を$b_0, b_1, \ldots, b_{4d-1}$とするときに，すべての t（$0 \leq t \leq d$-1）についてb_t=b_{t+d}=b_{t+2d}=b_{t+3d}が成り立つまで移動を継続する． このとき，k'_iの推定値は d となる．この推定はかならずしも正しいk'_iと一致するとは限らないが，誤った値を推定した場合には後述する方法で推定値を補正し，やがて正しいk'_iを得る．

次に，ステップ i において$c_i \stackrel{\text{def}}{=} \prod_{l=1}^{i} k'_l$個のエージェントが同一のノードに移動する方法を説明する．任意のビット列 X=$x_0, x_1, \ldots, x_l$を d'ビットだけ巡回シフトして得られるビット列$x_{d'}, x_{d'+1}, \ldots, x_l, x_0, x_1, \ldots, x_{d'-1}$を$X^{(d')}$と表記する．$k'_i$の計算後，エージェントは，前述の処理で観測した ID 列 B=$b_0, b_1, \ldots, b_{t-1}$に対して$B^{(d')}$が辞書順に最小の ID 列となるような $d'$$(0 \leq d' < t)$を計算し，$d'$ 回の移動を行う．こうすることで， k'_i個のエージェントが同一ノードへの移動できる．

ステップ i におけるk'_iの計算過程において， ID 列の周期の中に部分的な周期が 4 回以上繰り返される場合，エージェントがより小さい値を周期として誤認してしまい，k'_iの計算に失敗してしまう場合がある（図 2）．

図 2：周期の中に部分的な周期がある場合の図



この問題を避けるため，エージェントは移動のたびにステップ i におけるk'_iの現在の推定値 $k'[i]$を訪問先の各ノードの白板に書き込む．すでに白板にk'_iの推定値が保存されている場合，自身の推定値 $k'[i]$の方が大きい場合には上書きを行う．補題 2 で示されるように，ステップ i で k/c_i個の集合ノード各々に移動すべきc_i個のエージェントのうち，すくなくとも 1 体のエージェントは，ステップ i においてはじめて同じ ID 列を 4 回観測した時点で正しいk'_iを計算できていることがいえる．したがって，エージェントが周期より小さい推定値 $k'[i]$を算出し，どこかのノードで休止状態に移行したとしても，停止中のノードに正しいk'_iを算出したエージェントが訪れることで正しいk'_iを取得することができる．このとき，ステップ i の移動を再開し，集合すべきノードに移動する．また，ステップ i の移動中も，以前のステップ $i'$$(< i)$における ID 列の監視は継続し，その結果として推定値 $k'[i']$が拡大する場合にはステップ i'から再開することを行う．こうすることで，誤った $k'[i]$を記憶したまま，休止せずに次のステップに移行して移動を続ける場合も，やがて $k'[i]$が正しい値に修正されることが保証できる．

3.2 提案アルゴリズムの詳細

本節では提案アルゴリズムによるエージェントの動作の詳細について説明する．提案アルゴリズムの疑似コードを Algorithm1，2 に示す．

各エージェントが保持する変数について，i はステップ数，配列 $t[i]$ は各ステップを開始してからのエージェントの移動数，2 次元配列 $mem[i][]$ はステップ i で白板に書き込まれていた ID 列の記録，配列 $k'[i]$ はリングネットワークのステップ i における ID 列の周期k'_iの推定値， c はステップの完了時に同一ノードに集合するエージェントの数を記憶する変数である．各ノードv_j の白板で保持する変数について，$size_j[i]$はノードv_jを訪問したエージェントが保持していた推定値 $k'[i]$の最大値，$id_j[i]$ はノードv_jに割り当てられたステップ i における ID を記憶する変数である．また，Guess()，GatheringNode() は整数値を返す関数で，それぞれ，各ステップ i において記録した ID 列からk'_iを推定する関数，ステップを開始したノードから集合ノードまでに必要な移動数を計算する関数である． Check(i')は訪問中のノードのステップ i'における ID を確認し，これまでに観測したステップ i'の ID 列を照合することで，ステップ i'の ID 列の周期$k'_{i'}$の推定値 $k'[i']$に誤りがないかどうかの真偽を返す関数である． Move()はエージェントが現在存在するノードから，$initial_j$=*true* である次のノードまで移動する関数である．移動が完了した際には，

- AssignID()による ID の割り当て，

- 訪問ノードの ID の *mem* への記録,
- 各ステップ *i* の移動数 *t*[*i*]のインクリメント,
- 過去の全ステップ *i'*=0,1,...,*i*-1 に対しての Check(*i'*)による *k'*[*i'*]の検証
- 検証により *k'*[*i'*]の誤りが確認された際の GoBack(*i'*)の呼び出し

が行われる．AssignID()は，訪問中のノードに現在のステップ *i* における ID が割り当てられていない場合に，一様ランダムに 1 ビットを生成して ID として割り当てる関数である．関数 GoBack(*i'*)は，現在のステップを *i'*に戻すための処理を行う関数である．変数 *i'*より新しいステップの *k'*[]と *t*[]を 0 に初期化するとともに，変数 *i* を *i'*として現在のステップを *i'*に戻す．関数 SizeUpdate()は，ノードに記録されている推測値$size_j$ []を更新する関数である．ステップ 1 から現在のステップ *i* までの各ステップの推測値 *k'*[*i'*]について，$size_j$[*i'*]と比較する．推測値の方が大きい場合，それを新たな$size_j$[*i'*]とする．その場合，更新したステップ *i'*より大きなステップ *i''* > *i'*で記録した$size_j$[*i''*]が存在する場合，それらを削除する．

最初に Algorithm1 について説明する．まず，エージェントはステップ *i* と移動数 *t*[1]の初期化，関数 AssignID(1)による 1 ビットのランダム ID の書き込みを行う(1〜3 行目)．そして，1 ステップ目の ID をエージェントのメモリ *mem* に記録する (4 行目) ．他のエージェントによって既に ID が書き込まれていた場合，その ID を使用することに注意されたい．以降，ネットワークの移動を行い ID の列を観測することで周期k'_1を推定していく．まず，エージェントは関数 Move()によって初期状況でエージェントが存在したノードまで移動する．ID が書き込まれていなければ関数 AssignID(1)によって ID の書き込みを行う．そして，メモリ *mem* に ID の記録，移動数 *t*[1]のインクリメントを行う(7 行目の関数 Move())．このように移動を繰り返すと，エージェントは観測した ID 列に周期性を見つけることができる． ID に周期性を見つける，すなわち ID 列の繰り返しを 2 回観測すると，関数 Guess()によって ID 列の長さを推測できる．具体的には，ステップ 1 開始時から観測した ID 列の履歴が$B = b_0, b_1, \dots, b_l$であるとき，$\forall x, 0 \le x \le l - d: b_x = b_{x+d}$満たす最小の d $(1 \le d \le \lfloor l/2 \rfloor)$を$k'_1$の推定値とする．エージェントはノードを周回し，最終的に，同じ ID 列を 4 回連続で観測するまでリング上を移動する．移動中に自身の推定値 *k'*[1]よりも小さい推定値$size_j$が記載されているノードv_jを訪問した場合，自身の推定値を上書きする(9 行目)．これは，複数のエージェントが異なる推定をしていた場合，大きい方の推定が正しいということに基づく．ノードを 4 周して推定値を書き終えた後は，正しくk'_1を計算したエージェントは，観測した ID 列を，巡回シフトにより辞書式に最小のビット列に並びかえれば，同じ ID 列を共有している．このようにしてステップ 1 におけるエージェント数の推定を終える (6~9 行目)．次に，関数 GatheringNode()により集合ノードを決定し，関数 Move()を繰り返して，集合ノードまで移動する(11~13 行目)．*k'*[1]が g 以上の場合，*g*-部分集合に成功していると判断し，その場で休止状態に遷移する．エージェントの数が g 未満であった場合は *i* をインクリメントし，次のステップへ移る(15〜19 行目)．エージェントが，推定値 *k'*[1]が間違った状態で休止したときは，そのとき存在するノードをv_jとすると，やがて後続のエージェントが正しいk'_1の値を$size_j$[1]に書き込みにくる．すなわち，このとき，休止中のエージェントの推定値 *k'*[1]は $size_j$より小さくなるので，自身の推定値が間違っていることが分かる．このとき，エージェントは$size_j$ を自身の推定値 *k'*[1]に上書きする．この新しい *k'*[*i*]に基づいて，周期k'_1の推定からやり直す(20~24 行目).

次に，ステップ *i* の動作を考える．ステップ 1 ではエージェント数を推定していたが，ステップ *i* はそのステップにおけるグループ数を推定する．ステップ 1 と同様に，まず，エージェントは移動数の初期化，関数 AssignID(*i*)による 1 ビットのランダム ID の書き込みを行う．そして，ID をメモリ *mem* に記録する．他のエージェントによって既に ID が書き込まれていた場合，その ID を使用する． (18~19 行目)．その後，5 行目の while 文に戻り，エージェントは関数 Move()によって初期状態でエージェントが存在したノードまで移動する．ここで，各 *i'*<*i* について，*i'*ステップを開始してからの移動数（$initial_j$=*false* となるノードへの移動を除く）が，$\prod_{l=1}^{i'-1} k'[l]$ の整数倍となったとき，訪問中のノードはステップ *i'*-1 における集合ノードであると推測できる．よって，ステップ *i'*の ID が書き込まれていなければ ID の書き込みを行い，ステップ *i'*の移動数 *t*[*i'*]を 1 増加し， *mem* に ID を記録する（37〜39 行目）．この移動を繰り返し，ステップ 1 と同様に，同じ ID 列を 4 回連続で観測するまでリング上を移動する．移動中に自身の推定値 *k'*[*i*]よりも小さい推定値$size_j$が記載されているノードv_jを訪問した場合，自身の推定値を上書きする．このようにしてステップ *i* におけるエージェント数の推定を終える (6~9 行目)．次に，関数 GatheringNode()により，集合するノードを決定する．関数 Move()によって，移動数 *t*[1],…,*t*[*i*]のインクリメントを行いながら，集合ノードまで移動する(11~13 行目)．集合ノードに移動するエージェントの数は，1 ステップから *i* ステップまでの推測値が正しければ，その積 *k'*[1]* *k'*[2]*⋯* *k'*[*i*]となる．その値が g 以上の場合，その場で休止状態に遷移する． g 未満の場合はステップ数 *i* をインクリメントし，次のステップへ移る． ステップ 1 と同様に，任意の *i'*≤*i* について$k'_{i'}$の推定を誤って休止した場合には，やがて後続のエージェントから白板の変数$size_j$ [*i'*]を通して正しい *k'*[*i'*]の値を知ることができる．その場合，（*i'*<*i* の場合には）ステップを *i'*に戻し，ステップ *i'*における集合ノードに移動する．

このようにステップを繰り返すことで，やがて$c_i = \prod_{l=1}^{i} k'_l \ge g$となるステップの集合ノードで全エージェントが休止する．このとき，k 体のエージェントは *g*-部分集合問題を解いている．

Algorithm 1 提案アルゴリズム

Variables for Agent a
int *t*[] // *t*[*i*]はステップ *i* におけるエージェントの移動数
int *mem*[][] //白板の ID の記録用
int *k'* [] // *k'*[*i*]はステップ *i* における ID 列の推定周期
int h_{min} //集合ノードまでの移動距離
int c //現在のステップで集合するエージェントの数
int i //現在のステップ

Variables for Node v_j
int $size_j$[] //エージェントが推定したエージェントの数
int id_j[] //ランダム ID を格納
boolean $initial_j$

Main Routine of Agent a //エージェントが現在いるノードをv_jとする
1: $i \leftarrow 1$
2: $t[1] \leftarrow 0$
3: AssignID(1)
4: $mem[1][0] \leftarrow id_j[1]$
5: **while**(*true*) **do**
6: **while** Guess() = 0 or $t[i] < 4\ k'[i]$ **do**
7: Move()
8: $k'[i] \leftarrow \max(\text{Guess}(), k'[i])$
9: SizeUpdate()
10: **end while**
11: **while** $t[i] <$ GatheringNode() **do**
12: Move()
13: **end while**
14: $c \leftarrow k'[1] * k'[2] * \cdots * k'[i]$ //集合時に存在するはずのエージェント数
15: **if** $c < g$ **then**
16: $i \leftarrow i + 1$
17: $t[i] \leftarrow 0$
18: AssignID(i)
19: $mem[i][0] = id_j[i]$
20: **else**
21: $\forall i', k'[i'] = size_j[i']$である限り休止
22: $s = \min\{ i' \mid k'[i'] \neq size_j[i']\}$
23: $k'[i'] \leftarrow size_j[i']$ for all $i' = s, s+1, \ldots, i$
24: GoBack(s)
25: **end if**
26: **end while**

Algorithm 2 提案アルゴリズム

int Guess()
27: return ($\forall\ x (0 \le x \le t[i] - d)\ mem[i][x] = mem[i][x + d]$ 満たす最小の d $(1 \le d \le \lfloor t[i]/2 \rfloor)$. そのような d がなければ 0)

int GatheringNode()
28: return (長さ $k'[i']$のビット列 $mem[i][d'], mem[i][d'+1], \ldots, mem[i][d' + k'[i'] - 1]$が辞書順で最小となる d' $(0 \le d < k'[i']))+4k'[i]$

boolean Check(int i') //i'ステップ目の ID 列が正しいか確認する関数
29: return ($\forall\ l = k'[i'], k'[i']+1, \ldots, t[i'],\ mem[l]=mem[l - k'[i']]$)

void Move()
30: $initial_j = true$ となっているノードまで移動
31: AssignID(1)
32: $t[1] \leftarrow t[1] + 1$
33: $mem[1][t[1]] \leftarrow id_j[1]$
34: **for** $i' = 2$ **to** i **do**
35: **if** ($i' > 1$ and $t[i'-1]$が $k'[i'-1]$の整数倍) **then**
36: **if** Check($i'-1$) **then**
37: AssignID(i')
38: $t[i'] \leftarrow t[i'] + 1$
39: $mem[i'][t[i']] \leftarrow id_j[i']$
40: **else**

41: GoBack(*i'*-1)
42: **break**
43: **end if**
44: **end if**
45: **end if**

void AssignID(int *i'*)
46: **if** $id_j[i']$ = null **then**
47: $id_j[i']$ ← (1 ビットのランダム ID)
48: **end if**

void SizeUpdate()
49: **for** *i'*=1 to *i* **do**
50: **if** ($size_j[i']$ = null) **then**
51: $size_j[i']$ ←$k'[i']$
52: **if** ($k'[i']$ > $size_j[i']$) **then**
53: $size_j[i']$ ←$k'[i']$
54: $size_j[i'']$ ←null for all *i''*>*i'*
55: **else if** ($k'[i']$ < $size_j[i']$) **then**
56: **break**
57: **end if**
58: **end for**

void GoBack(int *i'*) //ステップ *i'*まで戻す際の動作
59: $t[l]$ ← 0 for all $l = i'+1, i'+2, \ldots, i$
60: $k'[l]$ ← 0 for all $l = i'+1, i'+2, \ldots, i$
61: $i \leftarrow i'$

3.3 提案アルゴリズムの正当性

本節では提案アルゴリズムの正当性を証明する．

[補題 1]（大下ら[2]）任意のビット列 *Y*，*Z* ににについて，*ZZZ* が *YYY* の接頭辞となるとき，*Z* は周期的であるか 2|*Z*|≤|*Y*|となる

$X = x_0, x_1, \ldots, x_{l-1}$を任意のビット列であるとする．任意の$i$ $(0 \le i \le n-1)$について$l_{i,4}$を次のように定義する．

$$l_{i,4}(X) = \min\{\, j > 0 \mid 0 \le j' \le j-1, x_{i+j'} = x_{i+j+j'} = x_{i+2j+j'} = x_{i+3j+j'} \,\}$$

ここでx_iの添え字についてなされた加算はすべて l を法とした加算である．

[補題 2]ビット列 $X = x_0, x_1, \ldots, x_{l-1}$が非周期的であるとき，ある i $(0 \le i < l)$について$l_{i,4}(X) = l$が成り立つ．

[証明] 以後，任意の i について$l_{i,4}(X)$を$l_{i,4}$と表記する．一般性を失わず$l_{0,4} = \max_j l_{j,4}$を仮定し，$t = l_{0,4}$とする．$t = l$がいえれば補題は証明できる．

まず，任意の整数$j \ge 0$について $l_{jt,4} = t$であることを示す．定義から$x_0,\ldots,x_{4t-1}$は長さ t のあるビット列 Y を用いて *YYYY* と表記できる．ここで *t'*= $l_{t,4}$として *t'*< *t* を仮定する．すると，$x_t,\ldots,x_{t+4t'-1}$は長さ *t'*のビット列 Z を用いて *ZZZZ* と表記できる．このとき当然ながら *ZZZ* は *YYY* の接頭辞となる． Z は定義から非周期的なビット列なので，補題 1 から *t'* ≤ *t*/2 がいえる．これは，*YY* が *ZZZZ* を接頭辞として持つことを意味するが，このことは$l_{0,4}$の最小性に矛盾する．したがって$l_{t,4} \ge t$がいえる．t は全ての j について，$l_{j,4}$のうち最大のものと仮定したため，$l_{t,4} = t$である．同様にして任意の j について$l_{jt,4} = t$がいえる．このとき，任意の整数$j \ge 0$について $l_{jt,4} = t$であるので，任意の j について$x_{jt}, x_{jt+1}, \ldots, x_{(j+4)t-1} = YYYY$がいえる．

t の定義より明らかに *t* ≥ 2 である．以降では，*t* < *l* を仮定して矛盾を導くことで *t* = *l* を示す．*t* が *l* の約数であれば，直ちに X の非周期性に矛盾する．*t* が *l* の約数でないとき，*d* = *l* mod *t* とする．このとき，0<*d*<*t* であり，$x_0, x_1, \ldots, x_{t-1} = x_{l-t+d}, x_{l-t+d+1}, \ldots, x_{d-1} = Y$であるから，$D = x_0, x_1, \ldots, x_{d-1}, E = x_d, x_{d+1}, \ldots, x_{t-1}$とすると，Y=DE=ED がいえる．したがって，次式を得る．

$$YYYY = DEDEDEDE$$
$$= DDEDEDEE$$

$= DDDEDEEE$
$= DDDDEEEE.$

これは$l_{0,4}$の最小性に矛盾する．これは $t < l$ と仮定したことによる．したがって $t = l$ を得る．

[系 1] $X = x_0, \ldots, x_{jt-1}$を周期 t のビット列とする．すなわち，$Y = x_0, \ldots, x_{t-1}$としたとき，X=$YY...Y$ である．このとき，ある i $(0 \leq i < t)$が存在して$l_{i,4}(X) = l_{i+t,4}(X) = \cdots = l_{i+(j-1)t,4}(X)$である．

補題 2 より明らか

[補題 3] i'を任意の正の整数とする．提案アルゴリズムのどのような実行においても，すべてのエージェントがステップ i' を開始した場合には，やがてすべてのエージェントがステップ i'を完了する，すなわち変数 i が i'より大きい，もしくは i=i' で休止状態であるような状況に到達する．そのとき，どのエージェントもk'_iを正しく計算できている，すなわち，変数 k'[i]にk'_iを記憶している．

[証明]ステップ i'-1 の集合ノードを$u_0, u_1, \ldots, u_l$とする(i'=1 のときは$initial_j$=$true$ となるノードを$u_0, u_1, \ldots, u_l$とする．)
系 1 より，ある d $(0 \leq d < t)$が存在して，任意の非負整数 s についてu_{d+st}からステップ i'を開始するエージェントは $4k'_i$回のステップ i'に関する移動（ステップ i'-1 のある集合ノードu_jからu_{j+1}への移動）を行い，正しいk'_iを計算することがいえる．さらに，この $4k'_i$回のステップ i'に関する移動のうち，$2k'_i$回以降の移動の際には，正しいk'_iを各ノードの白板の変数$size_j$[i']に書き込む．s は任意の非負整数であるから，これは，やがてすべてのノードの白板に正しいk'_iが書き込まれることが保証される．したがって，すべてのエージェントは自力でk'_iを正しく計算するか，あるいは白板に書かれた$size_j$[i']を確認することで正しいk'_iを k'[i']に記憶し，ステップ i'の集合ノードに移動してステップ i'を完了する．

[定理 1] n, k の前提知識が与えられていない場合に，提案アルゴリズムは確率 1 で g-部分集合問題を解決する．

[証明] i'を任意の正の整数とする．補題 3 より，すべてのエージェントはk'_1を正しく計算してステップ 1 を完了する．$k'_1 \geq g$ の場合，すべてのエージェントは休止状態になるが，エージェントが存在するどのノードにもk'_1体のエージェントがいるので，このとき g-部分集合問題は解決されている．$k'_1 < g$ のとき，すべてのエージェントがステップ 2 を開始する．同じく補題 3 より，すべてのエージェントはk'_2を正しく計算してステップ 2 を完了する．$c_2 = k'_1 k'_2 \geq g$ であればすべてのエージェントは休止状態となり，g-部分問題は解決される．$c_2 < g$ のときすべてのエージェントがステップ 3 を開始する．以下，同様に補題 3 を繰り返し適用することで，$c_i = k'_1 \ldots k'_i < g$ である限り次のステップを繰り返し行い，$c_i \geq g$ となったときには g-部分集合問題は解決されていることが保証される．各k'_iは確率変数となるが，1/2 以上の確率で 2 以上となるので，確率 1 でやがて g-部分集合問題は解決される．

4. まとめ

本稿では，非同期単方向リングネットワークにおけるエージェントの g-部分集合問題を確率 1 で解決する乱択アルゴリズムを提案した． 提案アルゴリズムは，ノード数 n やエージェント数 k を知ることなく，与えられた g をもとに g-部分集合問題を解く．

今後の課題として，提案アルゴリズムの期待総移動数の改善を考えている．提案アルゴリズムの期待総移動数は $O(kn)$となるが，これは自明な下界 $O(gn)$とは一致しない．$O(kn)$ よりも小さい期待総移動数で g-部分集合問題を解決するアルゴリズムの実現を検討する予定である．

謝辞

本研究は JSPS 科研費 JP26280022, JP26330084, JP16K00018, JP17K19977 の助成を受けたものです.

参考文献

[1] M. Shibata, S Kawai, F. Ooshita, H. Kakugawa, T. Masuzawa． Partial Gathering of Mobile Agents in Asynchronous Unidirectional Rings． Theoretical Computer Science, 2016.

[2] F. Ooshita, S. Kawai, H. Kakugawa, T. Masuzawa. Randomized Gathering of Mobile Agents in Anonymous Unidirectional Ring Networks， IEEE Transactions on Parallel and Distributed Systems， 2014.

[3] E. Kranakis, D. Krizanc, and E. Markou. The mobile agent rendezvous problem in the ring. Synthesis Lectures on Distributed Computing Theory, Vol. 1, pages 1-122, 2010.

[4] E. Kranakis, N. Santoro, C. Sawchuk, and D. Krizanc. Mobile agent rendezvous in a ring. Proc. of the 23rd International Conference on Distributed Computing Systems. pages 592-599, 2003.

[5] P. Flocchini, E. Kranakis, D. Krizanc, N.Santoro, and C.Sawchuk. Multiple mobile agent rendezvous in a ring. Proc. of the 6th Latin American Theoretical Informatics, LNCS, Vol. 2976. pages 599-608, 2004.

[6] L. Barriere, P. Flocchini, P. Fraigniaud, and N. Santoro. Rendezvous and election of mobile agents: impact of sense of direction. Theory of Computing Systems, Vol. 40, No. 2, pp. 143-162, 2007.

[7] T. Suzuki, T. Izumi, F. Ooshita, H. Kakugawa, T. Masuzawa. Move-optimal gossiping among mobile agents. Theoretical Computer Science Volume 393, lssues 1-3,20 March 2008,Pages 90-101

[8] L. Gasieniec, E. Kranakis, D. Krizanc, and X. Zhang. Optimal memory rendezvous of anonymous mobile agents in a unidirectional ring. Proc. of the 32nd International Conference on Current Trends in Theory and Practice of Computer Science, LNCS, Vol. 3831. pages 282-292, 2006.

[9] P. Flocchini, E. Kranakis, D. Krizanc, F. L. Luccio, N. Santoro, and C. Sawchuk. mobile agents rendezvous when tokens fail. Proc. of the 11th International Colloquium on Structural Information and Communication Complexity, LNCS, Vol. 3104. pages 161-172, 2004.