マルチメディア通信と分散処理 95－10
（１９９９．１１．１９）

動的Steiner木生成問題のための分散アルゴリズム

塚田 慎[†] 高井 昌彰[††]

[†]北海道大学大学院工学研究科
[††]北海道大学大型計算機センター

マルチキャストサービスの発展により、異なるマルチキャストパスが同一ネットワーク上に共存する状況も珍しくなくなっている。異なるマルチキャストパスの各々のアプリケーションにとって、通信のオーバーヘッドを最小にする経路を設定することは重要である。この問題は最小Steiner木生成問題を分散環境で複数同時生成することに相当する。本論文では、共通の制約で複数の最小Steiner木の近似解を同時生成する分散アルゴリズムについて論じる。また、マルチキャストメンバーノードが動的に変更しうる動的Steiner木生成問題に関する分散アルゴリズムについても論じる。

Distributed Algorithms for Dynamic Steiner Tree Problem

Makoto Tsukada [†] and Yoshiaki Takai[††]

[†]Graduate School of Engineering, Hokkaido University
[††]Hokkaido University Computing Center

Increase of multicast services brings the situation that different multicast paths coexist in the same computer network. It is important to arrange the multicast paths to minimize the communication overhead for each application on the different multicast paths. This problem is a minimum Steiner tree problem in the distributed environment. In this paper we discuss a distributed algorithm that generates multiple quasi-minimum Steiner trees at the same time under the common constraint. We also discuss a distributed algorithm for a dynamic Steiner tree problem in which multicast member nodes can be dynamically changed.

1 はじめに

テレビ会議室や遠隔教育などのネットワークアプリケーション使用時には、特定の複数ノードにデータを送信するマルチキャストサービスが必要不可欠である。また、ネットワーク技術の進歩にともない、その需要はますます拡大している。

これらのマルチキャストサービスでは、まずデータ送信の経路となるマルチキャストパスが設定される。異なるサービスが提供される場合では、結果としてそれぞれのマルチキャストパスが設定されることになる。そのような状況では、サービスがリンクを共有する可能性があり、通信遅延の悪化などの問題が生じる。

ネットワーク上で最適なマルチキャストパスを実現することは、グラフ理論における最小Steiner木生成問題を解決することに相当することが知られており、分散環境でこの問題の近似解を求めるアルゴリズムがすでに提案されている[1],[2]。

しかし、既に述べたように現実では単一のネットワーク上に複数のマルチキャストパス

を生成することが多く、これまでに提案されているアルゴリズムをそのまま適用することはできない。

そこで本論文では、異なるマルチキャストグループのSteiner木が1つのリンクを共有したとき、帯域幅共有によって通信遅延が増加するという前提の下、複数の最小Steiner木の近似解を同時生成する分散アルゴリズムを提案する。また、提案手法を評価するためにシミュレーション実験を行い、比較対象としてpruned MST法を逐次的に繰り返し使用するアルゴリズムとの比較結果について述べる。

複数マルチキャストパス生成後も、マルチキャストメンバーノードの動的参加・脱退の要求は起こりうる。この問題を扱う研究は重要であるにも関わらず、従来、あまり行われていない。我々はこの問題の解法として複数同時生成可能な動的Steiner木生成問題を解決する分散アルゴリズムを提案し、実装を試みる。

2 Steiner木生成問題

対象となるネットワークトポロジーを、コスト付き無向グラフ $G=(V,E,c)$ とする。ここで、コストとは各リンクに付けられた通信遅延であるとする。マルチキャストグループのメンバーノードをターミナルノードと呼び、ターミナルノードの集合 Z は V の部分集合である。

本研究で扱う最小Steiner木生成問題は以下のように定義される。

グラフ G、及び V の部分集合 Z が与えられたとき、Z の全頂点を閉路を構成しないように連結したものをSteiner木と呼び、この木を T とする。Z のそれぞれを連結して得られた T には、Z 以外の V に属する頂点（ノンターミナルノードと呼ぶ）が含まれていてもよい。T を構成するリンクのコストの総和を $C(T)$ としたとき、$C(T)$ が最小になるSteiner木を求める問題のことを最小Steiner木生成問題と呼ぶ。

続いてSteiner木生成終了後、マルチキャストメンバーノードの参加・脱退を表明する要求メッセージ列を $R=\{r_0,r_1,\cdots,r_k\}$ とする。ここで、時刻 i での要求 r_i は (v_i,p_i) のペアであり、$v_i \subseteq V, p_i \subseteq \{add, remove\}$ である。ターミナルノードの集合を Z_i とすると、動的Steiner木生成問題は以下のように定義される。

グラフ G、要求メッセージ列 R が与えられたとき、各時刻 i でのノード集合 Z_i の全頂点を張り、かつ、$C(T_i)$ を最小にするSteiner木 $\{T_1,T_2,\cdots,T_k\}$ を求める問題を動的Steiner木生成問題と呼ぶ[4]。

3 Steiner木の複数同時生成のための分散アルゴリズム

以下に提案する分散アルゴリズムでは、静的、動的いずれも2つのSteiner木を同時生成する。その際、2つのSteiner木のリンクのコストの合計を最小にする近似解を生成することを目的とする。2つのSteiner木の同時生成が可能であれば、それ以上の数のSteiner木生成への応用は容易であると考えられる。

3.1 2つの静的Steiner木の同時生成

ここでの目的は、グラフ G、V の部分集合 $Z_A, Z_B (Z_A \cap Z_B = \phi)$ が与えられたとき、Z_A, Z_B のそれぞれに対応する2つの最小Steiner木の近似解を生成することである。

以下に近似解を同時生成する分散アルゴリズムの概要を示す。

step1 アルゴリズムリーダノード v_{AL} は全ノードにアルゴリズム開始メッセージを送信する。ターミナルノードがこのメッセージを受信した場合、step2に移行する。

step2
各ターミナルノードは、SPF(Shortest Path Forest)の生成を行う。SPFとは、その構成要素となっている各ノードが、最小のコスト

でターミナルノードへ連結されている G のフラグメントである。

各ターミナルノードはSPFの生成が終了した時点で、step3を開始する。

step3 同じマルチキャストグループのフラグメント同士を、フラグメント内のターミナルノード間の経路のコストが最小になる一本の外向枝で接続するために、まず最小外向枝発見手続きを開始し、次に結合手続きを行う。

結合後の2つのフラグメントを新たな1つのフラグメントとして、再び最小外向枝の発見・結合を繰り返す。結合可能な外向枝がなくなったとき、各フラグメント内のターミナルノードはstep4に進む。

step4 各フラグメント間で探索メッセージを送信することにより、フラグメント間の最短経路、及びその経路の合計コストを探索し接続を行う。

接続後の2つのフラグメントを新たな1つのフラグメントとする。フラグメントに全てのターミナルノードが含まれた時点でアルゴリズムは終了する。

3.2 2つの動的Steiner木の同時生成法

静的な2つのSteiner木の生成終了後において、時刻 i、マルチキャストグループ $M \subseteq \{A, B\}$、要求メッセージ列を $R = \{r_0, r_1, \cdots, r_k\}$ とする。ただし、Mに参加・脱退を表明するノードをv_{Mi}とするとき、r_iとはv_{Mi}の表明する要求のことである。原則として、要求r_{i-1}の処理が終了しないうちに、r_iの処理を行うことはできないものとする。ここでの目的は、ある時刻iでの$Z_{Mi}(Z_A \cap Z_B = \phi)$に対応するそれぞれの2つの動的Steiner木 $T_{Mi} = \{T_{M1}, T_{M2}, \cdots, T_{Mk}\}$ を求めることである。

以下に動的Steiner木生成のためのアルゴリズムの概要を示す。

3.2.1 参加要求

step1 v_{Mi}はv_{AL}へ向けて*add*メッセージを送信する。このとき、*add*メッセージの通る経路と、時刻$i-1$のSteiner木である$T_{M(i-1)}$が交わっているかどうかを調べる。

step2 *add*メッセージの経路と$T_{M(i-1)}$が、あるノードで交わっていれば(図1の太い実線と細い実線の場合)、交わったノードからv_{Mi}までの経路を$T_{M(i-1)}$の経路に付け加えたものを求めるT_{Mi}として、要求r_iの処理は完了する。図1では、$v_{Mi} - v_2$が$v_1 - v_3$に付け加えられる。

*add*メッセージの経路と$T_{M(i-1)}$が交わっていなければ(図1の太い波線と細い実線の場合)、step3へ進む。

step3 $v_{AL} - v_{Mi}$間の経路を$T_{M(i-1)}$に付け加えるために、v_{AL}は、自分自身からSteiner木までの最短経路を求めるための*connect*メッセージをブロードキャストし、v_{AL}から$T_{M(i-1)}$までの最も近い距離にある経路(図1の$v_{AL} - v_2$、または$v_{AL} - v_{Mi} - v_2$間のリンクのコストの合計の小さい方)を発見する。この経路と、$v_{AL} - v_{Mi}$(図1の太い波線)を$T_{M(i-1)}$に付け加え、これを求めるT_{Mi}として、要求r_iの処理は完了とする。

3.2.2 脱退要求

step1 v_{Mi}はv_{AL}へ向けて*remove*メッセージを送信する。

また、v_{Mi}から$T_{M(i-1)}$上の最も近い、次数3以上のノード(中継点、図2のv_2, v_5)までの経路、または、最も近いターミナルノードまでの経路を開放し、自分自身もマルチキャストグループから脱退する。

図2では、$v_{Mi} - v_2$と$v_{Mi} - v_5$間の太い実線で示された経路を開放する。

step2 分離された2つの経路(図2、$v_1 - v_3$と$v_4 - v_6$)を接続するため、v_{AL}は*connect*メッセージを中継点v_2とv_5に送信する。

step3 脱退要求前の $v_2 - v_5$ 間と、脱退要求後の $v_2 - v_{AL} - v_5$ 間の経路のコストの合計値を比べ、小さい方の経路を $T_{M(i-1)}$ に付け加え、これを求める T_{Mi} として、要求 r_i の処理を終了する。

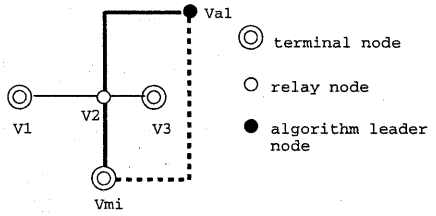

図 1: 参加要求

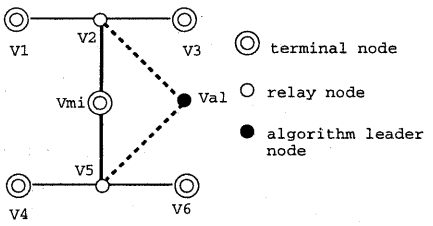

図 2: 脱退要求

4 評価実験

4.1 実験条件

実験で用いるネットワークトポロジーの生成には、一様乱数に基づく方法と、二点間のノードの距離をパラメータとする指数分布関数を用いる方法 [3]、[4] の 2 つを用いた。

一様乱数に基づく方法では、① アルゴリズムに参加する全ノード数、② 全ノード数に対する 2 つの Steiner 木のそれぞれのターミナルノード数の割合、③ 含まれるリンク数の完全グラフに対する割合の 3 つのパラメータを用いた。

指数分布関数を用いる方法では、まず各ノードをデカルト平面上の点 $\{p(x,y)|0 \leq x < N, 0 \leq y < N\}$ にランダムに配置し、次にノード間のリンクを確立する。すなわち、2 つのノード s,t 間の距離 $d_{s,t}$ に基づく指数分布関数 $Q(s,t)$ を

$$Q(s,t) = \beta exp(\frac{-d_{s,t}}{2\alpha N})$$

と定義し、この値と一様乱数で生成された値 $r(0 \leq r < 1)$ とを比較し、r が $Q(s,t)$ より小さければノード s,t 間のリンクを確立する。ここで、N はアルゴリズムに参加する全ノード数、$d_{s,t}$ はリンクに付けられたコストを意味する。

異なる 2 つの Steiner 木が 1 つのリンクを共有した場合には、そのリンクが各グループ毎に、それぞれ 2 倍のコストを持つものとして、2 つの Steiner 木のリンクのコストの合計値を計算する。

提案手法の比較対象として、pruned MST 法 [5] を 2 つの Steiner 木を生成するために用いる。この際、pruned MST 法を 2 回目に適用するときには、1 回目の Steiner 木がすでに完成しているという条件下で 2 回目の生成を行う。

以下に示す実験結果は、静的 Steiner 木生成問題について、同一パラメータでの一様乱数に基づく生成、及び指数分布関数を用いた生成の 100 個のネットワークトポロジーにおける平均値である。

4.2 結果・考察

4.2.1 静的 Steiner 木生成問題

一様乱数に基づくネットワークトポロジーでの 2 つの Steiner 木のリンクのコストの合計値 (解コスト) と計算時間をそれぞれ、図 3、4に示す。ターミナルノード数の割合 (=0.1)、及びリンク数の割合 (=0.05) は一定とする。

また、指数分布関数を用いた場合での 2 つの Steiner 木の解コストと計算時間をそれぞれ、図 5、6に示す。ここで、全ノード数 $N = 100, \alpha = 0.2, \beta = 0.35$ である。

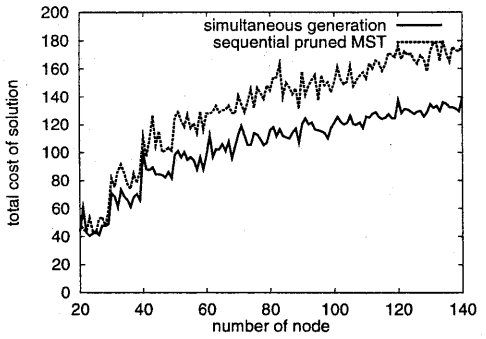

図 3: ノード数を変化させたときの解コスト
(一様乱数ネットワーク)

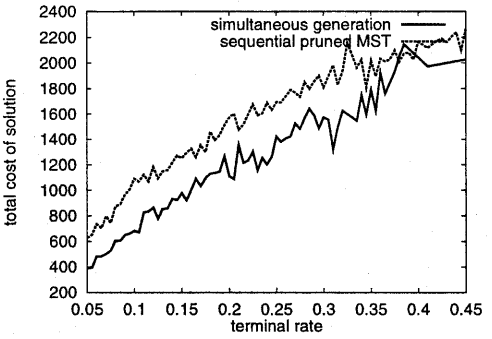

図 5: ターミナル数変化の解コスト
(一様乱数ネットワーク)

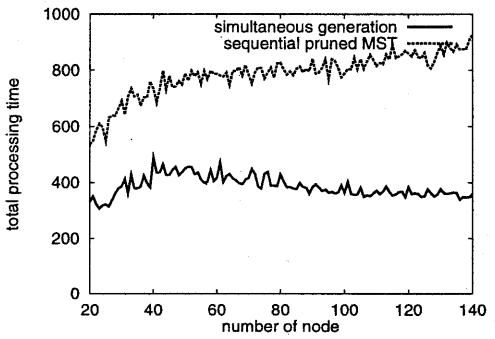

図 4: ノード数を変化させたときの計算時間
(指数分布ネットワーク)

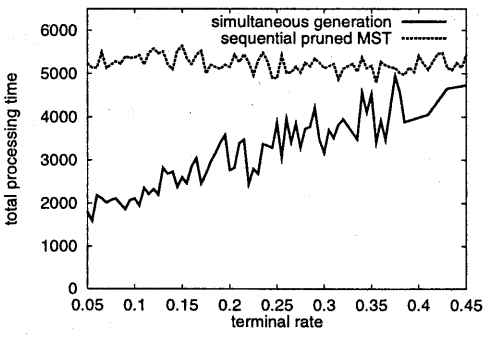

図 6: ターミナル数変化の計算時間
(指数分布ネットワーク)

図3、5より、提案手法は従来手法と比較して、約70%の解コストを持った解の生成が可能であることが分かる。

図4は、Nの増加に伴い、提案手法の計算時間が減少していることを示している。この理由は、ノード数が増加すると、リンク数も相対的に増加するためである。一方、従来手法ではNの増加にともない、計算時間も増加しており、提案手法の約2倍の計算時間を要している。

また、図6では、従来手法の計算時間にあまり変化が見られない。これは、従来手法の計算時間がノード数とリンク数の割合にのみ依存するためである。これに対して、提案手法ではターミナルノード数の割合が増加するに従って、計算時間も増加している。提案手法ではターミナルノード同士の結合に主に計算時間を要しているためである。

4.2.2 動的 Steiner 木生成問題

$N = 20$ としたときの固定的なネットワークトポロジー上での問題を解決する分散アルゴリズムの実装は終了している。現在、これを一様乱数ネットワーク及び、指数分布ネッ

トワークでの実験を行っている。

5 おわりに

本論文では、異なる複数のマルチキャストパスを生成するために、Steiner木の複数同時生成分散アルゴリズムを静的・動的それぞれについて提案した。

静的アルゴリズムについて、2つのネットワーク条件下でシミュレーション実験を行い、従来手法との比較を行った。その結果、提案手法を用いた場合、得られるSteiner木の解コストでは従来手法の約70%で解の生成が可能であることが分かった。必要な計算時間の比較においては、ターミナルノード数を一定にした場合では、従来手法の約50%で解の生成が可能であり、どちらの評価でも本手法の有効性が明らかになった。

現在、動的Steiner木生成問題を解決する分散アルゴリズムをネットワークシミュレータに実装、評価実験中である。

参考文献

[1] Fred Bauer and Anujan Varma, "Distributed Algorithms for Multicast Path Setup in Data Networks", *IEEE/ACM Transactions on Networking.*, vol.4, no.2, pp.181-191, 1996

[2] 大鷹 秀之、高井 昌彰、" 分散アルゴリズムを用いたMSTの構成法について"、情報処理学会第52回 (平成8年前期) 全国大会 講演論文集 (6)、pp.455

[3] Bernard M.Waxman, "Routeing of Multipoint Connections", *IEEE Journal on Selected Areas in Communications.*, vol.6, no.9, pp.1617-1622, 1998

[4] Ehud Aharoni and Reuven Cohen, "Restricted Dynamic Steiner Trees for Scalable Multicast in Datagram Networks", *IEEE/ACM Transactions on Networking.*, vol.6, no.2, pp.286-297, 1996

[5] R.G.Gallager, P.A.Humblet, and P.M.Spira, "A Distributed Alogorithm for Minimum-Weight Spanning Trees" *ACM Transactions on Programming Languages and Systems.*, vol.5, no.1, pp66-77, 1983