st-ordering問題を利用した極大DAG構成自己安定アルゴリズムに関する研究

大野 陽香[1]　片山 喜章[1]　増澤 利光[2]

1. はじめに

自己安定アルゴリズムとは, 任意の初期ネットワーク状況から実行を開始しても, 目的のシステム状況に到達することができる分散アルゴリズムである [1]. この性質から, 自己安定アルゴリズムとは, プロセスの一時的な故障によって分散システムがどのようなネットワーク状況に陥っても, 故障したプロセスが復旧すれば自動的に目的の状況 (正当な状況) に戻る. つまり, 自己安定アルゴリズムは一時故障に対して耐性のある分散アルゴリズムであり, 長期に渡ってネットワーク状況を安定に保ち, 一時故障に柔軟に対応することの求められる分散システムを動作させる場合に適している. 自己安定アルゴリズムは, どのような状況からでも有限時間内に安定するため, アルゴリズムを階層化することで, 複数のアルゴリズムを合成することができる. この手法を公平な合成という. 下位のアルゴリズムが安定すると, それを初期状況として上位のアルゴリズムが動作するため, 上位のアルゴリズムもやがて安定し, 目的の状況へと到達することができる. 公平な合成を用いることで, 任意のネットワークで問題を解くことも可能となる.

DAG(Directed Acyclic Graph) とは, 閉路のない有向グラフであり, 内向辺しか持たないプロセスをシンク (sink), 外向辺しか持たないプロセスをソース (source) と呼ぶ. DAG は分散アルゴリズムを動作させる初期グラフとしてよく利用される. また, 単一プロセス s,t が指定された 2 連結無向グラフ G において, s をソース, t をシンクとし, G 上の全ての辺が方向付けられているような DAG を Transport net といい, Transport net を構築する問題を Transport net 問題と呼ぶ.

st-ordering(st-numbering) 問題とは, 単一プロセス s,t が指定されたプロセス数 n の 2 連結無向グラフ G において, s に 1, t に n, s,t を除く各プロセスに 2 以上 $n-1$ 以下の整数 (st-order, st-number) を割り当てる問題である. ただし, s,t を除くプロセスには, 自分よりも大きい st-order を持つプロセスと, 自分よりも小さい st-order を持つプロセスの両方が隣接するように st-order を割り当てなければならない. st-ordering を行ったグラフ G の全ての辺に対して, 小さい st-order を持つプロセスから大きい st-order を持つノードへ向かうように方向づけを行うことで, Transport net を形成することができる. つまり, st-ordering 問題を解くことで Transport net 問題も解決することができる.

Transport net 問題, st-ordering 問題を解く自己安定アルゴリズムに関する既存研究について説明する.

Karaata ら [2] は, 二つの幅優先木を利用して Transport net を構築する自己安定アルゴリズムを提案した. Chaudhuri ら [3] は, 深さ優先木を利用して, st-ordering を行う自己安定アルゴリズムを提案した. 文献 [2], [3] で提案されているアルゴリズムは, いずれも相異なる識別子を持つプロセスで構成される連結度 2 以上の無向グラフでしか動作しない. また, 連結無向グラフの連結度が 1 である場合に, できるだけ多くの辺を方向づけるような DAG を構成するアルゴリズムは提案されていない. 本稿では, 連結度が 1 である無向連結グラフ上でも, できるだけ多くの辺を方向づけるような極大 DAG を構成するアルゴリズムを提案する. また, 文献 [2], [3] では, グラフ上の全てのプロセスが相異なる識別子を持っているが, 本稿では単一プロセス s,t と, 匿名プロセスで構成されるグラフを考える.

[1] 名古屋工業大学大学院工学研究科情報工学専攻
Graduate School of Computer Science and Engineering, Nagoya Institute of Technology
[2] 大阪大学大学院情報科学研究科コンピュータサイエンス専攻
Graduate School of Information Science and Technology, Osaka University

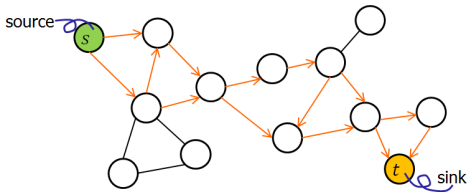

図 1　極大 DAG の例

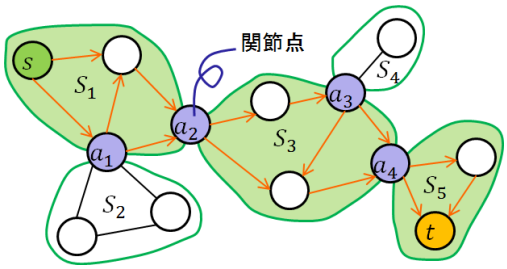

図 2　関節点で分解される連結成分

2.　極大 DAG 構成問題の定義

極大 DAG を次のように定義する.

定義 2.1.（極大 DAG） 単一プロセス s,t を持つ無向連結グラフ $G=(V,E)$ において, 次の条件を満たすように構成された DAG を極大 DAG という.

(1) s をソース, t をシンクとする

(2) s,t を除くプロセス $P_i \in V\backslash\{s,t\}$ は, シンクにもソースにもならない

(3) できるだけ方向付けられる辺を方向付ける

□

定義 2.1 のような DAG を構成する問題を, 極大 DAG 構成問題と定義する.

図 1 では, これよりも多く辺に方向づけを行うと, s,t 以外のプロセスがシンクやソースとなったり, 有向閉路が形成されたりするため, 極大 DAG の定義を満たさない. これ以上辺に方向付けることはできないため, "極大"であるといえる. また, 連結度 2 以上の無向グラフ上で極大 DAG を構成すると, 構成された極大 DAG は Transport net となる.

3.　提案アルゴリズム

本稿の提案アルゴリズムは関節点を利用する. 関節点とは, グラフ上から取り除くとそのグラフが非連結となるようなプロセスである. 図 2 より, 極大 DAG 上の方向を付けない辺は, 関節点に接続していることがわかる. 関節点は, 連結グラフを極大で連結な部分グラフである連結成分に分解することができる. 図 2 では, 関節点 $a_1,a_2,\cdots,a_4$ により連結成分 $S_1,S_2,\ldots,S_5$ に分解できる. 分割された連結成分は, 辺が全て有向辺で構成される連結成分と, 全て無向辺で構成される連結成分に分けられる. 辺が全て有向辺である各連結成分では, Transport net が構成されており, そのような連結成分内のプロセスと有向辺は全て, s から t への単純経路上にある. 図 2 では, 連結成分 S_1,S_3,S_5 でそれぞれ Transport net が構成されている. また, Transport net を構成している連結成分を挟むような関節点は, それぞれの Transport net のシンク, ソースとなっている. 図 2 では, 関節点 a_1 が S_1 のシンク, また S_3 のソースとなっており, a_4 は S_3 のシンク, また S_5 のソースとなっている. 一方, Transport net を構成する連結成分を挟まないような関節点 a_1,a_3 はそれぞれの Transport net 内のシンクやソースではない.

このことから, 関節点でグラフを連結成分に分解し, s から t への単純経路を含む各連結成分上で Transport net を構築することで極大 DAG を構成することができる. ただし, 極大 DAG を構成するには, Transport net を構築する連結成分に属するプロセスと構築しない連結成分に属するプロセス, つまり, 辺を方向付けるプロセスと, そうでないプロセスに分類する必要がある. そのため, 極大 DAG 上のプロセスを次の 3 種類に分類する.

st-node：　それぞれの連結成分上に構成される Transport net のソース, またはシンクとなるプロセス

normal：　Transport net を構成する連結成分上における st-node 以外のプロセス

NULL：　Transport net が構成されない連結成分上に存在するプロセス

この分類に基づき, 分割した連結成分上で Transport net 構築を行う. st-node, normal に分類されたプロセスは, 極大 DAG において有向辺が接続するプロセスであり, また NULL に分類されたプロセスに接続する辺はすべて無向辺である.

Transport net を構築するために, st-ordering を行う. st-ordering を実行し, 割り当てられた st-order の大小によって辺を方向づけることによって各連結成分上に Transport net が構成される.

3.1　提案アルゴリズムの概略

本稿の提案アルゴリズムの概略を次に示す.

(1) 深さ優先木による関節点を求めるアルゴリズム [4] を用いて, グラフの関節点を求める

(2) 深さ優先木を利用して, s から t への経路を探索する

(3) Transport net を構築する連結成分上のプロセスと, Transport net を構築しない連結成分上のプロセスを分類する

(4) プロセスの分類に基づき, st-ordering 問題を解くアルゴリズム [3] を用いて, Transport net を構築するそれぞれの連結成分上で st-ordering を実行する

(5) st-ordering を行った連結成分上のプロセスの st-order の大小によって辺を方向づけることにより, 各連結成分で Transport net を構築し, 極大 DAG を構成する

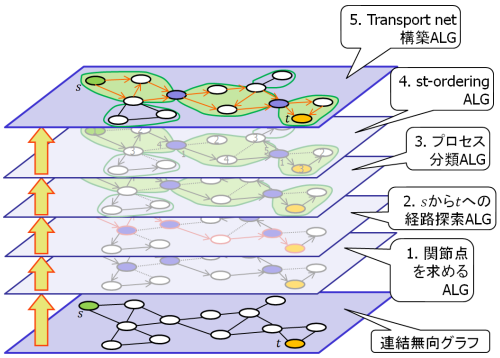

図 3　アルゴリズムの合成

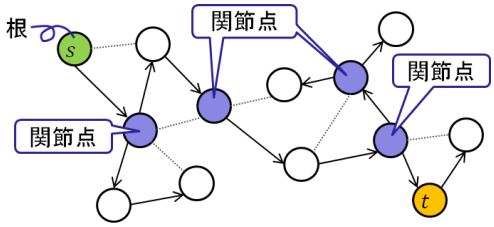

図 4　1. グラフの関節点を求める

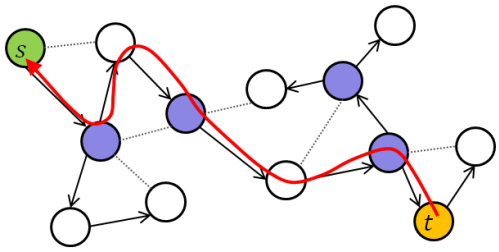

図 5　2. s から t への経路探索

これらを実現するために, 公平な合成を利用し, それぞれのアルゴリズムを合成する（図 3）.

それぞれのアルゴリズムについて, 説明する.

1. グラフの関節点を求める

[4] の提案アルゴリズムを用い, 深さ優先木を利用してグラフの関節点を求める. このとき, s を深さ優先木の根とする（図 4）.

2. s から t への単純経路の探索

先ほどの深さ優先木の木辺を利用して, t から根 s に向かって探索する（図 5）. これにより, Transport net を構築する連結成分と, Transport net を構築しない連結成分を区別することができる.

3. プロセスの分類

各プロセスを st-node, normal, NULL に分類する. プロセスを分類するために, 深さ優先木と関節点を用いて, 次のようなプロセスを定義する.

P_c : 深さ優先木において, ある関節点を取り除くと, s を含むグラフと分離してしまうような部分木の根プロセス

深さ優先木における P_c の親は, 必ず関節点である（図

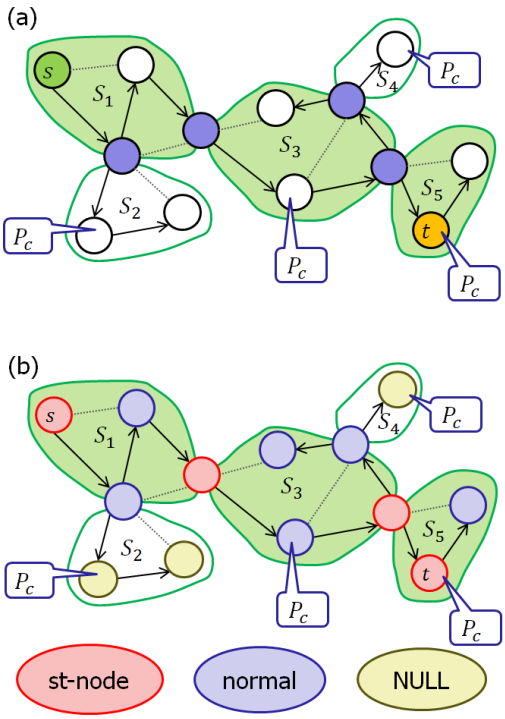

図 6　3. プロセスの分類

6（a））.

P_c は親である関節点を経由しないと, 根 s に到達できないプロセスであるため, P_c が s から t への経路上に存在しないならば, P_c の子孫には t が存在しない. つまり, P_c を根とする部分木上に Transport net が構築されることはなく, P_c とその子孫は NULL に分類される. 一方, P_c が s から t への経路上に存在するならば, P_c の親である関節点は s から t への任意の経路に必ず含まれる. よって, P_c は normal に分類され, P_c の親の関節点は st-node に分類される. P_c 以外のプロセスと st-node と判断されなかった関節点は, 親のプロセスの分類によって normal か, NULL かを判断する. 親が NULL であれば自分も NULL であり, そうでなかったら normal に分類される. また s, t は st-node に分類される（図 6（b））.

4. st-ordering

プロセスの分類に基づき, st-ordering を実行するアルゴリズム [3] を用いて, それぞれの連結成分で st-ordering を行う. このとき, NULL と分類されたプロセスは st-order を割り当てられない. st-node である関節点は, ソース s' としての変数と, シンク t' としての変数の両方を用意する. st-node である関節点に隣接する s に近い側のプロセスは関節点を t', t に近い側のプロセスは関節点を s' として扱う（図 7）.

5. Transport net を構築する各連結成分上で割り当てた st-order に基づき, Transport net をそれぞれ構築する. 小さい st-order を持つプロセスから, 大きい st-order

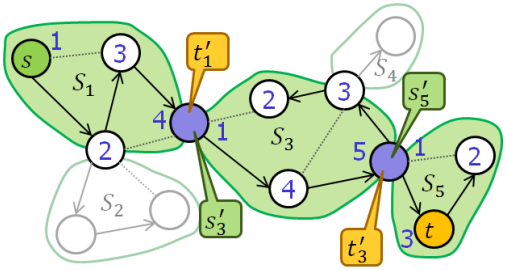

図 **7** 4. st-ordering

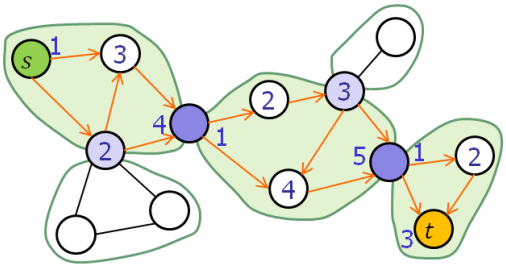

図 **8** 5. Transport net 構築

を持つプロセスへ向かうような方向付けを行うことで, Transport net が構築され, 極大 DAG が構成される（図 8）.

4. まとめと今後の課題

本稿では, 指定された単一プロセス s, t を持つ連結無向グラフ上に極大 DAG を構成するアルゴリズムを提案した.

今後の課題として, 正当性の証明と, アルゴリズム改良による時間計算量の削減が考えられる.

参考文献

[1] E. W. Dijkstra: *Self-stabilizing systems in spite of distributed control*, Comm. ACM 17 (11) pp.103-117 (1974).

[2] M. H. Karaata, P. Chaudhuri: *A Dynamic Self-Stabilizing Algorithm for Constructing a Transport Net*, Computing 68, pp. 143-161 (2002).

[3] Pranay Chaudhuri, Hussein Thompson: *A self-stabilizing algorithm for st-order problem*, The International Journal of Prallel, Emergent and Distributed Systems Vol. 23 (3), pp. 219-235 (2008).

[4] 大野陽香, 片山喜章: 深さ優先探索木によるグラフの関節点を求めるメッセージサイズ $\mathbf{O}(log_2 n)$ の自己安定アルゴリズムについて, 平成 26 年度東海支部連合大会 (2014).