Vol.2010-AL-128 No.3
2010/1/26

トポロジ変化に対して出力の変化数を最小化する全域木構成分散アルゴリズム

高田篤史[†1]　山内由紀子[†2]　大下福仁[†1]
角川裕次[†1]　増澤利光[†1]

動的な分散システムにおける重要な問題の一つに，トポロジ変化に対して全域木を再構成する問題 (全域木更新問題) がある．全域木は，分散システム内の各ノードが，木上の隣接ノード集合や木上の親ノードを出力として管理することによって表される．頻繁な出力の変更はシステムの可用性を低下させるため，出力の変化数を最小化して全域木更新問題を解くことが望ましい．これまでに，全域木更新問題を解く分散アルゴリズムは提案されているが，出力の変化数の最小化を目標としているものはない．本稿では，出力の変化数の最小化を目標とする 3 つの全域木更新問題を定式化し，これらの問題を解く分散アルゴリズムを提案する．

Distributed Algorithms to Update Spanning Trees with Minimizing Output Changes in Response to Topology Changes

Atsushi Takada ,[†1] Yukiko Yamauchi ,[†2]
Fukuhito Ooshita ,[†1] Hirotsugu Kakugawa [†1]
and Toshimitsu Masuzawa[†1]

One of the most important properties of distributed algorithms for dynamic distributed systems is to update a spanning tree (SPT) in response to topology changes. An SPT is represented by outputs of each node: tree neighbors and the parent on the tree. Some distributed algorithms to update an SPT have been presented but no algorithms aim to minimize the number of output changes. Frequent output changes reduce the availability of the system. Hence, it is desirable to update the SPT with minimizing the number of output changes. This report formalizes three SPT update problems with the aim of minimizing the number of output changes and presents distributed algorithms for the problems.

1. はじめに

分散システムとは，ネットワークで相互に接続された複数の計算機 (以降，ノードと呼ぶ) からなるシステムである．分散システム内の各ノードは出力変数 (以降，出力と呼ぶ) を管理し，ユーザや外部システムはノードの出力を用いて他のアプリケーションを実行する．近年，移動無線端末の普及に伴い，モバイルアドホックネットワークのような動的な分散システムが注目を集めている．動的な分散システムでは，ノードの参加，離脱もしくはノードの移動による通信リンクの出現，消滅により，ネットワークのトポロジが時々刻々と変化する．そのため，動的な分散システムでは，トポロジ変化に応じた出力の更新が必要となる．しかし，出力を頻繁に変更すれば，システムの可用性は大きく低下する．よって，トポロジ変化後のネットワークに対して出力を更新するとき，出力の変更を最小化することが望ましい．動的な分散システムにおける重要な問題の一つに，トポロジ変化後のネットワークに対して全域木を求める問題 (**全域木更新問題**) がある．全域木は，ネットワーク上でメッセージの放送を行う場合などに利用されており，全域木を構成するリンクを介してメッセージを配送することで，効率のよい放送が行える．これまでに，全域木更新問題を解く分散アルゴリズムは提案されているが，出力の変化数の最小化を目標としているものはない．

本稿では，出力の変化数の最小化を目標とする 3 つの全域木更新問題を新たに定式化する．全域木を表す出力として，ネットワーク内の各ノードは，木上の隣接ノード集合と木上の親ノードを管理する．第 1 の問題では，木上の隣接ノード集合に着目し，各ノードの木上の隣接ノード集合に追加される要素数と削除される要素数の総和を最小化する．第 2 の問題では，木上の親ノードに着目し，ネットワーク全体で親を変化させるノード数を最小化する．第 3 の問題では，木上の隣接ノード集合と木上の親ノードの両方に着目し，各ノードの木上の隣接ノード集合に追加される要素数と削除される要素数の総和の最小化 (第 1 の問題) を実現するもとで，ネットワーク全体で親を変化させるノード数の最小化 (第 2 の問題) を実現する．さらに，本稿では，定式化した 3 つの問題を解く分散アルゴリズムを提案する．各問題を解く分散アルゴリズムのメッセージ複雑度は，それぞれ $O(n \log k + m)$，$O(n^2)$，

†1 大阪大学 大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University
†2 奈良先端科学技術大学院大学 情報科学研究科
Graduate School of Information Science, Nara Institute of Science and Technology

$O(kn+m)$ である．ここで，n はノード数，m はリンク数，k はトポロジ変化直後の状況 (初期状況) に存在するフラグメント数を表す．フラグメントとは，初期状況において，解の全域木の連結な部分グラフ (部分木) で表されるネットワークである．

2. 諸 定 義

2.1 モ デ ル

分散システムのネットワークは，ノードを頂点，ノード間の通信リンク (以後，リンクと呼ぶ) を辺とみなしたグラフ $G=(V,E)$ で表す．ここで，V はノード集合，E はリンク集合を表し，V の要素数を $n=|V|$，E の要素数を $m=|E|$ とする．E は，V の異なる 2 要素の非順序対の集合である．ここで，$(v_i,v_j)\in E$ のとき，ノード v_i,v_j 間にリンクが存在し，v_i と v_j は**隣接**するという．各 v_i について，v_i の隣接ノード集合を $Neig_i$ とする．各 v_i,v_j について，$v_i\in Neig_j$ ならば $v_j\in Neig_i$ である．各ノード v_i は，固有の**識別子**といくつかの**局所変数**をもつ．ノードの局所変数は，システムに与えられる**入力変数**，外部で観測される**出力変数**と，計算の途中に利用される**内部変数**に分類する．

本稿では，通信モデルとして**非同期式メッセージ通信モデル**を用いる．つまり，各リンク (v_i,v_j) の各方向にメッセージチャンネル (以後，**チャンネル**と呼ぶ) が存在し，チャンネルを介したメッセージの送受信によってノード同士が通信する．各ノード v_i について，チャンネル (v_j,v_i) を v_i の入射チャンネル，チャンネル (v_i,v_j) を v_i の出射チャンネルと呼ぶ．本稿では，FIFO(First In First Out) チャンネルを仮定する．また，チャンネル上ではメッセージの損失が起こらないと仮定する．さらに，任意の送信されたメッセージは有限時間内に受信されるものと仮定する．

ノードの**状態**は，そのノードのすべての局所変数の値の組で定義する．システムの**状況**は，ネットワーク内のすべてのノードの状態と，すべてのチャンネル上のメッセージの組で定義する．各ノード v_i は，次の一連の動作を**原子動作**として実行する．

(1) ある入射チャンネル (v_x,v_i) からメッセージ m_x を取り出す (受信する)．
(2) m_x に基づく内部動作を行う．
(3) 各出射チャンネル (v_i,v_y) にいくつかのメッセージを挿入する (送信する)．ただし，必要がなければチャンネルにメッセージを挿入せずともよい．

非同期式メッセージ通信モデルでは，2 つ以上のノードが同時に原子動作を行っても，これらの原子動作は同時に実行された他の原子動作による影響を受けない．そのため，本稿では任意の時点でただ 1 つのノードが 1 つの原子動作を行うものと仮定する．アルゴリズムの**計算**は，状況の系列 $E=c_0,c_1,c_2,\ldots,c_t$ で定義する．c_{i+1} は c_i においてある 1 つのノードが 1 つの原子動作を行うことによって得られる状況である．c_0 は問題の入力から定まる状況であり，**初期状況**と呼ぶ．c_0 にはいくつかの始動ノードが存在し，始動ノードは自発的に自身の局所アルゴリズムの実行を開始できるものとする．また，始動ノード以外のノードは，他のノードからメッセージを受信することによって，自身の局所アルゴリズムの実行を開始できるものとする．

2.2 全域木更新問題の定義

本稿では，全域木が構成されているネットワークにおいて，トポロジ変化 (ノードの参加，離脱もしくはリンクの出現，消滅) が生じた際に，そのトポロジ変化に応じて全域木を更新 (再構成) する問題を**全域木更新問題**として定義する．ただし，トポロジ変化の前後いずれでもネットワークは連結とし，根ノード $v_0\in V$ が常にネットワーク内に存在する (離脱しない) ことを仮定する．さらに，本稿では，トポロジ変化によって隣接ノード集合に変化が生じたノードが始動ノードであることを前提とする．G 内の各ノード v_i は，ネットワークの (根付き) 全域木におけるノードの接続関係を表すため，親を表す出力変数 $P(v_i)$ と木上の隣接ノード集合を表す出力変数 $N^T(v_i)$ を管理する．$P(v_i)$ には，ある 1 つの隣接ノード $v_j\in Neig_i$ または $\perp$ を格納する．$N^T(v_i)$ には，全域木上で隣接するノードの集合を格納する．隣接する任意の 2 ノード v_i と v_j について，$P(v_j)=v_i$ を満たすとき，v_i を v_j の親ノード，v_j を v_i の子ノードと呼び，v_i と v_j の間に**親子関係**が成立しているという．ネットワーク G に対して，G の全域木が構成されている状況を**解状況**と呼び，以下のように定義する．ただし，以下では全域木の根となるノードを v_0 とする．

定義 1　解状況

ネットワーク $G=(V,E)$ において，すべてのノード $v_i\in V$ が次の全条件を満たす状況を**解状況**と呼ぶ．

- $P(v_0)=\perp$
- $v_i\neq v_0\Longrightarrow[P(v_i)\in Neig_i]$
- $N^T(v_i)=\{v_j\in Neig_i\mid P(v_j)=v_i\vee P(v_i)=v_j\}$
- v_i から親をたどれば，v_0 に到達する

全域木更新問題の初期状況 c_0 において，G 内の各ノード v_i はトポロジ変化直前における局所変数の値を保持している．また，v_i の隣接ノード集合 $Neig_i$ がトポロジ変化によって変化したとき，v_i はトポロジ変化後の $Neig_i$ を保持している．ただし，トポロジ変化直前

Vol.2010-AL-128 No.3
2010/1/26

において $P(v_i) = v_j$(または，$v_j \in N^T(v_i)$) を満たす v_j が，c_0 において $v_j \notin Neig_i$ を満たすとき，v_i は $P(v_i) = \perp$ を保持している (または，v_j が $N^T(v_j)$ から削除されている) ものとする．さらに，v_i が参加ノードならば，$P(v_i) = \perp$，$N^T(v_i) = \phi$ を保持しているものとする．初期状況 c_0 が満たす条件を以下に定義する．

定義 2　初期状況

ネットワーク $G = (V, E)$ の初期状況 c_0 において，すべてのノード $v_i \in V$ は以下の全条件を満たす．

- $P(v_0) = \perp$
- $v_i \neq v_0 \Longrightarrow [P(v_i) \in Neig_i \cup \{\perp\}]$
- $N^T(v_i) = \{v_j \in Neig_i \mid P(v_j) = v_i \vee P(v_i) = v_j\}$
- v_i から親をたどれば，$P(v_r) = \perp$ となるノード v_r に到達する

本稿では，与えられたネットワーク G に対して，定義 2 を満たす任意の初期状況 c_0 から解状況 c_f に到達させる問題を考える．さらに，本稿では，各ノード v_i の出力変数 $P(v_i), N^T(v_i)$ について，出力の変化数の最小化に関する 3 つの全域木更新問題を以下のように定義する．

定義 3　出力の変化数の最小化に関する全域木更新問題

各ノード v_i について，定義 2 を満たす初期状況 c_0 における $P(v_i), N^T(v_i)$ の値をそれぞれ $P_0(v_i), N_0^T(v_i)$ とする．また，解状況 c_f における $P(v_i), N^T(v_i)$ の値をそれぞれ $P_f(v_i), N_f^T(v_i)$ とする．与えられたネットワーク G に対して，最小化する目的関数の違いにより，3 つの全域木更新問題を以下のように定義する．

全域木更新問題 1　目的関数：$\mathcal{F}_\mathcal{N} = \sum_{v_i \in V} \left(|N_0^T(v_i) - N_f^T(v_i)| + |N_f^T(v_i) - N_0^T(v_i)|\right)$

全域木更新問題 2　目的関数：$\mathcal{F}_\mathcal{P} = |\{v_i \mid P_0(v_i) \neq P_f(v_i)\}|$

全域木更新問題 3　第 1 目的関数：$\mathcal{F}_\mathcal{N}$，第 2 目的関数：$\mathcal{F}_\mathcal{P}$

$\mathcal{F}_\mathcal{N}$ は，c_f において，c_0 から各ノードの木上の隣接ノード集合に追加された要素数と削除された要素数の総和を表し，$\mathcal{F}_\mathcal{P}$ は，c_f において，c_0 から親を変化させたノード数を表す．図 1 に各問題で求める全域木の例を示す．図 1 の矢印は，子ノードが親ノードを指す．図 1(a) はネットワーク G を，図 1(b) は全域木更新問題 1 で求める全域木を，図 1(c) は全域木更新問題 2 で求める全域木を，図 1(d) は全域木更新問題 3 で求める全域木を表す．

3. アルゴリズム

本節では，全域木更新問題 1，2，3 について，各問題を解く分散アルゴリズムを示す．

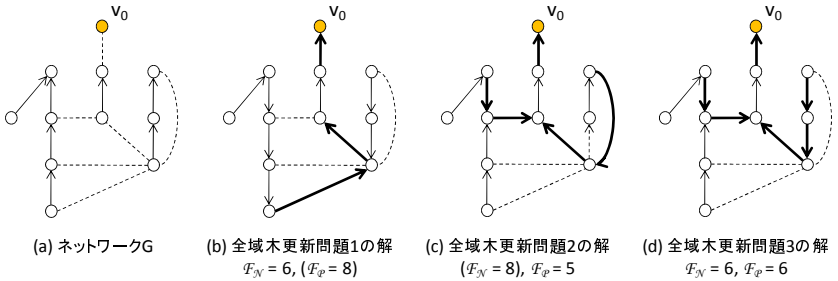

図 1　全域木更新問題 1，2，3 で求める全域木

3.1　全域木更新問題 1 を解く分散アルゴリズム UPD1

全域木更新問題 1 を解く分散アルゴリズム UPD1 は，目的関数 $\mathcal{F}_\mathcal{N}$ を最小化して全域木を求める．これは，G の初期状況 c_0 に存在する**フラグメント**を利用して全域木を構成することで実現する．フラグメントは以下のように定義する．

定義 4　フラグメント

与えられたグラフ $G = (V, E)$ について，E' を親子関係が成立している 2 ノード間のリンクの集合とする．G の部分グラフ $G' = (V, E')$ の極大な各連結成分を G の全域木のフラグメントという．

全域木更新問題の初期状況 c_0 では，各ノードから親をたどれば，$P(v_r) = \perp$ となる v_r に到達する．つまり，各フラグメントには閉路が存在せず，v_r を根とする木を構成する．以降，v_r をフラグメントの根ノードと呼ぶ．

辺 $e \in E$ について，e の両端が同じフラグメントに属すならば e を**内部辺**と呼び，そうでなければ e を**外部辺**と呼ぶ．ある外部辺によって接続される 2 つのフラグメントを 1 つのフラグメントに合併する操作をフラグメントを**結合**するという．UPD1 では，初期状況 c_0 のフラグメントを結合することによって全域木を構成する．c_0 には $k(\leq n)$ 個のフラグメントが存在するものとする．ネットワークは連結と仮定しているので，$k-1$ 個の外部辺を加えることによって全域木を構成できる．一方，全域木を構成するには $k-1$ 個の外部辺の追加が必要である．従って，この処理で得られる全域木は明らかに全域木更新問題 1 の解である．UPD1 では，フラグメントを結合して全域木を構成するために，朴らのアルゴリズム[4](PMHT 法と呼ぶ) を利用する．PMHT 法は，トポロジ変化後のネットワークに対して，フラグメントの結合を利用して重み最小全域木を求めるアルゴリズムである．UPD1

では，PMHT 法を用いてフラグメントの結合を行い，全域木を構成する．

アルゴリズムの性能

UPD1 では，PMHT 法を用いてフラグメントの結合を行うため，そのメッセージ複雑度は PMHT 法に等しい．PMHT 法は，$k(\leq n)$ 個のフラグメントが存在する状況から開始すれば，メッセージ複雑度 $O(n \log k + m)$ で全域木を求められる．以上より，UPD1 について以下の定理が導かれる．

定理 1 UPD1 はメッセージ複雑度 $O(n \log k + m)$ で，全域木更新問題 1 を解く．ここで，$k(\leq n)$ は初期状況 c_0 で存在するフラグメントの数である．

3.2 全域木更新問題 2 を解く分散アルゴリズム UPD2

全域木更新問題 2 を解く分散アルゴリズム UPD2 は，目的関数 $\mathcal{F}_{\mathcal{P}}$ を最小化して全域木を求める．これは，後述の方法で G に対して重み付き有向グラフ G' を構成し，G' 上でノード v_0 を根とする**重み最小全域有向木**(MDST と表す) T^d を求めることで $\mathcal{F}_{\mathcal{P}}$ の最小化を実現する．UPD2 の概略を以下に示す．

(1) 与えられたネットワーク $G = (V, E)$ に対して，重み付き有向グラフ $G' = (V, E')$ を以下のように定める．
- $(v_i, v_j) \in E$ のとき，かつそのときに限り，$(v_i, v_j) \in E', (v_j, v_i) \in E'$ とする．
- 各有向辺 $(v_i, v_j) \in E'$ について，v_j が v_i の親ノードの場合に限り (v_i, v_j) の重みを 0 とし，その他の場合は (v_i, v_j) の重みを 1 とする．

(2) G' 上で v_0 を根とする MDST T^d を構成する．

G' の構成法より，T^d に含まれる有向辺の重み総和は，G 内で親を変化させるノード数に一致する．従って，MDST T^d を構成すれば全域木更新問題 2 の解が得られる．UPD2 では，Humblet のアルゴリズム[2](H 法と呼ぶ) を用いて T^d を求め，全域木を構成する．

アルゴリズムの性能

H 法はメッセージ複雑度 $O(n^2)$ で MDST を求める．UPD2 において，G から G' を構成する際には，各ノードの局所計算のみで十分であり，メッセージを要しない．また，G' 上で T^d を求める際には H 法を利用する．よって，UPD2 に要するメッセージ数は H 法に等しい．以上より，UPD2 について以下の定理が導かれる．

定理 2 UPD2 はメッセージ複雑度 $O(n^2)$ で，全域木更新問題 2 を解く．

3.3 全域木更新問題 3 を解く分散アルゴリズム UPD3

3.3.1 アルゴリズムの概略

全域木更新問題 3 を解く分散アルゴリズム UPD3 は，第 1 目的関数として $\mathcal{F}_{\mathcal{N}}$ を，第 2 目的関数として $\mathcal{F}_{\mathcal{P}}$ を最小化して全域木を求める．UPD3 は，UPD1，UPD2 を実現するそれぞれのアイデアを組み合わせて実現する．まず，第 1 目的関数 $\mathcal{F}_{\mathcal{N}}$ を最小化するため，G の初期状況 c_0 に存在するフラグメントを結合することで全域木を構成する．次に，第 2 目的関数 $\mathcal{F}_{\mathcal{P}}$ を最小化するため，後述の方法で G に対して重み付き有向グラフ $G' = (V', E')$ を構成し，G' 上で MDST を求める．以下では，G 内のノードを $v_i \in V$，G' 内の頂点を $u_i \in V'$ と記述する．UPD3 は次の 2 フェーズからなる．

フェーズ 1 与えられたネットワーク $G = (V, E)$ に対して，重み付き有向グラフ $G' = (V', E')$ を以下のように定める．
- G の各フラグメント F_i を G' の 1 つの頂点 u_i とする．
- G の異なる 2 つのフラグメント間にリンクが存在するとき，かつそのときに限り，G' の対応する 2 つの頂点 u_i, u_j について $(u_i, u_j) \in E', (u_j, u_i) \in E'$ とする．
- 各 (u_i, u_j) について，u_i, u_j に対応する G のフラグメントをそれぞれ $F_i = (V_i, E_i), F_j = (V_j, E_j)$ とする．G 上でノード $v_j \in V_j$ に隣接し，かつ $dist(v_i)$ が最小となるノードを $v_i \in V_i$ とする．このとき，(u_i, u_j) の重みを $dist(v_i) + 1$ とする．ここで，$dist(v_i)$ は F_i の根 $v_r \in V_i$ から v_i までの F_i における距離である．

フェーズ 2 G' 上で $u_0 \in V'$ を根とする MDST T^d を構成する．ここで，u_0 は G 内の根ノード v_0 を含むフラグメントが対応する頂点である．このとき，G の全域木は，T^d の辺に対応する G のリンクと，初期状況 c_0 のフラグメントのリンクで構成される．

G 上の異なる 2 つのフラグメント $F_i = (V_i, E_i), F_j = (V_j, E_j)$ 間のリンク (v_i, v_j) について，$v_i \in V_i$ が $v_j \in V_j$ を親とするとき，F_i 内で F_i の根から v_i までの経路上に存在するすべてのノードが親を変化させる．つまり，F_i の根から v_i の親までの経路上に存在するすべてのノードが親子関係を逆転させ，v_i が v_j を新たな親として格納する．よって，G' の有向辺 $(u_i, u_j) \in E'$ の重みは，$u_i \in V'$ が対応する G のフラグメント内で親を変化させるノード数と一致する．また，G' の構成法より，T^d に含まれる有向辺の重み総和は，フラグメントの結合によって全域木を構成するときに，G 内で親を変化させるノード数に一致する．従って，MDST T^d を構成すれば，全域木更新問題 3 の解が得られる．

以下では，フェーズ 2 の概略について説明する．UPD3 では，UPD2 で用いた H 法[2] に

よる MDST の構成法を利用して G' 上の T^d を求める．UPD3 では，最初，G' の各頂点を 1 つのクラスタとする．その後，異なる 2 つ以上のクラスタの合併を繰り返し，G' 上にただ 1 つのクラスタを形成する．G' 上にただ 1 つのクラスタを形成したとき，T^d が求まる．以降では，u_0 が含まれるクラスタを**根クラスタ**と呼ぶ．有向辺 $(u_i, u_j) \in E'$ について，u_i と u_j が同じクラスタに属さないならば，(u_i, u_j) を u_i が属すクラスタの**外向辺**と呼ぶ．フェーズ 2 は以下の 5 つのステップからなる．以降，フェーズ 2 のステップ 1 からステップ 4 までの動作を 1 **ラウンド**と呼ぶ．フェーズ 2 は，ステップ 1 からステップ 4 の反復で T^d を求め，ステップ 5 で T^d の辺を用いて G の全域木を構成する．

ステップ 1　根クラスタ以外の各クラスタ C_i について，C_i の重み最小の外向辺 (MOE と表す) を選択する．

ステップ 2　各 C_i のすべての外向辺の重みから C_i の MOE の重みを減じる[⋆1]．

ステップ 3　各 C_i から MOE をたどって，根クラスタに到達可能か否かを判定する．

- 根クラスタに到達可能なすべてのクラスタを根クラスタに合併する．
- 根クラスタに到達不能なクラスタが存在する場合，各クラスタが選択した MOE からなる有向閉路を検出する (根クラスタに到達不能なクラスタが存在するとき，このような有向閉路は必ず存在する)．有向閉路に含まれるすべてのクラスタを合併し，1 つの新しいクラスタとする．

ステップ 4　ステップ 3 で根クラスタに合併されたクラスタについて，全域木のリンクとなる MOE を決定する．ただし，このラウンドで根クラスタとの合併が起こらなければ，ステップ 4 の処理は行わない．すべてのクラスタが根クラスタに合併された場合はステップ 5 に進み，その他の場合は次のラウンドのステップ 1 に進む．

ステップ 5　T^d の辺を用いて G の全域木を構成する．

3.3.2　アルゴリズムの詳細

本節では，前節で説明した UPD3 の実現法を示し，そのメッセージ複雑度が $O(kn + m)$ であることを示す．ここで，$k(\leq n)$ は初期状況 c_0 に存在するフラグメントの数を表す．

⋆1 あるクラスタについて，外向辺 e を (始点の頂点を u_f とする) を MOE として選択することを考える．MDST において各頂点は 1 つの外向辺をもつため，e によって MDST を構成するとき，u_f を含むクラスタがこれまでに選択してきた MOE は MDST の辺でなくなる．従って，MOE を選択する各時点において，MOE の候補となる外向辺の重みは，その始点の頂点を含むクラスタについて，MDST の辺でなくなる MOE の重みの総和に対する増加量といえる．そのため，UPD3 では，MOE が選択されるたびに，そのクラスタ内の頂点に接続するすべての外向辺の重みから MOE の重みを減じる．

フェーズ 1 の実現

説明を簡単にするため，初期状況 c_0 で $P(v_r) = \perp$ を満たす各ノード v_r を始動ノードと仮定する[⋆2]．フェーズ 1 では，各ノードに対して，フラグメントの根からの距離に 1 を加えた値を自身に接続する外向辺の重みの初期値とさせる．そのため，各フラグメントの根ノード v_r は，フラグメント内の木のリンクを介して，フラグメント内の全ノードに根からの距離を知らせる (メッセージ Initialize)．Initialize は根からの距離を含んでおり，親から子に伝搬されるたびにその距離の値を 1 増加させる．

フェーズ 2 の実現

説明を簡単にするため，すべてのノードが前節で示したステップと後述のサブステップを同期して実行するものと仮定する．つまり，すべてのノードが，あるステップ (あるいはサブステップ) を終えてから，次のステップ (あるいはサブステップ) が始まる[⋆3]．UPD3 では，クラスタ内のノードの協調動作のため，各クラスタを以下のように管理する．

- あるノード v_r をクラスタ内のノードのリーダとして選出し，v_r の識別子とラウンド数の組をクラスタの識別子 (CID と表す) とする．フェーズ 2 の開始時には，初期状況 c_0 の各フラグメントがクラスタを形成しており，$P(v_r) = \perp$ となるノードがリーダである．よって，フェーズ 2 の開始時の CID は $(id(v_r), 1)$ である．ここで，$id(v_r)$ はノード v_r の識別子を表す．クラスタの合併で新たなクラスタが形成される場合，根ノード v_0 を含むクラスタでは v_0 をリーダとする．v_0 を含まないクラスタでは，合併された (1 ラウンド前の) クラスタのリーダの中で，識別子が最小のリーダを新たなクラスタのリーダとする．
- リーダ v_r を根とする**クラスタ全域木**(クラスタ内の全ノードから構成される木) を構成する．フェーズ 2 の開始時では，初期状況 c_0 の各フラグメントがそれぞれのクラスタ全域木である．クラスタの合併で新たなクラスタが形成されるとき，合併されたクラス

⋆2 本稿では，トポロジ変化によって，隣接ノード集合に変化の生じたノードが始動ノードであることを前提としている．そのため，c_0 の各フラグメントには始動ノードが存在し，始動ノードから親ノードをたどることで，フラグメントの根ノード $v_r(P(v_r) = \perp$ を満たす) にアルゴリズムの実行開始を要求できる．この処理に要するメッセージ数 $O(n)$ であり，UPD3 のメッセージ複雑度に影響しない．

⋆3 ネットワークの根つき全域木が利用できれば，$O(n)$ のメッセージ複雑度で，ネットワーク全体をステップごとに同期させることが可能である．UPD3 では，ステップ 1 からステップ 4 を実行するたびにクラスタ数は減少するため，UPD3 のステップとサブステップの総数はいずれも $O(k)$ である．従って，この同期に要するメッセージ数はアルゴリズム全体で $O(kn)$ となり，UPD3 のメッセージ複雑度に影響しない．一方，v_0 を根とする全域木は，$O(m)$ のメッセージ複雑度で構成できる．ただし，この全域木は補助的に利用するものであり，全域木更新問題 3 の解とは異なることに注意する．

タのクラスタ全域木と合併に用いた MOE から，新たなクラスタのクラスタ全域木を構成する．MOE による有向閉路から新たなクラスタを形成した場合は，新たなクラスタのリーダを含むクラスタの MOE を取り除いてクラスタ全域木とする．クラスタ全域木を利用すれば，クラスタ内のノード数に比例したメッセージ数で，リーダはクラスタ内の全ノードへの情報散布やクラスタ内の全ノードからの情報収集を行える．

ステップ 1 の実現

ステップ 1 では，根クラスタ以外の各クラスタで MOE を求める．ただし，前回のラウンドで合併に関わらなかったクラスタは，MOE も変化しないため，ステップ 1 を実行する必要はない．ステップ 1 は，各クラスタで以下のサブステップを行うことで実現する．以降では，同じクラスタに属す辺を**内部辺**，異なるクラスタ間の辺を**外向辺**と呼ぶ．

サブステップ 1-1 リーダは，クラスタ内の全ノードに CID を通知する (メッセージ Initiate)．

サブステップ 1-2 リーダは，クラスタ内の全ノードに MOE の探索を要求する (メッセージ Request)．MOE の探索はクラスタ全域木の葉ノードから開始し，クラスタ全域木のリンクを介して，リーダーに探索結果を伝える．MOE の探索を開始したノードは，自身の接続辺の中から重み最小の外向辺を求め，自身とその子孫の中が求めた外向辺の中で重み最小の外向辺をリーダーに伝える．各ノードは，自身の接続辺の中から重み最小の外向辺を以下のように求める．

(1) まだ内部辺と確定していない接続辺の中から重みが最小の辺 e を選び，e を介して隣接するノードに CID を問い合わせる (メッセージ Test)．
(2) Test に対する返答をもとに，e が外向辺であるか内部辺であるかを判定する．
(3) 外向辺が見つかるか，あるいはすべての接続辺が内部辺であることが確定するまで (1) から (2) を繰り返す．

ステップ 2 の実現

リーダは，クラスタ内の全ノードにステップ 1 で求めた MOE の重みを通知する (メッセージ Decide)．各ノードは，内部辺と確定していない各接続辺に対して，MOE の重みを減じた重みを新たな重みとする．また，MOE の始点ノード (以降，**幹ノード**と呼ぶ) は，MOE に対して自身のクラスタの CID をラベル付けする．ここで，あるラウンド h のステップ 3 で合併されなかったクラスタは，ラウンド $h+1$ でもラウンド h と同じ MOE を選択する．つまり，CID のラウンドが異なる複数のラベルが，同じ MOE に付けられる場合がある．MOE のラベルはステップ 4 で利用する．

ステップ 3 の実現

ステップ 3 は，次に示す動作を行うことで実現する．

(1) リーダーは，ステップ 1 で求めた MOE をたどって到達できる他のクラスタのリーダーに自身の CID を知らせる (メッセージ Circulate)．ただし，前回のラウンドで合併に関わらなかったクラスタのリーダーは，Circulate を送信しない．Circulate は，リーダーから幹ノードまで伝搬し，幹ノードは MOE を介して Circulate を転送する．他のクラスタからの Circulate を受信したノードは，リーダーまでその Circulate を転送する．リーダーは，Circulate を受信すると自身のクラスタの MOE を介してその Circulate を転送する．

(2)
- クラスタ C_i の CID を含んだ Circulate が v_0 に伝搬すれば，v_0 は C_i が MOE をたどって根クラスタに到達可能であることを検出する．このとき，v_0 は C_i を根クラスタに合併するため，C_i のリーダーに合併の事実を知らせる (メッセージ Absorb)．
- クラスタ C_i のリーダーが C_i の CID を含んだ Circulate を受信すれば，C_i のリーダーは C_i が MOE をたどって根クラスタに到達不能で，かつ有向閉路に属すことを検出する．このとき，C_i のリーダーは，その有向閉路に属す他のクラスタを検出するため，MOE をたどって到達できる他のクラスタのリーダーに再び C_i の CID を知らせる (メッセージ Detect)．C_i の Detect は，Circulate と同じ経路を伝搬し，有向閉路に属すクラスタのリーダーから同じ有向閉路に属す他のクラスタのリーダーに伝搬する．いずれ，有向閉路に属す各クラスタのリーダーは，自身のクラスタの CID を含んだ Detect を受信する．このとき，リーダーは有向閉路に属すすべてのクラスタを検出したことを知るため，自身が新しいクラスタのリーダーになるか否かを判定する (有向閉路に属すクラスタのリーダーの中で，最小の識別子をもつか否かを判定する)．新しいクラスタのリーダーは，有向閉路内のクラスタを合併した新しいクラスタを管理する．
- クラスタ C_i のリーダーがメッセージ Wait を受信すれば，C_i のリーダーは C_i が MOE をたどって根クラスタに到達不能で，かつ C_i が MOE からなる有向閉路に属さないことを検出する．Wait は，上記の新しいクラスタのリーダが，有向閉路に属さないクラスタの Circulate を受信した (あるいはリーダー選出時点ですでに受信していた) とき，これらのクラスタのリーダーに対して送信する．Wait は，新しいクラスタのリーダーから，C_i の MOE を逆方向にたどって，C_i

Vol.2010-AL-128 No.3
2010/1/26

のリーダーに伝搬する．Wait を受信した C_i のリーダは，次回のラウンド以降に，新しいクラスタのリーダとして選出されるまで，自身の CID を含む Circulate を送信しない．ただし，この間も他のクラスタの CID を含む Circulate の転送は行う．Wait を受信した C_i のリーダは，いずれ Absorb あるいは Detect を受信する．Detect を受信した場合，C_i のリーダーは，C_i が有向閉路に属すことを検出する．このとき，上記と同様に自身の CID を含む Detect を送信し，その有向閉路に属す他のクラスタの検出を行う．

ステップ 4 の実現

ステップ 4 では，根クラスタに合併されるクラスタについて，全域木のリンクとなる MOE を確定する．ただし，このラウンドのステップ 3 で根クラスタが関わる合併が起こらなかった場合は，次のラウンドのステップ 1 に進む．

はじめに，全域木のリンクに含まれる MOE の選択方法を示す．クラスタ C_f がラウンド h で根クラスタに合併されたとき，ラウンド h で選択した C_f の MOE は全域木のリンクに含まれる．しかし，ラウンド h における C_f の幹ノードについて，幹ノードの属すクラスタがラウンド 1 からラウンド $h-1$ までに選択した MOE は，全域木のリンクに含まれない．そこで，全域木のリンクに含まれない MOE のラベルを削除し，ラベルの付いている MOE のみを全域木のリンクとする．削除するラベルの選択方法を以下に示す．

(1) ラウンド h で根クラスタに合併されるクラスタ C_f の幹ノードを v_f とする．

(2) v_f が属すクラスタについて，ラウンド 1 からラウンド $h-1$ までに選択した MOE のラベルを削除する．

(3) ラウンド $h'(1 < h' < h)$ で v_f が属すクラスタと合併した各クラスタ C_i を考える．再帰的に，h' の値を h，C_i の幹ノードを v_f として (2) から (3) を繰り返す．

この処理の終了後に，ラベルが残っている MOE が全域木のリンクに含まれる．図 2 に，ステップ 4 の処理を行うクラスタに対し，全域木のリンクに含まれるか否かで MOE を区別した例を示す．ここでは，リーダの識別子が x のクラスタをクラスタ x と呼ぶ．図 2 において，クラスタ 1 からクラスタ 6 までのクラスタを合併したクラスタが根クラスタに合併される．まず，根クラスタと合併するラウンド 4 において，幹ノードが存在するクラスタ 6 がラウンド 3 までに選択した MOE のラベル $(6,1),(5,2..3)$ を削除する．次に，ラウンド 3 において，クラスタ 6 が含まれるクラスタと合併するクラスタ (クラスタ 1 からクラスタ 4 で形成) について，幹ノードが属するクラスタ 2 に着目する．そして，クラスタ 2 がラウンド 1 と 2 で選択した MOE のラベル $(2,1),(1,2)$ を削除する．ラウンド 2 において，クラスタ

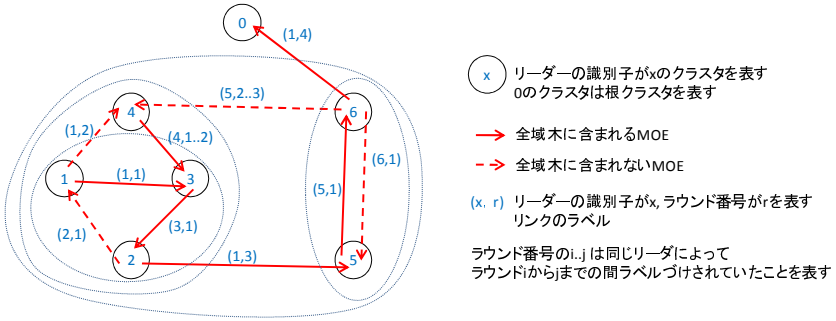

図 2 全域木のリンクとなる MOE を決定したクラスタ

2 が含まれるクラスタと合併するクラスタ 4 に着目する．そして，クラスタ 4 がラウンド 1 で選択した MOE のラベル $(4,1)$ を削除する．ここで，MOE のラベル $(4,2)$ は削除されないことに注意する．以上の処理により，全域木のリンクとなる MOE にのみラベルが残る．

ステップ 4 は以下の 3 つのサブステップを行うことで実現する．ステップ 4 では，サブステップ 4-1 を実行後，全域木のリンクとなる MOE が確定するまでサブステップ 4-2 を反復する．そして，サブステップ 4-2 の反復終了後，サブステップ 4-3 を実行する．以下では，ラウンド h でステップ 4 を行うことを考える．また，ラウンド x のラベル (CID にはラウンド番号が含まれている) が付けられた MOE の始点ノードを，ラウンド x の幹ノードと呼ぶ．ただし，以下のサブステップの実行中に，ラウンド x のラベルが削除された MOE の始点ノードは，ラウンド x の幹ノードと呼ばない．

サブステップ 4-1 v_0 は，ラウンド h の幹ノードに対して，ラウンド 1 からラウンド $h-1$ までの CID を要求する (メッセージ Inquire)．Inquire を受信したラウンド h の各幹ノードは，ラウンド 1 からラウンド $h-1$ までの自身の CID を v_0 に報告する．サブステップ 4-2 の説明のため，反復に用いる h' の初期値を $h-1$ とする．

サブステップ 4-2 v_0 は，ラウンド h' の各幹ノードに対して，ラウンド 1 からラウンド $h'-1$ までの CID を要求する．さらに，すべての幹ノード (ラウンド h' の幹ノードに限らない) に対して，v_0 が前回のサブステップで得た CID($S(I_{h'+1})$ と表す) をラベルとする MOE の削除を要求する (メッセージ Delete)．そのため，Delete は $S(I_{h'+1})$ を含んでいる．Delete を受信した各幹ノードは，自身に接続する MOE の中で，$S(I_{h'+1})$ に含まれるラベルの MOE を削除する．さらに，ラウンド h' の各幹ノードは，ラウンド 1 からラウンド $h'-1$ までの自身の CID を v_0 に報告する．

以降，ラウンド $h'-1$ をラウンド h' とみなし，サブステップ 4-2 を反復する．ここで，$h'-1$ が 0 となるとき，反復を終了し，サブステップ 4-3 に進む．

サブステップ 4-3 各幹ノードは，自身に接続する MOE の中でラベルが残った MOE が存在すれば，その MOE を全域木のリンクとして確定する．そして，その MOE が全域木のリンクとして確定することを MOE の終点ノードに知らせる (メッセージ Branch)．

ステップ 5 の実現

v_0 は，クラスタ内のすべてのノードに対して，解の全域木における親と隣接ノード集合を格納するよう要求する (メッセージ Construct)．Construct は，フラグメントを形成するリンクと全域木のリンクとして確定した MOE のみを介して送信される．従って，各ノードは，全域木の親ノードから Construct を受信し，各子ノードに Construct を転送する．このとき，各ノードは，全域木における親と隣接ノード集合を格納する．Construct を受信した全域木の葉ノードは，局所アルゴリズムを停止する．局所アルゴリズムの停止は，葉ノードから根ノード v_0 に向かって進む．以上により，解の全域木が求まり，その後すべてのノードの局所アルゴリズムが停止する．

アルゴリズムの性能

以降では，初期状況 c_0 で $k(\leq n)$ 個のフラグメントが存在するものとして議論を進める．クラスタ全域木を介して送信されるメッセージは，Initialize，Initiate，Request，Decide，Inquire，Construct であり，それぞれクラスタ内のノード数に比例した数のメッセージが送信される．また，アルゴリズム全体のラウンド数は $O(k)$ である．Initialize，Construct による複雑度は $O(n)$ である．一方，Initiate，Request，Decide，Inquire は，各ラウンドで送信されるため，これらのメッセージによる複雑度は $O(kn)$ である．次に，メッセージ Circulate，Detect，Wait のメッセージ複雑度を示す．最初のラウンドでは，根クラスタ以外の各クラスタのリーダーが Circulate の送信を開始する．2 回目以降の各ラウンドでは，前回のラウンドでの有向閉路内のクラスタの合併によって形成された新しいクラスタのリーダーのみが送信を開始する．ここで，アルゴリズム全体で合併に関わるクラスタ数は高々 $2(k-1)$ である[⋆1]．よって，アルゴリズム全体で Circulate を送信するリーダーの数は高々 $2(k-1)$ である．同様に，メッセージ Detect は，有向閉路内のクラスタの合併が行われるとき，合併に関わる各クラスタのリーダーが送信する．よって，アルゴリズム全体で Detect を送信するリーダーの数は高々$2(k-1)$ である．メッセージ Wait は，有向閉路内のクラスタの合併後の新しいクラスタのリーダーが，その合併に関わらなかったクラスタのリーダーのうち，Circulate を受信した各クラスタのリーダーに対して送信する．つまり，Wait を受信するリーダーの数は，Circulate を送信するリーダーの数を超えないため，高々$2(k-1)$ である．また，各ラウンドで，Circulate，Detect，Wait はネットワーク全体で高々n 回送信される．よって，Circulate，Detect，Wait による複雑度は $O(kn)$ である．メッセージ Delete は，各反復において v_0 を根とするクラスタ全域木を介して送信される．また，反復回数はラウンド数に依存するため $O(k)$ である．よって，Delete による複雑度は $O(kn)$ である．メッセージ Branch は，全域木のリンクとなる各 MOE を介して 1 回送信されるため，アルゴリズム全体で $k-1$ 回送信される．メッセージ Test は，各ノードが自身の親と子を除く隣接ノードに高々1 回送信するため，アルゴリズム全体で高々$2m$ 回送信される．以上より，UPD3 について以下の定理が導かれる．

定理 3 UPD3 はメッセージ複雑度 $O(kn+m)$ で，全域木更新問題 3 を解く．ここで，$k(\leq n)$ は初期状況 c_0 で存在するフラグメントの数である．

4. ま と め

本稿では，出力の変化数の最小化を目標とする 3 つの全域木更新問題を定式化した．各問題で対象とする出力は，木上の隣接ノード集合 (問題 1)，親ノード (問題 2)，木上の隣接ノード集合と親ノード (問題 3) である．さらに，これらの問題を解く分散アルゴリズムを提案した．各問題を解く分散アルゴリズムのメッセージ複雑度は，それぞれ $O(n\log k+m)$，$O(n^2)$，$O(kn+m)$ である．ここで，k は初期状況 c_0 で存在するフラグメントの数である．

参 考 文 献

1) Gallager, R. G., Humblet, P. A. and Spira, P. M.: A distributed algorithm for minimum-weight spanning trees, *ACM TTOPLAS 5(1)*, pp.66–77 (1983).
2) Humblet, P. A.: A distributed algorithm for minimum weighted directed spaning trees, *IEEE Transactions on Communications*, Vol.COM-31, No.6, pp.756–762 (1983).
3) Yamauchi, Y., Ooshita, F., Kakugawa, H. and Masuzawa, T.: Output stability of self-stabilizing protocols against topology changes and transient faults, *International Conference on Application and Principles of Information Science (APIS)*, pp.306–310 (2009).
4) 朴 政鎬，増澤利光，萩原兼一，都倉信樹：重み最小生成木更新問題を解く分散アルゴリズム，電子情報通信学会論文誌 (D-I)， Vol.J73-D-I, No.10, pp.802–810 (1990).

⋆1 1 つの合併に関わるクラスタ数を $x_i \leq k$ とする．合併によって減少するクラスタ数は $k-1$ を超えないため，$\sum(x_i-1) \leq k-1$ が成り立つ．さらに，クラスタの合併は高々$k-1$ 回行われるため，$\sum(x_i-1) \geq \sum x_i-(k-1)$ が成り立つ．以上より，アルゴリズム全体で合併に関わるクラスタ数について $\sum x_i \leq 2(k-1)$ が成り立つ．