

相互観測性を持つブロックリーダ決定問題を解く 自己安定アルゴリズムについて

尾関 豊大[†] 金 鎔煥[†] 片山 喜章[†]

[†] 名古屋工業大学大学院工学研究科工学専攻情報工学系プログラム

E-mail: [†]t.ozeki.743@stn.nitech.ac.jp, ^{††}{kim,katayama}@nitech.ac.jp

あらまし グラフ $G = (V, E)$ (V : ノード集合, E : 辺集合) とノードの部分集合 $P \subseteq V$ が与えられたとき, 任意の 2 つのノード $i, j \in P$ の間に P に含まれる他のノードを含まない最短パスが 1 つ以上存在する場合, 集合 P を相互観測集合という. 本研究ではグラフ G が与えられたとき, G を二連結ブロックに分割して, 相互観測性を満たすように各ブロックにおいて高々 1 つのリーダを選ぶことを考える. すなわち, 全てのブロックのリーダから成るノード集合は相互観測集合となる. 本稿では, 最大相互観測ブロックリーダ決定問題を定義し, 任意の連結無向グラフにおいて最大サイズの相互観測ブロックリーダ集合を構築する自己安定アルゴリズムを提案する.

キーワード 相互観測性, 自己安定アルゴリズム, ブロックリーダ決定問題

1. はじめに

複数の計算機が通信リンクによって接続され, 連携して作業を分担して共通のタスクを達成するシステムを分散システムという. この分散システムを正しく動作させるために各計算機が実行するアルゴリズムを分散アルゴリズムという. 多くの分散アルゴリズムはある決められた初期状況から実行した場合に正しく動作することを保証する. しかし, 実際の分散システムにおいては各計算機が保持する変数やプログラムカウンターの値の一時的変化や通信リンクの切断のような一時的な故障 (一時故障) が発生することでシステムの状況が一時的に変更され, アルゴリズムの動作の正しさが保証されない場合が存在する. Dijkstra が文献 [1] で初めて提案した自己安定アルゴリズム (self-stabilizing algorithm) は任意のシステム状況から実行を開始し, 有限時間内に目的とする状況 (正当な状況) に到達することの出来る分散アルゴリズムである. つまり, 一時故障が発生してネットワーク状況が変更されることで正当な状況から任意のネットワーク状況に遷移しても, その状況を初期状況として再び有限時間内に正当な状況に到達する, 一時故障に対して故障耐性を持つ分散アルゴリズムであるため, 一時故障に対して柔軟に対応する必要がある分散システムの動作に適している. そのため WSN (Wireless Sensor Network) のように計算機が移動し, ネットワークの構造が変化する可能性のあるネットワークに利用することが出来る. WSN のような無線通信ネットワーク上での効率的な通信や低消費電力化のために用いられる手法の 1 つにクラスタリングがある [2]. クラスタリングとはネットワーク全体をいくつかの部分ネットワーク (クラスタ) に分割し, クラスタ内とクラスタ間の通信に分けることで無駄な通信を減らす構造である. 1 つのクラスタは 1 つの代表となる計

算機 (クラスタヘッド) とその他の計算機 (メンバプロセス) によって構成される. クラスタリングはエネルギー効率, 通信の効率化, トポロジーの効率的な管理等多くの利点を持っている [3]~[5]. クラスタリングにおける問題点として, 通信による負担がクラスタヘッドに集中することが挙げられる. WSN は電力やストレージなどに限りがあるためリソース的な制約が大きい. そのためクラスタ間の通信においてクラスタヘッドを経由しない最短経路での通信を行うことでクラスタヘッドの無線通信の回数を減らし, クラスタヘッドの負担を軽減することが出来るため, クラスタヘッドの決定は重要な問題である.

グラフ理論における相互観測性について述べる. グラフ $G = (V, E)$ (V : ノード集合, E : 辺集合) とノード集合 $P \subseteq V$ が与えられたとき, 任意の 2 つのノード $i, j \in P$ について i, j 間に P に含まれる他のノードを含まない最短パスが 1 つ以上存在するとき, i, j は相互観測 (Mutually-Visible) であるといい, 集合 P の任意の 2 ノードが相互観測である場合, P を相互観測集合 (Mutual-Visibility set) という. このとき, G 上で最も要素数の多い相互観測集合を最大相互観測集合 (Maximum Mutual-Visibility set) といい, 最大相互観測集合の要素数を相互観測数 (Mutual-Visibility Number) という [6]. 図 2 は最大相互観測集合の一例である. 図のグラフにおいて要素数が 6 以上の相互観測集合は存在しないため, 要素数が 5 である相互観測集合が最大相互観測集合であり, 相互観測数は 5 となる.

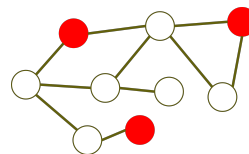


図 1: 相互観測集合の例

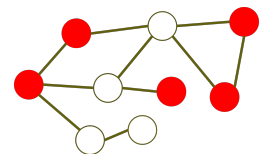


図 2: 最大相互観測集合の例

ネットワークをグラフ、クラスタヘッドを相互観測集合のノードと見立てることで相互観測集合をクラスタリングに応用することが出来る。しかし、要素数が最大の相互観測集合を見つけることは難しく、特に以下の相互観測問題として知られる判定問題は NP 完全として知られている。

[定義 1] (相互観測問題 (*Mutual-visibility problem*)) グラフ G と整数 K が与えられたとき、 $|S| \leq K$ であるような相互観測集合 S が存在するかどうかを判定する。このような問題を**相互観測問題**という。

そこで、グラフを二連結ブロックに分割し、各ブロックで相互観測性を保証するリーダを決定することを考える。ブロックをクラスタ、リーダをクラスタヘッドと見立てることでブロック単位でのクラスタリングが可能となる。そしてクラスタヘッド同士が相互観測性を保証することで2つのクラスタヘッド間に他のクラスタヘッドが含まれない最短経路で通信を行うことが可能になり、分散システムにおいてクラスタヘッドの負担を軽減することが可能になる。しかし、全てのブロックにおいて相互観測であるリーダを決定するのは不可能な場合が多い。例えば図3のグラフでは円で囲まれた部分のブロックにおいて他のブロックリーダの相互観測性を保証したブロックリーダを選ぶことは不可能である。

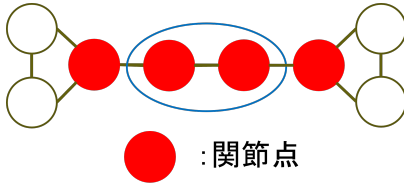


図3: ブロックリーダが選出出来ないブロックの例

そこで、本研究ではグラフ上で相互観測性を保証するブロックリーダの集合で要素数が最大であるものを最大相互観測ブロックリーダ集合と定義し、任意の連結無向グラフ上に最大相互観測ブロックリーダ集合を構築する自己安定アルゴリズムを提案する。

2. 関連研究

2.1 相互観測集合の性質に関する研究

文献 [6] では相互観測問題や与えられたノードの部分集合が相互観測集合かどうかを判定する相互観測確認問題 (*Mutual-Visibility Test*) の計算複雑度の解析、木や完全二部グラフ等の特定のグラフクラスにおける相互観測数の境界について研究している。

2.2 General Position Set の性質に関する研究

文献 [7] において、相互観測問題と類似した性質を持つ集合として General Position Set が提案されている。相互観測集合は集合内の任意の2点間の最短経路上に相互観測集合のノードを3点以上含まないような経路が1つ以上存在するのに対し、General Position Set は集合内の任意の2点間の全最短経路が相互観測集合のノードを3点以上含まないような経路である。グラフ G 上に存在する要素数最大の General Position Set の要素

数を general position number($gp(G)$) といい、特殊なグラフクラスにおいて general position number の解析に関する研究が行われている [8]~[10]。このように、特定のグラフクラスにおける相互観測数の境界や *generalPositionnumber* の解析に関する研究についてはについては数多く行われているが、グラフにおける相互観測集合や General Position Set を構築するアルゴリズムに関してはほとんど研究されていない。

3. 諸定義

本章では、本論文で取り扱うグラフ、関節点とブロック、相互観測性、モデル、自己安定アルゴリズム、公平な合成、問題定義について説明する。

3.1 グラフ

無向グラフ G は2項組 $G = (V, E)$ で定義され、 V は空でないノード集合、 E は無向辺の集合である。2ノード i, j の間に無向辺が存在している場合、 i と j は隣接しているといい、 $(i, j) \in E$ と表す。ノード i に隣接するノードの集合を i の開近傍といい、 $N(i)$ と表す。ノード i に隣接するノードの数を i の次数といい、 $deg(i) (= |N(i)|)$ と表す。無向グラフ G について相異なるノードの系列 (x_0, x_1, \dots, x_n) が $(x_i, x_{i+1}) \in E (0 \leq i \leq n)$ を満たすとき、この系列をパスといい、 n をパスの長さという。2つのノード i, j の間に存在するパスの中で最も長さの短いパスを i, j の最短パスという。 i, j の最短パスの長さを i, j の距離といい、 $dist(i, j)$ と表現する。

3.2 関節点とブロック

グラフ $G = (V, E)$ の関節点 $v \in V$ とは取り除かれたときにグラフ G が非連結になるノードのことを関節点という。また、関節点を含まないようなグラフ G の極大部分グラフのことをグラフ G のブロックという。図4は関節点及びブロックの例である。

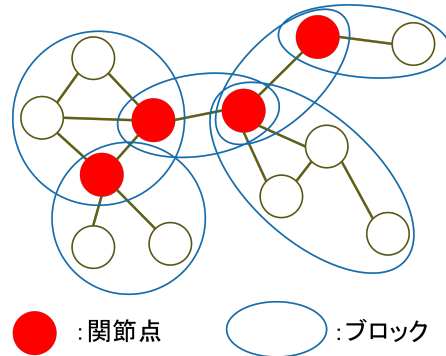


図4: 関節点・ブロックの例

3.3 相互観測性

グラフ $G = (V, E)$ 、ノードの部分集合 $P \subseteq V$ が与えられたとき、2つのノード $i, j \in P$ について i, j 間に $P \setminus \{i, j\}$ のノードを1つも含まないような最短パスが存在するとき、 i, j は相互観測であるといい、集合 P の任意の2ノードが相互観測である場合、 P を相互観測集合という。

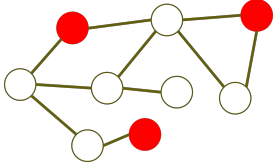


図 5: 相互観測集合である例

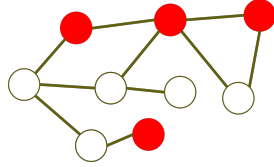


図 6: 相互観測集合でない例

3.4 システムモデル

グラフ $G = (V, E)$ においてノードをプロセス (以下ノードと表す), 辺を通信リンクとすることで分散システムをグラフで表す. このとき, G は単純連結無向グラフとする. G には特別なノード $s \in V$ が存在すると仮定する. このとき s は自身が特別なノードであることを知っているものとする. 各ノードは相異なる ID (識別子) を持たないものとする. ノード間の通信は状態通信モデルを仮定する. 状態通信モデルとはノード i において自身と隣接するノード ($N(i)$) の内部状態, すなわち変数や ID の読み込みが可能であり, 自身の内部状態にのみ書き込みが可能なモデルである.

[定義 2] (グラフの状況) グラフにおける状況は各ノード p_i の状態 q_i を列挙することによって表す. つまり, グラフに n 個のプロセスが存在する場合, あるグラフの状況を $c = (q_0, q_1, \dots, q_{n-1})$ と表す. また, G の取り得る全ての状況の集合を C と表す. プロセス p_i の取り得る内部状態の集合を Q_i とした場合, $C = (Q_0 \times Q_1 \times \dots \times Q_{n-1})$ である.

$c \in C$ をグラフの任意の状況, $P \subseteq V$ をノードの部分集合と表す. P に属するノードが任意のアルゴリズム A に従って 1 原子動作を行いグラフの状況が c' になるとき, $c \mapsto (P, A)c'$ と表す.

[定義 3] (スケジュールとアルゴリズムの実行) 空でないノードの部分集合の無限系列 $T = (P_0, P_1, \dots)$ をスケジュールという. ネットワーク状況の無限系列 $E = (c_0, c_1, \dots)$ とする. このとき, 各 $i (0 \leq i)$ について $c_i \mapsto (P_i, A)c_{i+1}$ を満たすのであれば, E を「初期状況 c_0 , スケジュール T におけるアルゴリズム A の実行」という.

[定義 4] (公平なスケジュール) スケジュール T において全てのノード $p_i \in V$ が無限回現れるとき, T は公平であるという.

本論文では公平なスケジュールを仮定し, 以降は公平なスケジュールを単にスケジュールという. ノードの部分集合 P_i に含まれるノードは 1 原子動作のみを行うことが出来る. スケジュール T の各 $P_i (i \leq 0)$ に含まれるノードの数と 1 原子動作によっていくつかのモデルが考えられる. 本論文では D デモンを扱う. D デモンでは $P_i (i \leq 0)$ に含まれるノードの数と 1 原子動作について以下のように示される.

- (ノード数) スケジュール T の各 $P_i (i \leq 0)$ において, $|P_i| \leq 1$ を満たす
- (原子動作) 全ての隣接ノードから内部状態を読み込み, アルゴリズム A に従って自身の内部状態を書き換える

3.5 自己安定アルゴリズム

アルゴリズム A があるグラフの状況の集合 $C_L \subseteq C$ について次の 2 つの条件を満たすとき「アルゴリズム A は C_L に対して

自己安定である」という.

- (1) (到達可能性) 任意のグラフの状況 $c_0 \in C$ と任意のスケジュール T に対し, 初期状況 c_0 , スケジュール T におけるアルゴリズム A の実行 $E = (c_0, c_1, \dots)$ において $c_i \in C_L$ であるような $c_i (i < 0)$ が存在する.
- (2) (閉包性) 任意の状況 $c \in C_L$ と任意の空でないノードの集合 $P \subseteq V$ に対し, $c \mapsto (P, A)c'$ であれば $c' \in C_L$ である

すなわち, 任意の初期状況 $c_0 \in C$, スケジュール T におけるアルゴリズム A の実行において, 有限時間内に $c' \in C_L$ に到達し, 1 度 C_L 内の状況に達した後はずっと C_L 内の状況であり続けるとき, アルゴリズム A は状況の集合 C_L に対して自己安定であるとする. このとき, $c \in C_L$ をアルゴリズム A に対して正当な状況という.

3.6 公平な合成

複数の自己安定アルゴリズムを合成することを公平な合成といい, 自己安定アルゴリズム A_1, A_2, \dots, A_k が正当な状況に到達した時の出力を A_{k+1} の入力として用いることが出来る. 公平な合成により 2 つの自己安定アルゴリズム A_1, A_2 を合成すると次の条件を満たす.

- A_1 が書き込む変数を A_2 は読み込むが書き込まない
 - A_2 が書き込む変数を A_1 は読み込みも書き込みもしない
- 各ノードは 1 原子動作で A_1, A_2 それぞれの原子動作をこの順で実行する. A_2 は A_1 が正当な状況に到達したかどうかにかかわらず実行される. A_1 が正当な状況に到達した後, A_2 も有限時間内に正当な状況に到達する. これは 3 つ以上の自己安定アルゴリズムの公平な合成についても明らかである.

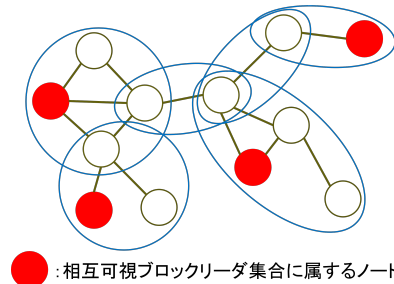
3.7 問題定義

最大相互観測ブロックリーダ集合を以下のように定義する.

[定義 5] (最大相互観測ブロックリーダ集合) グラフ $G = (V, E)$ が与えられたとき, 以下の条件を満たすノードの集合 $S \subseteq V$ を最大相互観測ブロックリーダ集合という.

- S は G における相互観測集合である.
- 各ブロックは S のノードを高々 1 つ含む
- $|S| < |S'|$ であるような S' は存在しない

任意の単純連結無向グラフ $G = (V, E)$ と特別なノード $s \in V$ が与えられたとき, 任意の初期状況から最大相互観測ブロックリーダ集合を出力する問題を最大相互観測ブロックリーダ決定問題という. 図 7 は最大相互観測ブロックリーダ集合の例である.



●: 相互可視ブロックリーダ集合に属するノード

図 7: 最大相互観測ブロックリーダ集合の例

また、 G 上のある最大相互観測ブロックリーダ集合 MVB_G とブロックの集合 B_G について、 $|MVB_G| \leq |B_G|$ が成り立つ。

4. 提案アルゴリズム

本章では、任意の単純無向連結グラフ G が与えられたとき、 G 上に最大相互観測ブロックリーダ決定問題を解く自己安定アルゴリズムについて示す。

4.1 基本戦略

まず各ブロックで最大相互観測ブロックリーダ集合に含むことの出来るノードが存在するかを判定するため、与えられたグラフをブロック単位に分割して各ブロックごとに関節点間の最短経路に含まれているか判定を行う。しかし、関節点は複数のブロックに属しているために各ブロックを区別することが出来ない。そのため全ブロックに相異なる識別子 (ブロック ID) を割り振ることでブロックを区別可能にする。ブロック ID にはノードの識別子 (ID) が用いられるが、与えられたグラフの各ノードは相異なる ID を持たないため、まず特別なノード s を根として DFS (深さ優先探索) 木を構築し、この DFS 木における各ノードの訪問順を ID として利用する。この DFS と各ノードの ID を用いて関節点かどうかの判定とブロック ID の割り当てを行う。ブロック ID は 2 つのノードの ID の組で表現され、図 8 は DFS を行い、訪問順とブロック ID を割り当てた例である。実線は DFS 木に含まれている辺を表し、点線は含まれていない辺を表す。各ノードにある数字は DFS 木における訪問順を表し、各辺にある数字の組はブロック ID (この例ではブロックに属しているノードの中で最小の訪問順と 2 番目に小さい訪問順の順序対) を表している。

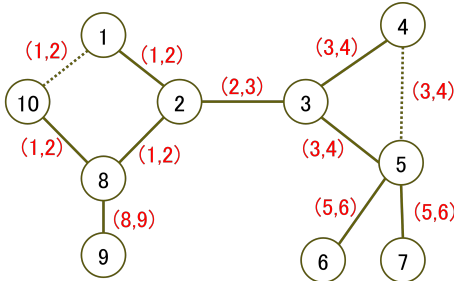


図 8: 訪問順とブロック id を割り当てた例

次に、各ブロックに属するノードについて最大相互観測ブロックリーダ集合に含むことの出来るノードかどうかを判定する。この際に含むことの出来るノードの条件として以下の条件を満たすかどうかを判定する。

- 関節点または関節点間の全最短パスに含まれるノードでない

関節点かどうかはブロック ID 割り当ての際に判定をしているため、各ノードは関節点間の全ての最短パスに含まれるノード (以下 Essential Point とする) かどうかを判定し、最大相互観測ブロックリーダ集合に含むことが出来るノードかどうか判定を行う。最後に、各ブロックにおいて最大相互観測ブロックリーダ集合に含むことの出来るノードの中から最小 ID のノードを

ブロックリーダとし、最大相互観測ブロックリーダ集合に含める。全てのブロックにおいてブロックリーダが決定したとき、最大相互観測ブロックリーダ集合が構築され、アルゴリズムは終了する。

4.2 アルゴリズムの流れ

任意の単純無向連結グラフ上に最大相互観測ブロックリーダ決定問題を解くための手順について以下に示す。

step1 s を根とする DFS 木を構築する。

step2 関節点を特定する。

step3 全ブロックにブロック ID を割り当てる。

step4 各ノードが Essential Point かどうかを判定する

- (1) 各ブロックの関節点からの最短距離を計算し、2 つの関節点間の最短パスに含まれるか判定する。
- (2) 関節点間で重みを伝搬し、Essential Point かどうかを判定する。

step5 最大相互観測ブロックリーダ集合を構築する

- 各ブロックにおいて関節点または Essential Point でないノードでリーダ選挙を行う

これらの手順を実現するために、複数の自己安定アルゴリズムを公平な合成により合成する。step1 については任意のグラフ上に $t \in V$ を根とする自己安定アルゴリズム [11] を用いる。step2,3 については [12] の *BlockAssign()* アルゴリズムを用いる。本章では step4,5 の自己安定アルゴリズムを提案する。また、各アルゴリズムにおける変数、マクロ、関数はノード p_i のものを表し、アルゴリズムは p_i にて実行されるものとする。

4.3 step1: s を根とする DFS 木構築アルゴリズム

step1 では特別なノード s を根とする DFS 木を構築し、各ノードに訪問順を割り当てる。この訪問順を識別子とみなすことで、 G 上の全ノードに相異なる識別子 (ID) を割り当てることが出来る。以降は s を根とする DFS 木における訪問順と各ノードの ID を区別せずに用いる。step1 の変数は以下の通りである。

変数:

$parent_i \in N(i) \leftarrow S$ を根とする DFS 木の親ノードを保存する変数

$id_i \in (N) \leftarrow$ DFS 木における訪問順を変数

マクロ:

$Child_i \stackrel{\text{def}}{=} \{j \in N(i) | parent_j = i\} \leftarrow$ 子ノードの集合

アルゴリズムについては既存のアルゴリズム [11] を用いる。step1 において有限時間内に到達する正当な状況を以下のように定義する。

[定義 6] (step1 の正当な状況) G 上に s を根とする DFS 木が構築され、 $parent_i$ が DFS 木における親ノード、 id_i が DFS 木における訪問順である状況を step1 における正当な状況という。

4.4 step2: 関節点特定アルゴリズム

step2 では G 上の各ノードが関節点かどうかを特定する。関

節点の特定には Lowlink を用いる。Lowlink とは自身の隣接ノードの訪問順と自身の子孫ノード、すなわち DFS 木上で自身を根とする部分木における自身以外のノードの隣接ノードの訪問順の最小値である。DFS 木において、根は子ノードが 2 つ以上存在すれば関節点である。根以外のノードは自身の訪問順と Lowlink の値が一致する子ノードが存在するならば関節点である。

step2 の変数は以下の通りである。

step1 からの入力:
$parent_i \in N(i)$
$id_i \in \mathbb{N}$
変数:
$ap_i \in \{0, 1\} \leftarrow$ ノード i が関節点かどうかを保存する変数
$low_i \in \mathbb{N} \leftarrow$ ノード i の lowlink の値を保存する変数
マクロ
$smallID_i \stackrel{\text{def}}{=} \{j \in N(i) id_i < id_j\} \leftarrow$ ノード i の ID よりも小さい ID を持つノード i の隣接ノードの集合

アルゴリズムは Lowlink の計算に既存研究 [12] の $BlockAssign()$ アルゴリズム、関節点かどうかの判定に $Algorithm1$ を用いる。これらのアルゴリズムの実行順は最初に $BlockAssign()$ アルゴリズムを実行し、 $BlockAssign()$ アルゴリズムの実行が完了した後に $Algorithm1$ を実行するような順となる。

Algorithm 1 ap_i 決定アルゴリズム

```

1: if  $i = s$ 
2:   if  $|Child_i| \geq 2$ 
3:     then  $art_i \leftarrow 1$ ;
4:   else  $art_i \leftarrow 0$ ;
5: else if  $\exists j \in Child_i : id_i = low_j$ 
6:   then  $art_i \leftarrow 1$ ;
7: else  $art_i \leftarrow 0$ ;

```

step2 において有限時間内に到達する正当な状況を以下のよう

[定義 7] (step2 の正当な状況) G において関節点である全ノードの $art_i = 1$ 、関節点でない全ノードの $art_i = 0$ である状況を step2 の正当な状況という。

4.5 step3: ブロック ID 割り当てアルゴリズム

step3 では G の全ブロックに相異なる識別子 (ブロック ID) を割り当てる。割り当てには既存研究 [12] の $BlockAssign()$ アルゴリズムを用いる。このアルゴリズムではブロック ID は step1 で構築した DFS 木上でのノードの訪問順の順序対で表され、各ノードは自身に接続する各辺ごとにブロック ID を割り当てる。step3 の変数は以下の通りである。

step1 からの入力:

$art_i \in \{0, 1\}$

変数:

$block_i[j] \in \mathbb{N}^2 \leftarrow$ ノード i に接続する辺 (i, j) の属するブロックの ID

マクロ

$N_i(B) \stackrel{\text{def}}{=} \{block_i[j] | j \in N(i)\} \leftarrow$ ノード i が属しているブロックのブロック ID の集合

アルゴリズムについては既存研究 [12] の $BlockAssign()$ アルゴリズムを用いる。step3 において有限時間内に到達する正当な状況を以下のように定義する。

[定義 8] (step3 の正当な状況) G 上に存在する全ブロックに相異なるブロック ID が割り当てられている状況を step3 の正当な状況という。

各ブロックにおけるブロック ID はブロック内のノードの中で最小の訪問順と 2 番目に小さい訪問順の順序対となる。

4.6 step4: Essential Point 特定アルゴリズム

step4 では、 G 上のブロックの各ノードにおいて、関節点間の Essential Point かどうかを判定する。2 点間の Essential Point について以下に定義する。

[定義 9] (Essential Point) グラフ上の各ブロックに存在する任意の 2 点の関節点 i, j において、 ij 間の全ての最短パスに含まれるノードを i, j の Essential Point という。

step4 の変数は以下の通りである。

step1 からの入力

$ap_i \in \{0, 1\}$

変数

$dis_i[|V|] \in \mathbb{N} \leftarrow$ 関節点までの最短距離を保存する配列

$SDP_i[|V|] \in \{0, 1\} \leftarrow$ 関節点まで最短距離で辿ることが出来るノードかどうかを保存する配列

$SPP_i[|V|][|V|] \leftarrow$ 関節点 j, k 間の少なくとも 1 つの最短パスに含まれているかどうかを保存する二重配列

$Num_i[|V|][|V|] \in \mathbb{N} \leftarrow$ 関節点 j, k 間で自身が持つ重みの分子を保存する二重配列

$Den_i[|V|][|V|] \in \mathbb{N} \leftarrow$ 関節点 j, k 間で自身が持つ重みの分母を保存する二重配列

$reportNum_i[|V|][|V|] \in \mathbb{N} \leftarrow$ 関節点 j, k 間の最短経路上で隣接するノードが持つ重みの分子を保存する二重配列

$reportDen_i[|V|][|V|] \in \mathbb{N} \leftarrow$ 関節点 j, k 間の最短経路上で隣接するノードが持つ重みの分母を保存する二重配列

$EP_i[|V|][|V|] \leftarrow$ 関節点 j, k 間の全最短パスに含まれているかどうかを保存する二重配列

マクロ

$Forward_i(j) \stackrel{\text{def}}{=} \{k \in N(i) | dis_i[k] = dis_j[k] - 1\} \leftarrow$ 関節点 j までの距離が 1 小さいノードの集合

$Backward_i(j) \stackrel{\text{def}}{=} \{k \in N(i) | dis_i[k] = dis_j[k] + 1\} \leftarrow$ 関節点 j までの距離が 1 大きいノードの集合

関数

$MinDis_i(j) \stackrel{\text{def}}{=} \underset{k \in N(i)}{\text{argmin}} dis_k[j] \leftarrow$ 隣接ノードからノード j までの最短距離を返す関数

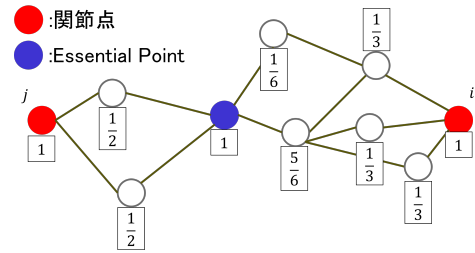
Algorithm2 では、各ブロックに属するノードがブロック内の 2 つの関節点間の少なくとも 1 つの最短パスに含まれているかどうかを判定する。 i が関節点の場合、自身までの距離が 0 であるため、関節点である自身までの距離を保存する変数である $dis_i[i]$ を初期化する。 i が関節点でない場合、隣接するノードから関節点までの距離を参照し、その中で最小の距離に 1 を加えた数を自身までの距離として保存することで関節点までの距離 dis_i を更新する。ある関節点 j までの距離の情報が他のある関節点 k に伝搬したとき、関節点 k に隣接するノードは自身が k から j までの最短パスに含まれていることが分かるため、 $SDP_i[k]$ を 1 に更新し、 i に隣接するノードが関節点までの距離を用いて SDP_i を伝搬することで最短パスに含まれているかどうかを伝搬していく。この際、関節点 j から同様に最短パスに含まれているかどうかの情報を伝搬し、パスの両端の関節点 j, k について $SDP_i[j] = 1$ かつ $SDP_i[k] = 1$ であるとき、関節点間の最短パスに含まれているため $SPP_i[j][k]$ を 1 にする。

```

1: if  $ap_i = 1$ 
2:    $dis_i[i] \leftarrow 1$ ;
3: else
4:    $SDP_i[i] \leftarrow 0$ ;
5:   for each  $jinV$ 
6:      $dis_i[j] \leftarrow MinDis_i[j] + 1$ ;
7:   if  $\exists j \in N[i] : ap_j = 1 \wedge dis_j[k] = dis_i[k] + 1$ 
8:      $SDP_i[k] \leftarrow 1$ ;
9:   else  $SDP_i[k] \leftarrow 0$ ;
10:  if  $\exists j \in N[i] : SDP_i[k] = 1 \wedge dis_j[k] = dis_i[k] + 1$ 
11:     $SDP_i[k] \leftarrow 1$ ;
12:  else  $SDP_i[k] \leftarrow 0$ ;
13:  if  $SDP_i[j] = 1 \cap SDP_i[k] = 1$ 
14:    then  $SPP_i[j][k] \leftarrow 1$ ;
15:  else  $SPP_i[j][k] \leftarrow 0$ ;

```

は $\frac{1}{3}$ となる.



- 1: **if** $ap_i = 1 \cap SPP_i(i, j) = 1$
- 2: $reportNum_i(i, j) \leftarrow 1$;
- 3: $reportDen_i(i, j) \leftarrow |Forward_i(j)|$;
- 4: **else**
- 5: **for each** $j \in V, k \in V$
- 6: $Den_i(j, k) \leftarrow \underset{l \in Backward_i(k)}{\operatorname{argmax}} \quad reportDen_l(j, k)$;
- 7: $Num_i(j, k) \leftarrow \sum_{l \in Backward_i(k)} reportNum_l(j, k) * (Den_i(j, k) / Den_l(j, k))$;
- 8: $reportDen_i(j, k) \leftarrow Den_i(j, k) * |Near_i(j)|$
- 9: $reportNum_i(j, k) \leftarrow Num_i(j, k)$;
- 10: **if** $\exists j \exists k \in N(i) : Num_i(j, k) = Den_i(j, k)$
- 11: **then** $EP_i \leftarrow 1$;
- 12: **else** $EP_i \leftarrow 0$;

[定義 10] (step4 の正当な状況) G 上に存在する各ブロックに属する関節点のうち 2 点を結ぶ最短パスに対して, 全最短パスに含まれている全てのノードの $EP_i = 1$, そうでない全てのノードの $EP_i = 0$ である状況を step4 の正当な状況という.

step5 では G 上に最大相互観測ブロックリーダ集合の構築を行う。全ブロックにおいて step1 で判定した関節点, そして step4 で判定した Essential Point を除いたノードの中から 1 つリーダを決定し, そのノードを最大相互観測ブロックリーダ集合内のノードとすることで最大相互観測ブロックリーダ集合を構築する。

$$block_i[j] \in \mathbb{N}^2$$

変数

$report_i(block_i[j]) \in \mathbb{N} \cup \{\perp\} \leftarrow$ 辺 (i, j) が属するブロックにおけるリーダー候補の ID

$leader_i(block_i[j]) \in \mathbb{N} \cup \{\perp\} \leftarrow$ 辺 (i, j) が属するブロックにおいて選ばれたリーダーの ID

$MVB_i \leftarrow \{0, 1\} \leftarrow$ 最大相互観測ブロックリーダー集合に含まれているかどうかを保存する変数

関数

ブロック B に属する隣接ノードのうち ID が自身よりも小さいノードのリーダー候補の中で ID が最小のリーダー候補の ID を返す関数：

$$Election_i(B) = \begin{cases} \underset{j \in smallID_i}{\operatorname{argmin}} \quad report_j(B) & \text{if } \exists j \in N_j(B) : report_j(B) \neq \perp \\ report_i(B) & \text{otherwise} \end{cases}$$

algorithm4 では各ノードがブロック内でのリーダー候補となるようなノードの id を $report_i$ に保存し、その中で ID が最小のノードをブロック内のリーダーとして決定する。 i が関節点または Essential Point である場合は最大相互観測ブロックリーダー集合の候補にならないため、 $report_i$ を空 (\perp) に更新する。 そうではない場合は $report_i$ を自身の $ID(id_i)$ に更新する。 全ノードにおけるリーダー候補が決定すると、各ブロックにおけるリーダーの決定を行う。 各ブロックの中で ID が最小のノードは自身のリーダー候補をリーダーとして決定する。 ID が最小ではないノード自身より ID が小さいノードが決定したリーダーと自身のリーダー候補を比較し、その中で ID が最小のノードをリーダーとして決定し、 $leader_i$ を更新する。 $leader_i$ が id_i と一致したノードはブロックにおけるリーダーとなるため、 MVB_i を 1 に更新し、ブロック内の他の全ノードは MVB_i を 0 に更新する。

Algorithm 4 MVB_i 決定アルゴリズム

```
1: if  $ap_i \neq 1 \wedge EP_i \neq 1$ 
2:    $report_i \leftarrow id_i$ ;
3: else
4:    $report_i \leftarrow \perp$ ;
5: for each  $j \in BlockSet_i$ 
6:   if  $\forall k \in BlockSet_i : id_k < id_i$ 
7:     then  $leader_i \leftarrow report_i$ ;
8:   else
9:     if  $Election_i(j) < report_i(j)$ 
10:       $leader_i(j) \leftarrow Election_i(j)$ ;
11:   else  $leader_i(j) \leftarrow report_i(j)$ 
12: if  $leader_i(j) = id_i$ 
13:    $MVB_i \leftarrow 1$ ;
14: else
15:    $MVB_i \leftarrow 0$ ;
```

step5 において有限時間内に到達する正当な状況を以下のよう

に定義する。
[定義 11] (step5 の正当な状況) G 上に最大相互観測ブロックリーダー集合が構築され、集合に属している全ノードの $MVB_i = 1$ 、属していない全ノードの $MVB_i = 0$ である状況を step5 の正当

な状況という。

5. おわりに

本論文では、最大相互観測ブロックリーダー決定問題を定義し、問題を解くアルゴリズムを提案した。今後の課題として、提案アルゴリズムの正当性の証明、時間計算量・空間計算量の解析が考えられる。

文 献

- [1] E.W. Dijkstra, “Self-stabilizing systems in spite of distributed control,” Communications of the ACM, vol.17, no.11, pp.643–644, 1974.
- [2] A. Zeb, A.K.M.M. Islam, M. Zareei, I.A. Mamoon, N. Mansoor, S. Baharun, Y. Katayama, and S. Komaki, “Clustering analysis in wireless sensor networks: The ambit of performance metrics and schemes taxonomy,” International Journal of Distributed Sensor Networks, vol.12, no.7, p.4979142, 2016.
- [3] D. Karaboga, S. Okdem, and C. Öztürk, “Cluster based wireless sensor network routings using artificial bee colony algorithm,” **, vol.18, no.17, pp.847–860, 2012.
- [4] C.E. Perkins, E.M. Belding-Royer, S.R. Das, and M.K. Marina, “Performance comparison of two on-demand routing protocols for ad hoc networks,” IEEE Personal Communications, vol.8, pp.16–28, 2001.
- [5] N. Vlahic and D. Xia, “Wireless sensor networks: to cluster or not to cluster?,” 2006 International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM’06), pp.258–268, 2006.
- [6] G.D. Stefano, “Mutual visibility in graphs,” Applied Mathematics and Computation, vol.419, no.15, p.126850, 2022.
- [7] P. Manuel and S. Klavzar, “The general position problem in graph theory,” Bulletin of the Australian Mathematical Society, vol.850, pp.177–187, 2018.
- [8] P.D. Manuel and S. Klavzar, “The graph theory general position problem on some interconnection networks,” Fundamenta Informaticae, vol.163, no.4, pp.339–350, 2018.
- [9] B.S. Anand, S.V.U. Chandran, M. Changat, S. Klavzar, and E.J. Thomas, “Characterization of general position sets and its applications to cographs and bipartite graphs,” Applied Mathematics and Computation, vol.359, no.15, pp.84–89, 2019.
- [10] S. Klavzar and B. Patkos, “On general position sets in cartesian products,” Result in Mathematics, vol.76, no.123, pp.**–**, 2021.
- [11] S. Devismes, “A silent self-stabilizing algorithm for finding cut-nodes and bridges,” Parallel Processing Letters, vol.15, no.01n02, pp.183–198, 2005.
- [12] Y. Kim, M. Shibata, Y. Sudo, J. Nakamura, Y. Katayama, and T. Masuzawa, “A self-stabilizing algorithm for constructing a minimal reachable directed acyclic graph with two senders and two targets,” Theoretical Computer Science, vol.874, no.12, pp.1–14, 2021.