Vol.2014-AL-149 No.2
2014/9/12

サイクルグラフ上の経路長を短くする一方通行決定問題に対する $O(n + q \log n)$ アルゴリズム

澤 道彦[†1,a)]

概要：n 頂点の重み付き無向グラフ G と頂点対の集合 $(s_1, t_1), \ldots, (s_q, t_q)$ に対し, G の orientation D で, D 内での s_i から t_i への最短路長の, i に関する和や最大値を最小化するものを探す問題を, それぞれ min-sum, min-max 型の route-enabling graph orientation problem と呼ぶ. 本文では, 特に G が cycle の時に, min-sum, min-max 型共に $O(n + q \log n)$ で解くアルゴリズムを提案する. さらに, そのアルゴリズムは G が cacti の場合の時間複雑度も $O(n^2 + nq \log q)$ に改善する.

An $O(n + q \log n)$ algorithm for the route-enabling graph orientation problem on cycles

Sawa Michihiko[†1,a)]

Abstract: For a given edge weighted undirected graph G with n vertices and a set of pairs of vertices $(s_1, t_1), \ldots, (s_q, t_q)$, the min-sum (resp. min-max) route-enabling graph orientation problem is to find an orientation D of G which minimizes the sum (resp. max.) of the shortest distances from s_i to t_i in D. In this paper, we propose $O(n + q \log n)$ algorithm for the problem on cycles. An $O(n^2 + nq \log n)$ algorithm on cacti is introduced by our algorithm in the straight forward way.

1. はじめに

無向グラフ $G = (V, E)$ に対し, G の各無向辺 $\{u, v\}$ を有向辺 (u, v) 又は (v, u) の一方に向き付けした有向グラフ D を, G の *orientation* と呼ぶ.

無向グラフ G と, その辺に付随する距離 $d : E(G) \to \mathbb{R}_{\geq 0}$ に対し, ある G の orientation D 上での d による最短距離を $d_D : V^2 \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ で表す. 与えられた G, d と s-t ペアの列 $(s_1, t_1), \ldots, (s_q, d_q) \in V^2$ に対し, $\forall i : d_D(s_i, t_i) < \infty$ なる D を求める問題を, *route-enabling (graph) orientation problem* と呼ぶ. 更に,

$$\sum_{1 \leq i \leq q} d_D(s_i, t_i),$$

$$\max_{1 \leq i \leq q} d_D(s_i, t_i)$$

を最小化する D を求める問題を, それぞれ *min-sum*, *min-max* 型の route-enabling (graph) orientation problem と呼ぶ.

表 1 既存の結果

	min-sum	min-max
平面グラフ	強 NP 困難	強 NP 困難
cacti	$O(nq^2)$	$q = 2$ でも NP 困難
cycle	$O(n + q^2)$	$O(n + q^2)$

これらの問題は, 一般には計算困難である事や, 特殊な G に対しては多項式時間で解ける事が知られている. 具体的には, 平面グラフ, cacti, cycle に対して, **表 1** の結果が知られている [1]. しかし, cycle 上の min-sum と min-max, cacti 上 min-sum について知られているアルゴリズムは, 時間複雑度の意味で最良であるとは示されていない.

本論文では, G が cycle の時, min-sum, min-max 型共に $O(n + q \log n)$ で解くアルゴリズムを提案する. また, G が cacti の場合, min-sum 型の問題は, cycle の場合を

†1 現在, 上智大学 理工学研究科
Presently with Graduate School of Science and Technology, Sophia University
a) michihiko@sophia.ac.jp

$O(n)$ 回解く事に帰着される [1]. 従って, これについても $O(n^2 + nq\log n)$ のアルゴリズムが得られる.

2. 準備

以後, G は $1,2,\ldots,n$ を頂点とし, $\{1,2\},\ldots,\{n-1,n\},\{n,1\}$ を辺とする n 頂点のサイクルとする. 簡単の為, $\{s,s+1,\ldots,t-1,t\}$ で

$$\begin{cases}\{s,s+1,\ldots,t-1,t\} & \text{if} \quad s \le t,\\ \{s,s+1,\ldots,n,1,2,\ldots,t\} & \text{if} \quad s > t\end{cases} \tag{1}$$

を表すなどとする. また, s-t ペアは重複が無いものとし, $(s_i,(s_i-t_i) \mod n)$ の辞書順で番号付けしておく.

以下, 準備として, いくつか用語と記法を定義し, 基本的な性質を導く.

定義 1: s から t への 2 通りのパスに含まれる辺を,

$$\begin{aligned}\mathrm{cw}(s,t) &= \{\{s,s+1\},\ldots,\{t-1,t\}\},\\ \mathrm{ccw}(s,t) &= \{\{s,s-1\},\ldots,\{t+1,t\}\}\end{aligned}$$

とおき, その長さを

$$\begin{aligned}d_{\mathrm{cw}}(s,t) &= \sum_{\{u,v\}\in\mathrm{cw}(s,t)} d(u,v),\\ d_{\mathrm{ccw}}(s,t) &= \sum_{\{u,v\}\in\mathrm{ccw}(s,t)} d(u,v)\end{aligned}$$

とおく. ♣

定義 2: 写像 $x : E(G) \to \{\mathrm{cw},\mathrm{ccw}\}$ に対し, G の *orientation* D_x を,

$$\begin{aligned}V(D_x) &= V(G),\\ E(D_x) &= \{f(e) \mid e \in E(G)\}\end{aligned}$$

で定める. 但し,

$$f(\{u,u+1\}) = \begin{cases}(u,u+1) & \textit{if} \quad x(\{u,u+1\}) = \mathrm{cw},\\ (u+1,u) & \textit{o.w.}\end{cases}$$

とする. ♣

G の orientation と $x \in \mathrm{Map}(E(G),\{\mathrm{cw},\mathrm{ccw}\})$ は一対一に対応するので, 以後同一視する.

定義 3: s-t ペアへの割り当て $\mathbf{y} = (y_1,\ldots,y_q) \in \{\mathrm{cw},\mathrm{ccw}\}^q$ が

$$y_i \neq y_j \implies y_i(s_i,t_i) \cap y_j(s_j,t_j) = \emptyset \tag{2}$$

を満たす時, $\mathbf{y}$ は *route-enabling orientation* を誘導すると言い,

$$x(e) = \begin{cases}\mathrm{cw} & \textit{if} \quad \exists i : y_i = \mathrm{cw} \text{ かつ } e \in y_i(s_i,t_i),\\ \mathrm{ccw} & \textit{o.w.}\end{cases}$$

で定まる x, 又は D_x を, $\mathbf{y}$ に誘導された *route-enabling orientation* と呼ぶ. 但し,

$$y_i(s_i,t_i) = \begin{cases}\mathrm{cw}(s_i,t_i) & \textit{if} \quad y_i = \mathrm{cw},\\ \mathrm{ccw}(s_i,t_i) & \textit{if} \quad y_i = \mathrm{ccw}\end{cases}$$

とする. ♣

命題 4: $\mathbf{y}$ が *route-enabling orientation* を誘導する時, $\mathbf{y}$ に誘導された D_x は実際に *route-enabling orientation* である. ♣

証明: x の定め方から, $\forall i, \forall e \in y_i(s_i,t_i) : x(e) = y_i$. 従って, G の s_i-t_i パス $y_i(s_i,t_i)$ は D_x 上でも s_i-t_i パスになる. よって, $\forall i : d_D(s_i,t_i) < \infty$. ■

命題 5: 任意の *route-enabling orientation* $D = D_x$ に対し, 目的値が D_x の目的値以下である *route-enabling orientation* を誘導する割り当て $\mathbf{y}$ が存在する. ♣

証明: $\mathbf{y} = (y_1,\ldots,y_q)$ を

$$y_i = \begin{cases}\mathrm{cw} & \textit{if} \quad \forall e \in \mathrm{cw}(s_i,t_i) : x(e) = \mathrm{cw}\\ & \qquad \text{かつ } d_D(s_i,t_i) = d_{\mathrm{cw}}(s_i,t_i)\\ \mathrm{ccw} & \textit{o.w.}\end{cases}$$

で定めれば, これが誘導する *orientation* D' において, $\forall i : d_D(s_i,t_i) \ge d_{D'}(s_i,t_i)$ となる. ■

命題 6: *route-enabling orientation* を誘導する割り当て $\mathbf{y}$ は $O(n+q)$ 種類. ♣

証明: $\forall i : y_i = \mathrm{cw}$ と, $\forall i : y_i = \mathrm{ccw}$ なる二つの割り当ては明らかに *route-enabling orientation* を誘導する. これらを自明な割り当てと呼ぶ.

自明な割り当てを除けば, $x(\{u-1,u\}) = \mathrm{ccw}$ かつ $x(\{u,u+1\}) = \mathrm{cw}$, すなわち, $(u,u-1),(u,u+1) \in E(D_x)$ なる $u \in V(G)$ が存在する. この注目点 u をひとつ固定し, $(u,u-1),(u,u+1) \in E(D_x)$ なる *route-enabling orientation* を誘導する $\mathbf{y}$ を列挙する事を考える.

$s_i,t_i \neq u$ なる i について, $\{u,u+1\} \in \mathrm{ccw}(s_i,t_i)$ と $\{u-1,u\} \in \mathrm{cw}(s_i,t_i)$ の一方のみが成立する. 従って, 前者の場合は $y_i = \mathrm{cw}$, 後者の場合は $y_i = \mathrm{ccw}$ が必要である.

$t_i = u$ なる i が存在する時, $\{u-1,u\} \in \mathrm{cw}(s_i,t_i)$ かつ $\{u,u+1\} \in \mathrm{ccw}(s_i,t_i)$ であるから, $(u,u-1),(u,u+1) \in E(D_x)$ なる *route-enabling orientation* を誘導するのは不可能である.

$l_u = \min\{i \mid s_i = u\}$, $r_u = \max\{i \mid s_i = u\}$ とする. 番号付けから, $l_u \le i < j \le r_u \implies \mathrm{cw}(s_i,t_i) \cap \mathrm{ccw}(s_j,t_j) \neq \emptyset$ であるから, *route-enabling orientation* を誘導する可能性のある $\mathbf{y}$ の $l_u,\ldots,r_u$ への割り当ては, $k = l_u,\ldots,r_u+1$ に対し,

$$y_i = \begin{cases}\mathrm{ccw} & \textit{if} \quad l_u \le i < k\\ \mathrm{cw} & \textit{if} \quad k \le i \le r_u\end{cases}$$

とする, $r_u - l_u + 2$ 通りのみである.

従って, 注目点 u を動かせば, $\sum_u (r_u - l_u + 2) = O(n+q)$ 種類となる. ■

定義 7: $u \in V(G)$ とし, $l_u = \min\{\, i \mid s_i = u \,\}$, $r_u = \max\{\, i \mid s_i = u \,\}$ とする.

$l_u \le k \le r_u + 1$ なる k に対し, $\mathbf{y}^{u,k}$ を

$$y_i^{u,k} = \begin{cases} \mathrm{cw} & \text{if } s_i, t_i \neq u \text{ かつ } \{\, u, u+1 \,\} \in \mathrm{ccw}(s_i, t_i) \\ \mathrm{ccw} & \text{if } s_i, t_i \neq u \text{ かつ } \{\, u-1, u \,\} \in \mathrm{cw}(s_i, t_i) \\ \mathrm{ccw} & \text{if } s_i = u \text{ かつ } i < k \\ \mathrm{cw} & \text{if } s_i = u \text{ かつ } i \ge k \\ \mathrm{cw} & \text{if } t_i = u \end{cases}$$

で定める. ♣

命題 8: 定義 7 の $\mathbf{y}^{u,k}$ について, 次が成り立つ.
(1) $i \neq k$ ならば $y_i^{u,k} = y_i^{u,k+1}$.
(2) $t_i \neq u$ ならば $y_i^{u-1,r_{u-1}+1} = y_i^{u,l_u}$. ♣

証明: $\mathbf{y}^{u,k}$ の定義から明らか. ■

命題 9: $\mathbf{y}$ に対し, $z_{\mathrm{cw}}, z_{\mathrm{ccw}} \in \mathrm{Map}(E(G), \mathbb{Z}_{\ge 0})$ を

$$z_{\mathrm{cw}}(e) = \#\{\, i \mid e \in y_i(s_i, t_i),\ y_i(s_i, t_i) = \mathrm{cw} \,\},$$
$$z_{\mathrm{ccw}}(e) = \#\{\, i \mid e \in y_i(s_i, t_i),\ y_i(s_i, t_i) = \mathrm{ccw} \,\}$$

で定め, その内積を $z_{\mathrm{cw}} \cdot z_{\mathrm{ccw}} = \sum_e z_{\mathrm{cw}}(e) z_{\mathrm{ccw}}(e)$ で定める. この時, $\mathbf{y}$ が *route-enabling orientation* を誘導するならばその時に限り $z_{\mathrm{cw}} \cdot z_{\mathrm{ccw}} = 0$ となる. ♣

証明: 式 *(2)* から明らか. ■

3. アルゴリズム

3.1 アルゴリズムの概要

アルゴリズムは, 定義 7 による割り当て $\mathbf{y}^{u,k}$ 全てを (u,k) の辞書順に試すものである. これら全てが route-enabling orientation を誘導するとは限らないため, 式 (2) を満たすか判定する必要がある.

提案するアルゴリズムのメインルーチンを Algorithm 1 に示す. メインルーチンで使われているサブルーチン SET, CHECK_FEASIBILITY, CURRENT_OBJVAL については, 3.2, 3.3, 3.4 節で説明する.

3.2 目的関数値の更新

目的関数が min-sum 型の時, y_i を cw から ccw に変えた時の目的関数値の増分は $d_{\mathrm{ccw}}(s_i, t_i) - d_{\mathrm{cw}}(s_i, t_i)$ である. 同様に, y_i を ccw から cw に変えた時の増分は $d_{\mathrm{cw}}(s_i, t_i) - d_{\mathrm{ccw}}(s_i, t_i)$. 従って, 現在の目的関数値を保持しておくだけでよい.

一方, 目的関数が min-max 型の時は, 紐付けされた max-heap を使えばよい.

Algorithm 1 メインルーチン

```
(s_i, t_i) を (s_i, (s_i − t_i) mod n) の辞書順で基数ソート
// 自明な割り当て
result ← min{ Σ_i d_cw(s_i, t_i), Σ_i d_ccw(s_i, t_i) }
// 初期化
INITIALIZE_STRUCTURE()
for i ∈ {1, ..., q} do
    if s_i = 1 or t_i = 1 or {n, 1} ∈ ccw(s_i, t_i) then
        SET_CW(i)
    else
        SET_CCW(i)
    end if
end for
// 非自明な割り当て
for u ∈ {1, ..., n} do
    a ← min{ i | 1 ≤ i ≤ q, s_i = u }
    b ← max{ i | 1 ≤ i ≤ q, s_i = u }
    for i ∈ { i | 1 ≤ i ≤ q, t_i = u } do
        SET_CW(i)
    end for
    if CHECK_FEASIBILITY() then
        result ← min{ result, CURRENT_OBJVAL() }
    end if
    for i ∈ {a, ..., b} do
        SET_CCW(i)
        if CHECK_FEASIBILITY() then
            result ← min{ result, CURRENT_OBJVAL() }
        end if
    end for
end for
Return result
```

3.3 Segment Tree

$z_{\mathrm{cw}}, z_{\mathrm{ccw}}$ は, *segment-tree* と呼ばれるデータ構造を用いて管理する. このアルゴリズムで用いる segment-tree T は, 列 $\mathbf{a} = (a_1, \ldots, a_n)$ を管理するデータ構造で, 次の二つのクエリに対応するものである.

- ADD(T, l, r, v) : $a_l, \ldots, a_r$ を v 増やす.
- GET(T, l, r) : $\sum_{l \le i \le r} a_i$ を返す.

簡単の為, $n = 2^k$ とする. この時, segment-tree は, 各節点に区間 $[l, r] \subset \{\, 1, \ldots, n \,\}$ を対応させた, 葉が $n = 2^k$ 個の完全二分木である. segment-tree の根には $[1, n]$ を対応させ, $[l, r]$ の区間に対応する節点の二つの子には, それぞれ $[l, \lfloor (l+r)/2 \rfloor], [\lceil (l+r)/2 \rceil, r]$ を対応させる.

また, $[l, r]$ に対応する節点には, $[l, r]$ を全て含み, 親に対応する区間を全ては含まないような区間に対する ADD クエリの和と, 親に対応する区間を全ては含まないような区間に対する ADD クエリの, $[l, r]$ との共通部分での和を保持する. これらの値は Algorithm 2 のように更新でき, GET クエリに利用出来る.

3.4 feasibility の判定

feasibility の判定には, 命題 9 を用いる. y_i を cw から ccw へ変えた時, z_{cw} は $z'_{\mathrm{cw}} = z_{\mathrm{cw}} - \mathbf{1}_{\mathrm{cw}(s_i, t_i)}$ に, z_{ccw} は $z'_{\mathrm{ccw}} = z_{\mathrm{ccw}} + \mathbf{1}_{\mathrm{ccw}(s_i, t_i)}$ に変化する. 但し,

Algorithm 2 Segment Tree

procedure ADD(T, l, r, v)
  if $[l, r] \cap T.interval = \emptyset$ **then**
    return
  end if
  $T.sum \leftarrow T.sum + v * \#(T.interval \cap [l, r])$
  if $[l, r] \supset T.interval$ **then**
    $T.added \leftarrow T.added + v$
  else
    ADD($T.left, l, r, v$)
    ADD($T.right, l, r, v$)
  end if
end procedure
procedure GET(T, l, r)
  if $[l, r] \cap T.interval = \emptyset$ **then**
    return 0
  else if $[l, r] \supset T.interval$ **then**
    return $T.sum$
  else
    return GET($T.left, l, r$) + GET($T.right, l, r$) + $T.added *$ $\#(T.interval \cap [l, r])$
  end if
end procedure

Algorithm 3 feasibility の更新

procedure SET_CCW(i)
  if $y_i =$ ccw **then**
    return
  end if
  $dot \leftarrow dot +$ GET($T_{\mathrm{cw}}, \mathrm{ccw}(s_i, t_i)$) − GET($T_{\mathrm{ccw}}, \mathrm{cw}(s_i, t_i)$)
  ADD($T_{\mathrm{ccw}}, \mathrm{ccw}(s_i, t_i), +1$)
  ADD($T_{\mathrm{cw}}, \mathrm{cw}(s_i, t_i), -1$)
  $y_i \leftarrow$ ccw
end procedure
procedure SET_CW(i)
  if $y_i =$ cw **then**
    return
  end if
  $dot \leftarrow dot +$ GET($T_{\mathrm{ccw}}, \mathrm{cw}(s_i, t_i)$) − GET($T_{\mathrm{cw}}, \mathrm{ccw}(s_i, t_i)$)
  ADD($T_{\mathrm{cw}}, \mathrm{cw}(s_i, t_i), +1$)
  ADD($T_{\mathrm{ccw}}, \mathrm{ccw}(s_i, t_i), -1$)
  $y_i \leftarrow$ cw
end procedure
procedure CHECK_FEASIBILITY
  return $dot = 0$
end procedure

$$\mathbf{1}_A(e) = \begin{cases} 1 & \text{if} \quad e \in A, \\ 0 & \text{o.w.} \end{cases}$$

とする. 従って,

$$\begin{aligned} (z'_{\mathrm{cw}} \cdot z'_{\mathrm{ccw}}) &- (z_{\mathrm{cw}} \cdot z_{\mathrm{ccw}}) \\ &= (z_{\mathrm{cw}} - \mathbf{1}_{\mathrm{cw}(s_i,t_i)}) \cdot (z_{\mathrm{ccw}} + \mathbf{1}_{\mathrm{ccw}(s_i,t_i)}) \\ &\quad -(z_{\mathrm{cw}} \cdot z_{\mathrm{ccw}}) \\ &= (z_{\mathrm{cw}} \cdot z_{\mathrm{ccw}}) - (\mathbf{1}_{\mathrm{ccw}(s_i,t_i)} \cdot \mathbf{1}_{\mathrm{cw}(s_i,t_i)}) \\ &\quad +(z_{\mathrm{cw}} \cdot \mathbf{1}_{\mathrm{ccw}(s_i,t_i)}) - (z_{\mathrm{ccw}} \cdot \mathbf{1}_{\mathrm{cw}(s_i,t_i)}) \\ &\quad -(z_{\mathrm{cw}} \cdot z_{\mathrm{ccw}}) \\ &= (z_{\mathrm{cw}} \cdot \mathbf{1}_{\mathrm{ccw}(s_i,t_i)}) - (z_{\mathrm{ccw}} \cdot \mathbf{1}_{\mathrm{cw}(s_i,t_i)}) \\ &= \sum_{e \in \mathrm{ccw}(s_i,t_i)} z_{\mathrm{cw}}(e) - \sum_{e \in \mathrm{cw}(s_i,t_i)} z_{\mathrm{ccw}}(e) \qquad (3) \end{aligned}$$

である. 逆に, y_i を ccw から cw へ変える時も同様である. z_{cw}, z_{ccw} に対応する segment-tree を, それぞれ T_{cw}, T_{ccw} とし, これらに対する ADD, GET クエリを, 式 (1) と同様に拡張しておく. 式 (3) を用いれば, Algorithm 3 で示すように, $\mathbf{y}$ の更新に対応出来る.

3.5 時間複雑度の評価

Algorithm 1 の時間複雑度を評価する. Algorithm 3 の SET_CW, SET_CCW は, 定数回の segment-tree の操作であるから, 時間複雑度は $O(\log n)$ である. 目的値の更新については, min-sum 型の時は $O(1)$, min-max 型の時は $O(\log q)$ である. また, 目的値の取得は, min-sum 型, min-max 型共に $O(1)$ である. 定義 7 の $\mathbf{y}^{u,k}$ は, $O(n+q)$ 種類であり, 命題 8 から, y_i の更新は $O(q)$ 回である. 以上をあわせて, Algorithm 1 の時間複雑度は $O(n+q(\log n+\log q))$ であるが, s-t ペアに重複は無いから, $q = O(n^2)$. 従って, Algorithm 1 の時間複雑度は $O(n + q \log n)$ となる.

謝辞 興味深い問題を教えていただき, また, 議論にお付き合いいただいた, 上智大学の宮本裕一郎准教授に深く感謝したい.

参考文献

[1] Ito, T., Miyamoto, Y., Ono, H., Tamaki, H. and Uehara, R.: Route-Enabling Graph Orientation Problems, *Algorithmica*, Vol. 65, No. 2, pp. 317–338 (2013).