オンライン木探索アルゴリズム

八神 貴裕[1,a)]　山内 由紀子[2]　来嶋 秀治[2]　山下 雅史[2]

概要：グラフ探索問題は，連結なグラフ内を逃げ回る侵入者を複数の探索者が移動しながら発見する問題である．グラフの形を知っている（グラフの外にいる）プレイヤーが複数の探索者を動かしてグラフを探索するオフライン探索問題が盛んに研究されている．本稿では，初期状態ではグラフの形状を与えられていない（グラフの中にいる）一人のプレイヤー（探索者）が，侵入者の行動を制限するバリケードをいくつか用いて木をオンライン探索するアルゴリズム TSB を提案する．TSB は木 T をバリケード r 個で探索可能かどうかを正しく判断し，可能であれば探索する．つまり，TSB で与えられた木 T を k 個のバリケードで探索ができなかったとき，どのようなオフラインのアルゴリズムでも T を k 個のバリケードで探索することはできないことを示す．

キーワード：グラフ探索問題，オンライン木探索，オフライン木探索

Online Tree Search Algorithm

Abstract: Given a tree T and k barricades, we investigate the problem of searching T for an intruder moving in T with unbounded speed by a searcher walking in T with a bounded speed using the k barricade. When barricade has been placed on a vertex, intruder cannot go through the vertex. We assume that the searcher has no access T at least at initiation time. We propose a search algorithm for the searcher such that it successfully search T when it is possible, and otherwise, the searcher knows that T is not searchable. We also show that this algorithm is optimal in the search, namely if T is not searchable by this algorithm then T is not searchable by k barricades in the offline setting.

Keywords: graph search problem, online tree search, offline tree search

1. はじめに

グラフ探索問題 (graph search problem) は，グラフ内を逃げ回る侵入者を探索者が移動しながら発見する問題である．この問題は Parsons[7] によって初めて考察された．元々は，洞窟で仲間とはぐれた友人（探索対象）を見つけ出すためには何人の**探索者** (searcher) が必要で，それぞれがどのように行動すれば発見できるか，という疑問から生じた問題である．洞窟を道が枝分かれしている場所を頂点，それらをつなぐ通路を辺としたグラフとして解釈し，そのグラフに対して，探索者の動きを考えることが提案された [7]．グラフ探索問題で注意しなければならない点は，侵入者はどこにいるか分からない，侵入者は常にグラフ内を動き回っている，侵入者の移動の速さは不明であることである．このことは一度探索者によって探索された場所に，侵入者が訪れる可能性があることを意味する．

グラフ内を移動する対象を探索する問題だけでなく，多角形内を移動する対象を探索する問題も研究が行われている．**多角形探索問題** (polygon search probrem) は，多角形内を逃げ回る侵入者を探索者が発見する問題である [1][4]．文献 [6] は複数の探索者による多角形探索問題について，グラフ探索問題に帰着し探索者数の上界の 1 つを得ている．グラフ探索問題や多角形探索問題のような移動する対象を発見する問題は，例えばロボットによる建物の警備や，市街地に出没した動物の捕獲に役立つだろう．

文献 [1][5] では，探索者を石とみなしてグラフ探索問題をグラフ上の石置きゲームとしてモデル化した．複数の

[1] 九州大学電気情報工学科
Department of Electrical Engineering and Computer Science, Kyushu University
[2] 九州大学大学院システム情報科学研究院
Graduate School of Information Science and Electrical Engineering, Kyushu Univercity
[a)] yakami@tcslab.csce.kyushu-u.ac.jp

© 2015 Information Processing Society of Japan

探索者によるグラフ探索問題には，**辺探索モデル** (edge searching)，**頂点探索モデル** (node searching) や**混合探索モデル** (mixed searching) と呼ばれるものも存在する [1][3]．連結で閉路を持たないグラフである**木** (tree) に関しては，辺探索モデルと混合探索モデルのどちらについても線形時間で探索に必要な最小の探索者数を求めることができる [2][5]．これらの複数探索者のモデルは全て石置きゲームのプレイヤーがグラフの形を知っていることが前提である．プレイヤーがグラフの形を知っている探索を**オフライン** (offline) 探索と呼ぶ．しかし，実際洞窟で誰かが遭難したとき，救助隊員が必ずしもその洞窟の構造を知っているとは限らない．このような状況において，グラフの中にいる探索者がグラフの形が分からなくてもグラフを探索できるアルゴリズムは非常に有用である．初期状態でグラフの中にいるプレイヤー（探索者）がグラフの形や自分の位置を知らない探索を**オンライン** (online) 探索と呼ぶ．

本稿では，一人の探索者がいくつかのバリケードを用いて探索するバリケードモデルを提案する．そして，バリケードモデルを用いて連結で閉路を持たないグラフである木のオンライン探索について考える．まず，2 章で諸定義を行う．3 章ではバリケードモデルを用いて，木のオンライン探索アルゴリズム TSB を提案する．そして，4 章で TSB で木 T をバリケード k 個で探索できないとき，任意のオフラインのアルゴリズムで木 T をバリケード k 個で探索できないことを示す．

2. 準備

複数探索者のモデルの一つである辺探索モデル（オフライン）の定義の紹介と，バリケードモデルのオンラインモデルの定義を行う．初期状態で探索者がグラフの形や自分の位置を知らないことをオンライン探索と言う．

2.1 グラフ探索問題

グラフ探索問題 (graph search problem) の入力は連結な無向グラフ $G = (V, E)$ である．探索開始時刻のグラフの全ての辺は**非クリア** (contaminated) と呼ばれる状態である．非クリアな辺は探索者の通過によって**クリア** (cleared) になる．しかし，クリアな辺と非クリアな辺の両方が接続する頂点 $v \in V$ があり，v に探索者（またはバリケード）がない場合，v に接続するクリアな辺は全て非クリアに戻る．このことを**再汚染** (recontaminated) という．グラフ G の全ての辺が同時にクリアになれば探索は終了する．直感的に言うと，非クリアな辺は侵入者がいる可能性があり，クリアな辺は侵入者がいない．

2.2 辺探索モデル

辺探索モデルの定義を紹介する [5]．辺探索モデルにおける探索者を本稿ではガードと呼ぶ．k 個のガードを用いたグラフの全ての辺をクリアにする手順が存在するとき，その手順を探索スケジュールと呼ぶ．辺探索モデルでは，グラフ G の全ての辺をクリアにできる最小のガード数を求めることと，探索スケジュールを求めることがその目的である．G を辺探索モデルで探索するときの最小のガード数を $es(G)$ と表す．

辺探索モデルにおけるガードの操作は以下の 3 つである．

- ガードを頂点に置く
- ガードを頂点から取り除く
- ガードをある頂点から隣接する頂点へ辺を通過して移動する

非クリアな辺 $e = (u, v)$ は以下の 2 つの方法でクリアになる．一般性を失わず，ガードは頂点 u から頂点 v に移動すると仮定する．

- 頂点 u に 2 つ以上のガードが置かれていれば，そのうちの 1 つのガードを辺 e を通過して移動することで e はクリアになる．
- 頂点 u に接続する e 以外の全ての辺がクリアかつ $u \neq v$ であれば，u に置かれているガードが 1 つだけでも辺 e を通過して移動することで e はクリアになる．木探索の場合，木には自己ループがないので $u \neq v$ の条件は特に気にする必要はない．

2.3 バリケードモデルのオンライン探索の定義

バリケードモデルのオンライン探索の定義を行う．k 個のバリケードを用いたグラフの全ての辺をクリアにする手順が存在するとき，その手順を探索スケジュールと呼ぶ．バリケードモデルでは，グラフ G の全ての辺をクリアにできる最小のバリケード数を求めることと，探索スケジュールを求めることがその目的である．G をバリケードモデルで探索するときの最小のバリケード数を $r(G)$ と表す．探索者が探索開始時刻に持っているバリケードの個数を r とする．探索者はグラフの形が分からないので，探索しながら得た情報をメモリに記録しながら探索を行う．

探索者は以下の 5 個の基本動作を組み合わせて探索する．

- 今いる頂点から，隣接する頂点に辺を通過して移動する．
- 今いる頂点内で向きを変える．
- 今いる頂点にバリケードを設置する．
- 今いる頂点のバリケードを回収する．
- 今いる頂点のバリケードのメモリの読み出しと書き込み．

また，探索者は次の情報を得ることができる．

- 今いる頂点の次数
- 今いる頂点にバリケードがあるかどうか

非クリアな辺 $e = (u, v)$ は以下の 2 つの方法でクリアになる．一般性を失わず，探索者は頂点 u から頂点 v に移動すると仮定する．

© 2015 Information Processing Society of Japan

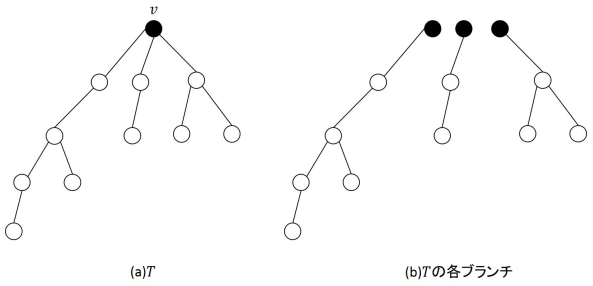

図 1: 木 T の v のブランチ

- 頂点 u にバリケードが設置されていれば，探索者が辺 e を通過することで e はクリアになる．
- 頂点 u に接続する e 以外の全ての辺がクリアかつ $u \neq v$ であれば，探索者が辺 e を通過することで e はクリアになる．木探索の場合，木には自己ループがないので $u \neq v$ の条件は特に気にする必要はない．

3. 木のオンライン探索

本章では連結で閉路を持たないグラフである木 $T = (V, E)$ を扱う．また，木 T は平面に埋め込まれたグラフを仮定する．本章では，バリケードモデルで T のオンライン探索を行うアルゴリズム TSB を提案する．次のように記号を定義する．

- b_i は今グラフ上に設置されているバリケードの中で i $(1 \leq i \leq r)$ 番目に設置されたバリケード
- $v(b_i)$ は b_i が設置されている頂点
- $T(v)$ は頂点 v を根とする根付き木

定義 1 ([7])．木 T の頂点 v のブランチ S とは，v を次数 1 の頂点と見なしたときの極大の T の部分木のことを指す．(図 1(a) は木 T，(b) は v のブランチ)

つまり，$T(v)$ の v の子の 1 つを根とした部分木 T' を $T' = (V', E')$ と表すとき，T の v のブランチ S は $S = (V' \cup \{v\}, E' \cup \{(u, v)\})$ と表すことができる．また，$v(b_i)$ $(2 \leq i \leq r)$ のブランチとは，$v(b_{i-1})$ のブランチのうち $v(b_i)$ を含むブランチを木と見なしたときの $v(b_i)$ のブランチとする．$v(b_1)$ のブランチは木 T の $v(b_1)$ のブランチとする．

特に，辺探索モデルにおいてハブとアベニューは以下のように定義されている．

定義 2 ([5])．木 T のハブとは，頂点 h のブランチが全て $es(T) - 1$ 個のガードで探索できるような頂点 h のことである．(図 2(a))

定義 3 ([5])．木 T のアベニューとは以下の 2 つの条件を満たす長さ 2 以上の道 $\langle v_1, v_2, ..., v_a \rangle$ のことである．(図 2(b))

- v_i $(2 \leq i \leq a-1)$ はガードが $es(T)$ 個必要なブランチをちょうど 2 つ持つ．
- v_1 と v_a はガードが $es(T)$ 個必要なブランチをちょうど 1 つ持つ．

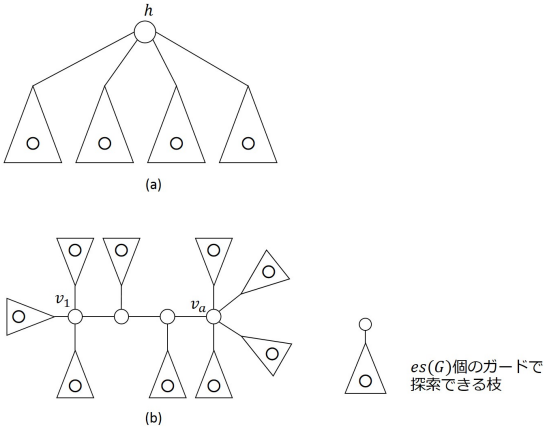

図 2: ハブ h とアベニュー $\langle v_1, v_2, ..., v_a \rangle$

3.1 提案アルゴリズム TSB

提案アルゴリズム TSB は，与えられた木 T の頂点にバリケードを置くことで，木をいくつかのブランチに分解する．そして，各ブランチを 1 つの木と見なして，同様に探索をしていく再帰的なアルゴリズムである．このアルゴリズムのポイントは，ブランチを重複なく順に探索することや，バリケードを設置する最適な位置を探索者自身で見つけ出すこととである．また，辺探索モデルにおいて，以下の定理，補題が既に知られている．これらを参考にオンラインアルゴリズムを設計する．

定理 1 ([7])．$k \geq 1$ について，木 T の探索が探索者 $k+1$ 人以上必要であるとき，かつそのときに限り，T には頂点 $v \in V$ のブランチのうち探索者が k 人以上必要なブランチが少なくとも 3 つ存在するような v が存在する．

補題 1 ([5])．任意の木 T について，$es(T) = s$ ならば，T は次のどちらかを満たす．(i)v の全てのブランチは s 個未満のガードで探索可能（v はハブである），または (ii) アベニューを 1 つだけ持つ．

このアルゴリズムでは探索者と各バリケードにいくつかメモリを与えている．ここではそれらのメモリを紹介する．

探索者の持つメモリ

- r
  探索者が探索開始時刻に持っているバリケードの個数．整数型でサイズは $\lceil \log_2 r \rceil$ ビット．
- $mode$
  ブランチの探索の結果を記録する．$mode$ = E(xecute) は探索中であることを表す．$mode$ = S(uccess) はブランチの探索に成功したことを表す．$mode$ = F(ailure) はブランチの探索に失敗したことを表す．E, S, F の値をとるように記述しているが，それぞれ異なる整数を割り当ててもよいので整数型とする．3 つの値が区別できればよいのでサイズは 2 ビット．
- $flag$

© 2015 Information Processing Society of Japan

それぞれのバリケードが最適な位置にいるか記録する配列である．配列の大きさは r で，各バリケードと 1 対 1 で対応する．バリケードに割り当てていない理由は，探索開始直後に全ての $flag$ を 0 に初期化するからである．$flag = 0$ は初期値である．$flag = -1$ のときはバリケードを設置する最適な頂点を探しているときである．$flag = 1$ のときはバリケードを置くのに最適と思われる頂点を見つけているときである．バリケード b_i に対応する $flag$ を $flag[i]$ と表す．$-1, 0, 1$ の 3 つの値を区別できればよいので整数型で各サイズは 2 ビット．全体で $2r$ ビット．

- bn
  探索者が今持っているバリケードの個数 (barricade number) を表す．0 から r までの自然数が区別できればよいので整数型でサイズは $\lceil \log_2(r+1) \rceil$ ビット．
- $deg1$
  次数 1 の頂点を訪れた回数を記録する．木またはブランチの最高次数を調べるときに使われる．0 から 2 までの整数が記録できればよいので整数型でサイズは 2 ビット．

バリケードのメモリ

- $counter$
  未探索のブランチを順に重複なく探索するために用いる．T 内の頂点の最大次数を d とすると，0 から d までの整数が書き込めればよい．整数型でサイズは $\lceil \log_2(d+1) \rceil$ ビット．
- $name$
  設置されているバリケードを区別するためのメモリである．バリケード b_i の添え字の i を $name$ とする．バリケードに 1 から r まで順番に整数で名前を付ける．整数型でサイズは $\lceil \log_2 r \rceil$ ビット．
- $failure$
  $v(b_i)$ のブランチの探索に失敗し，$v(b_i)$ に戻ってきたときの $counter$ の値を書き込む．よって，型やサイズは $counter$ と同じ．

メモリの値を変更できるのは探索者だけで，$name = r - bn$ が成立するバリケードのメモリの値のみ書き換えることができる．探索者がバリケードのメモリを操作しているとき，b_{r-bn} のメモリを操作していることは自明であるので，b_{r-bn} の $counter$ や $failure$ などを操作するときは，単に $counter$ や $failure$ ということにする．もし，探索者が $name \neq r - bn$ となるバリケードが設置されている頂点にたどり着いたとき，その頂点は次数 1 の頂点と見なす．

探索者の基本的な移動の仕方について説明する．このアルゴリズムでは頂点を部屋，辺を通路のようにとらえている．探索者には向きがあり，移動するときは進行方向を向いている．探索者が向きを変えるのは頂点にいるときのみで，進行方向を決定するために行う．向きを変える動作は

Algorithm 1 SELECT_EDGE(j)

```
1: //探索者は入口となった辺を向く
   180 度反時計回りに回転する;
2: i := j;
3: while i > 0 do
4:     反時計回りに回転し，次の辺を見つけたら停止する;
5:     i := i − 1
6: end while;
7: if バリケードがあり，かつ name = bn then
8:     counter := counter − j
9: end if;
```

アルゴリズム SELECT_EDGE(j) によって行う．探索者が頂点 u から頂点 v に移動したときに通過した辺 (u, v) のことを入口となった辺と呼ぶ．探索者は今いる頂点を離れる前に必ずアルゴリズム SELECT_EDGE(j) によって，進行方向を決定する．アルゴリズム SELECT_EDGE(j) の説明をする．まず，探索者は 180 度回転し，入口となった辺の方を向く．そこから反時計回りに向きを変えていき，j 回辺を発見した時点で停止する．SELECT_EDGE(0) は単に 180 度回転することを表している．直感的にいうと，入り口となった辺から反時計回りに j 番目の辺の方向を向いている．

$v(b_i)$ のブランチの探索を行い成功または失敗が分かると (このとき $mode$ = S または F)，探索者は $v(b_i)$ に戻る必要がある．このとき，各頂点でアルゴリズム SELECT_EDGE(1) を行い，次に進む頂点を決定する．木は閉路を持たないので，このように移動すれば必ず $v(b_i)$ にたどり着ける．

メモリ $counter$ と探索者の動作の関係について説明する．探索者はバリケード b_i を設置するとき，設置するバリケードの $counter$ に，今いる頂点の次数を書き込む（ただし，探索中木に最初にバリケードを置いたときのみ次数より 1 だけ大きい値を書き込む）．探索者は $v(b_i)$ のブランチの探索の直前に SELECT_EDGE(1) を行い，向きを変え，$counter$ の値を 1 減らす．ブランチの探索後 $v(b_i)$ に戻ってきたとき，$counter > 1$ であれば，再び SELECT_EDGE(1) を行い次のブランチを探索する．$counter = 1$ であれば探索すべきブランチを全て探索したことになる．このように探索したとき，$v(b_i)$ の各ブランチを反時計回りに順に選んで探索している．

オンライン木探索を行うアルゴリズムを TSB(Tree Separation by Barricade) とする．入力は木 T，探索を開始する頂点 s，探索者の持つバリケード数 r で，出力は探索成功，もしくは失敗である．アルゴリズム TSB(T, s, r) とサブルーチンとして呼び出す各アルゴリズムは 3.1 節の最後にまとめて示す．

アルゴリズム TSB の動作のイメージは木の頂点にバリケードを設置することで，木をいくつかのブランチに分け，各ブランチを順に探索する．各ブランチについて，1 つの

© 2015 Information Processing Society of Japan

ブランチを木と見なしてバリケードを設置し，さらにいくつかのより小さいブランチに分けて同様に探索していく．このときの探索者の移動は木の深さ優先探索に似ている．

アルゴリズム中で用いる変数 $k(0 \leq k \leq r)$ はメモリの bn を表す．アルゴリズム TSB 全体の概要とメモリ $flag$ の関係を，特に TSB の出力とバリケード b_1 に対応する $flag[1]$ に注目して説明する (説明中の (1)–(3) は図 3 と対応)．また，アルゴリズム TSB の停止判定についての説明でもある．(1) 探索開始時刻，全ての $flag$ を 0 に初期化する．TSB ではまず，探索者は T の次数 3 以上の頂点を発見するまで移動する．もし，T に次数 3 以上の頂点がなければ T の探索は成功する．T の次数 3 以上の頂点を発見すると，そこにバリケード b_1 を設置する．バリケードを設置できない，つまり $r = 0$ であれば T の探索は失敗になる．この説明中 $v(b_1)$ を探索の起点と呼ぶことがある．T の $v(b_1)$ の各ブランチの探索がバリケード $r-1$ 個で十分なら，T の探索は成功する．バリケード $r-1$ 個で探索に失敗する $v(b_1)$ のブランチを発見すると，それを 1 つまで b_1 の $failure$ に記録する．バリケード $r-1$ 個で探索に失敗する $v(b_1)$ のブランチが 1 つだけなら，他のブランチを探索した後で探索に失敗したブランチにバリケード b_1 を持回収して移動する．そして，同様に次数 3 以上の頂点にバリケード b_1 を設置し，各ブランチを探索する．ただし，このとき 1 つ前の探索の起点を含むブランチの辺は全てクリアなので探索しない．このようにして T 内の全ての辺をクリアにできれば探索は成功になる．T の $v(b_1)$ のバリケード $r-1$ 個で探索に失敗する 2 つ目のブランチを発見したら，$flag[1]$ を -1 に変更し，すぐにバリケード b_1 を回収して 2 つ目に失敗したブランチに移動する．$flag[1] = 0$ のとき，T の探索失敗は判断できない．

(2)$flag[1] = -1$ のとき，$v(b_1)$ の各ブランチを探索する．ただし，1 つ前の探索の起点を含むブランチは $r-1$ 個のバリケードで探索に失敗するブランチを含んでいるので探索しない．探索した $v(b_1)$ のブランチに $r-1$ 個のバリケードで探索できないブランチがあれば，すぐに b_1 を持ってそのブランチに移動し，同様に探索する．探索した $v(b_1)$ のブランチが全て $r-1$ 個のバリケードで探索に成功したら，$flag[1]$ を 1 に変更し，$v(b_1)$ の探索が終わっていない，1 つ前の探索の起点を含むブランチに移動する．$flag[1] = -1$ のときは T の探索の成功，失敗は分からない．

(3)$flag[1] = 1$ のとき，探索の仕方は $flag[1] = 0$ と似ている．$flag[1] = 0$ のときと同じ理由で，1 つ前の探索の起点を含む $v(b_1)$ のブランチは探索しない．探索した $v(b_1)$ のブランチが全て $r-1$ 個のバリケードで十分なら T の探索は成功する．$r-1$ 個のバリケードで探索に失敗する $v(b_1)$ のブランチを発見すると，そのブランチを $failure$ に記録する．$r-1$ 個のバリケードで探索に失敗する $v(b_1)$ のブランチが 1 つだけならば，他のブランチを探索後，b_1 を持ってそのブランチに移動し，同様に探索する．2 つ目の $r-1$ 個のバリケードで探索に失敗する $v(b_1)$ のブランチを発見したら，その時点で探索は終了する（T は探索失敗）．T の探索失敗が分かるのは $flag[1] = 1$ のときのみである．

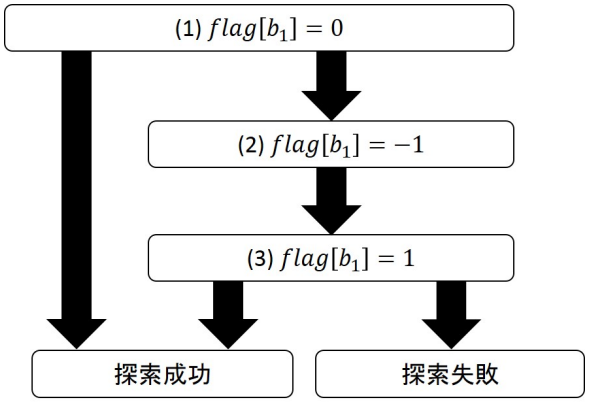

図 3: アルゴリズム TSB の概要 ($flag[1]$ に注目して)

Algorithm 2 TSB(T, s, r)

```
1: ∀i (1 ≤ i ≤ r)flag[i] := 0;  //flag の初期化
2: TREE_SEARCH();
3: //結果の出力
4: if mode = S then
5:     return “探索成功”
6: else if mode = F then
7:     return “探索失敗”
8: end if;
```

また，各ブランチの探索はブランチを 1 つの木とみなして，この説明とほぼ同様にして探索する．異なる部分は $flag$ が 0 のときの動作である．最初に木 T にバリケード b_1 を置いたときは $v(b_1)$ の全てのブランチを探索する．しかし，それ以降バリケードを置いたとき，ブランチの 1 つはその時点で全てクリアかどうか分かっているので，全てのブランチを探索する必要はない．例えば，$v(b_i)$ のブランチ S を探索するとき，S に初めて b_{i+1} を置いたときについて $v(b_i)$ から $v(b_{i+1})$ に移動するまでに通過した辺は既にクリアである．

アルゴリズム TSB(G, s, r) とサブルーチンとして用いる各アルゴリズムの説明と詳細を示す．

アルゴリズム TSB(T, s, r) は木のオンラインアルゴリズムの main 関数である．配列 $flag$ の初期化をし，木全体を探索するアルゴリズム TREE_SEARCH() を呼び出す．TREE_SEARCH() 終了後の探索者の $mode$ の値によって T の探索が成功か失敗か出力する．

アルゴリズム TREE_SEARCH() は，木 T 全体を探索するアルゴリズムである．まず木 T 内の次数 3 以上の頂点に移動するためにアルゴリズム MOVE_TO_DEG3() を呼び出す．もし，T に次数 3 以上の頂点がなければ，$mode$ = S で MOVE_TO_DEG3() は終了する．そして，TREE_SEARCH() は $mode$ = S で終了する．次数 3 以

© 2015 Information Processing Society of Japan

Algorithm 3 TREE_SEARCH()

```
 1: mode := E;
 2: MOVE_TO_DEG3();
 3: if mode = E かつ r = 0 then
 4:     mode := F
 5: else if mode = E then
 6:     //b_1 を設置する
        SET_BARRICADE();
 7:     counter := counter + 1;
 8:     SEARCH_FLAG_NON-NEGATIVE(r)
 9: end if;
10: if mode = E then
11:     辺を通過する;
12:     BRANCH_SEARCH(r)
13: end if;
14: //TREE_SEARCH() 終了時，T の探索成功なら mode = S，そ
    うでないなら mode = F になっている．
```

上の頂点があれば，探索者は最初に発見した次数 3 以上の頂点にいて，$mode$ = E である．アルゴリズム SET_BARRICADE() を呼び出してバリケード b_1 を設置し，メモリ $counter$ の値を 1 増加する．$flag[1] = 0$ なので，アルゴリズム SEARCH_FLAG_NON-NEGATIVE(r) を呼び出し $v(b_1)$ の各ブランチの探索を行う．$v(b_1)$ のブランチ全ての探索が成功なら，SEARCH_FLAG_NON-NEGATIVE(r) 終了時 $mode = \mathrm{S}$ である．1 つでも $r-1$ 個のバリケードで探索に失敗するブランチがあれば $mode = \mathrm{E}$ で SEARCH_FLAG_NON-NEGATIVE(r) は終了する．そして，バリケード $r-1$ 個で探索に失敗したブランチをバリケード r 個で探索するために BRANCH_SEARCH(r) を実行する．

アルゴリズム BRANCH_SEARCH(k) はブランチを k 個のバリケードで探索するアルゴリズムである．まず，アルゴリズム MOVE_TO_DEG3() で次数 3 以上の頂点に移動する．次数 3 以上の頂点を発見した後，バリケードを持っていれば設置する．設置したバリケード b_{r-k+1} の $flag[r-k+1]$ が -1 であれば，アルゴリズム SEARCH_FLAG_NEGATIVE(k) を呼び出す．$flag[r-k+1]$ が 0 または 1 であれば，アルゴリズム SEARCH_FLAG_NON-NEGATIVE(k) を呼び出す．呼び出したアルゴリズムが終了後 $mode = \mathrm{E}$ の場合，$v(b_{r-k+1})$ には探索が終わっていないブランチがあるので，b_{r-k+1} を回収しそのブランチを探索する（探索者は既に進むべき方向を向いている）．$mode \neq \mathrm{E}$ の場合，$v(b_{r-k})$ のブランチの 1 つもしくは T の探索結果が分かったことになる．b_{r-k+1} を回収し，$k \neq r$ なら $v(b_{r-k})$ まで移動する．

SEARCH_FLAG_NON-NEGATIVE(k) は $flag[r-k+1] = 0$ または 1 のときに呼び出される．$v(b_{r-k+1})$ の各ブランチをアルゴリズム BRANCH_SEARCH($k-1$) で探索する．もし，バリケード $k-1$ 個で探索に失敗する $v(b_{r-k+1})$ のブランチを発見したら，$v(b_{r-k+1})$ に戻った後で $failure$ に $counter$ の値を書き込み，残りのブランチの探索を続ける．ただし，既に $failure$ に 0 以外の値が書き込まれていたら，バリケード $k-1$ 個で探索に失敗する 2 つ目のブランチを発見したことになる．このとき，$flag[r-k+1] = 0$ なら $flag[r-k+1] = -1$ に，$mode$ を E に変更し，2 つ目に探索に失敗したブランチの方向を向いて終了する．$flag = 1$ なら今探索者がいる $v(b_{r-k})$ のブランチの探索はバリケード k 個で失敗することが分かるので，$mode$ を F にし終了する．$v(b_{r-k+1})$ の探索すべきブランチを全て探索した後 (つまり探索者が $v(b_{r-k+1})$ に戻ってきたとき $counter = 1$ の場合)，$failure = 0$ なら $v(b_{r-k+1})$ のブランチは全て探索成功している．つまり，$v(b_{r-k})$ の探索者がいるブランチは探索成功になるので $mode$ を S に変更し終了する．$failure \neq 0$ なら $k-1$ 個のバリケードで探索に失敗した $v(b_{r-k+1})$ のブランチが 1 つだけあるので，$mode$ を E にし，そのブランチの方向を向いて終了する．

Algorithm 4 BRANCH_SEARCH(k)

```
 1: mode := E;
 2: MOVE_TO_DEG3();
 3: if mode = E かつ k = 0 then
 4:     mode := F
 5: end if;
 6: while mode = E do
 7:     //b_{r-k+1} を設置する
        SET_BARRICADE();
 8:     if flag[r-k+1] = -1 then
 9:         SEARCH_FLAG_NEGATIVE(k)
10:     else if flag[r-k+1] = 0 または 1 then
11:         SEARCH_FLAG_NON-NEGATIVE(k)
12:     end if;
13:     if mode = E then
14:         b_{r-k+1} を回収する;
15:         辺を通過する;
16:         MOVE_TO_DEG3()
17:     else
18:         flag[r-k+1] := 0;
19:         b_{r-k+1} を回収する;
20:         if k ≠ r then
21:             v(b_{r-k}) に到達するまで移動する
22:         end if;
23:     end if;
24: end while;
25: //このとき，探索成功なら mode = S，そうでないなら mode = F
    になっている．
```

SEARCH_FLAG_NEGATIVE(k) は $flag[r-k+1] = -1$ のときに呼び出される．$v(b_{r-k+1})$ を探索の起点と呼ぶこともある．$v(b_{r-k+1})$ の各ブランチをアルゴリズム BRANCH_SEARCH($k-1$) で探索する．もし，バリケード $k-1$ 個で探索に失敗するブランチがあれば，$v(b_{r-k+1})$ に戻った後で $mode$ を E に変更し，探索に失敗したブランチの方向を向いて終了する．1 つ前の探索の起点を含むブランチ以外を全て $k-1$ 個で探索に成功したら，$flag[r-k+1]$ を 1 に，$mode$ を E に変更し，1 つ前の探索の起点を含む

© 2015 Information Processing Society of Japan

Algorithm 5 SEARCH_FLAG_NON-NEGATIVE(k)

```
 1: while counter > 1 do
 2:    SELECT_EDGE(1) の後，辺を通過する;
 3:    BRANCH_SEARCH(k − 1);
 4:    if mode = F then
 5:       if failure = 0 then
 6:          failure := counter
 7:       else if failure ≠ 0 then
 8:          if flag[r − k + 1] = 0 then
 9:             flag[r − k + 1] := −1;
10:             mode := E;
11:             SELECT_EDGE(0)
12:          else if flag[r − k + 1] = 1 then
13:             mode := F;
14:             SELECT_EDGE(counter)
15:          end if;
16:          break
17:       end if;
18:    end if;
19:    if counter = 1 then
20:       if failure = 0 then
21:          mode := S;
22:          SELECT_EDGE(1)
23:       else
24:          SELECT_EDGE((今いる頂点の次数)−failure + 1);
25:          mode := E
26:       end if;
27:    end if;
28: end while;
```

Algorithm 6 SEARCH_FLAG_NEGATIVE(k)

```
 1: while counter > 1 do
 2:    SELECT_EDGE(1) の後，辺を通過する;
 3:    BRANCH_SEARCH(k − 1);
 4:    if mode = F then
 5:       flag[r − k + 1] := −1;
 6:       SELECT_EDGE(0);
 7:       mode := E;
 8:       break
 9:    else if mode = S かつ counter = 1 then
10:       flag[r − k + 1] := 1;
11:       mode := E;
12:       SELECT_EDGE(1)
13:    end if;
14: end while;
```

ブランチの方向を向いて終了する．

アルゴリズム MOVE_TO_DEG3() はブランチ内の次数 3 以上の頂点に移動するアルゴリズムである．もし，ブランチ内に次数 3 以上の頂点がないことが分かったとき，既にブランチ内は探索済みであるので $mode$ を S に変更し終了する．次数 3 以上の頂点を発見するとアルゴリズムは終了する．

アルゴリズム SET_BARRICADE() はバリケードを設置し，バリケードのメモリを初期化するアルゴリズムである．探索者がバリケードを 1 つ以上持っているときのみ呼び出される．

Algorithm 7 MOVE_TO_DEG3()

```
 1: //次数 3 以上の頂点が見つかる前に異なる 2 つの次数 1 の頂点
    を見つければ T の最高次数は 2 以下である
 2: deg1 := 0;
 3: while 今いる頂点の次数が 1 または 2，かつ deg1 < 2 do
 4:    deg1 := deg1 + 2−(今いる頂点の次数);
 5:    SELECT_EDGE(1);
 6:    辺を通過する
 7: end while;
 8: if deg1 ≥ 2 then
 9:    mode := S
10: end if;
```

Algorithm 8 SET_BARRICADE()

```
1: バリケードを設置する;
2: k := k − 1;
3: counter に今いる頂点の次数を書き込む;
4: failure := 0;
5: name := r − k;
```

4. アルゴリズム TSB の解析

観察 1. オンラインアルゴリズム TSB から分かることを以下にまとめる．

- TSB で探索失敗が分かるのは，$flag[1] = 1$ のときに，バリケード $r-1$ 個で探索に失敗する探索が終わっていない $v(b_1)$ のブランチを 2 つ発見したときである．このとき，$v(b_1)$ のブランチには全ての辺がクリアになっているものが少なくとも 1 つあり，そのうちの 1 つは $r-1$ 個のバリケードでは探索できないブランチである．よって，バリケード $r-1$ 個で探索に失敗する $v(b_1)$ のブランチは少なくとも 3 つ存在する (図 4(a))．
- TSB で探索成功したとき，木の形は以下の 2 つに分類される．

  (i) 与えられた開始位置を v_0 とすると，v_0 のブランチが全て $r-1$ 個のバリケードで探索に成功する (図 4(b))．また，この場合 $flag[1] = 0$ である．

  (ii) 以下のように道 $\langle v_1, v_2, ..., v_p \rangle$ を定義する (図 4(c))．TSB でバリケード b_1 が最後に設置されていた頂点を v_p とする．$flag[1] = 0$ で探索成功が分かった場合，入力で与えられた開始位置を v_1 とする（ただし，$v_1 \neq v_p$）．$flag[1] = 1$ で探索成功が分かった場合，$flag[1]$ が 1 に切り替わった頂点を v_1 とする．v_1 と v_p をつなぐ道を $\langle v_1, v_2, ..., v_p \rangle$ とする．このとき，v_i $(2 \leq i \leq p-1)$ の v_1 または v_p を含まないブランチは，バリケード $r-1$ 個で探索可能である．v_1 の v_p を含まないブランチ，v_p の v_1 を含まないブランチもバリケード $r-1$ 個で探索可能である．

TSB で探索失敗が出力されたとき，探索開始位置 s に他の頂点を与えれば探索に成功するのではないかと予想できる．しかし，このアルゴリズムの出力結果は開始位置に

© 2015 Information Processing Society of Japan

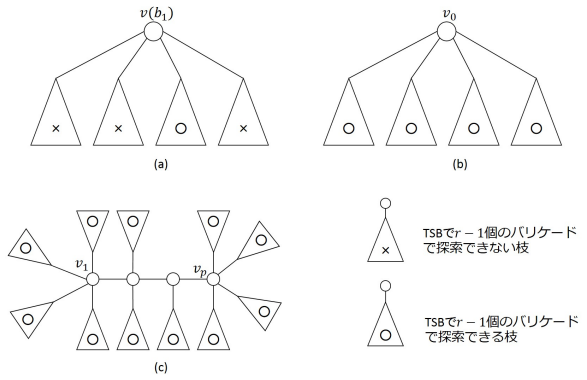

図 4: TSB の実行結果

よって変化しないことを示す．

補題 2. オンラインアルゴリズム TSB の出力は探索開始時刻の探索者の位置によって変化しない．

証明のアイデアは，TSB でバリケード r 個で探索に失敗した木に観察 1 の (i) の v_0 のような頂点や観察 1 の (ii) の $\langle v_1, v_2, ..., v_p \rangle$ のような道が存在すると仮定し矛盾を導く．

オンラインアルゴリズム TSB と他の任意のオフラインのアルゴリズムとの関係について次の定理 2 が成り立つ．

定理 2. TSB で r 個のバリケードで T の探索に失敗するならば，どのようなオフラインのアルゴリズムでも r 個のバリケードで T の探索に失敗する．

証明. 証明の概略のみ示す．数学的帰納法で証明する．

$r = 0$ のとき，TSB を用いてバリケードなしで探索に失敗した木 T を任意に 1 つ選ぶ．このとき，T には次数 3 以上の頂点が含まれているので，どのようなアルゴリズムでもバリケードなしで探索することはできない．

$r = k - 1$ で命題成立を仮定し，$r = k$ のときを考える．TSB で $r = k$ で失敗した木 T を任意に選ぶ．TSB で最後に b_1 を設置していた頂点を u とする．観察 1 より，バリケード $k - 1$ 個で探索に失敗する u のブランチが少なくとも 3 つ存在する．バリケード k 個で T を探索するときそのような 3 つのブランチの全ての辺が同時にクリアになることはない（証明は省略）．よって，T を k 個のバリケードで探索できるアルゴリズムは存在しない． □

定理 2 より，TSB は任意の木について，与えられたバリケードが $r(T)$ 個以上であれば，正しく探索し，$r(T)$ 個未満であれば探索に失敗するアルゴリズムであると言える．

定理 3. オンラインアルゴリズム TSB の実行時間は，探索者の辺の通過を 1 ステップとして計算する．木の頂点数を n，バリケード数を r とすると，$r = 0$ のとき $O(n)$ 時間，$r \geq 1$ のとき $O(n^r)$ 時間で実行できる．

TSB はバリケードを少しずつ移動しながら探索するアルゴリズムである．次数 3 以上の頂点をの個数を $m(= O(n))$ とすると，$\binom{m}{r} = O(n^r)$ 通りのバリケードの置き方を試していると考えることができる．

5. おわりに

本稿では一人の探索者がいくつかのバリケードを用いてグラフを探索するバリケードモデルを提案し，それを用いて木のオンライン探索アルゴリズム TSB を設計した．そして，アルゴリズム TSB で与えられた木 T をバリケード r 個で探索できなかったとき，他のどのようなオフラインのアルゴリズムでも T をバリケード r 個で探索できないことを示した．今後は一般のグラフもしくは特定のグラフクラスについて，オンラインアルゴリズムを設計し，与えられたグラフについてオフライン探索とオンライン探索で必要な最小のバリケード数を比較する．そして，もし両者が必ずしも一致するとは限らないのであれば，一致するの条件や近似比を求めることを目標とする．

参考文献

[1] 山下雅史 ,「捜索—移動する対象を探索する」, 室田一雄編, 『離散構造とアルゴリズム III』, 近代科学社, pp. 115–162, 1994.

[2] A. Takahashi, S. Ueno and Y. Kajitani. “Mixed searching and proper-path-width”, Theoretical Computer Science 137.2, pp. 253–268, 1995.

[3] Bienstock, Daniel and Paul Seymour. “Monotonicity in graph searching”, Journal of Algorithms 12.2, pp. 239–245, 1991.

[4] I. Suzuki and M. Yamashita. “Searching for a mobile intruder in a polygonal region”, SIAM Journal on computing 21.5, pp. 863–888, 1992.

[5] N. Megiddo, S. L. Hakimi, M. R. Garey, D. S. Johnson, and C. H. Papadimitriou, “The complexity of searching a graph”, Journal of the ACM 35.1, pp. 18–44, 1988.

[6] M. Yamashita, H. Umemoto, I. Suzuki and T. Kameda, “Searching for Mobile Intruders in a Polygonal Reagion by Group of Mobile Searchers”, Algorithmica, pp. 208–236, 2001.

[7] T. D. Parsons, “Pursuit-evasion in a graph”, In *Proceedings of the Theory and Applications of Graphs*, pp. 426–441, 1976.

© 2015 Information Processing Society of Japan