A-032

フローネットワークの脆弱性を判定するアルゴリズムの高速化
Faster algorithm for deciding the vulnerability of flow network

阿部 和樹[1)]　山田 敏規[1)]
Kazuki Abe　Toshinori Yamada

1 はじめに

ブライスのパラドックス (Braess paradox) とは，移動時間の短縮を目的としてネットワークの中に新たな道を作ったにもかかわらず，逆に移動時間が増加する場合があるという交通工学におけるパラドックスである．これは，道を取り去ることで移動時間が短縮される場合があることと等価である．

プライスのパラドックスは何十年にもわたって研究されてきた．とりわけ，Roughgarden [2] は有向マルチグラフ，辺の遅延関数，フローの総量からなるネットが与えられたとき，ネットがブライスのパラドックスに陥るか否かを判定する問題が NP 困難であることを証明した．そこで Roughgarden [2] は，与えられた有向マルチグラフがブライスのパラドックスに陥るような辺の遅延関数とフローの総量を持つ (この性質を脆弱性と呼ぶ) か否かを判定する問題について研究することを提案した．Cenciarelli ら [1] は，与えられた有向マルチグラフ $G = (V, E)$ が脆弱であるか否かを $O(|V| \cdot |E|^2)$ 時間で判定するアルゴリズムを提案した．

小文では，Cenciarelli ら [1] のアルゴリズムを改良し，与えられた有向マルチグラフ $G = (V, E)$ が脆弱であるか否かを $O(|E|^2)$ 時間で判定するアルゴリズムを提案する．

2 ブライスのパラドックス

本節ではブライスのパラドックスを考える上での用語・定義を述べる．これらの定義は Roughgarden [2] に従っている．

有向マルチグラフ $G = (V, E)$ は頂点の集合 V と辺の集合 E からなり，すべての辺 $e \in E$ は頂点対 (u, v) $(u, v \in V)$ と関連づいている．パスは，すべての $i < k$ において e_i が (u_i, u_{i+1}) と関連づいている頂点と辺の交互列 $u_1 e_1 u_2 \cdots u_{k-1} e_{k-1} u_k$ である．このとき，u_1, u_k を端点，$u_2, \ldots, u_{k-1}$ を内部点と呼ぶ．パス p の端点のみに関心がある場合，$u_1 \overset{p}{\rightsquigarrow} u_k$ と書き，$u_1 u_k$-パスと呼ぶ．パスの名前 p を無視するときは $u_1 \rightsquigarrow u_k$ と書く．$u_1 \overset{p}{\rightsquigarrow} u_2$ と $u_2 \overset{q}{\rightsquigarrow} u_3$ が与えられたとき，パス $u_1 \overset{p,q}{\rightsquigarrow} u_3 = u_1 \overset{p}{\rightsquigarrow} u_2 \overset{q}{\rightsquigarrow} u_3$ を p, q で表す．

有向マルチグラフ G，始点 s，終点 t の組 (G, s, t) をネットと呼ぶ：これを単に G とも書く．2 回以上現れる頂点や辺がない st-パスを単純 st-パスと呼ぶ．G 上の st-パスの集合を $P(G)$，単純 st-パスの集合を $SP(G)$ で表す．各頂点を少なくとも 1 本の st-パスが通るとき，G は st-連結であるという．

ネット (G, s, t) のフローは関数 $\varphi : SP(G) \to \mathbb{R}^+$ である．ここで，$\mathbb{R}^+$ は非負実数の集合である．フロー φ の値は $\sum_{p \in SP(G)} \varphi(p)$ である．任意の $e \in E$ に対して $\varphi(e) = \sum_{p \in SP(G): e \in p} \varphi(p)$ である．遅延関数 $l_e : \mathbb{R}^+ \to \mathbb{R}^+$ は各辺 e に辺を流れるフローの関数として遅延を与える：通常，l_e は連続かつ非減少であるとする．フロー φ の下での st-パス $p \in SP(G)$ の遅延は $l_p(\varphi) = \sum_{e \in p} l_e(\varphi(e))$ として定義される．H が G の部分グラフであるとき，遅延関数 l の H の辺への制限を $l|_H$ で表す．

ネット (G, s, t)，実数 $r \in \mathbb{R}^+$，G 上の遅延関数 l の組 $((G, s, t), r, l)$ または単に (G, r, l) をインスタンスと呼ぶ．値が r であるフロー φ は (G, r, l) に対して実行可能であると言われる：明らかに，任意のインスタンスに対して実行可能であるフローが存在する．

$\varphi(p) > 0$ である任意の st-パス $p, q \in SP(G)$ に対して $l_p(\varphi) \le l_q(\varphi)$ を満たすとき，(G, r, l) に対して実行可能であるフロー φ はウォードロップ均衡にある (もしくはウォードロップフローである) と言われる．このとき，$\varphi(p), \varphi(q) > 0$ であるならば $l_p(\varphi) = l_q(\varphi)$ であることに注意されたい．この値を $L((G, s, t), r, l)$ もしくは単に $L(G, r, l)$ で表す．ここで，$r = 0$ であるとき，$L(G, r, l) = 0$ と定義する．

インスタンス $((G, s, t), r, l)$ は，$L((H, s, t), r, l|_H) < L((G, s, t), r, l)$ を満たす s, t を含む G の部分グラフ H が存在するならば，脆弱であると言われる．また，ネット (G, s, t) は，$((G, s, t), r, l)$ が脆弱であるような実数 $r \in \mathbb{R}^+$，遅延関数 l が存在するとき，脆弱であると言われる．

与えられたインスタンスが脆弱であるか否かを判定する問題は NP 困難であることが知られている [2]．一方，Cenciarelli ら [1] は与えられたネット $G = (V, E)$ が脆弱であるか否かを判定する問題を解く $O(|V| \cdot |E|^2)$ 時間アルゴリズムを提案した．次の節ではこのアルゴリズムを紹介する．

3 従来のアルゴリズム

Cenciarelli ら [1] が考案したアルゴリズムを，関数を用いた擬似コードで述べる．まず，部分グラフ同型と，st–埋め込みの定義を述べる．

H から G への部分グラフ同型は，単射な写像 (ϕ, ψ) の組であり，それぞれ V_H から V_G へ，E_H から G の単純パスへの，各 $e = x \to y \in E_H$ に対して $\psi(e) = \phi(x) \rightsquigarrow \phi(y)$ となるような写像の組である．このような ψ の像に含まれるすべてのパスが端点まで対でノード非共有である場合，同型をノード非共有と呼ぶ．

図 1 のホイートストンネットワーク W(Wheatstone network) のネット (G, s', t') への st–埋め込みとは，W から G へのノード非共有部分グラフ同型 (ϕ, ψ) であり，次のようなものである．s' から $\phi(s)$ まで，および $\phi(t)$ から t' までのノード非共有の単純パスで，ψ の像に含まれるすべてのパスについて，その端点までがノード非共有な対になって (空の可能性もあるが) 存在する．

次に，アルゴリズムで用いられる定理を述べる．

定理 3.1　(G, s, t) を st–連結なネットとする．G が脆弱である必要十分条件は，G が図 1 の st–埋め込みを含んで

1) 埼玉大学 大学院理工学研究科. Graduate School of Science and Technology, Saitama University

いることである.

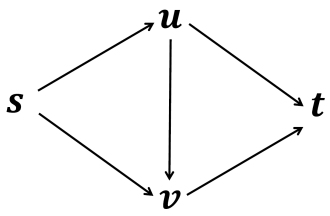

図 1　ホイートストンネットワーク W

図 1 を W と呼ぶことにする.

このアルゴリズムでは, 定理 3.1 を用いて, G の閉路が W の st–埋め込みを含んでいるか, G から閉路がなくなるまで処理を行うことで脆弱性を判定する。

Algorithm 1 脆弱性の判定

Input: 有向マルチグラフ $G = (V, E)$, 始点 s と終点 t
function: $isVulnerable(G, s, t)$
1: $G = makeSTConnected(G, s, t)$
2: **while** G に閉路がある間 **do**
3: $\langle C, \varepsilon^* \rangle = s\text{-}minimalCycle(G, s, t)$
4: **repeat**
5: $C' = C$; $Vuln = false$
6: $E_n = enrtyNodes(G, C, s)$
7: $E_x = exitNodes(G, C, t)$
8: **if** $E_n = \{\varepsilon\}$ **then**
9: $D = \{\varepsilon$ に入る C の辺 $\}$
10: **else if** $E_x = \{\xi\}$ **then**
11: $D = \{\xi$ から出る C の辺 $\}$
12: **else if** $isSplittable(C, E_n, E_x, G)$ **then**
13: $\langle Vuln, C, D \rangle = splittableAnalysis(C, E_n, E_x, G)$
14: **else**
15: $\langle Vuln, C \rangle = nonSplittableAnalysis(C, E_n, E_x, G)$
16: **end if**
17: **until** $C \neq C'$
18: **if** $Vuln$ **then**
19: **return** *true*
20: **else**
21: $G = makeSTConnected((V, E \backslash D), s, t)$
22: **end if**
23: **end while**
24: **if** G が直並列グラフ **then**
25: **return** *false*
26: **else**
27: **return** *true*
28: **end if**

このアルゴリズムは, 有向マルチグラフ G が脆弱であるとき, すなわちブライスのパラドックスを発生させる可能性があるときに *true* を出力し, そうでないときに $false$ を出力する.

while ループ (2–23 行目) では, 各反復で G の辺の数は少なくとも 1 減少するので, このループは最大でも $O(|E|)$ 回の繰り返しで終了する. また, 4–17 行目のループでは各反復で閉路 C は少なくとも距離 1 は終点 t に近づくので, このループは最大でも $O(|V|)$ 回の繰り返しで終了する. 以上のことから, 1 行目と 21 行目の計算では $O(E)$, while ループ (2–23 行目) で $O(|E|)$, 3 行目の計算で $O(|V| * |E|)$, 4–17 行目のループで $O(|V|)$, そのループの中の閉路の計算で $O(|E|)$, そして 24 行目は線形時間の時間計算量 [3] であることから, while ループの処理に一番時間がかかり, アルゴリズム全体では $O(|V| \cdot |E|^2)$ となる.

4　改良手法の提案

従来のアルゴリズムを $O(|E|^2)$ に改良することを目指して考察を行った. 改良する点は 2 ヶ所ある. アルゴリズム 1 の 3 行目と, 4–17 行目の繰り返しである. 両者において改良手法を提案する.

4.1　3 行目の *s-minimalCycle* の改良案

まず, 3 行目の $O(|V| * |E|)$ である *s-minimalCycle* を, 強連結成分分解を用いることで $O(|E|)$ に改良する. 強連結成分とは, 有向グラフにおいて互いに行き来が可能な頂点の集合のことであり, 強連結成分分解は有向グラフを強連結成分でグループ分けをすることである. 改良後のアルゴリズムを擬似コードで述べる.

Algorithm 2 *s-minimalCycle* の改良

Input: 有向マルチグラフ $G = (V, E)$, 始点 s と終点 t
1: s から深さ優先探索を行い, 進めなくなった頂点から順に番号付けをする
2: **repeat**
3: 有向辺の向きを入力と逆にし, 番号が 1 番大きい頂点から深さ優先探索を行う
4: **until** 1 番小さい番号まで探索を終えるまで
5: s から幅優先探索を行い, s からすべての頂点への距離を求める
6: s から最も近い 2 以上の強連結成分において, s から最も近い頂点から深さ優先探索を行い閉路を求める
7: $C = \{s$ から最も近い 2 以上の強連結成分における閉路 $\}$
8: $\varepsilon^* = \{C$ 内の頂点で, かつ最も s からの距離が短い頂点 $\}$
9: **return** $\langle C, \varepsilon^* \rangle$

4.2　4–17 行目の繰り返しの改良案

次に, 4–17 行目の繰り返しを改良する. この繰り返しは, 新たな s–最小閉路 C' が存在する際に発生する. s–最小閉路とは, G のすべての閉路の中で始点 s からの距離が一番短い閉路のことである. 従来のアルゴリズムでは, この s–最小閉路のうち最も t に近いもの, すなわち一番大きい閉路を $O(|V|)$ で求めていたが, これを適切な頂点を閉路に加えることで一度の繰り返しのみで見つけるように改良した.

5　まとめと今後の課題

Cenciarelli らによる有向マルチグラフの脆弱性を判定する $O(|V| \cdot |E|^2)$ の多項式時間アルゴリズムを $O(|E|^2)$ へ改良した. 現実世界のネットワークへの応用が可能かどうかを考えることが今後の課題である.

参考文献

[1] P. Cenciarelli, I. Salvo, and D. Gorla. A polynomial-time algorithm for detecting the possibility of braess paradox in directed graphs. *Algorithmica*, 81:1535 – 1560, 2019.
[2] T. Roughgarden. On the severity of braess's paradox: designing networks for selfish users is hard. *Journal of Computer and System Sciences*, 72(5):922 – 953, 2006.
[3] B. Schoenmakers. A new algorithm for the recognition of series parallel graphs. *Technical report*, 1995.