Vol.2016-AL-158 No.9
2016/6/25

分散コンピューティング制御効率化のための平方分割手法による動的グラフにおける最小全域木クエリ処理

山崎 一明[a)]

概要：モバイル機器での分散コンピューティング制御はデータ転送に無線通信を利用するが，環境の影響を受けコストや可用性が変化する無線通信においては，最適なネットワークも変化する．最適なネットワークとはグラフ理論における最小全域木に対応する．本研究は最小全域木に基づいてネットワークを再構築することで通信コストを最小化できることを示し，そのための効率的な手法を提案することを目的とする．グラフ $G = (V, E)$ の最小全域木を求める Prim や Kruskal のアルゴリズムがあるが，どちらも $O(|V|^2)$ 時間がかかる．提案手法では，グラフ変化時の最小全域木の変化は局所的であることを利用する．頂点集合 V を $\sqrt{|V|}$ 個程度のクラスタに分割し，それに基づいて辺集合を分割することで辺の探索を効率化できる．グラフの変化に対しては数個のクラスタの分割・統合により，高速探索が可能な辺集合分割を維持する．理論解析により提案手法が $O(|V|\sqrt{|V|})$ 時間で動作することを示し，計算機実験により既存手法と比較して十分高速に動作することを示した．これにより，モバイル機器での分散コンピューティング制御における通信コスト最小化のための手法の提案ができた．

キーワード：動的グラフ，最小全域木

MST Query on Dynamic Graph with Square-Root-Decomposition for Optimization for Controlling Distributed Computing

Yamazaki Kazuaki[a)]

Abstract: Distributed computing on mobile entities uses wireless communication for data transportation. Costs and availability of wireless communication changes affected by surroundings, thus optimal network (corresponds a MST) also change. Dynamic network can be modeled in dynamic graphs. This paper proposes a method for calculating Minimum Spanning Tree (MST) on dynamic graphs efficiently. The method is based on square-root-decomposition on tree nodes and dividing edge set into several sets according vertex dividing. Then, number of edges that candidate for MST is decreased and we can search edge in shorter time.

Keywords: Dynamic Graphs, Minimum Spanning Trees

1. まえがき

1.1 背景

近年，スマートフォンなどをはじめとするモバイル機器の演算性能が向上している．また，モバイル機器は，無線通信を用いて相互に通信することができる．こうした状況により，モバイル機器を利用した分散コンピューティングが現実的なものとなった．

モバイル機器を取り巻く環境は人や物体の移動によって変化する．それに対応して，機器間の直接通信のコストや可用性も周囲の環境の変化に対応して変化する．

あるモバイル機器群が利用可能な無線通信は，機器を頂点とし，機器間の通信を辺としたグラフとして表現できる．

a) torus711@jaist.ac.jp

しかし，先に述べたように無線通信のコストや可用性は周辺環境の変化の影響を受け変化するため，それを表現したグラフも変化する．従って，グラフの変化を扱う手法が必要となる．

1.2 本研究の目的

本研究で行うことは以下の点である．

(1) 周辺環境の無線通信への影響と分散コンピューティング制御について述べ，通信コスト最小化の意義を示す
(2) 問題をグラフに定式化し，変更クエリを導入する
(3) グラフ変更クエリを効率的に処理する手法を提案する
(4) 提案手法の理論解析を行う
(5) 既存手法と提案手法を実装し，計算時間を比較する

2. 分散コンピューティング制御効率化

2.1 モバイル機器上の分散コンピューティング制御

モバイル機器上の分散コンピューティングでは，モバイル機器群がネットワークを構成する．

電波通信において，アンテナの性能や距離と電力との関係を表す式が $\frac{P_r}{P_t} = \left(\frac{\lambda}{4\pi D}\right)^2 G_t G_r$（Friis の伝達公式）である [1]．ここで，P_r は受信電力を表し，P_t は送信電力を表す．G_t, G_r, λ はアンテナの性能や電波の波長を表す変数であるが，利用する端末や周波数を固定すると定数と見做せる．式 **??** から，無線通信によるモバイル機器間の直接通信は，端末の移動の影響を受けて可用性や必要電力が変化すると言える．また，障害物の影響を受けて通信可能距離や伝送速度が変化する [2].

2.2 制御の効率化のためのネットワーク構造

2.2.1 ネットワーク上の接続状態をあらわす木

分散コンピューティング制御においては，木というトポロジのネットワークを構成することで，効率的な制御を実現できる [3]．木は，モバイル機器群を相互に接続する構造としては構造が最も単純なものであると言える．トポロジが木となっているとき，ネットワーク中の任意の機器から別の任意の機器への伝送経路はただ一つに定まる．

2.2.2 ネットワーク構造の変化への対応問題

2.1 節で述べたように，無線通信は周囲の環境によって通信に必要な電力や通信可能な距離が変化する．

機器間の通信には消費電力や速度でコストを定義できるので，直接通信の集合であるネットワーク全体のコストは通信のコストの和として定義できる．通信のコストや可用性が動的に変化する中で，各時点において最小コストで実現できるネットワークを構成することが，コストを最小化することに繋がる．最小コストで実現できるネットワークに対応する木を最小全域木と言う．短時間で変化後の最小全域木を求めネットワークを再構成することで，非効率的なネットワークを利用する時間が短縮し，通信に必要な電力や時間が減少する．

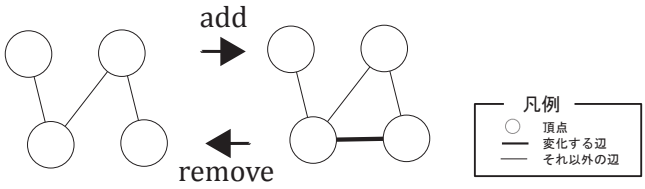


図 1 2 種類のクエリ

3. 問題の定式化

3.1 グラフ・MST

グラフとは，頂点集合 V と辺集合集合 $E \subseteq V^2$ の組 $G = (V, E, w)$ である．重み付きグラフとは，グラフに，重み関数 $w : E \mapsto \mathbb{Z}^+$ を加えた (V, E, w) である．本論文で扱うグラフは無向グラフのみであるため，以降無向グラフのことを単にグラフと書き，$(u, v) = (v, u)$ であるとする．また，同一の頂点を結ぶ辺 $(v, v) \in E$ は存在せず，また，グラフは常に連結であると仮定する．

連結なグラフであって閉路を含まないグラフ (V, E) を木とよぶ．グラフが木のとき，$|E| = |V| - 1$ である．木 (V, T) が含む辺の重みの総和 $\sum_{e \in T} w(e)$ を木の重みと呼ぶ．あるグラフ $G = (V, E)$ に対し，木 (V, T) を G の全域木と呼ぶ．G の全域木の内，重みが最小のものを G の最小全域木 (MST : Minimum Spanning Tree) と呼ぶ．以降は MST と書く．

3.2 グラフ変更クエリへの帰着

モバイル機器の集合を V とし，集合 $E = \{(u, v) \mid$ 2 機器 $u, v \in V$ 間で通信可能 $\}$ をとる．関数 $w : E \to \mathbb{Z}^+$ を $w((u, v)) :=$ u, v 間の通信にかかるコスト である関数とすれば，グラフ $G = (V, E, w)$ はある時点で利用可能な通信を表す．最適なネットワークは，G の MST に対応する．

ネットワークの変化を扱うために，グラフに対する以下の二つの変更操作を定義する．

(1) $add(u, v, x) := E$ に (u, v) を加え $w((u, v)) \leftarrow x$ とする
(2) $remove(u, v) := E$ から (u, v) を削除する

2 種類のクエリを図 1 に示す．図において，太い線で描かれている辺がクエリにより変化する辺である．2 つの操作 *add* と *remove* は互いに逆の操作となっている．

ある時間範囲における通信の状態変化は，上記クエリの列として表現できる．変化するグラフを動的グラフと呼ぶ．

4. グラフ変更の手法

4.1 既存手法

動的でないグラフの MST を求める手法として，Prim のアルゴリズム [4], [5] と Kruskal のアルゴリズム [4], [6] がある．Prim のアルゴリズムを用いて動的グラフ上の MST

クエリを処理する場合，変更クエリを Q 回処理するためには $O(Q|E|\log|V|)$ 時間がかかる．Kruskal のアルゴリズムを用いて動的グラフ上での MST クエリを処理する場合，変更クエリを Q 回処理するためには $O(Q|E|\log|E|)$ 時間がかかる．

Kruskal のアルゴリズムを利用したより効率的なアルゴリズムとして，辺重みに関して昇順ソート済みの辺リストを保持し，変更クエリのたびにソート済みリストへ挿入・削除することによって新たなソート済み辺リストを得るアルゴリズムを構成できる．この手法を Enhanced Kruskal と呼ぶことにする．Enhanced Kruskal を用いて変更クエリを Q 回処理するには $O(Q|E|)$ 時間がかかる．

漸近的にもっとも高速に動作するのは Enhanced Kruskal である．このことから，既存手法を用いてグラフ変更クエリを Q 処理するには $O(Q|E|)$ 時間が必要であると言える．

4.2 提案手法

グラフ $G=(V,E)$ の変化は，以下の 4 通りに分類できる．

(1) 辺が追加され，変更後の MST がその辺を含むとき
(2) 辺が追加され，変更後の MST がその辺を含まないとき
(3) 辺が削除され，変更前の MST がその辺を含むとき
(4) 辺が削除され，変更前の MST がその辺を含まないとき

(2) と (4) の場合は MST は変化しない．

(1) のとき，MST に辺を加えると閉路ができる．この場合，できた閉路から重みが最も大きい辺を取り除くことで，グラフ変化後の MST となるので，この場合は辺 2 本だけ MST が変化する．

(3) のとき，MST から辺を取り除くと二つの連結成分からなるグラフになる．この場合，異なる連結成分に属する頂点間を結ぶ辺の内で，辺重みが最小のものを加えることで，グラフ変化後の MST となる．従って，この場合も辺 2 本だけ MST が変化する．

以上により，(1), (2), (3), (4) のいずれの場合でも，MST の変化は高々 2 本の辺である．

4.1 節で示したアルゴリズムはいずれもグラフが変化するたびに全ての辺を調べ MST を再計算しているが，上で示したことを踏まえると無駄な計算をしている．提案手法では各時点における MST を保持し，変更クエリにより変化する辺を効率的に探索することで効率化する．

候補となる辺の数は (3) のケースが最も多いため，(3) のケースを高速化することが高速化のための主題となる．

4.2.1 辺集合の平方分割

4.2 節の (3) のケースにおいて辺削除後に MST に加える辺の探索に $O(|V|^2)$ 時間が必要となる．これは，探索対象となる辺リストの長さが $O(|V|^2)$ となるためである．提案手法では，辺リストを分割して複数の辺リストを保持する．

辺集合を分割するにあたり，まずは MST $S=(V,T)$ の構造に基づいて頂点集合を分割し，それを辺集合分割の基準とする．着目している (3) のケースは T からある辺 (u,v) を取り除いた場合を含むが，この場合は $T'=T\setminus\{(u,v)\}$ としてグラフ $S'=(V,T')$ の構造を基準とする．分割においては，頂点集合を n 個の部分集合 $V_0,V_1,\ldots,V_{n-1}\subseteq V$ に分割し，$\forall i,j(0\le i,j<n,i\neq j),V_i\cap V_j=\emptyset$ となるようにする．更に各 V_i について，V_i による S' の誘導部分グラフが連結となるようにする．この分割をクラスタリングと呼び，分割後の各頂点集合をクラスタと呼ぶ．クラスタリングの例を図 2 に示す．図のグラフにおいて，太く示した辺がグラフの MST が含む辺である．破線で描かれた枠で囲われている頂点群がそれぞれのクラスタである．

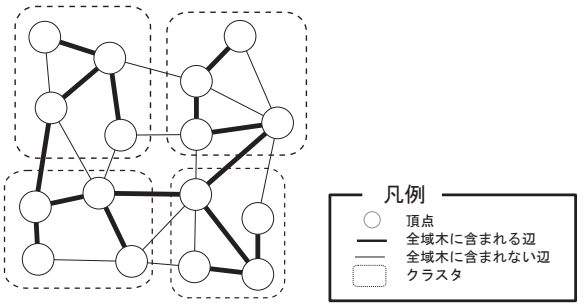


図 2 頂点のクラスタリング

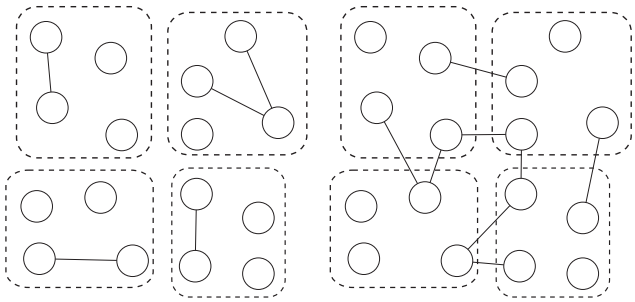


図 3 内部辺と外部辺を抽出した図

辺集合の分割では，各辺集合が次の条件のいずれかを満たすようにする．

(1) ある i ($0\le i<n$) が存在して，辺集合が含む各辺について端点がどちらも V_i に属する
(2) ある i,j ($0\le i,j<n,i\neq j$) が存在して，辺集合が含む各辺の端点の一方が V_i に属し，他方が V_j に属する

前者の条件を満たす辺を，クラスタの内部を結ぶことから内部辺と呼ぶ．後者の条件を満たす辺を，異なるクラスタ間を結び，クラスタの外部を通ることから外部辺と呼ぶ．各クラスタ，クラスタの対について対応する辺集合が存在し，全ての辺はいずれかの辺集合に該当する．MST が含む辺については，T' に加えて MST を構築する際に使う辺となり得ないので，分割した辺集合に含めない．図 2 から内部辺と外部辺を抽出した図を図 3 に示す．

分割した各辺集合を表す辺リストは，4.1 の Enhanced

Kruskal と同様に重みに関して昇順ソート済みの状態を保つ．この場合，(3) のケースで MST 分断後に MST に加える辺はいずれかの辺リストの先頭にある．従って，辺集合を m 個に分割したとすれば，MST に加える辺の探索は外部辺集合に対応する辺リストについて先頭要素だけを対象にすればよいので $O(m)$ 時間で実行可能である．

頂点集合の分割の方法によって辺の集合の分割が変わり，辺の探索に必要な計算量も変わる．提案手法では，クラスタ数 n が $\sqrt{|V|}$ 程度となるように分割する．構造が恣意的でないグラフでこれを実現するには，所属クラスタが決定されていない適当な頂点から MST 上での幅優先探索を行い，訪問済みかつクラスタ未決定な頂点が $\lfloor \sqrt{|V|} \rfloor$ 個になった時点でそれらの頂点を一つのクラスタとする処理を，未訪問頂点が無くなるまで繰り返せばよい [7]．これにより，各クラスタに属する頂点数が平均的に $\sqrt{|V|}$ 個となる．クラスタ数は頂点数 $|V|$ をクラスタの平均サイズ $\sqrt{|V|}$ で割った値程度であるため，$\frac{|V|}{\sqrt{|V|}} = \sqrt{|V|}$ より，クラスタ数は $\sqrt{|V|}$ 個程度となる．クラスタリングの計算に続き，計算したクラスタリングに基づいて辺集合を分割する．全ての辺を重みに関して昇順ソートし，辺リストの先頭から順に辺を振り分ければ，辺集合の分割は $O(|E| \log |E|) = O(|V|^2 \log |V|)$ 時間で完了する．

4.2.2 クエリの平方分割

5 章で示すように，提案手法ではグラフトポロジの変化に合わせてクラスタリングを更新する．このとき，グラフの全体を調べることなく，変化した部分の周辺のみを再クラスタリングすることで計算時間を短縮する．そのため，複数の変更クエリの処理を通して，クラスタリングが理想的な状態ではなくなる．そこで提案手法では，クエリ列をある程度の長さの周期で分割し，周期の開始時にグラフ全体への幅優先探索を行ってクラスタリングを再計算する．周期は $|V|$（辺数の平方根）程度とする．

このとき，クエリ一回あたりの時間計算量が $\Omega(|V|)$ であれば，変更クエリ $|V|$ 回ごとの幅優先探索にかかる時間コストは定数倍の差に収まる．6.1 節で示すようにこの条件は充足され，$|V|$ 回ごとの再クラスタリングの償却時間計算量は $O(1)$ 時間となる．

5. グラフ変更手法の詳細

提案手法では，グラフの変更に合わせて一部のクラスタのみを変更する．これにより，グラフ変更クエリのたびにクラスタリングの再計算をすることなく変更後の辺集合分割を得る．クラスタリングの変化に合わせて辺集合を統合・分割することで，変更クエリ受理後にも 4.2 節で示した辺探索が可能である．

5.1 辺集合の統合

二つのクラスタ c_i, c_j を c_i に統合し c'_i とするとき，三つの辺集合

(1) c_i の内部辺集合
(2) c_j の内部辺集合
(3) c_i, c_j 間の外部辺集合

を統合し c'_i の内部辺集合とする．その他のクラスタ c_k ($k \notin \{i, j\}$) について，二つの辺集合

(1) c_j, c_k 間の外部辺集合
(2) c_i, c_k 間の外部辺集合

を統合し c'_i, c_k 間の外部辺集合とする．この結果，統合により消滅するクラスタ c_j に関わる内部辺・外部辺は全て新たな辺集合へ移動する．

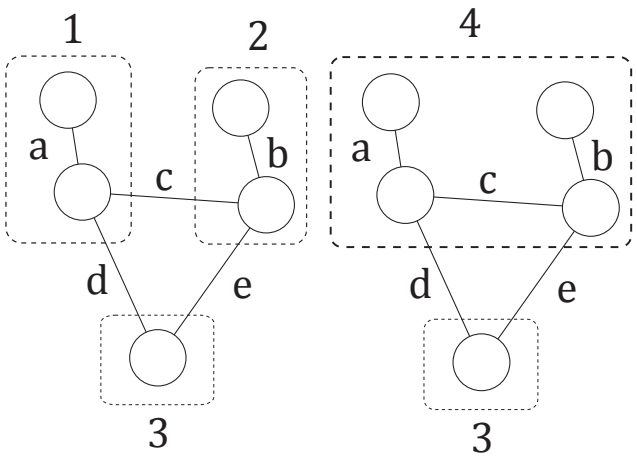
辺集合の統合の例を図 4 に示す．(i) におけるクラスタ 1, 2 を統合すると，(ii) におけるクラスタ 4 になる．両図におけるクラスタ 3 は統合に直接関わらないクラスタで，上記の c_k にあたる．図 4 における辺 a, b, c, d, e の分類を示すと以下のようになる．

(1) a : クラスタ 1 の内部辺
(2) b : クラスタ 2 の内部辺
(3) c : クラスタ 1, 2 間の外部辺
(4) d : クラスタ 1, 3 間の外部辺
(5) e : クラスタ 2, 3 間の外部辺

クラスタ 1, 2 の統合の結果，(ii) においては分類が変化し，以下のようになる．

(1) a, b, c : クラスタ 4 の内部辺
(2) d, e : クラスタ 3, 4 間の外部辺

以上で示したように，クラスタ統合の実装には辺集合を統合する操作が必要となる．二つの辺集合を統合するアルゴリズムがあれば，このアルゴリズムを複数回適用することで三つ以上の辺集合を統合できる．二つの辺集合を統合



(i) クラスタの分割　(ii) クラスタの統合

図 4 クラスタの分割・統合

Algorithm 1 MergeEdgeLists

Require: $es1, es2$ are sorted edge list
Ensure: $es1, es2$ will be empty list, $result$ will be sorted

```
function MergeEdgeLists( es1, es2 )
    i = 0, j = 0
    result = []                                  ▷ [] is a empty list
    while True do
        if i = |es1| then
            result.append(es2)
            break
        else if j = |es2| then
            result.append(es1)
            break
        end if
        if w(es1[i]) ≤ w(es2[j]) then
            result.appned(es1[i])
            i = i + 1
        else
            result+.appned(es2[j])
            j = j + 1
        end if
    end while
    es1 = es2 = []
    return result
end function
```

Algorithm 2 MergeClusters

```
procedure MergeClusters ( i, j )
    inner_i = MergeEdgeLists (innter_i, inner_j)
    inner_i = MergeEdgeLists (inner_i, outer_{i,j})
    for all k ∈ {c | c ∈ AllClusters \ {i,j}} do
        outer_{i,k} = MergeEdgeLists (outer_{i,k}, outer_{j,k})
    end for
end procedure
```

するアルゴリズムの擬似コードを Algorithm 1 に示す．関数 MergeEdgeLists は，$es1, es2$ 内の要素を昇順ソートした列を返す．関数 MergeEdgeLists を用いて二つのクラスタ i, j の統合を実装することができる．アルゴリズムを Algorithm 2 に示す．擬似コード中の $inner_i$ はクラスタ i に対応する内部辺集合を表す．$outer_{i,j}$ はクラスタ i, j 間の外部辺を表す．以降の擬似コードも同様である．

5.2 辺集合の分割

辺集合 c を分割し二つの辺集合 c_i, c_j とするとき，c の内部辺集合を分割し三つの辺集合

(1) c_i の内部辺集合
(2) c_j の内部辺集合
(3) c_i, c_j 間の外部辺集合

とする．その他のクラスタ c_k ($c_k \neq c$) について，c, c_k 間の外部辺集合を分割し，

(1) c_i, c_k 間の外部辺集合
(2) c_j, c_k 間の外部辺集合

の二つの辺集合とする．クラスタ統合はクラスタ分割の逆操作なので，図 4 の (ii) の状態から (i) の状態に変化する．

Algorithm 3 SplitEdgeList

Require: es is soted, f is a function that $f : E \to \{True, False\}$
Ensure: each element $e \in result1$ will satisfy $f(e) = True$ and each element $e \in result2$ will not

```
function SplitEdgeList( es, f )
    result1 = [], result2 = []
    for all e ∈ es do
        if f(e) then
            result1.append(e)
        else
            result2.append(e)
        end if
    end for
    return (result1, result2)
end function
```

Algorithm 4 SpliteCluster

```
procedure SplitCluster ( i, j )
    (inner_i, inner_j) = SplitEdgeList (inner_i, (λe.True ⇔ e is inner edge of cluster i ))
    (inner_j, outer_{i,j}) = SplitEdgeList (inner_i, (λe.True ⇔ e is inner edge of cluster j ))
    for all k ∈ {c | c ∈ AllClusters \ {i,j}} do
        (outer_{i,k}, outer_{j,k} = SplitEdgeList (outer_{i,k}, (λe.True ⇔ e is outer edge between clusters i, k ))
    end for
end procedure
```

クラスタを分割するためには辺集合を分割する操作が必要となる．一つの辺集合を，辺に関する条件に基づいて二つの辺集合に分割するアルゴリズムがあれば，そのアルゴリズムを連続的に適用することで三つ以上の辺集合への分割も可能となる．辺集合と辺に関する条件を表す高階関数を入力とし，分割後の辺集合の組を返すアルゴリズムの擬似コードを Algorithm 3 に示す．関数 SplitEdgeList により辺集合の分割が実現できる．関数 SplitEdgeList を用いてクラスタ i を二つのクラスタ i, j に分割する処理を実装することができる．アルゴリズムの擬似コードを Algorithm 4 に示す．

5.3 辺の削除

削除クエリ $remove(u, v)$ を処理する手順について述べる．$remove(u, v)$ の処理は，MST が辺 (u, v) を含むか否かによって異なる．(u, v) が MST に含まれていないときは，(u, v) が属する辺集合から (u, v) を取り除くのみである．

そうでない場合は，まず MST から (u, v) を取り除いて MST を二つの連結成分に分ける．このとき，(u, v) が内部辺であれば u, v が属するクラスタを分割する．その後，連結成分間を結ぶ辺の内最も軽いものを追加することで再び連結にする．分割した連結成分間を結ぶ辺は，異なる連結成分に属する頂点が属するクラスタ間の外部辺である．従って，追加する辺は外部辺集合を走査することで発見で

Algorithm 5 RemoveEdge

```
1: procedure RemoveEdge ( u, v )
2:     if Not MST.hasEdge(u, v) then
3:         if cluster(u) = cluster(v) then
4:             inner_cluster(u).erase(u, v)
5:         else
6:             outer_cluster(u),cluster(v).erase(u, v)
7:         end if
8:         return
9:     end if
10:    MST.remove_edge((u, v))
11:    if cluster(u) = cluster(v) then
12:        assign new cluster for vertices such that connected from v on
       MST and belongs to cluter(u)
13:        SplitCluster(cluster(u), cluster(v))
14:    end if
15:    c = cluster which has lightest head edge
16:    MST.add_edge(c[0])
17:    c.remove(c[0])
18:    MergeClusters (cluster(u), cluster(v))
19: end procedure
```

きる．アルゴリズムを Algorithm 5 に示す．

5.4 辺の追加

追加クエリ $add(u, v, x)$ を処理する手順について述べる．重み w の辺 (u, v) を MST に加えたグラフを考えると，辺 (u, v) により閉路ができる．この閉路は，MST 上で 2 頂点 u, v 間を結ぶパスの端に，(u, v) を加えたものである．新たな MST を得るために閉路から最も重い辺を取り除く必要があるが，取り除く辺は次の 2 通りに場合分けできる．

(1) (u, v) を取り除く
(2) それ以外の辺を取り除く

これは，(u, v) の重みと u-v パス中で最も重い辺の重みを比較すれば判別できる．

追加クエリによりグラフに追加した辺である (u, v) を閉路から取り除く場合は，(u, v) を対応する辺リストに挿入する．そうでない場合は，u-v パス中で最も重い辺を MST から取り除いて MST を二つの連結成分に分断してから，(u, v) を MST に加える．パスから取り除いた辺は，対応する辺リストへ挿入する．アルゴリズムの擬似コードを Algorithm 6 に示す．

Algorithm 6 AddEdge

```
1: procedure AddEdge ( u, v, x )
2:     modify w to be w((u, v)) = x
3:     e = (p, q) = most heavy edge in u-v path in MST
4:     if w(e) ≤ x then
5:         if cluster(u) = cluster(v) then
6:             inner_cluster(u).insert((u, v))
7:         else
8:             outer_cluster(u),cluster(v).insert((u, v))
9:         end if
10:        return
11:    end if
12:    MST.remove_edge(e)
13:    if cluster(p) = cluster(q) then
14:        assign new cluster for vertices such that connected from v on
       MST and belongs to cluter(u)
15:        SplitCluster (cluster(p), cluster(q))
16:    end if
17:    outer_cluster_p,cluster_q.insert(e)
18:    MST.add_edge((u, v))
19:    MergeClusters (cluster(u), cluster(v))
20: end procedure
```

6. 効率化に関する実験結果と評価

6.1 計算量解析結果と考察

平均的な状態について考察するため，本節では以下の二つの仮定を置いた上で提案手法を解析する．

(1) クラスタの総数は $\lfloor \sqrt{|V|} \rfloor$ 個である
(2) 各クラスタ内の頂点数は $\lfloor \sqrt{|V|} \rfloor$ 個である

表記が煩雑になるのを避けるため，以下の議論では $\lfloor \sqrt{|V|} \rfloor$ を単に $\sqrt{|V|}$ と書く．また，配列を使用する際には動的な表 [4] を用いた可変長配列を用いるとする．

まずは基本的な操作について解析する．次の値のオーダは全て $O(|V|)$ である．

(1) 一つのクラスタの内部辺の本数
(2) 外部辺集合の個数
(3) 一つの外部辺集合が含む辺の本数

Proof. あるクラスタの内部辺の総数は，クラスタ内部から二つの頂点を選ぶ組合せの総数であるため，二項係数 $\binom{\sqrt{|V|}}{2}$ に等しい．$\binom{\sqrt{|V|}}{2} = O(|V|)$ であるので，あるクラスタの内部辺の総数は $O(|V|)$ 本である． □

Proof. 外部辺集合の総数は，異なるクラスタ二つを選ぶ組合せの総数である．クラスタ数は $O(\sqrt{|V|})$ 個であるので，内部辺集合サイズに関する議論と同様に $O(|V|)$ 個となる． □

Proof. ある外部辺集合が含む辺の本数は，対応する二つのクラスタのそれぞれから一つずつ頂点を取り出す組合せの総数である．一つのクラスタから頂点を取り出す選び方は $O(\sqrt{|V|})$ 通りである．それぞれのクラスタから頂点を取り出す操作は独立であるため，二つのクラスタから一つずつ頂点を取り出す選び方はそれぞれのクラスタからの取り出し方の総数の積である従って $O(\sqrt{|V|^2}) = O(|V|)$ 通りである． □

MST への以下の操作は，MST を隣接リストで表現しているとすれば時間計算量 $O(|V|)$ 時間である．

(1) 辺の追加
(2) 辺の削除
(3) ある辺を含む否かを調べる

(4) 2 頂点 u, v 間のパスを求める

次に，5 で定義した関数・手続きを解析する．MergeEdgeLists ($es1, es2$) は $O(|es1|+|es2|)$ 時間で動作する．

Proof. MergeEdgeLists の 12 〜 18 行目の if 文により，両配列の各要素を高々一回ずつ *result* の末尾に挿入する．償却 $O(1)$ 時間の処理を $|es1| + |es2|$ 回実行するので，償却 $O(|es1| + |aes2|)$ 時間となる．

片方の配列が空になったとき，他方の配列を *result* の末尾に連結する．この処理は，どちらのリストが残ったとしても償却 $O(\max(|es1|, |aes2|))$ 時間で実行できる．

以上より， MergeEdgeLists は償却 $O(|es1| + |es2|)$ 時間で動作する． □

また，辺リストの長さは $O(|V|)$ であるため，MergeEdgeLists は償却 $O(|V|)$ 時間である．

MergeClusters (i, j) は償却 $O(|V|\sqrt{|V|})$ 時間で動作する．

Proof. 第 2, 3 行では， MergeEdgeLists により $inner_j$ と $outer_{i,j}$ を $inner_i$ に統合する．これらの処理は，償却 $O(|V|)$ 時間で実行できる．

第 4 〜 6 行の for 文において，k は i, j を除いた全てのクラスタに関してループし，その対象の個数は $O(\sqrt{|V|})$ である．各クラスタについて MergeEdgeLists により $outer_{i,k}$ と $outer_{j,k}$ を統合する．クラスタ数は $O(\sqrt{|V|})$ であるため，償却 $O(|V|)$ 時間の処理を $O(\sqrt{|V|})$ 回実行する．

以上より， MergeClusters は $O(|V|\sqrt{|V|})$ 時間で動作する． □

SplitEdgeList (es, f) は，条件関数 f が定数時間で処理できるならば，償却 $O(|V|)$ 時間で動作する．

Proof. 第 3 〜 9 行の for 文により，e は es の全ての要素に関してループする．各 e について，$f(e)$ の値によって *result1*, *result2* のいずれかの末尾に連結する処理をする．配列末尾への単一要素の追加は償却 $O(1)$ 時間であり，連結を $|es|$ 回実行する．

$|es| = O(|V|)$ であるため， SplitEdgeList は $O(|V|)$ 時間で動作する． □

SplitCluster (i, j) は償却時間計算量 $O(|V|\sqrt{|V|})$ 時間で動作する．

Proof. 所属クラスタに関する条件関数は，頂点番号に対する所属クラスタの情報を配列に記憶しておくことで定数時間で動作するよう実装できる．従って，処理に利用する SplitEdgeList は償却 $O(|V|)$ 時間で動作する．

第 2, 3 行では， SplitEdgeList により $inner_i$ を 2 回処理する．これらの処理は，償却 $O(|V|)$ 時間で実行できる．

第 4 〜 6 行の for 文において，k は i, j を除いた全てのクラスタに関してループし，その対象の個数は $O(\sqrt{|V|})$ である．各クラスタについて SplitEdgeList を実行する．クラスタ数は $O(\sqrt{|V|})$ であるため，償却 $O(|V|)$ 時間の処理を $O(\sqrt{|V|})$ 回実行する．

以上より， SplitCluster は $O(|V|\sqrt{|V|})$ 時間で動作する． □

6.1.1 辺の削除

削除クエリ *remove*(u, v) を処理する関数 RemoveEdge の実装に必要となる操作は以下のものである．

(1) MST がある辺を含むか否かを調べる
(2) u, v について，属するクラスタをそれぞれ調べる
(3) 辺リストから (u, v) を削除する
(4) MST から (u, v) を削除する
(5) MST から辺を一本取り除いたグラフ上で，v と連結かつ所属クラスタが u と等しい頂点を列挙し，所属クラスタを変更する
(6) SplitCluster
(7) リストの先頭にある辺が最も軽い頂点集合を求める
(8) MST に辺を加える
(9) 辺リストの先頭要素を削除する
(10) MergeClusters を実行する

MST から辺を一本取り除いたグラフ上で条件を満たす頂点を列挙する操作は，v から幅優先探索などを行うことで実現できる．

他の操作については，既に計算量を示した．操作の内最も時間がかかるものは SplitCluster の実行と MergeClusters の実行であり，それぞれ償却時間計算量 $O(|V|\sqrt{|V|})$ 時間である．従って，削除クエリの処理にかかる償却時間計算量は $O(|V|\sqrt{|V|})$ 時間である．

6.1.2 辺の追加

追加クエリを *add*(u, v, x) を処理する関数 AddEdge の実装に必要となる操作の内，削除クエリに現れなかったものは以下の操作である．

(1) $w((u, v))$ の値が x となるように w を変更する
(2) MST 上の u-v パス上で最も重い頂点を探す

w を変更する処理は，各 u, v に対する $w((u, v))$ の値を 2 次元配列で保持していれば $O(1)$ 時間で実行できる．MST 上の u-v パスを求める処理は，上で示したように $O(|V|)$ 時間で実行できる．u-v パスが含む辺の本数は最大で $|V| - 1$ であり，そこから最も重い頂点を探す操作は線形探索により $O(|V|)$ 時間で実行できる．従って，上に挙げた操作はいずれも $O(|V|)$ 時間で実行できる．

SplitCluster と MergeClusters の実行は追加クエリの処理にも含まれており，これらの実行に最も時間がかかる．従って，追加クエリにかかる償却時間計算量も削除クエリと同様に $O(|V|\sqrt{|V|})$ 時間である．

表 1　既存手法及び提案手法の実行時間比較（ランダムケース）

手法	時間計算量	実行時間（実測） [sec]
Prim のアルゴリズム	$O(\|E\| \log \|V\|) = O(\|V\|^2 \log \|V\|)$	32.71
Kruskal のアルゴリズム	$O(\|E\| \log \|E\|) = O(\|V\|^2 \log \|V\|)$	787.31
Enhanced Kruskal	$O(\|E\|) = O(\|V\|^2)$	60.41
提案手法	$O(\|V\| \sqrt{\|V\|}) = O(\|V\|^{1.5})$	6.57

6.2　定量的解析結果と考察

提案手法の評価のため，既存手法と提案手法を C++ で実装し，実行時間を計測する実験を行った．実験内容は，1024 頂点のグラフに対しランダムなグラフ変更を 1024 回行う処理を 20 回繰り返し，実行時間を計測するものである．

実験は Windows 7 にインストールした VMWare 上で動作する Ubuntu 15.04 上で実施した．各プログラムはプログラミング問題作成補助ツール Rime [8] を用いて自動実行し，実行時間を計測した．

実験結果を表 1 に示す．表中には比較のために計算量も示した．分かりやすさのため，変形したものも併記している．既存手法の内最も高速に動作したのは Prim のアルゴリズムであるが，提案手法は Prim のアルゴリズムの約 5 倍高速に動作した．

Prim のアルゴリズムと Kruskal のアルゴリズムの計算量が等しいにも関わらず実行時間に大きな差があるが，これは Prim のアルゴリズムにおいて優先順位付きキューに push されない辺が多いためである．一方，Kruskal のアルゴリズムは全ての辺について操作する．

提案手法と同様にソート済み辺リストを扱う Enhanced Kruskal と比較すると $\frac{O(|V|^2)}{O(|V|^{1.5})} = O(\sqrt{|V|})$ より $\sqrt{1024} = 32$ 倍の高速化が期待できるが，実際は 10 倍程度となっている．しかし，アルゴリズム実装がより複雑であるにも関わらず定数倍の差が 3 倍程度に収まっているとも言え，十分に効率的である．

理論解析及び計算機上での実験の結果，提案手法は両観点において既存手法よりも効率的であると言える．これにより，モバイル機器上の分散コンピューティングにおいて，以下のような効果が期待できる．

(1) 通信での電力消費が減少することによる稼働時間の延長
(2) 制御やタスクに関するデータ転送の所要時間減少による処理時間短縮

7. むすび

7.1　結論

本研究では以下のことを行った．

(1) 周辺環境が無線通信に与える影響と分散コンピューティング制御のための通信について述べ，通信コスト最小化の意義を示した
(2) モバイル機器間の無線通信をグラフ変更クエリに定式化した
(3) 最適なネットワークがグラフにおける MST に対応することを示した上で，グラフ変更時の MST の変化が局所的であることに着目し，グラフ変更クエリを高速に処理する手法を開発した
(4) 提案手法の理論解析を行い，既存手法である Prim のアルゴリズムや Kruskal のアルゴリズムより漸近的に高速であることを示した
(5) 提案手法と既存手法を実装し，計算機実験での実測により提案手法が既存手法より 5 〜 120 倍程度高速であることを示した

従って，モバイル機器による分散コンピューティング制御において，通信により発生するコストを最小化する手法を示すことができたと言える．これにより，処理の時間短縮や省電力化が期待できる．

7.2　展望

本研究の展望として以下の点がある．

(1) グラフの形に抽象化して扱ったことで，同様の形式化ができる問題への応用が期待できる
(2) 平均計算量についてのみ分析したため，クラスタリングの変化を分析し最悪ケースでの解析を行いたい
(3) 計算機上での実験がランダムケースのみであったため，性能が悪化するケースを発見し追試することでより詳細な分析を行いたい

参考文献

[1] Shaw, J. A.: Radiometry and the Friis transmission equation, *American Journal of Physics*, Vol. 81, No. 1, pp. 33–37 (2013).
[2] ruuroo: 無線 LAN に対する障害物の影響 - パソコントラブルと自己解決，http://pctrouble.lessismore.cc/network/wirelesslan_obstacle.html，(2016/01/19 閲覧).
[3] Abbas, S., Mosbah, M. and Zemmari, A.: Distributed computation of a spanning tree in a dynamic graph by mobile agents, *Engineering of Intelligent Systems, 2006 IEEE International Conference on*, IEEE, pp. 1–6 (2006).
[4] Cormen, T. H.，Leiserson, C. E.，Rivest, R. L.，Stein, C.，浅野哲夫，岩野和生，梅尾博司，山下雅史，和田幸一：アルゴリズムイントロダクション: 世界標準 MIT 教科書，近代科学社，第 3 版 edition (2012).
[5] Prim, R. C.: Shortest connection networks and some generalizations, *Bell system technical journal*, Vol. 36, No. 6, pp. 1389–1401 (1957).
[6] Kruskal, J. B.: On the shortest spanning subtree of a graph and the traveling salesman problem, *Proceedings of the American Mathematical society*, Vol. 7, No. 1, pp. 48–50 (1956).
[7] 秋葉拓哉：完全制覇・ツリー上でのクエリ処理技法 - (iwi){ 反省します - TopCoder 部，http://topcoder.g.hatena.ne.jp/iwiwi/20111205/1323099376 (2016/01/14 閲覧).
[8] Takahashi, S.: Rime – Rime 1.0 documentation, "http://nya3jp.github.io/rime/, (2016/01/14 閲覧).