2003－AL－91　(7)
2003／9／19

スタイナー木問題を解く自己安定分散アルゴリズム

亀井 清華 *　　角川 裕次 *

概要

自己安定とは、耐故障性のある分散アルゴリズムの理論的な枠組みの１つである。

本研究では、分散システム上におけるスタイナー木問題を考え、既存の発見的分散アルゴリズムの１つである Chen らのアルゴリズム [2] に基づいて問題を解く自己安定分散アルゴリズムを提案する。分散システム上においてのスタイナー木問題とは、マルチキャストのルーティング問題であると言える。本アルゴリズムは、(1) 最短経路森を構成し、(2) ネットワークを変換し、(3) 最小重み生成木を構成、(4) 必要でない枝や節点を刈り取る、という 4 つの階層的な構造になっており、本研究では (1)、(2)、(4) の 3 つの自己安定分散アルゴリズムを提案する。

A Self-Stabilizing Algorithm for the Steiner Tree Problem

Sayaka Kamei *　　Hirotsugu Kakugawa*

Abstract

In this paper, we investigate the Steiner tree problem in distributed systems, and propose a self-stabilizing heuristic solution.

Our algorithm uses fair composition technique. The algorithm is constructed by four layered modules (sub-algorithms): construction of a shortest path forest, transformation of the network, construction of a minimum spanning tree, and pruning unnecessary links and processes. The third algorithm, for construction of a minimum spanning tree, has been proposed so far. Therefore, in this paper, we propose three algorithms for other layers. Competitiveness of this algorithm is $2(1-1/l)$, where l is the number of leaves of optimal solution.

*広島大学大学院工学研究科
*Department of Information Engineering, Hiroshima University, Japan

1 はじめに

1.1 分散システムとその故障

分散システムとは、プロセスの集合とそれらの間の通信リンクの集合である。近年のネットワークシステムの急成長と需要の増加により、分散アルゴリズムの設計が重要になってきた。

分散システムは、その性質により不安定なものである。それはそのサイズが大きくなるほど顕著になるといえる。その不安定さは、結果として故障となって現われる。したがって、耐故障性のある分散システムを設計することは分散システムを考える上で重要なことである。

1.2 自己安定

自己安定アルゴリズムは 1974 年に Dijkstra [3] によって提案された、耐故障性のある分散アルゴリズムの理論的な枠組みの 1 つである。

一般的には、分散アルゴリズムは実行開始時に決められた初期状態にある必要がある。その初期状態ではネットワークを伝搬中のメッセージは存在しないと仮定する。しかし、これら初期状態を考慮しない、つまりアルゴリズム実行開始時の各プロセスの状態や伝搬中のメッセージの有無やその内容などに関わらず問題を解くことのできる分散アルゴリズムを自己安定アルゴリズムという。自己安定アルゴリズムは、あらゆる初期化を必要としない。そして、あらゆる種類の一時故障が何度起ころうとも、それに対する耐性を有する。また、その最も大きな性質として、アルゴリズム実行中にネットワークトポロジーが変化するような動的ネットワークにおいても、十分に長い間トポロジーが変化しなければ問題を解決することができるということも挙げられる。

1.3 スタイナー木問題

V を節点集合、E を枝の集合、$w : E \to \mathbb{R}$ をコスト関数とする、グラフ $G = (V, E, w)$ を考える。多くの応用において G の最小重み生成木を用いて、例えば、ブロードキャストの経路探索などが行なわれている。G の生成木 G_T においては、G のすべての節点を網羅する。しかし、いくつかのアプリケーションにおいてはすべての節点を網羅する必要の無いものもある。ここで、$Z \subseteq V$ をいくつかの節点の集合とする。G の部分グラフで、Z のメンバーを網羅した木であり、かつ重みの最小のものを Z のスタイナー木 G_z と呼ぶ。$|Z| = 2$ のときは G_z は Z のメンバーの間の最短経路であり、$|Z| = n$ (n; G の節点の数) のときは G_z は G の最小重み生成木である。

一般的には、最小コストのスタイナー木を求める問題は NP 完全問題の 1 つとして知られている [7]。したがって、実際には代わりに発見的な近似アルゴリズムを考えるのが適当であり、これまでにもたくさんの逐次的発見的アルゴリズムが提案されてきた。その多くのアルゴリズムにおいて、発見的な操作として、まず最小重み生成木を求め、それから必要でない節点や枝を刈り取っていくということがなされる。このような操作を Pruned-MST という [11]。この Pruned-MST のみを行なったときの解は最適解である保証がないが、とても単純な手法である。このコンペティティブネスは $n - |Z| + 1$ である。このコンペティティブネスというのは、C_{alg} をその近似アルゴリズムの解の最悪の場合の重みとし、C_{opt} を最適解の重みとしたときの C_{alg}/C_{opt} の割合をいう。Pruned-MST のコンペティティブネスは決してよくはないが、これは他の発見的アルゴリズムの中に 1 操作として組みこまれて使われる。そういったものの例として、Distance Network Heuristic が Kou らによって提案され [8]、Wu らによって改良された [12]。このコンペティティブネスは、l を G と Z のスタイナー木の最適解の葉の数としたとき、$2(1 - 1/l) < 2$ である。

たくさんの逐次的な発見的アルゴリズムが提案され、現在では [9] など、コンペティティブネスが 1.55 のものもある。このスタイナー木問題のサーベイ論文として、[11] と [5] を挙げておく。

1.4 分散システム上におけるスタイナー木問題

分散システム上では、最適なマルチキャストの経路探索問題はスタイナー木問題によってモデル化される。Z のスタイナー木は Z を 1 つの送信元と複数の受信先の集合とみなしたとき、最小重みのマルチキャストの経路である。しかし、スタイナー木を求める効率のよいアルゴリズムがまだ知られておらず、分散システム上では最小重み生成木を構成する方が効率がよい。この理由により、最小重み生成木がスタイナー木の実用的な近似解として用いられている。たとえばインターネットでのマルチキャストのプロトコルである DVMRP もその 1 つである [10]。多くの発見的逐次アルゴリズムは提案されてはいるが、そ

れらを分散アルゴリズムとして実装することは大変難しい。

文献 [2] において、Chen らは Wu らのアルゴリズムに基づいた分散アルゴリズムを提案している。このアルゴリズムは Wu らのものとその方針など本質的には変わらないので、コンペティティブネスは $2(1-1/l)$ である。このアルゴリズムは非同期分散システムを仮定しており、故障は認めていない。

分散システム上では耐故障性を考えることは最も重要な課題の一つであり、また、インターネットなどの動的なシステムにおけるマルチキャストなどの応用を考えると、Z のメンバーのようなシステムの動的な変化に適応できることも必要なことであるといえる。しかし、スタイナー木問題に対する自己安定分散アルゴリズムは存在していなかった。よって、本研究では、スタイナー木を求める自己安定分散アルゴリズムを提案する。

自己安定の性質により、本アルゴリズムは Z のメンバーが動的に変化するような動的スタイナー木問題にも適用できるアルゴリズムであるといえる。また、スタイナー木は特定のプロセス (Z のメンバー) 間での通信路といえ、自己安定アルゴリズムはその性質により、その通信路になんらかの故障があった場合でも、そのプロセス間の通信路を自動的に確保することができるといえる。

本論文ではまずシステムモデルについての定義をし、Chen らの分散アルゴリズム [2] に基づいてスタイナー木問題を解く自己安定分散アルゴリズムを提案する。そして最後にその解のコストをより小さくするための簡単な自己安定アルゴリズムを提案する。

2 定義

$V=(P_1,P_2,...,P_n)$ をプロセスの集合とし、$E \subseteq V \times V$ を双方向通信リンクの集合とする。そして、それぞれのリンクに対して重み (コスト) を定義する。この重みは、E の非負の数の集合への写像であるコスト関数 w によって計算される。分散システムのトポロジーは重み付き無向グラフ $G=(V,E,w)$ によって表される。プロセスの数は n で表す。そして、グラフは連結で単純なものであると仮定する。本研究では「グラフ」と「分散システム」を同様の意味で用いる。また、「節点」と「プロセス」、「枝」と「リンク」も同様に考える。

N_i をプロセス P_i の隣接プロセスの集合とする。すべての $P_i,P_j \in V(P_i \neq P_j)$ について $id(P_i) \neq id(P_j)$ が成り立つ場合に限り、プロセス識別子は一意 (unique) である、といい、それぞれのプロセスは一意なプロセス識別子を持つとする。id をプロセスの名前付け関数とする。$id(P_i)$ をそれぞれのプロセス P_i の識別子とする。プロセス識別子については、明らかな場合には $id(P_i)$ で P_i を表す。

局所変数の集合はプロセスの局所状態とする。q_i によってプロセス $P_i \in V$ の局所状態を表す。それぞれのプロセスの局所状態の組 $(q_1,q_2,...,q_n)$ が、分散システムの状況 (configuration) を表す。Γ をすべての状況の集合とする。

通信モデルはそれぞれのプロセスは隣接プロセスの局所状態を遅延なく読んでこれるものとする。このモデルは状態通信モデルと呼ばれる。プロセスは隣接プロセスの局所状態を読んでくることはできるが、それらを書換えることは出来ない。プロセスはそれ自身の局所状態のみを書換えることが出来る。

それぞれのプロセス P_i のアルゴリズムはガード・コマンドの集合で与えられる。

$$*[\, g_1 \rightarrow c_1; g_2 \rightarrow c_2; g_3 \rightarrow c_3; \cdots \,]$$

各 g_j $(j=1,2,...)$ はガードと呼ばれ、隣接プロセスの局所状態と自身の局所状態に関する前提条件である。各 c_j はコマンドと呼ばれ、自身の局所状態を書換える。次の局所状態は現在の自身の局所状態と隣接プロセスの局所状態から計算される。各プロセスには、自身の識別子と隣接プロセスの集合 N_i がパラメータとして与えられる。これらの値はガード・コマンドにおいて用いられる。

状況 γ において P_i について少なくとも 1 つのガードが真である場合に限り、P_i が特権を持つ。

各プロセス P_i の一原子動作は (1) 隣接プロセスの局所状態を読み、ガードを評価し、(2) 真となったガードに対するコマンドを実行し、(3) 自身の局所状態を書換える、という 3 つの動作から成り立つ。

動作するプロセスを決定するスケジューラには C デーモンと呼ばれるものを仮定する。これは自己安定アルゴリズムにおいてはしばしば用いられるものである。各ステップにおいて、C デーモンは特権を持つプロセスの中から一度に任意の 1 つのプロセスを選択し、選ばれたプロセスのみが一原子動作を実行する。また、スケジューラは必ずしもシステムが安

定するようにプロセスを選ばないという意味で不公平であると仮定する。アルゴリズムは仮定されたスケジューラによって可能である実行スケジューリングに関係なく正しくなければならない。つまり、スケジューラはアルゴリズムに対する敵 (adversary) であるといえる。

いくつかの状況 γ について、γ' を γ に続く状況とする。このとき、この遷移関係を $\gamma \to \gamma'$ で表す。状況 γ_0 について、γ_0 から始まった計算 X は、各 $t \geq 0$ について $\gamma_t \to \gamma_{t+1}$ であるとしたとき、出来る限り無限な状況の列 $X = \gamma_0, \gamma_1, \gamma_2, \ldots$ である。

定義 1 Γ をすべての状況の集合とする。アルゴリズム A が次の *2* つの条件を満たす場合に限り、A は $\Lambda \subseteq \Gamma$ について**自己安定** (self-stabilization) であるといえる。

- 収束性：任意の状況から始まって、状況はいつかは Λ のうちの 1 つとなる。
- 閉包性：状況 $\lambda \in \Lambda$ について、λ に続くあらゆる状況 γ もまた Λ の状況である。

各 $\lambda \in \Lambda$ は解状況 *(legitimate configuration)* と呼ばれる。そして Λ は解状況の集合と呼ぶ。 □

一般に、スタイナー木は次の様に定義される。

定義 2 V を節点の集合、E を枝の集合、$w : E \to \mathbb{R}$ を枝のコスト関数とし、$G = (V, E, w)$ を連結な重み付き無向グラフとする。空集合でないあらゆる部分集合 $Z \subseteq V$ について、$G_z = (V_z, E_z, w)$ が Z と G に対する**スタイナー木**であるということは、G_z は連結で $Z \subseteq V_z$ である G の最小重みの部分グラフの木である。 □

本研究では分散システム上でスタイナー木を求める問題を解くことを考える。各プロセスはネットワーク全体の情報は知らないものとし、局所的な情報のみを知るものとする。

定義 3 分散スタイナー木問題を次のように定義する。

- 入力：各プロセスは自身に接続されているリンクの重みと自身が Z のメンバーであるかどうかの情報が与えられる。
- 出力：各プロセスは自身がスタイナー木の節点になるかどうかと各枝についてスタイナー木の枝とするかどうかを決定する。 □

3 Chen らの分散アルゴリズム [2]

ここで Chen らのアルゴリズムについて説明する。このアルゴリズムはスタイナー木の近似解の一つである、一般化最小重み生成木を求めるというのがその方針である。

定義 4 $G = (V, E, w)$ と $Z \subseteq V$ について、$E_c = \{(P_i, P_j) \mid P_i, P_j \in Z, P_i \neq P_j\}$ で、各 $P_i, P_j \in Z$ $(P_i \neq P_j)$ について $w_c(P_i, P_j)$ が G の $dist(P_i, P_j)$ に等しいような重みつき完全グラフを $G_c = (Z, E_c, w_c)$ とする。また、G_c の最小重み生成木を $G_s = (Z, E_s, w_c)$ とする。このとき、次の条件を満たすような G の部分グラフ $G_z = (V_z, E_z, w)$ を一般化最小重み生成木という。

- 節点 p が V_z の要素であることと、p が G において節点 u と節点 v の間の最短経路における節点であるような枝 $(u, v) \in E_s$ が存在することとは必要十分である。
- 枝 e が E_z の要素であることと、e が G において節点 u と節点 v の間の最短経路における枝であるような枝 $(u, v) \in E_s$ が存在することとは必要十分である。
- G_z の全ての葉が Z のメンバーである。 □

このアルゴリズムの実行例を図 1 に示す。黒い節点が Z のメンバーで、各枝のラベルはその枝の重みを表している。(a) は元のネットワーク $G = (V, E)$ である。

このアルゴリズムはまず、次のように定義された最短経路森を構成する (b)。

定義 5 V を節点の集合、E を枝の集合、$w : E \to \mathbb{R}$ を枝のコスト関数とし、$G = (V, E, w)$ を連結な重み付き無向グラフとする。節点の部分集合 $Z \subseteq V$ を与えたとき、G における Z に対する**最短経路森** (SPF) を次の条件を満たすような互いに素な木々 $T_i = (V_i, E_i, w)$ $(i = 1, 2, \ldots, |Z|)$ から成る部分グラフ $G_F = (V, E_F, w)$ とする。

- すべての $i \neq j$ に対して、$\bigcup_{i=1}^{|Z|} V_i = V$ かつ $V_i \cap V_j = \emptyset$ である。
- $\bigcup_{i=1}^{|Z|} E_i = E_F \subseteq E$
- すべての $1 \leq i \leq |Z|$ について、V_i は Z のちょうど 1 つの節点 P_z^i を含み、各プロセス $P_z^i \in Z$ は T_i に含まれる。
- すべての V_i について、P_z^i を V_i における Z のメンバーとする。$P_z \in Z$ に対して $dist(P_j, P_z^i) \leq$

$dist(P_j, P_z)$ である場合に限り、$P_j \in V_i$ である。ここで $dist(P_k, P_l)$ は P_k と P_l の距離とする。

- すべての $1 \leq i \leq |Z|$ について、P_z^i は T_i の根である。 □

森の各木は 1 つの Z のメンバーといくつかの Z のメンバーでない節点のグループとなっている。各 Z のメンバーでないプロセスは自身からもっとも近いところに存在する Z のメンバーと同じグループに属する。

そして次に、各グループをそれぞれ 1 つの節点と見立てて、最小重み生成木を構成する。このとき、各グループ間の距離を、それらのグループの Z のメンバー間の距離とする。このために、最短経路森をもとに、枝を次のように 3 つに分類する (c)。

- G の枝 e が G_F の枝でもあるならば、*tree edge* である。
- G の枝 e が G_F の枝でなく、その両端が G_F の同じ木の中にあるならば、*intra-tree edge* である。
- G の枝 e が G_F の枝でなく、その両端が G_F のそれぞれ異なる木の中にあるならば、*inter-tree edge* である。

そして、G_P に対して、最小重み生成木を構成する (d)。このとき、最小重み生成木の枝として選出される枝は、すべての tree edge といくつかの inter-tree edge である。つまり、グループ間をつなぐ inter-tree edge を選出することになる。

その後、必要でない葉や枝を刈り取っていく (e)。こうして、スタイナー木を得ることが出来る。

4 アルゴリズム

提案する自己安定アルゴリズム LSS-ST は、Chen らの分散アルゴリズムにもとづいている。入力としてもとのネットワーク G を与え、出力としてネットワーク G_z を返す。

一般的に、大きな自己安定アルゴリズムを設計し、その正当性を証明することは大変難しいものである。そこで、自己安定アルゴリズムの設計、証明、解析にはさまざまな手法が提案されてきた。その一つの手法として、公平な合成というのが挙げられる。ここで、A_1 と A_2 をそれぞれ正しい自己安定アルゴリズムとする。アルゴリズム A_1, A_2 を公平に実行させる。例えば、A_1 と A_2 の実行を各プロセスにおいて交互に行なう。A_2 は A_1 の状態を参照するが、A_1 は A_2

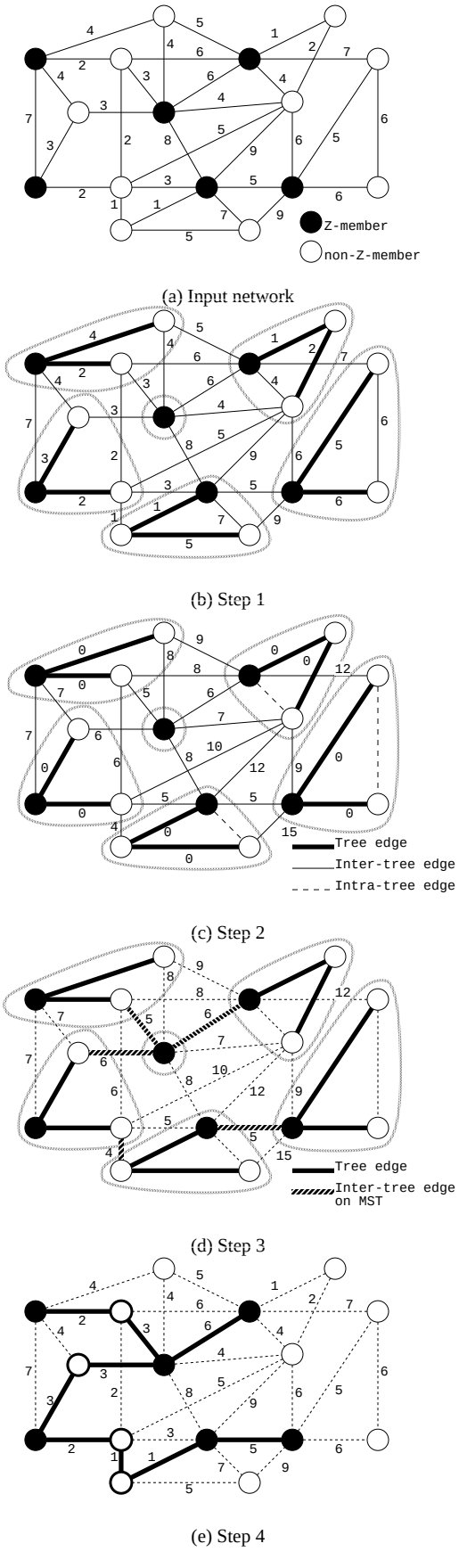

図 1: An Execution Example

の状態を参照しない。A_1 はいつかは正当な状況 (解状況) に収束するので、その状況を参照している A_2 もいつかは正当な状況に収束する。このような合成方法により、ひとつの自己安定アルゴリズム A が得られ、この方法を公平な合成という [4]。この方法は合成するアルゴリズムの数が増えたとしても、同じ議論が成り立つ。

提案する自己安定アルゴリズム LSS-ST は次の 4 つの階層から成り立ち、次の 4 つの自己安定アルゴリズムを公平な合成により合成して得られる。以下

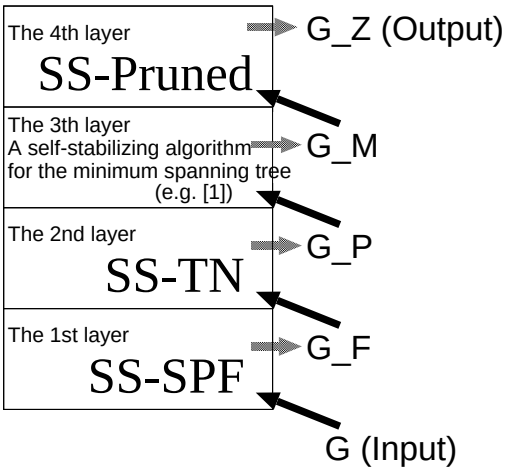

図 2: 階層的自己安定アルゴリズム LSS-ST

で 4 つの自己安定アルゴリズムを示すが、その証明については紙面の都合上省略する。くわしくは、文献 [6] を参照せよ。

1 段目：SS-SPF

$G=(V,E,w)$ と Z に対する最短経路森 $G_F=(V,E_F,w)$ を構成する。各木は Z のメンバーを根とする。

各プロセス P_i への入力は、(1) 隣接プロセスの集合 N_i、(2) P_i が Z のメンバーであるかどうかを表す定数 z_i、(3) P_j を P_i の隣接プロセスとしたときの枝の重み $w(P_i)[P_j]$、である。各プロセスの出力は、(1) 最も近い Z のメンバーからの距離 (最短経路の枝の重みの和)$D(P_i)$、(2) 最も近い Z のメンバーの識別子 $R(P_i)$、(3) 最短経路森における親 $F(P_i)$、である。

アルゴリズムを図 3 に載せる。各 Z のメンバーのプロセスは GC1 によって、木の根となる。他のプロセスは GC2 を実行し、Z のメンバーからの距離の小さいものを自身の親とする。

定理 1 アルゴリズム *SS-SPF* は最短経路森を形成する自己安定アルゴリズムである。 □

Constants
$1 \le w(P_i)[P_j]$: P_i と P_j の間の枝の重み
$z(P_i) \in \{0,1\}$: $z(P_i)=1$ の場合に限り、P_i は Z のメンバー
Local Variables
$D(P_i)$: 最も近いと思われる Z のメンバーからの距離
$R(P_i)$: 最も近いと思われる Z のメンバーの識別子
$F(P_i)$: 最短経路木上の親
Macros
$Distance(P_i) \equiv \min_{P_j \in N_i}(D(P_j)+w(P_i)[P_j])$
$Father(P_i) \equiv$
$\min\{id(P_j) \mid P_j \in N_i, Distance(P_i)=D(P_j)+w(P_i)[P_j]\}$
A Set of Guarded-Commands:
*[
// GC1: Z のメンバーは最短経路木の根となる。
$z(P_i)=1 \wedge \{D(P_i)\neq 0 \vee F(P_i)\neq P_i \vee R(P_i)\neq P_i\}$
$\rightarrow D(P_i)=0; F(P_i)=P_i; R(P_i)=P_i;$
/* GC2: Z のメンバーでないものは、
隣接プロセスの中から親を選び、距離を書換える。*/
$z(P_i)=0 \wedge \{D(P_i)\neq Distance(P_i)$
$\vee F(P_i)\neq Father(P_i) \vee R(P_i)\neq R(Father(P_i))\}$
$\rightarrow D(P_i)=Distance(P_i); F(P_i)=Father(P_i);$
$R(P_i)=R(Father(P_i));$
]

図 3: SS-SPF

定理 2 各枝に与えるコストの上限を定めたとき、最悪計算時間は $O(n^4)$ である。 □

2 段目：SS-TN, 3 段目：最小重み生成木を構成

2 段目のアルゴリズムは、枝の分類にしたがって、その重みを次のように書換え、ネットワーク $G_P=(V,E,w_P)$ を導きだす。

$$w_P(e)=\begin{cases} 0 & e \text{ が tree edge である場合} \\ \infty & e \text{ が intra-tree edge である場合} \\ \mathrm{dist}(P_i,P_z)+w(P_i,P_j)+\mathrm{dist}(P_j,P'_z) & \\ & e=(P_i,P_j) \text{ が } P_i \text{ (または } P_j\text{) の} \\ & \text{根が } P_z \text{ (または } P'_z \text{ であるような} \\ & \text{inter-tree edge である場合} \end{cases}$$

2 段目と 3 段目で、最短経路森上の各木を個々の節点とみたてたような最小重み生成木を構成する。このために、2 段目では、G_F を $G_P=(V,E,w_P)$ に変換し、3 段目では G_P の最小重み生成木 $G_M=(V,E_F,w_P)$ を構成する。

2 段目のアルゴリズムにおける各プロセス P_i に対する入力は、(1) P_j を P_i の隣接プロセスとしたとき

Constants
$w(P_i)[P_j]$: P_i と P_j の間の枝の重み
$R(P_i)$: 最も近い Z のメンバーの識別子
$D(P_i)$: 最も近い Z のメンバーからの距離
$F(P_i)$: 最短経路森上の親
Local Variable
$W(P_i)[P_j]$: P_i と P_j の間の枝の新しい重み
A Set of Guarded-Commands:
*[
// GC1: Tree edge
$\exists P_j \in N_i[R(P_i) = R(P_j) \wedge$
$(P_j = F(P_i) \vee P_i = F(P_j)) \wedge W(P_i)[P_j] \neq 0]$
$\rightarrow W(P_i)[P_j] = 0;$
// GC2: Intra-tree edge
$\exists P_j \in N_i[R(P_i) = R(P_j) \wedge$
$\neg(P_j = F(P_i) \vee P_i = F(P_j)) \wedge W(P_i)[P_j] \neq \infty]$
$\rightarrow W(P_i)[P_j] = \infty;$
// GC3: Inter-tree edge
$\exists P_j \in N_i[R(P_i) \neq R(P_j) \wedge$
$W(P_i)[P_j] \neq w(P_i)[P_j] + D(P_i) + D(P_j)]$
$\rightarrow W(P_i)[P_j] = w(P_i)[P_j] + D(P_i) + D(P_j);$
]

図 4: SS-TN

の枝の重み $w(P_i)[P_j]$、(2) 最も近い Z のメンバーの識別子 $R(P_i)$、(3) 最も近い Z のメンバーからの距離 $D(P_i)$、(4) 最短経路森上の P_i の親 $F(P_i)$、である。各プロセス P_i の出力は P_j を隣接プロセスとしたとき、P_i と P_j の間の枝の新しい重み $W(P_i)[P_j]$ である。W の集合は G_P のコスト関数 w_P となる。

アルゴリズムを図 4 に載せる。各プロセスは自身に接している枝をそれぞれ分類して、そのコストを付け変える。枝が tree edge なら GC1 を、intra-tree edge なら GC2 を、inter-tree edge なら GC3 をそれぞれ実行する。

定理 3 アルゴリズム *SS-TN* はコストを変換する自己安定アルゴリズムである。 □

定理 4 最悪計算時間は $O(n^2)$ である。 □

3 段目：最小重み生成木を構成する自己安定アルゴリズム

例えば文献 [1] の生成木を計算する自己安定アルゴリズムを使用すればよい。

4 段目：SS-Pruned

最小重み生成木 G_M の必要でない節点や枝を刈り取る。

各プロセスへの入力は最小重み生成木 G_M における子供の集合 $CH(P_i) \subseteq N_i$ と P_i が Z のメンバーであるかないかを表す定数 z_i である。各プロセスは自身がスタイナー木のメンバーであるかないかをブーリアン変数 $S(P_i)$ に出力する。

アルゴリズムを図 5 に載せる。Z のメンバーは GC1 によってスタイナー木のメンバーとなる。G_M の葉で、Z のメンバーでないものは GC2 によって、スタイナー木のメンバーとはならない。そのほかのプロセスは GC3 と GC4 を実行する。すべての子がスタイナー木のメンバーでないなら、GC3 によってスタイナー木のメンバーでなくなる。そうでなければ、GC4 によって、スタイナー木のメンバーとなる。

ここで、$S(P_i) = 1$(スタイナー木のメンバー) のプロセスを最小重み生成木 G_M の枝でむすび、各枝の重みをもとの重みに戻し、得られた $G_z = (V_z, E_z, w)$ が G のスタイナー木の近似解である。

定理 5 アルゴリズム *SS-SPF* は最小重み生成木を刈り取ってスタイナー木を構成する自己安定アルゴリズムである。 □

定理 6 最悪計算時間は $O(n^2)$ である。 □

4.1 考察

最小重み生成木を求めるアルゴリズムと SS-Pruned の 2 つの自己安定アルゴリズムを合成しただけでも、Pruned-MST を単独で実行した場合の自己安定分散近似アルゴリズムであると言える。この場合はコンペティティブネスは $n - |Z| + 1$、計算時間は $O(n^2)$ かあるいは最小重み生成木を求めるアルゴリズムの計算時間に等しい。

4 つ全てのアルゴリズムを合成すると、コンペティティブネスが $2(1 - 1/l) < 2$、計算時間は SS-SPF の計算時間である $O(n^4)$ か、あるいは最小重み生成木を求めるアルゴリズムの計算時間に等しい自己安定分散アルゴリズムが得られる。これは Pruned-MST を実現するよりは複雑にはなるが、よりよい近似解が得られると言える。この計算時間複雑度の解析には、枝の数や初期状態など、すべてが最悪の場合を考えてあり、自己安定ではけっして大きすぎるものでもないことを付け加えておく。

Constants
$CH(P_i)$: G_M 上の子供の集合
$z(P_i) \in \{0,1\}$: $z(P_i) = 1$ の場合に限り P_i は Z のメンバー
Local Variables
$S(P_i) \in \{0,1\}$ —$S(P_i) = 1$ (resp. 0) の場合に限り
P_i はスタイナー木のメンバー
(resp. メンバーでない)
A Set of Guarded-Commands:
*[
// GC1: Z のメンバーはスタイナー木に加わる。
$S(P_i) = 0 \wedge z(P_i) = 1 \rightarrow S(P_i) := 1;$
/* GC2: Z のメンバーでない G_M の葉は
スタイナー木からはずれる。*/
$S(P_i) = 1 \wedge z(P_i) = 0 \wedge CH(P_i) = NULL \rightarrow S(P_i) := 0;$
/* GC3: Z のメンバーでないもので、
すべての子供がスタイナー木からはずれていたら、
自身もスタイナー木からはずれる。*/
$S(P_i) = 1 \wedge z(P_i) = 0 \wedge \forall P_j \in CH(P_i)[S(P_j) = 0]$
$\rightarrow S(P_i) := 0;$
/* GC4: Z のメンバーでないもので、
少なくとも 1 つはスタイナー木のメンバーである
子供が存在したら、自身はスタイナー木に加わる。*/
$S(P_i) = 0 \wedge z(P_i) = 0 \wedge \exists P_j \in CH(P_i)[S(P_j) = 1]$
$\rightarrow S(P_i) := 1;$
]

図 5: SS-Pruned

しかし、$|Z| = 2$ の場合は最短経路を計算するだけで十分であり、$|Z| = n$ は最小重み生成木を計算するだけで十分であるので、本アルゴリズムでの計算は冗長であると言える。よって、よりよい自己安定アルゴリズムの設計がのぞまれる。

5 おわりに

本研究では最小重みスタイナー木の近似解を求める自己安定アルゴリズムを提案し、その正当性を証明した。本アルゴリズムは最短経路森を求める SS-SPF、ネットワークを変換する SS-TN、最小重み生成木を求めるアルゴリズム、そして刈り取るアルゴリズム SS-Pruned の 4 つの階層によって成り立つ。

提案したアルゴリズムのコンペティティブネスは $2(1-1/l) < 2$ (l は最適解の葉の数) である。

参考文献

[1] Gheorghe Antonoiu and Pradip K. Srimani. Distributed self-stabilizing algorithm for minimum spanning tree construction. *Euro-Par'97 Parallel Proceeding, LNCS*, 1300:480–487, 1997.

[2] Gen-Huey Chen, Michael E. Houle, and Ming-Ter Kuo. The Steiner problem in distributed computing systems. *Information Sciences*, 74:73–96, 1993.

[3] Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, 1974.

[4] Shlomi Dolev. *Self-Stabilization*. The MIT Press, 2000.

[5] Clemens Gropl, Stefan Hougardy, Till Nierhoff, and Hans Jurgen Promel. Approximation algorithms for the Steiner tree problem in graphs. *In Steiner Trees in Industry, X. Cheng and D.-Z. Du (Eds.)*, pages 235–279, 2001.

[6] Sayaka Kamei. A self-stabilizing algorithm for the steiner tree problem. Master's thesis, Department of Information Engineering, Hiroshima University, Japan, 2003.

[7] Richard Manning Karp. *Reducibility among combinatorial problems*. New York:Plenum Press, 1972.

[8] L. Kou, George Markowsky, and Leonard Berman. A fast algorithm for Steiner trees. *Acta Informatica*, 15:141–145, 1981.

[9] Gabriel Robins and Alexander Zelikovsky. Improved Steiner tree approximation in graphs. *In Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2000*, pages 770–779, 2000.

[10] D. Waitzman, C. Partridge, and S. Deering. Distance vector multicast routing protocol. *RFC:1075*, November 1988.

[11] Pawel Winter. Steiner problem in networks: A survey. *Networks*, 17:129–167, 1987.

[12] Y. F. Wu, Peter Widmayer, and C. K. Wong. A faster approximation algorithm for the Steiner problem in graphs. *Acta Informatica*, 23:223–229, 1986.