G-01

モバイルエージェントによる自己安定グラフ探索

Self-Stabilizing Graph Search by a mobile agent

原 悠樹 †　首藤 裕一 †　角川 裕次 †　増澤 利光 †
Yuki Hara　Yuichi Sudo　Hirotsugu Kakugawa　Toshimitsu Masuzawa

1. はじめに

モバイルエージェント（以下，**エージェント**）とは，ネットワーク中の計算機を移動しながら，自律的に動作するソフトウェアである．

グラフ探索問題（以下，**探索問題**）とは，エージェントシステムにおける最も基本的な問題で，エージェントにネットワーク中の全てのノードとリンクを訪問させることである．グラフ探索により，ネットワーク・トポロジの認識やネットワーク中でのデータ検索が可能である．また，ノードやリンクの故障の検知が可能である．[1, 2, 3, 4, 5]

ロータールータは，有向グラフ上を移動するエージェントがグラフ探索を繰り返し行うためのアルゴリズムのひとつである．ノードvの出次数を$\delta(v)$とすると，ノードv上に存在するエージェントは，vの外向辺をポート番号ρ_v:$\{1,2,\dots,\delta(v)\}$で区別することが可能である．各ノードはエージェントが次に訪問する辺のポート番号をノードの変数π_v（以降，ポインタとよぶ）で保持する.エージェントがノードvに訪れると，ポインタπ_vが指定する辺にエージェントを送り出す. その後，π_vの値を次のポート番号に変更する$\left(\pi_v \leftarrow \left(\pi_v \bmod \delta(v)\right)+1\right)$.

ロータールータでは，エージェントの移動経路は，エージェントの現在地と各ノードのポインタの値によってのみ決定される．以降では，エージェントの現在地と全ノードのポインタの値の組をグラフ状況と呼ぶ．あるグラフ状況において，その後のエージェントの移動経路がグラフのオイラー閉路となるとき，グラフ上に**オイラー閉路が生成された**という．オイラー閉路とは，グラフ上のすべての辺を一度だけ通る閉路のことである．

任意の双方向有向グラフ上で，ロータールータの実行は，探索開始から$O(mD)$回の移動でオイラー閉路が生成されることが証明されている[6]. ここに，双方向グラフとは，有向辺(u,v)がある場合に必ず逆向きの有向辺(v,u)が存在する有向グラフのことであり，mはグラフ中の有向辺の数，Dはグラフの直径である．一度オイラー閉路が生成されると，エージェントは同じオイラー閉路に沿ってグラフ探索を繰り返し行う. また，オイラー閉路が生成されている状況でグラフ上の 1 組の隣接頂点u,v間の有向辺(u,v)および(v,u)が削除された場合，その後の$O(\gamma m)$回のエージェントの移動で新たなオイラー閉路が生成されることが証明されている[7]. ここでγは，辺の削除以前のグラフ上における，削除した辺を一度だけ通る閉路のうち最小の閉路の長さのことである．

本稿ではロータールータモデルを改良することによって，辺の削除からの復旧に要する$O(\gamma m)$時間を改善する手法を提案する．提案手法では，各ノードがエージェントが訪れたポート番号の順序を記憶することによって，$O(\gamma m)$時間から，$O(m)$時間に復旧時間の短縮を実現する．

既存研究でのロータールータと，本研究で提案するアルゴリズムとの比較を表 1 に示す．提案アルゴリズムでは，エージェントとノードの記憶容量を既存手法より多く使用することによって 1 リンクの削除からの復旧時間を$O(\gamma m)$から$O(m)$に改善する．具体的には，既存手法ではエージェント記憶領域とノード記憶領域の空間計算量はそれぞれ0, $O(\log\delta(v))$であるが，提案手法ではそれぞれ$O(\log N)$, $O(\delta(v))$となる．改善にはノード数の上界Nの情報と根ノードの存在が必要である．

本稿の構成は以下の通りである．2 節では，本稿で用いる用語の定義を行う．3 節では，1 リンク削除からの復旧時間を改善したアルゴリズムを提案する．3.1 節では，提案アルゴリズムの概要の説明を行い，3.2,3.3 節では，提案アルゴリズムで新たに使用する変数の定義，3.4 節では，提案アルゴリズムの説明を行う．4 節では，本稿の結果をまとめる．

表 1：本研究と既存研究の比較

	既存研究[1]	提案手法
任意の初期状態からの収束時間	$O(mD)$	$O(mD)$
1 リンク削除からの復旧時間	$O(\gamma m)$	$O(m)$
エージェントの記憶領域	0	$O(\log N)$
ノードの記憶領域	$O(\log\delta(v))$	$O(\delta(v))$
Nの知識の有無	なし	あり
根ノードの有無	なし	あり

2. 諸定義

2.1 ネットワークモデル

任意の単純連結無向グラフ$G=(V,E)$を考える．ここでVはn個ノードの集合であり，Eは無向辺の集合である．

双方向有向グラフ$\vec{G}=(V,\vec{E})$は，Gの各辺を向きの異なる 2 本の有向辺に置き換えることによって生成されたグラフである．すなわち，$\vec{E}$は弧の集合であり，$\vec{E}=\{(u,v),(v,u):(u,v)\in E\}$である．本稿では任意の双方向有向グラフ$\vec{G}$に対する単一エージェントによるグラフの探索を考える．

Gにおいて各ノードvに接続する無向辺の集合$\{\{u,v\}\in E|u\in V\}$をI_vで表す．同様に，$\vec{G}$において各ノードvに接続する内向辺の集合$\{(u,v)|(u,v)\in\vec{E}\}$および外向辺の集合$\{(v,u)|(v,u)\in\vec{E}\}$をそれぞれ$I_v^-$および$I_v^+$ で表す．また，各ノードの次数$|I_v|$を$\delta(v)$で表す．ノードvにおいてI_v^-およびI_v^+の各辺にはポート番号と呼ばれる局所的なラベル$p_v^-:I_v^-\mapsto\{1,2,\dots,\delta(v)\}$および$p_v^+:I_v^+\mapsto\{1,2,\dots,\delta(v)\}$が存在する．異なる内向辺，外向辺は$v$においてそれぞれ異なるポート番号を持つ．また，同一の無向辺$\{u,v\}$に対応する内向辺(u,v)および外向辺(v,u)はノードvにおいて同一のポート番号が割り当てられる．すなわち，任意の$\{u,v\}\in E$

†大阪大学大学院情報科学研究科，Graduate School of Information Science and Technology, Osaka University

について$p_v^-((u,v)) = p_v^+((v,u))$である．以下では，$p_v^-((u,v))\left(= p_v^+((v,u))\right)$を$p_v(u)$と表す．

2.2 エージェントモデル

本稿では単一のエージェントによる任意の双方向有向グラフ$\vec{G}$の永続探索（グラフ探索を繰り返し行うこと）を考える．エージェントは常にグラフ上のひとつのノードに存在する有限状態機械であり，1 ステップごとにグラフ上の辺をひとつ通って移動する．また，各ノードは**白板**と呼ばれるメモリを保有する．また，エージェントは，事前知識としてノード数のある上界$N(\geq n)$を保持しているものとする．アルゴリズムはエージェントの状態の集合，白板の状態の集合，各ステップにおいてエージェントが行う動作を規定する．ステップ$t(\geq 0)$においてエージェントが存在するノードをv^tとする．各ステップ$t(\geq 0)$においてエージェントは，

- 現在のエージェントの状態
- ノードv^tの白板の内容
- ノードv^tの次数$\delta(v^t)$
- 前ステップにおいてv^tに移動するために用いた辺のv_tにおけるポート番号$p_{in} = p_{v^t}(v^{t-1})$

を入力として，与えられたアルゴリズムに従って，

- エージェントの状態を更新し，
- ノードv^tの白板を書き換え，
- 移動先ポート番号$p_{out} \in \{1,2,\dots,\delta(v^t)\}$を決定し，
- p_{out}に対応する外向辺を経由して隣接ノードに移動する．

なお，初期ステップ（ステップ 0）におけるp_{in}は，$\{1,2,\dots,\delta(v^t)\}$中の任意の値であるとする．また，本稿では$V$中にただひとつの根ノード$r$が存在することを仮定する．アルゴリズムは根$r$以外の通常ノードと根$r$に異なる状態空間を与えることができる．

2.3 永続探索問題

グラフのノード数がnであるとき，各ノードを$(v_0, v_1, \dots, v_{n-1})$と表す．グラフのステップ$t$における状況$C$は，

- エージェントの状態$mode$
- 各ノードの白板の状態$(w_0, w_1, \dots, w_{n-1})$
- エージェントの位置$v^t \in V$（エージェントが存在するノードv^t）
- v^tへの移動の際に用いた内向辺のポート番号p_{in}

の 4 つで構成される．アルゴリズムが与えられると状況の集合は一意に決まる．アルゴリズムAのすべての状況の集合を$\mathcal{C}_A$で表す．状況$C \in \mathcal{C}_A$においてエージェントが 1 ステップの動作を行うことでグラフの状況がC'に遷移するとき，$C \to C'$と書く．初期状況$C_o \in \mathcal{C}_A$が与えられたときのアルゴリズムAの実行$\xi_A(C_0)$を次式で定義される状況の無限系列とする．

$$\xi_A(C_0) = C_0, C_1, C_2, \dots, s.t. \forall i \geq 0; C_i \to C_{i+1}$$

$\xi_A(C)$におけるエージェントの移動経路（位置の系列）がグラフ$\vec{G}$上のあるオイラー閉路の無限回の周回となっているとき，状況Cにおいて**オイラー閉路が生成されている**という．

[定義 1]

下記を満たす状況の集合$\mathcal{C} \subseteq \mathcal{C}_A$が存在するとき，アルゴリズム A は永続探索問題を解く自己安定アルゴリズムであるという．

- 任意の状況$C_0 \in C_A$についてアルゴリズムAの実行$\xi_A(C_0) = C_0, C_1, \dots$が$\mathcal{C}$に属す状況を含む．すなわち，ある$i \geq 0$について$C_i \in \mathcal{C}$
- $\mathcal{C}$に属する状況から，$\mathcal{C}$に属さない状況には遷移しない．すなわち，$\forall C \in \mathcal{C}[(C \to C') \Longrightarrow C' \in \mathcal{C}]$が生成される．
- 任意の状況$C \in \mathcal{C}$においてオイラー閉路が生成されている．

永続探索問題を解く任意の自己安定アルゴリズムAについて，定義 1 における$\mathcal{C}$に含まれる状況を，アルゴリズムAの正当な状況と呼ぶ．

[定義 2]

永続探索問題を解く自己安定アルゴリズムAについて，任意の初期状態から開始するAの実行がiステップ以内に正当な状況に到達するならば，Aの収束時間は高々iであるという．

2.4 1 組の辺の削除からの復旧時間

本稿では，永続探索アルゴリズムの（任意の状況からの）収束時間に加え，正当な状況において任意の 1 組の辺$(u,v),(v,u)$を削除した際，再びオイラー閉路を生成するのに要する復旧時間を考える．

いま，グラフ$\vec{G}$から辺(u,v)および(v,u)が削除されてできたグラフを$\vec{G'}$とする．ただし，辺$(u,v),(v,u)$の削除によってグラフ$\vec{G}$の連結性は失われないものとする．以降では，辺削除に伴うグラフの変化により，元の状況Cがどのように変化するかを定義する．辺の削除に伴い，ノードuおよびvにおいてポート番号の再割り当ておよび白板の内容変更がアルゴリズムによって即座に行われる．ノードu,vのポート番号の再割り当てはそれぞれ$\alpha_u: \{1,2,\dots,\delta(u)\} - p_u(v) \to \{1,2,\dots,\delta(u)-1\}$および$\alpha_v: \{1,2,\dots,\delta(v)\} - p_v(u) \to \{1,2,\dots,\delta(v)-1\}$によって行われるものとする．すなわち，ノード$u$(resp.$v$)の各接続辺$(u,w),(w,u)$$\left(\text{resp.}(v,w'),(w',v)\right)$に対し，グラフ$\vec{G'}$においてはラベル$\alpha_u(p_u(w))$$\left(\text{resp.}\,\alpha_v(p_v(w'))\right)$が割り当てられるものとする．2.2 節でアルゴリズムはエージェントの状態空間，白板の状態空間，各ステップのエージェントの動作規則を与えるものと定義したが，アルゴリズムはさらに，グラフから辺(u,v)および(v,u)が削除されたときに，消失辺$(u,v),(v,u)$およびα_u, α_vの情報をもとにノードuおよびノードvの白板の内容を書き換える規則を与えるものとする．

削除される 1 組の辺$(u,v),(v,u)$およびα_u, α_v，アルゴリズムA，グラフの状況Cが与えられると，$(u,v),(v,u)$を削除した結果生じる新たな状況C'が一意に決まる．このようなC'を$R_{u,v,\alpha_u,\alpha_v,A}(C)$と表す．アルゴリズムの 1 組の辺の削除からの復旧時間を以下で定義する．

[定義 3]

アルゴリズムAを，永続探索問題を解く任意の自己安定アルゴリズムとする．また，u,vをグラフGにおいて隣接する任意の 2 ノードとし，α_u, α_vをそれぞれ辺$(u,v),(v,u)$を削除したときのノードu,vにおける任意のポート再割り当

て関数 $\alpha_u: \{1,2,\dots,\delta(u)\} - p_u(v) \to \{1,2,\dots,\delta(u)-1\}$, $\alpha_v: \{1,2,\dots,\delta(v)\} - p_v(u) \to \{1,2,\dots,\delta(v)-1\}$とする．グラフ$G$におけるアルゴリズム$A$の任意の正当な状況$C$について，実行$\xi_A\left(R_{u,v,\alpha_u,\alpha_v,A}(C)\right) = C_0, C_1, \dots$が$i$ステップ以内に正当な状況に到達するとき，1 組の辺削除に対するアルゴリズムAの復旧時間は高々iステップであるという．

3. 提案アルゴリズム

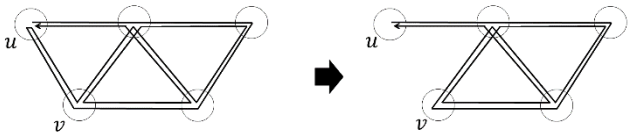
3.1 アルゴリズムの戦略・全体像

本節では 1 組の辺削除に対して$O(m)$ステップで復旧する自己安定永続探索アルゴリズムを提案する．1 節で述べた通り，従来のロータルータアルゴリズムでは 1 組の辺削除からのオイラー閉路の復旧に$O(\gamma m)$ステップを要する．γは最悪時$O(D)$となるので，従来手法の 1 組の辺削除に対する復旧時間は最悪時$O(mD)$となり，任意の初期状況からの収束時間と漸近的に等しいステップ数を要することになる．従来手法が 1 組の辺$(u,v),(v,u)$の削除から$O(\gamma m)$ステップの復旧時間を要するのは，ノードu,vの近傍を中心にオイラー閉路を大幅に修正することが起こり得るからである．提案アルゴリズムは，辺削除前の正当な状況で生成されていたオイラー閉路を最大限活用することで$O(m)$ステップでの復旧を実現する．

初めにアルゴリズムの戦略について簡潔に述べる．辺の削除が発生する前，エージェントはオイラー閉路eを探索している．eはグラフ上のノードの順列によって$e = u_0, u_1, \dots, u_m (u_m = u_0)$と表現できる．$e$を探索中にノード$(u,v)$間の辺$(u,v),(v,u)$が削除された場合，削除された辺によって 2 通りに分けられる．辺(u,v)も(v,u)もオイラー閉路eに現れるので、$(u,v) = (u_i, u_{i+1}), (v,u) = (u_j, u_{j+1})$とし，一般性を失うことなく，$i < j$と仮定する．

1 つ目は，削除された 2 つの有向辺$(u,v),(v,u)$がeで連続する場合，つまり，$j = i+1$の場合である．この時，図 1 に示すように，eから(u,v)と(v,u)を削除して得られるe'はオイラー閉路となるため，容易に新たなオイラー閉路を構築することが可能である．

2 つ目は，削除する有向辺$(u,v),(v,u)$がeで連続しない，つまり，$j > i+1$場合である．図 2 に示すように，オイラー閉路eは，辺$(u,v),(v,u)$の削除によって互いに辺素な 2 つの閉路に分割される．このとき，グラフの各辺はいずれかの閉路に含まれる．ここで，この 2 つの閉路うち，辺の削除が発生した状況でエージェントがロータールータアルゴリズムにしたがって移動する際に最初に通過する有向辺を含む閉路をe_1とし，もう一方をe_2とする．最初にエージェントが通過する有向辺が削除された場合，提案アルゴリズムによって通過が決まった有向辺を含む閉路をe_1とする．辺削除後のグラフG'の連結性から，e_1とe_2は少なくともひとつのノードを共有する．提案アルゴリズムは，各頂点のポート番号の移動順を入れ替えることで，共有ノードのひとつwを介してe_1とe_2を連結し，新しいオイラー閉路e'を構築する．具体的には，$e_1 = u_0, u_1, \dots, u_k, \dots, u_{s-1}, e_2 = x_0, x_1, \dots, x_{k'}, \dots, x_{s'-1}$としたとき$(u_0 = u_{s-1}, x_0 = x_{s'-1}, u_k = x_{k'} = w), e' = u_0, u_1, \dots, u_k (= x_{k'}), \dots, x_{s'-1} (= x_0), x_1, \dots, x_{k'} (= u_k), \dots, u_{s-1} (u_0)$となるオイラー閉路$e'$を構築する（図 3 参照）．以降，削除された辺に接続する 2 ノードu,vを辺消失点と呼ぶ．また，上記のノードwを結合点と呼ぶ．



オイラー閉路e　　オイラー閉路e'

図 1　辺削除後，オイラー閉路eが分割されない例

e_1とe_2の連結を実現するため，エージェントは各ノードsの白板に 2 つの変数$s.\mathrm{in}, s.\mathrm{out}$を用意することで，エージェントが通過したポート番号の順序を記憶させる．詳細は後述するが，ポート番号の順序について，エージェントが過去にどの内向辺を辿ってsに到達したかを$s.\mathrm{in}$に，どの外向辺を使ってsから移動したかを$s.\mathrm{out}$に保存する．ポート番号の順序を保存しておくことで，エージェントは巡回中の閉路がオイラー閉路でないときにそれを検知することができる．また，この順序を辿ることによって，辺削除の発生後にe_1およびe_2 を巡回すること，および，e_1とe_2の結合を可能にする．$s.\mathrm{in}, s.\mathrm{out}$はどちらも$\delta(s)$個の要素の配列で構成されている．つまり，$s.\mathrm{in} = \{s.\mathrm{in}_1, s.\mathrm{in}_2, \dots, s.\mathrm{in}_{\delta(s)}\}$である．

次に，提案アルゴリズムの全体像を述べる．提案アルゴリズムの実行においてエージェントは以下の 4 つのモードを切り替えて動作する．

- *normal*: 生成されているオイラー閉路を巡回するモード
- *searchW*: e_1を巡回しながら結合点wを探索するモード
- *trace*: 結合後，e_2を巡回してe_2の各辺を 1 回ずつ移動し，その後 *searchW* で巡回しなかったe_1の各辺を 1 回ずつ移動するモード
- *cleaning*: 初期状況からのオイラー閉路の生成，および辺の削除から新しいオイラー閉路の生成が終了後，ノードの白板の変数の初期化を行うモード

エージェントは通常 *normal* モードで動作する．*normal* モード時，エージェントは，従来のロータールータアルゴリズムと同様の移動を行う．具体的には，ノードvを訪問した際，vの外向辺の移動履歴$v.\mathrm{out}$を参照し，最も過去に移動した外向辺を選択して移動する．アルゴリズムの開始時，$v.\mathrm{out}$の値は$\delta(s)$個の相違なるポート番号の任意の並びが格納されている．後述するように，$(u,v),(v,u)$の削除が発生した際，2 ノードu,vの白板に辺削除を示すラベルが生成される．そのため，1 組の辺削除が発生したのち，*normal* モードのエージェントはm回の移動中に辺消失点u,vのいずれかを訪問する．ここでは，一般性を失わずにはじめに辺消失点uを訪問したものとする．辺消失点uを発見したエージェントは *searchW* モードに切り替わる．*searchW* モードでは，エージェントはe_1に沿ってグラフ上の移動を行う．ただし，後述する方法でe_1とe_2に共通して現れるノードwを発見した際には，wを結合点としてe_1とe_2を結合するため$w.\mathrm{in}$の要素の値を以下の手順で変更する．

- ノードwに移動する際に通過した内向辺のポート番号p_{in}に関して，$p_{in} = w.\mathrm{in}_k$となるkを探す．
- $w.\mathrm{in}_1 \leftrightarrow w.\mathrm{in}_k$（$w.\mathrm{in}_1$に$w.\mathrm{in}_k$を交換）．

以上の変更によって，e_1とe_2の互いに辺素な 2 つのオイラー閉路を 1 つに結合することができる．

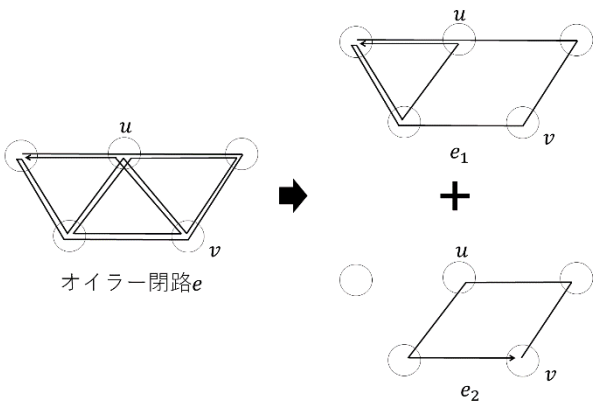


図 2　辺の削除後，互いに辺素な 2 つのオイラー閉路に分割される例

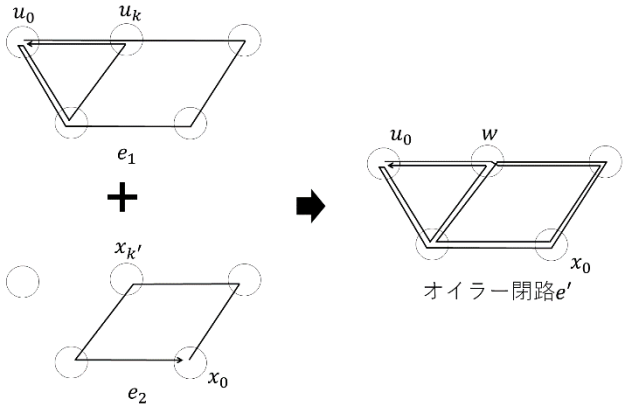


図 3　2 つのオイラー閉路から新たなオイラー閉路の生成例

白板の値の変更後，エージェントは*trace*モードに移行する．*trace* モードのエージェントは，各ノードsの白板を参照することで，e_2の各辺を 1 回ずつ移動し，その後，*searchW* モードで移動しなかったe_1の辺を 1 回ずつ移動する．これらの移動は，各内向辺からノードsに移動してきた場合に次の移動先に選択した外向辺の履歴が$s.\mathrm{out}$の中に残っているため行うことが可能となる．

searchW モードおよび *trace* モードでは，辺消失点 u の外向辺を通るたびにuの白板$u.\mathrm{check}$にその外向辺を通過したことを記憶しておく．*trace* モードでuに到着した時，すべての外向辺が移動済みであれば，*cleaning* モードに移動する．

cleaning モードではエージェントは，オイラー閉路が生成されている場合，ローターループアルゴリズムと同様に移動を行うことでその閉路に従ってすべての辺をちょうど 1 回移動し，*normal* モードに移行する．オイラー閉路が生成されていない場合，ローターループアルゴリズムと同様に$O(mD)$回の移動でオイラー閉路を生成し，*normal* モードに移行する．いずれの場合も，移動の際，各ノードsの白板に対して$s.\mathrm{in}, s.\mathrm{out}$以外の変数をリセットしていく．オイラー閉路を移動できたかどうかの判定は，後述するが，根ノードの各外向辺に移動済みのマークを付けるとともに，*cleaning* モードで根ノードのポート番号 1 が示す外向辺を通過してから，再びポート番号 1 の外向辺を通過するまでの移動回数をエージェントメモリに記憶させることで容易に実現できる．

cleaning モード以外の 3 モードにおいて，p_{in}や辺消失点のフラグなど白板の各種変数の値に不整合を発見した場合，*cleaning* モードに移行する．これにより，正当な状況で 1 組の辺削除が発生した場合には$O(m)$回の移動でオイラー閉路を再構築するとともに，任意の初期状況から開始された実行で$O(mD)$の移動でオイラー閉路を生成することを保証する．

以降では，エージェントおよび白板が管理する変数，および，各モード実行時のエージェントの処理を詳細に説明する．

3.2　エージェントの保持する変数

エージェントはモードを表す変数 $\mathrm{mode} \in \{$*normal, searchW, trace, cleaning*$\}$を持つ．また，オイラー閉路が生成されたことを確認するために 2 つの変数$t_1, t_2 \in \{1,2,\ldots, N(N-1)\}$（$N$はノード数の上界) を持つ．

3.3　白板の保持する変数

ノードsの白板は，5 つの変数$s.\mathrm{in}, s.\mathrm{out}, s.\mathrm{check}, s.\mathrm{d}, s.\mathrm{r}$を保持する．変数$s.\mathrm{in}, s.\mathrm{out}$は$\delta(s)$個の相違なるポート番号の並びである．

$s.\mathrm{in}$は，ノードsの内向辺について，その最新の訪問時刻が最も古いものから順に$\delta(s)$個のポートを並べたものである．$s.\mathrm{in}$のi番目の要素を$s.\mathrm{in}_i$で表す．すなわち，$s.\mathrm{in} = s.\mathrm{in}_1, s.\mathrm{in}_2, \ldots, s.\mathrm{in}_{\delta(s)}$である．$s.\mathrm{in}_1$は，ノード$s$の内向辺のなかで最も長い間エージェントの移動に利用されなかった内向辺のポート番号である．

$s.\mathrm{out}$は，ノードsの外向辺について，その最新の訪問時刻が最も古いものから順に$\delta(s)$個のポートを並べたものである．$s.\mathrm{out}$のi番目の要素を$s.\mathrm{out}_i$で表す．すなわち，$s.\mathrm{out} = s.\mathrm{out}_1, s.\mathrm{out}_2, \ldots, s.\mathrm{out}_{\delta(s)}$である．$s.\mathrm{out}_1$は，ノード$s$の外向辺のなかで最も長い間エージェントの移動に利用されなかった外向辺のポート番号である．

$s.\mathrm{in}, s.\mathrm{out}$はそれぞれ，各要素の値がすべて異なる．つまり，$\forall i, j \in \{1,\ldots,\delta(s)\}(i \neq j), s.\mathrm{in}_i \neq s.\mathrm{in}_j, s.\mathrm{out}_i \neq s.\mathrm{out}_j$となる．

$s.\mathrm{check}$は$\delta(s)$個の要素を持つ配列で，各要素は 0,1 のいずれかの値を持つ．$s.\mathrm{check}$の$\delta(s)$個の要素を$s.\mathrm{check}_1, \ldots, s.\mathrm{check}_{\delta(s)}$で表し，$s.\mathrm{check}_k$は，ノード$s$におけるポート番号$k$に対応する外向辺を mode = *searchW* または mode = *trace* のいずれかで移動したことを記録する．

$s.\mathrm{d}$はノードsが削除された辺に接続するノードかどうかを示す（$s.\mathrm{d} \in \{0,1\}$）．通常，$s.\mathrm{d} = 0$である．辺$(u,v), (v,u)$が削除されると，辺消失点u, vにおいてそれぞれ$u.\mathrm{d}, v.\mathrm{d}$の値が 1 に変更される．

$s.\mathrm{r}$はノードsが根ノードかどうかを示す（$s.\mathrm{r} \in \{0,1\}$）．2 節で定義した通り，根ノードは有向グラフ中でただひとつ存在するノードである．$s.\mathrm{r} = 1$のとき，ノードsは根ノードであることを，$s.\mathrm{r} = 0$のとき，sは根ノードでないことを示す．$s.\mathrm{r}$の値は固定である．すなわち，アルゴリズムの実行中，$s.\mathrm{r}$の値が変化することはない．

3.4　提案アルゴリズムの詳細

Algorithm1 に提案アルゴリズムの疑似コードを示す．Algorithm1 はエージェントがノードに移動したときに実行される．p_{in} は，ノードsに入ってきたときのポート番号を表す．Algorithm1 の行番号ごとに，主な動作について以下に示す．

- 1-4 行目・・・異常検知.
- 6-9 行目・・・辺消失点を検知する.
- 10-13 行目・・・結合点を検知する.
- 14-16 行目・・・通過した辺を記録する.
- 17-28 行目・・・*cleaning* モードの動作を行う.
- 29-31 行目・・・白板の書き換え，別のノードに移動.

Algorithm1 の各動作について以下に詳しく述べる.

1-4 行目では，不整合が発生した場合に *cleaning* モードに移行する．不整合は初期状況が正当な状況でない場合に発生する．オイラー閉路が生成されていない場合，mode = *normal* かつ$s.\text{in} \neq p_{in}$となる．$s.\text{check}$の具体的な役割については 6-9 行目の処理の説明の所で述べる.

5 行目は，他のノードからノードsに来た時に通過した内向辺のポート番号と，$s.\text{in}_k$が同じ値となるkを定義する．正当な状況の場合，kは 1 となる.

6-9 行目では，辺の削除を初めて検知した場合，*normal* モードから *searchW* モードに移行する．エージェントが辺の削除を検知するのは，$s.\text{d} = 1$，つまり辺消失点にエージェントが来たときである．*searchW* モードに移行することでオイラー閉路の復旧を開始する．オイラー閉路の復旧を開始してから終了までに，グラフの全ての辺を一度だけ通過する．ただし，この復旧作業は，正当な状況から一組の辺削除が発生した状況から実行を開始することを前提としている．そのため，そうでない状況から実行を開始した場合には，グラフのある辺を 2 回以上通過することが起こりうる．このとき，これを検知して *cleaning* モードに遷移するために，$s.\text{check}_{s.\text{out}_1}$に 1 を代入し，移動を記録しておく必要がある.

10-16 行目で，*searchW* モードと *trace* モードで行う動作について記述している．*searchW* モードは結合点を発見するまでの動作，*trace* モードは結合点の発見からe_2（3.1 節）の全ての辺を 1 度だけ通過するまでの動作を行う．13 行目と 16 行目で，*searchW* モードと *trace* モードで通過したことを記録するため，次に通過する外向辺に対応するポート番号をpとして，$s.\text{check}_p$に 1 を代入する.

10-13 行目では，エージェントが結合点を発見した場合の動作を行う．*searchW* モードを開始してから初めて$s.\text{in}_1 \neq p_{in}$となるノードが結合点であり，結合点を発見したエージェントは，*searchW* モードから *trace* モードに移行する．移行時，$s.\text{in}_1$と$s.\text{in}_k$を入れ替える．本稿では，紙面の都合で証明を省略するが，この処理により，辺削除により分断されたふたつのオイラー閉路が結合される.

14-16 行目では，*searchW* モードと *trace* モード共通の動作を行う．*searchW* モードと *trace* モードではp_{in}の値によってエージェントの次の移動先を決定する．具体的には，$(s.\text{in}_1, s.\text{out}_1)$を$(s.\text{in}_k, s.\text{out}_k)$と入れ替え，$s.\text{out}_1$に対応する外向辺を移動することとする．この移動方法は，sが管理する移動履歴$s.\text{in}$および$s.\text{out}$において，sに移動してきた際に用いた内向辺に対応する外向辺を用いて移動することを意味する．したがって，この移動方法によって，エージェントは辺削除により分断されたふたつのオイラー閉路を巡回することができる.

17-28 行目では，*cleaning* モードでの動作を説明している．*cleaning* モードでは 2 つの役割が存在する.

1 つ目は，オイラー閉路の再構築のために用いた白板の変数の値を元に戻すことである．具体的には 18,19 行目で全てのノードの白板の変数checkとdの値を 0 にする.

2 つ目は，初期状況でオイラー閉路が形成されていない場合，もしくは切断以外の何かしらの理由によってオイラー閉路が崩れた場合に，オイラー閉路を復旧することである.

Algorithm 1 提案アルゴリズム

Main Routine:

1:**if** mode $= normal \wedge (s.\text{in}_1 \neq p_{in})$ **or** mode $\neq cleaning \wedge (\exists k, \exists p; p_{in} = s.\text{in}_k \wedge p = s.\text{out}_k \wedge s.\text{check}_p = 1)$ **then**
2: mode $\leftarrow cleaning$
3: $t_2 \leftarrow 0$
4: **end if**

5: Let k be the interger such that $s.\text{in}_k = p_{in}$.
6: **if** mode $=$ $normal$ **and** $s.\text{d} = 1$ **then**
7: mode $\leftarrow searchW$
8: $(s.\text{in}_k, s.\text{out}_k) \leftrightarrow (s.\text{in}_1, s.\text{out}_1)$
9: $s.\text{check}_{s.\text{out}_1} \leftarrow 1$
10: **else if** mode $=$ $searchW$ **and** $s.\text{in}_1 \neq p_{in}$ **then**
11: mode $\leftarrow trace$
12: $s.\text{in}_k \leftrightarrow s.\text{in}_1$
13: $s.\text{check}_{s.\text{out}_1} \leftarrow 1$
14: **else if** mode $\in \{searchW, trace\}$ **then**
15: $(s.\text{in}_k, s.\text{out}_k) \leftrightarrow (s.\text{in}_1, s.\text{out}_1)$
16: $s.\text{check}_{s.\text{out}_1} \leftarrow 1$
17: **else if** mode $= cleaning$ **then**
18: $s.\text{check}_p \leftarrow 0$ for all $p = 1.2, \dots, \delta(s)$
19: $s.\text{d} \leftarrow 0$
20: $s.\text{in}_k \leftrightarrow s.in_1$
21: **if** s.r $= 1$ **and** $s.\text{out}_1 = 1$ **and** $0 < t_1 = t_2$ **then**
22: mode $\leftarrow normal$
23: **else if** $s.\text{r} = 1$ **and** $s.\text{out}_1 = 1$ **then**
24: $t_2 \leftarrow t_1$
25: $t_1 \leftarrow 0$
26: **end if**
27: $t_1 + +$
28: **end if**

29: $p_{out} \leftarrow s.\text{out}_1$
30: Lotate()
31: Move throuth port p_{out}

Algorithm 2 提案アルゴリズム

Lotate():
1: $(\text{s.in}_1, s.\text{in}_2, \dots, s.\text{in}_{\delta(s)}) \leftarrow (\text{s.in}_2, s.\text{in}_3, \dots, s.\text{in}_{\delta(s)}, \text{s.in}_1)$
2: $(\text{s.out}_1, s.\text{out}_2, \dots, s.\text{out}_{\delta(s)}) \leftarrow (\text{s.out}_2, s.\text{out}_3, \dots, s.\text{out}_{\delta(s)}, \text{s.out}_1)$

20 行目はinの配列にオイラー閉路の順序を記録する.

21-27 行目では，オイラー閉路が生成されたかを確認する．根ノード s のポート番号 1 に対応する外向辺を移動してから，1 周して再びポート番号 1 の外向辺を通過しようとするまでの移動回数を記憶しておくことでオイラー閉路が生成されたかを確認することが可能である．移動回数を

数えるため，エージェントに変数t_1, t_2を用意する．t_1に現在の 1 周のエージェントの移動回数を，t_2に前回の一周の移動回数を記録する．1 周が終わると，t_1, t_2を比較し，同じ値の場合に，normal モードに移行する．

29-31 行目で白板の変数を書き換えてエージェントが移動を行う．29 行目でエージェントが次に移動する外向辺のポート番号をp_{out}に記憶しておく．その後，関数 Lotate() によって，ノードsの白板の内容を書き換える．関数 Lotate は$s.\mathrm{in}_1$と$s.\mathrm{out}_1$をキューの後ろに回すことで通過した順にポート番号を保持する．詳しくは Algorithm2 に示す．その後，p_{out} のポート番号が示す外向辺を通過してエージェントが移動する．

辺の削除によってオイラー閉路が 2 つに分解されなかった場合，$normal$モードから$searchW$モードに移行するが既にオイラー閉路が生成されているため，$s.\mathrm{in}_1 \neq p_{in}$は発生しないため$searchW$モードで全ての辺を通過後$cleaning$モードに遷移する．

$cleaning$モードに遷移後，オイラー閉路はすでに生成されているので，$cleaning$モードは$O(m)$ステップで終了し，正当な状況に移行する．

3.5 削除された辺に接続するノード（辺消失点）の処理

辺消失点の処理アルゴリズムを Algorithm3 に示す．ノードsに接続した辺(s,t)が削除された場合を考える．ポート番号の再割り当てが必要であり，ポート再割り当て関数α_uを用いて$s.\mathrm{in}, s.\mathrm{out}$の値を変更する．

辺の削除によって，ノードsに接続されている内向辺と外向辺の本数はどちらも 1 減る．そのため，最初に，$s.\mathrm{in}, s.\mathrm{out}, s.\mathrm{check}$から，辺$(s,t)$を削除する．

Algorithm 3 提案アルゴリズム（辺削除発生時）

1: Delete $p_s(t)$ from $s.\mathrm{in}, s.\mathrm{out}$ and $s.\mathrm{check}$
2: **for** $i = 1,2,\dots,\delta(s)-1$ **do**
3: $\quad \mathrm{s.in}_i \leftarrow \alpha_s(\mathrm{s.in}_i)$
4: **end for**
5: $\mathrm{s.d} \leftarrow 1$

上記の提案アルゴリズムは，表 1 に示す計算量での永続探索問題を解く．本稿においては，計算量の解析について，紙面の都合上割愛する．

4. まとめ

本稿では双方向有向グラフにおいてグラフ探索を実現するロータールータについて，1 辺の削除からのオイラー閉路の復旧までの時間を，$O(\gamma m)$から $O(m)$に改善する方法を示した．一方，このために，エージェントおよびノードが要する記憶容量が増加している．

今後の課題は，エージェントが記憶容量もたない場合での辺削除からの復旧時間の改善について考えたい．また，根ノードが存在しない状況での同様の改善を考えたい．

謝辞 本研究は一部 JSPS 科研費 17K19977, 16K00018, 18K18000, JST 戦略的国際共同研究プログラム(SICORP)の助成を受けたものです．

参考文献

[1] Y. Sudo, D. Baba, J. Nakamura, F. Ooshita, H. kakugawa, and T. Masuzawa, “A Single Agent Exploration in Unknown Undirected Graphs with Whiteboards,” IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, vol.E98, no.10, p. 2117-2128, 2015.

[2] S. Das, P. Flocchini, S. Kutten, A. Nayak, and N. Santoro, “Map construction of unknown graphs by multiple agents,” Theor. Comput. Sci., vol.385, no1-3, pp.34-48, 2007.

[3] S. Albers and M.R. Henzinger, “Exloring an unknown environments,”Proc. Twenty-Ninth Annual ACM Symposium on Theory of Computing - STOC'97, pp416-425, 1997

[4] X. Deng andC.H. Papadimitriou, "Exploring an unknown graph," J. Graph. Theor, vol.32, no.3, pp.265-297, 1999

[5] R. Fleischer and G. T rippen, “Exploring an unknown graph efficiently,” Algorithms ESA 2005, Lecture Notes in Computer Science, vol.3669, pp.11-22, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005

[6] Vladimir Yanovski, Israel A. Wagner, Alfred M. Bruckstein, “A Distributed Ant Algorithm for Efficiently Patrolling a Network,” Algorithmica in Springer-Verlag, vol.37, no.2, pp.165-186, 2003.

[7] Evangelos Bampas, Leszek Gasieniec, Nicolas Hanusse, David Ilcinkas, Ralf Klasing, Adrian Kosowski, Tomasz Radzik, “Robustness of the Rotor-Router Mechanism,” Algorithmica in Computer Science, vol 5923, pp 345-358, 2017.