2023 年度 卒業論文

橋を架けろの組合せゲーム理論に基づく調査

岐阜大学 工学部 電気電子・情報工学科 草刈研究室

1203033037　内田宗明

指導教員　草刈圭一朗教授

2024 年 2 月 8 日

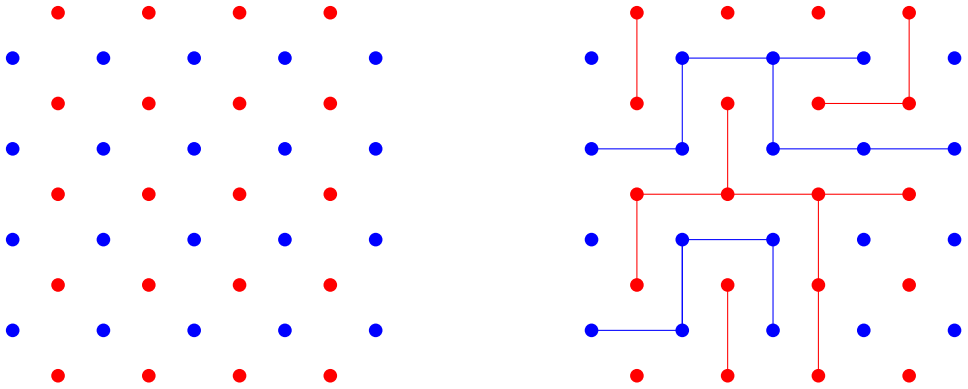
目次

第 1 章　はじめに

最初に，本研究で扱う橋を架けろについて紹介し，本論文の概要を紹介する．

1.1　橋を架けろ (Bridg-It)

「橋を架けろ (Bridg-It)」とは，1960 年にハズブロ (Hasbro) によって販売されたボードゲームである．二人の対局者 (二人の対局者を左と右と呼ぶ) が図 1.1a のような格子状に支柱を配置した盤上で行う．図中で青点が左対局者の支柱，赤点が右対局者の支柱である．各対局者は隣接する自分の支柱間に橋ユニットを交互に架橋する．図中では，左対局者の架橋した橋ユニットを青，右対局者の架橋した橋ユニットを赤で表している．このとき，既に架橋しているところに重ねたり，交差して架橋することはできない．左対局者は，盤の左端から右端まで橋 (連続した橋ユニットの道) を架ければ勝利となる．同様に，右対局者は盤の上端から下端まで橋を架ければ勝利となる．以下の図 1.1b は，左対局者 (青) が勝利局面となる例である．



(a) 橋を架けろの盤　　**(b)** 左対局者の勝利局面

図 1.1: 局面の例

1.2　本研究の目標

現在，「橋を架けろ」は全域木戦略と呼ぶことにする全域木という概念を用いた必勝法が発見されている [3]．全域木戦略は左対局者が初期局面からの実行をするときのみを考慮した戦略であることから，途中の局面からは実行できず，右対局者は実行できない．本研究では，任意の局面に対応し，両対局者とも実行できるよう拡張した随意全域木戦略を提案する．また，随意全域木戦略は任意の局面で必勝戦略でありどちらかの対局者が実行できると予想する．この系として引き分けが存在しないことが導ける．

随意全域木戦略が与える選択肢は複数存在することがある．選択肢の優劣を判別するために優劣判別関数を導入し，優劣判別手法を提案する．

「橋を架けろ」のようなゲームのことを組合せゲームと呼ぶ．最後に手を打った対局者が勝利し，手の打てなかった対局者が敗北するようにすることで正規形組合せゲームとして定式化した仕様を CGSuite 上に実装し，提案した判別手法の正当性を組合せゲーム理論に基づき調査する．

1.3　本論文の構成

本論文の構成は次の通りである．

2 章では，本研究で取り扱う組み合わせゲーム理論と用語の定義や定理，CGSuite について紹介する．

3 章では，「橋を架けろ」の規則と，それと矛盾することなく組合せゲーム理論に基づいて作成した仕様について説明する．

4 章では，CGSuite を用いた実装について紹介する．

5 章では，既知の必勝法である全域木戦略を説明する．

6 章では，全域木戦略を任意の局面に対応し，両対局者とも実行できるよう拡張した随意全域木戦略を提案する．

7 章では，選択肢の優劣を判別するために優劣判別関数を導入し，優劣判別手法を提案する．

第 2 章　組合せゲーム

本章では，本研究で取り扱う組合せゲームと必要となる組合せゲーム理論の定義，定理を紹介する [1].

2.1　組合せゲーム

本研究で取り扱う組合せゲームとは，二人零和有限確定完全情報ゲームのことである．

二人: 二人の対局者が交互に手を打つゲーム．
零和: 利得の総和が常にゼロになるゲーム．すなわち，ある対局者が利得を得ると，それと同量の損失を他の対局者が被ることになるゲーム．
有限: 次の二つの意味で有限なゲーム:

1. 有限の手数によって必ずゲームが終了する．
2. どの局面からでも着手可能な手は有限通りしかない．

確定: サイコロを振るなどの偶然要素を含まないゲーム．
完全情報: ゲームの局面や状態が全て公開されているゲーム．

また，組み合わせゲームには，次の重要な 2 つのクラスが存在する．

正規形: 最後に着手したほうが勝ちとなるゲーム．
逆形: 最後に着手したほうが負けとなるゲーム．

さらに，組み合わせゲームは対局者による選択肢の偏りに関して次の 2 つに分けられる．

不偏ゲーム: 対局者によって着手可能な手に違いのないゲーム．
非不偏ゲーム: 対局者によって着手可能な手が異なることもあるゲーム

「橋を架けろ」は最後に着手したほうが勝利であり，対局者によって着手可能な手が異なることもあることから，正規形の非不偏ゲームである．

2.2　組合せゲーム理論

組合せゲーム理論とは組合せゲームを解析する理論である．以下で本研究で使用する組合せゲーム理論の定義を紹介する．

定義 2.1 (局面値)
ゲームの局面 G の局面値とは，左が手を打った後の局面の局面値全体からなる集合 $\mathcal{G}^L$ と，右が手を打った後の局面の局面値全体からなる集合 $\mathcal{G}^R$ を用いて次のように表す．

$$\{\mathcal{G}^L|\mathcal{G}^R\}$$

以降では混乱のない限り，局面と局面値を同一視し以下の記述を許す．

$$G = \{\mathcal{G}^L|\mathcal{G}^R\}$$

また，$\mathcal{G}^L$ に含まれる局面を左選択肢と呼び，$\mathcal{G}^R$ に含まれる局面を右選択肢と呼ぶ．

定義 2.2 (終局値)
ゲーム G の左終局値および右終局値とは，それぞれ次の式で互いに再帰的に定義される $LS(G)$ および $RS(G)$ のことである．

$$LS(G) = \begin{cases} G & G \text{ が数のとき} \\ max\{RS(G^L)\} & G \text{ が数でないとき} \end{cases}$$
$$RS(G) = \begin{cases} G & G \text{ が数のとき} \\ min\{LS(G^R)\} & G \text{ が数でないとき} \end{cases}$$

定義 2.3 (数)
局面値には 2 進有理数を割り当てることができ，正の整数 n，m と正の奇数 j を用いて以下のように定義する．

$$\begin{aligned} 0 &\stackrel{\text{def}}{=} \{|\} \\ n &\stackrel{\text{def}}{=} \{n-1|\} \\ -n &\stackrel{\text{def}}{=} \{|-n+1\} \\ \frac{m}{2^j} &\stackrel{\text{def}}{=} \{\frac{m-1}{2^j}|\frac{m+1}{2^j}\} \end{aligned}$$

2 進有理数が割り当てられた局面のことを数と呼ぶ．

定義 2.4 (ゲームの帰結類)
ゲームの局面は先手必勝局面，後手必勝局面，左必勝局面，右必勝局面の 4 つに分類でき，それぞれ以下で定義される．

先手必勝局面 G を初期局面と考えたときの先手番が必ず勝つ．
後手必勝局面 G を初期局面と考えたときの後手番が必ず勝つ．
左必勝局面 G を初期局面とすると，どちらが先手でも左が必ず勝つ．
右必勝局面 G を初期局面とすると，どちらが後手でも右が必ず勝つ．

また，これらを帰結類と呼ぶ．

定義 2.5 (ゲームの比較)
ゲーム G の 0 との比較を以下のように定義する．

$$\begin{aligned} G > 0 &\stackrel{\text{def}}{\iff} G\text{は左必勝} \\ G < 0 &\stackrel{\text{def}}{\iff} G\text{は右必勝} \\ G = 0 &\stackrel{\text{def}}{\iff} G\text{は後手必勝} \\ G \parallel 0 &\stackrel{\text{def}}{\iff} G\text{は先手必勝} \end{aligned}$$

ここで，$\parallel$ は比較不能を表す．

定義 2.6 (選択肢の優劣)
ゲーム $G = \{\mathcal{G}^L|\mathcal{G}^R\}$ を考える.

- 左選択肢 G_1^L, $G_2^L \in \mathcal{G}^L$ に対して $G_1^L \leq G_2^L$ となるとき, G_1^L が G_2^L より劣位な選択肢であるという.
- 右選択肢 G_1^R, $G_2^R \in \mathcal{G}^R$ に対して $G_1^R \geq G_2^R$ となるとき, G_1^R が G_2^R より劣位な選択肢であるという.

ここで, G が H より劣位な選択肢のとき, H は G より優位な選択肢とも呼ぶ.

2.3　CGSuite

定式化した仕様を CGSuite 上に実装し,「橋を架けろ」を調査する. ここで, CGSuite とは組合せゲーム理論における様々な代数計算を行えるソフトウェアである [1]. 第 4 章では GridGame という用意されているクラスを継承し, CGSuite の言語を用いて「橋を架けろ」の仕様記述を作成する.

第 3 章　仕様

「橋を架けろ」の仕様を定式化し，組合せゲーム理論として再定式化する．

3.1　橋を架けろの規則

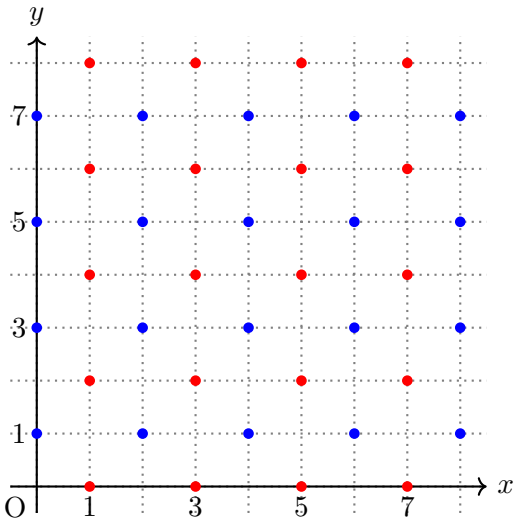


図 3.1: 座標上で考えた初期局面

- ゲームの局面
  - 盤

    初期盤は図 3.1 のように格子上に設置された支柱のみで構成されている．座標に着目すると (偶数, 奇数) には青，(奇数, 偶数) には赤の支柱が設置されている．

    この資料では，8×8 の格子上に支柱が設置されている盤 (各色の支柱に着目すると 4×5) を基本として考える．
  - 支柱

    青点が左対局者の支柱，赤点が右対局者の支柱である．
    - 隣接支柱

      ある点の座標を (x, y) としたとき，もう一つの点の座標が $(x, y+2), (x, y-2), (x+2, y)$, または $(x-2, y)$ のとき隣接支柱と呼び，それら二つの支柱は隣接しているという．
  - 橋ユニット

    隣接支柱の間には橋ユニットを架橋することができる．橋ユニットは各隣接支柱の間に一本しか架橋することはできず，すでに架橋されている橋ユニットの上に重ねたり，交差して架橋することはできない．
  - 勝利局面

    盤の左 (上) 端から右 (下) 端まで橋 (連続した橋ユニットの道) を架けている局面を勝利局面と呼ぶ．ここにおける道とは，支柱と橋ユニットを交互に並べた列のことである．

- 着手可能手
  組み合わせゲーム理論では伝統的に先手番を左手側，後手番を右手側と呼ぶ.
  - 左手側
    青の隣接支柱の間にひとつ橋を配置する.
  - 右手側
    赤の隣接支柱の間にひとつ橋を配置する.

  以上の手を交互に打つ.
- 勝利条件
  先に勝利局面となった対局者の勝利.

3.2　組合せゲームとしての仕様

橋を架けろ (Bridg-It) の本来の規則を正規形組み合わせゲームとして再定式化する.

1. 本来の規則と矛盾しないことを前提とする.

組み合わせゲームとするために,

2. 片方の対局者が連続して手を打つような本来ない選択肢を追加する.

正規形とするために,

3. 最後に手を打った対局者が勝利し，手の打てなかった対局者が敗北するようにする.

本研究では，盤の形状や支柱の数を変更する可能性も考慮し，支柱が任意の配置，個数にも適応できるよう定式化する.

本来の着手可能手をそのまま正規形組み合わせゲームの規則に反映しようと試みる.

そのようにして橋を架けろを正規形組み合わせゲームとして再定式化した仕様を以下に示す.

仕様

- ゲームの局面
  - 盤

    盤は，青と赤の支柱が平面 xy 座標の格子上に設置されている．青は (偶数, 奇数)，赤は (奇数, 偶数) にのみ設置される．
  - 支柱

    支柱を役割に応じて二種類に分類し名前をつけ，扱う．
    - 端支柱

      勝利の条件となる支柱.

      端支柱は，1 と 2 で区別されるが役割は同じである．
    - 経由支柱

      端支柱以外の支柱.

    支柱には隣接するという位置関係が存在する．
    - 隣接支柱

      ある支柱の座標を (x, y) とすると，座標が $(x, y+2), (x, y-2), (x+2, y)$, または $(x-2, y)$ のとき隣接支柱と呼び，それら二つの支柱は隣接しているという．
  - 橋ユニット

    隣接支柱の間には橋ユニットを架橋することができる．橋は各隣接支柱の間に一本しか架橋することはできず，すでに架橋されている橋の上に重ねたり，交差して架橋することはできない．
  - 勝利局面

    端支柱 1 と端支柱 2 の間に橋 (連続した橋ユニットの道) を架けている局面を勝利局面と呼ぶ．ここにおける道とは，支柱と橋ユニットを交互に並べた列のことである．
- 選択肢
  - 左手側

    右が勝利局面でなければ, 青の隣接支柱の間にひとつ橋ユニットを架橋することができる.
  - 右手側

    左が勝利局面でなければ, 赤の隣接支柱の間にひとつ橋ユニットを架橋することができる.

3.3　橋を架けろにおける局面の優劣

局面に対する代数的な優劣を導入することで，選択肢に明確な優劣を与える事ができる．

定義 3.1

局面の優劣を局面値の比較演算に従うと定義する．局面値が比較不能であるとき終局値と呼ばれる近似値を用いて比較を行う．

第 4 章　CGSuite を用いた実装

CGSuite に用意されているクラス GridGame を継承し，抽象アルゴリズムに従って CGSuite の言語を用いて仕様記述を実装する．

4.1　CGSuite 上で扱うアイコン

局面を与える文字と入された文字に対応するアイコンの定義を以下の仕様記述で与える．

```
1  override def CharMap := ".ox/-=abcdt"; //ゲームの局面を与える文字の定義
2
3  override def Icons := //ゲームの局面を可視化したときに表示するアイコンの定義
4  [
5    Icon.Blank,          //0[.]. 空白
6    Icon.BlackStone,     //1[o]. 左の経由点
7    Icon.WhiteStone,     //2[x]. 右の経由点
8    Icon.GraySquare,     //3[/]. 未架橋
9    Icon.BlackSquare,    //4[-]. 左の橋ユニット
10   Icon.RedSquare,      //5[=]. 右の橋ユニット
11   Icon.BlackKing,      //6[a]. 左の端支柱 1
12   Icon.BlackKing,      //7[b]. 左の端支柱 2
13   Icon.WhiteKing,      //8[c]. 右の端支柱 1
14   Icon.WhiteKing,      //9[d]. 右の端支柱 2
15   Icon.BlackRook       //10[t]. 探索済用
16 ];
17 end
```

2 行目のように CharMap を用いることで局面を与える文字を定義できる．入力された文字がプログラム内で使用される値となる．

5 から 19 行目のように Icons を用いることで CharMap で定義した文字に対するアイコンを定義できる．

まとめると以下のようになる．

入力文字	アイコン	ゲームでの仕様	プログラム内で使用される値
.	Blank	空白	0
o	BlackStone	左の経由点	1
x	WhiteStone	右の経由点	2
/	GraySquare	未架橋の支柱間	3
-	BlackSquare	左の橋ユニット	4
=	RedSquare	右の橋ユニット	5
a	BlackKing	左の端支柱 1	6
b	BlackKing	左の端支柱 2	7
c	WhiteKing	右の端支柱 1	8
d	WhiteKing	右の端支柱 2	9
t	BlackRook	探索済み用	10

以上の Icon を用いて CGSuite 上で以下のコードを打つことで変数 G に局面を与え，Explorer に表示する．Icon を定義したクラスを BridgIt とすると

G:=BridgIt(".c.c.c.c.|a/o/o/o/b|.x/x/x/x.|a/o/o/o/b|.x/x/x/x.|a/o/o/o/b|.x/x/x/x.|a/o/o/o/b|.d.d.d.d.")
ex:=Explorer.NewExplorer(G)

実際に CGSuite に出力した結果は以下のようになる．

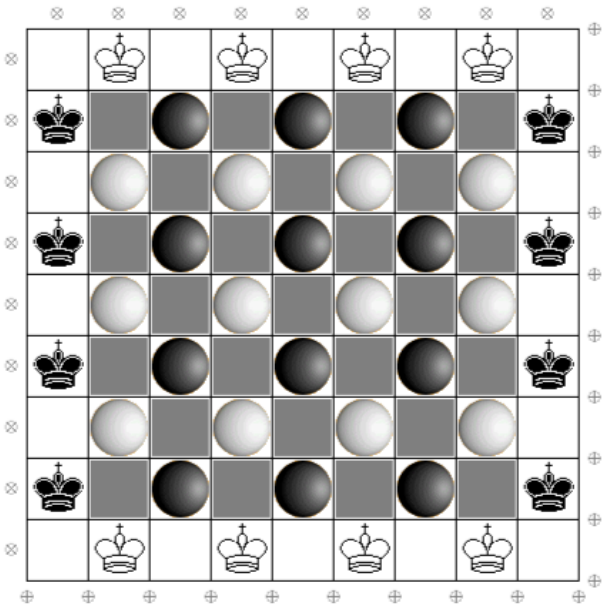


図 4.1: 8 × 8の初期盤

4.2　仕様に基づくアルゴリズム

勝利局面の判定に用いる勝利局面判定関数の抽象アルゴリズムを以下に示す.

勝利局面の判定抽象アルゴリズム

局面は (Row,Column) の二次元配列 grid に保存されているものとする
/*勝利局面の判定*/
勝利局面を表す初期値 false である変数を用意する．(ここでは win)
grid のコピーである変数 copy を用意する
for 各直交座標を変数 direction に代入する **do**
　if grid([y,x]+direction) にあるのが自分の橋ユニット **then**
　　if grid([y,x]+direction) の先にあるのが探索していない経由支柱 **then**
　　copy の ([y,x]+direction) の先を探索済みに変更
　　if win == false **then**
　　win:= 橋ユニットの先の支柱について再帰呼び出し;
　　else if grid([y,x]+direction) の先にあるのが端支柱 **then**
　　win:= true;
　　break;
return win;

勝利局面判定関数を用いて，選択肢を返す抽象アルゴリズムを以下に示す．

選択肢を求めるアルゴリズム

選択肢を保存するための初期値が空である変数 options を用意する
勝利局面判定関数を左と右それぞれ win_L,win_R とする
/*選択肢*/
勝利局面を表す初期値 false である変数を用意する．(winL,winR)
if 左手番 **and** winR == false **then**
　for 全ての左の端支柱の座標を変数 coord に代入 **do**
　　if win_R(grid) == true **then**
　　　winR := true;
　if winR == false **then then**
　　ゲームの規則と同様の手を選択肢として options に追加する
elseif 右手番 **and** winL == false **then**
　for 全ての右の端支柱の座標を変数 coord2 に代入 **do**
　　if win_L(grid) == true **then**
　　　winL := true;
　if winL == false **then**
　　ゲームの規則と同様の手を選択肢として options に追加する
return options;

以上の抽象アルゴリズムに基づいて CGSuite に実装する．

CGSuite に用意されているクラス GridGame を継承し，CGSuite の言語を用いて「橋を架けろ」を作成する．その際，端支柱同士の間に橋ユニットを架橋することは何もせず相手に手番を渡すのと同等であると考え，端支柱同士の間に橋ユニットを架橋するという選択肢を考慮せず，実装する．

```
1   // GridRuleset のサブクラスとして Bridgit を定義する。
2   singleton class BridgIt extends game.grid.GridRuleset
3
4   // 特定の位置を表すネストされたクラスを定義する
5   mutable class Position(grid as Grid) extends GridGame
6
7       mutable var copy := grid;//grid のコピーを用意する
8
9       //勝利局面判定に用いる関数を定義
10      //左の勝利局面判定関数
11      def win_L(y as Integer,x as Integer) begin
12          var L := false;
13          for direction in Coordinates.Orthogonal do //各直交座標について
14              if copy[(y,x) + direction] == 4 //左の橋ユニットだったら
15                  then if copy[(y,x) + direction*2] == 1 //左の経由支柱だったら
```

```
16                          then copy := copy.Updated((y,x) + direction*2,10); //その
                            ↪ 支柱を探索済みに変更
17                                if L == false
18                                  then L :=
                                    ↪ win_L(y+direction.Row*2,x+direction.Col*2);
                                    ↪ //橋ユニットの先の支柱について再帰呼び出し
19                                end
20                          else if copy[(y,x) + direction*2] == 7 //左の端支柱だっ
                            ↪ たら
21                                  then L := true; //変数を true に
22                                end
23                      end
24              end
25          end
26          L //true or false を返す
27      end
28
29      //右の勝利局面判定関数
30      def win_R(y as Integer,x as Integer) begin
31          var R := false;
32          for direction in Coordinates.Orthogonal do //各直交座標について
33                      if copy[(y,x) + direction] == 5 //右の橋ユニットだったら
34                          then if copy[(y,x) + direction*2] == 2 //右の経由支柱だっ
                            ↪ たら
35                                  then copy := copy.Updated((y,x) +
                                    ↪ direction*2,10); //その支柱を探索済みに変更
36                                       if R == false
37                                          then R :=
                                            ↪ win_R(y+direction.Row*2,x+direction.Col*2);
                                            ↪ //橋ユニットの先の支柱について再帰呼び
                                            ↪ 出し
38                                       end
39                                  else if copy[(y,x) + direction*2] == 9 //右の端支
                                    ↪ 柱だったら
40                                          then R := true; //変数を true に
41                                       end
42                                end
43                      end
44          end
45          R //true or false を返す
```

```
46        end
47
48        // このポジションのオプションを定義する
49        override def Options(player as Player) begin
50
51            var us := player.Ordinal;//各左の支柱
52            var them := player.Opponent.Ordinal;//各右の支柱
53
54            var options := {}; //選択肢を入れる空のリストを用意
55
56            var winL := false;    //左の勝利局面を表す変数
57            var winR := false;    //右の勝利局面を表す変数
58
59            //選択肢
60            if player == Player.Left //左手番だったら
61                then if winR == false //右の勝利局面でなかったら
62                        then for coord2 in (grid FindAll 8) do //全ての左の端支柱と一致
                          ↪ する各座標について
63                                copy:= grid; //copy に一時保管
64                                var r := coord2.Row;
65                                var c := coord2.Col;
66                                if win_R(r,c) == true //右の勝利局面だったら
67                                    then winR := true; //右の勝利局面とする
68                                end
69                             end
70                    end
71                    if winR == false //右の勝利局面でなかったら
72                    then for coord3 in grid FindAll 1 do  //左の経由支柱と一致する各座
                      ↪ 標について
73                            for direction in Coordinates.Orthogonal do//そして各直交方
                              ↪ 向について
74                                if (grid[coord3 + direction] == 3) //もし未架橋だっ
                                  ↪ たら
75                                    then copy:= grid; //copy に一時保管
76                                         copy := copy.Updated(coord3 + direction,4);
                                           ↪ //copy の未架橋を左の橋ユニットに変える
77                                         options := options.Adjoin(BridgIt(copy));
                                           ↪ //copy の局面を options に格納
78                                end
79                            end
```

```
80                          end
81                      end
82                  else if player == Player.Right //右手番だったら
83                          then if winL == false //左の勝利局面でなかったら
84                                  then for coord1 in (grid FindAll 6) do //全ての右の端
                                    ↪ 支柱と一致する各座標について
85                                      copy:= grid; //copy に一時保管
86                                      var r := coord1.Row;
87                                      var c := coord1.Col;
88                                      if win_L(r,c) == true //左の勝利局面だったら
89                                          then winL := true; //左の勝利局面とする
90                                      end
91                                  end
92                              end
93                              if winL == false //左の勝利局面でなかったら
94                                  then for coord4 in grid FindAll 2 do //右の経由支柱と
                                    ↪ 一致する各座標について
95                                      for direction in Coordinates.Orthogonal do
                                        ↪ //そして各直交方向について
96                                          if (grid[coord4 + direction] == 3) //もし
                                            ↪ 未架橋だったら
97                                              then copy:= grid; //copy に一時保管
98                                                   copy := copy.Updated(coord4 +
                                                     ↪ direction,5); //copy の未架橋
                                                     ↪ を右の橋ユニットに変える
99                                                   options :=
                                                     ↪ options.Adjoin(BridgIt(copy));
                                                     ↪ //copy の局面を options に格納
100                                         end
101                                     end
102                                 end
103                             end
104                     end
105             end
106             options
107         end
108     end
```

第 5 章　全域木戦略

橋を架けろは左必勝であり，左の応手として定式化される全域木を用いた必勝戦略が知られている [3]. この戦略を全域木戦略と呼ぶことにする．以下，全域木戦略に必要な知識および必勝手順を示す.

5.1　全域木

連結グラフ G が与えられたとき，閉路を選びその 1 本の辺を除去しても，残りのグラフは連結である．閉路がなくなるまでこのことを続けると，残るグラフは木であり，G のすべての点を連結している．これを G の全域木という [2].

5.2　全域木の交換可能性

全域木戦略に用いる 2 つの全域木に関して，必勝戦略の鍵を握る辺 e, e' に関する命題を紹介する.

命題 5.1
T, T' という 2 つの全域木が連結グラフ G に含まれていて，$e \in E(T) - E(T')$ であるとき，$T - e + e'$ が G の全域木となる辺 $e' \in E(T') - E(T)$ が存在する.

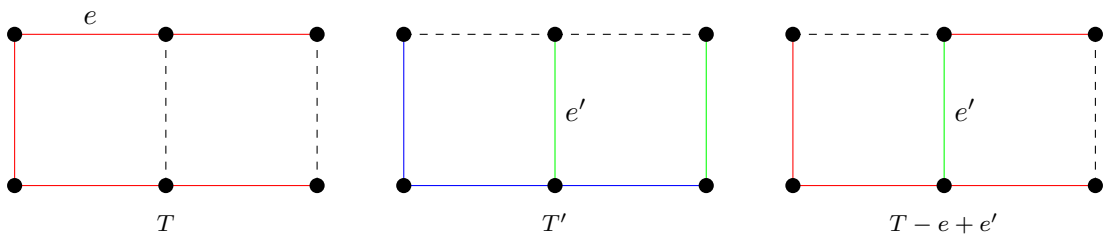


図 5.1: 全域木の交換可能性の例

証明:
T から e を除去すると，T は木であることから，連結でありすべての辺は橋であるため，T の点集合は互いに素な 2 つの集合 V_1, V_2 に分割される．ここにおける橋とは，それを除去すると G が非連結になるような辺のことである．T' も全域木であるから，V_1, V_2 を結ぶ橋が必ず 1 本存在する．その橋を e' とすると，$(T - e) \cup e'$ すなわち $T - e + e'$ は必ず全域木となる.

5.3　全域木戦略

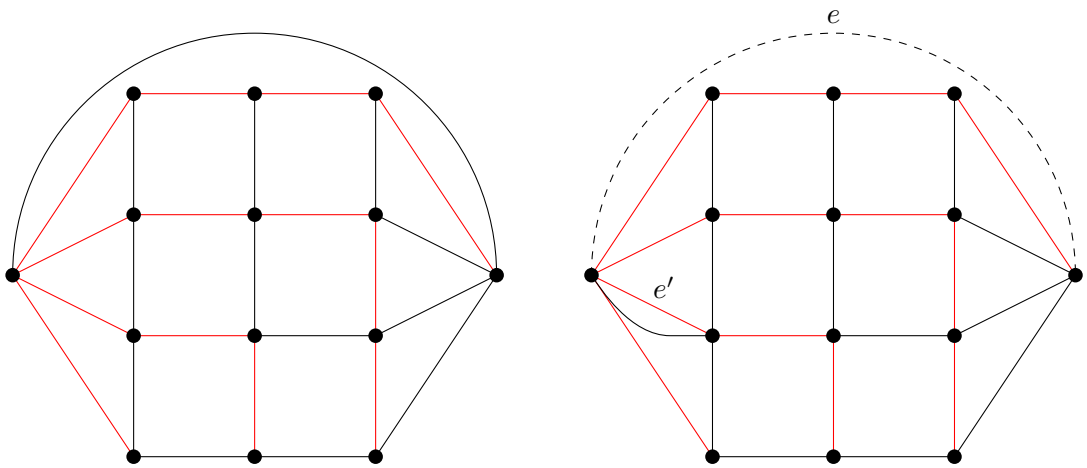
初期局面に対応するグラフを以下のように定義する: 頂点を左対局者の支柱とし，辺を左対局者の隣接支柱に対応する頂点間としたグラフを考える．その際，盤の左端の支柱と右端の支柱をそれぞれ一つの単一な支柱とみなし，左端の支柱と右端の支柱をつなぐ補助辺が存在するとみなす.

この補助辺がはじめに削除されると考えることで，命題 5.1 を初手にも適用することができる.

左の全域木戦略実行手順を以下に示す.

1. 初期局面に対応するグラフ G を作成する.
2. グラフ G 中の 2 つの辺素な全域木を求める.
3. ゲームが終了するまで手順 4,5 を繰り返す.
4. グラフ G 中の右が架橋した橋ユニットに対応する辺 e(初手の場合は補助辺とする) を削除し，命題 5.1 に従って対応する辺 e' を選択し，その辺を二重辺に変更する.
5. e' に対応する橋ユニットを架橋する.

下図 5.2a は初期局面である図 1.1a に対応するグラフで，2 つの辺素な全域木をそれぞれ黒と赤で表している．図 5.2b は左対局者が初手で全域木戦略に従って手順 4 を行った例である．右に補助辺を切断され，左が 2 つに分割された黒の全域木を接続し直している.



(a) 初期局面に対応するグラフ　　**(b)** 全域木戦略に従う初手の例

図 5.2

左が勝利局面となる．もしくは，右が切断する辺が 1 つもない場合プロセスは終了する．後者の場合，残った辺はすべて二重辺となり左によって 2 つの辺素な全域木が作られている．したがって，どちらの場合も左対局者が左端と右端の支柱に橋を架けることができ，勝利する.

右が橋ユニットを架橋することは，グラフ G において対応する辺が削除されることと同等である．右の動きに対して，グラフ G において左は命題 5.1 に従って辺 e' を選択し二重辺とする．これは，辺 e' に対応する橋ユニットを架橋することと同等である．ただし，両対局者とも端同士の支柱間に橋ユニットを架橋したとき，グラフ G の辺に影響はないものと考えられる.

辺 e が削除され，2 つの全域木に共通する二重辺 e' が作られたとき，グラフ G は未だ 2 つの辺素な全域木を含んでいる．したがって，この手順を繰り返すことで左が常に 2 つの全域木の存在を維持することができる.

第 6 章　随意全域木戦略

前章で紹介した全域木戦略は，初期局面に対応するグラフから常にその戦略に従うことで勝利できるという左対局者用の戦略であった．よって，一度全域木戦略から外れた手を打ってしまった場合は実行できず，右対局者はそもそも実行することができない．

本研究では，任意の局面に対応し，両対局者とも実行できるよう拡張した随意全域木戦略を提案する．提案戦略を設計するために，左対局者グラフと右対局者グラフ，および生成分と死成分という概念を導入する．

6.1　対局者グラフと生成分 (死成分) の導入

定義 6.1 (左対局者グラフと右対局者グラフ)
ゲームの局面 G に対応する左対局者グラフ $LG(G)$ とは，頂点を左対局者の支柱とし，辺を左対局者の隣接支柱に対応する頂点間としたグラフである．その際，盤の左端の支柱と右端の支柱をそれぞれ一つの単一な支柱とみなす．また，架橋していない隣接支柱間を一重辺，左対局者が架橋している隣接支柱間を二重辺，右対局者が架橋している隣接支柱間の辺はないものとする．

同様に，ゲームの局面 G に対応する右対局者グラフ $RG(G)$ とは，頂点を右対局者の支柱とし，辺を右対局者の隣接支柱に対応する頂点間としたグラフである．その際，盤の上端の支柱と下端の支柱をそれぞれ一つの単一な支柱とみなす．また，架橋していない隣接支柱間を一重辺，右対局者が架橋している隣接支柱間を二重辺，左対局者が架橋している隣接支柱間の辺はないものとする．

定義 6.2 (生成分と死成分)
任意の局面を考える際，対応する対局者グラフが非連結な場合がある．そこで，ゲームの局面に対応した左 (右) 対局者グラフにおいて，盤の左 (上) 端と右 (下) 端の支柱に対応する頂点をともに含む成分を生成分と呼び，そうでない成分を死成分と呼ぶ．ここで成分とは，任意の非連結なグラフに含まれる各々の連結グラフのことである [2].

6.2　随意全域木戦略

随意全域木戦略実行手順を以下に示す.

1. 局面に対応するグラフ G(左対局者の場合は左対局者グラフ，右対局者の場合は右対局者グラフ) を作成する.
2. グラフ G 中の生成分 C を求める．求められないときは勝ち手なしとして終了する.
3. 生成分 C において，二重辺に変更することで 2 つの辺素な全域木を取ることができる一重辺を選択する．その辺を二重辺に変更する．選択できないときは勝ち手なしとして終了する.
4. 手順 3 で選択した辺に対応する橋ユニットを架橋する.

この手順は以下の予想に基づき設計している.

予想 6.3

- 生成分が存在しないときは相手の勝利局面である.
- 生成分において，2 つの辺素な全域木を取ることができないときその対局者は必勝手をもたない.

提案した随意全域木戦略に対して，我々は以下の予想を立てる．

予想 6.4

- 随意全域木戦略は実行できるとき必勝戦略である．
- どちらかの対局者が随意全域木戦略を実行できる．

なお予想 6.4 が正しければ，任意の局面でどちらかの対局者が必勝戦略である随意全域木戦略を実行できることから「橋を架けろ」には引き分けが存在しないことが直ちに導ける．

6.3　提案戦略の検証

随意全域木戦略を実際の局面を例に検証を行う．

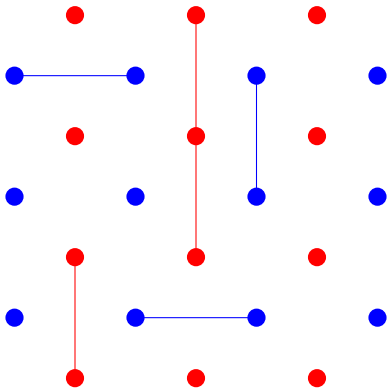
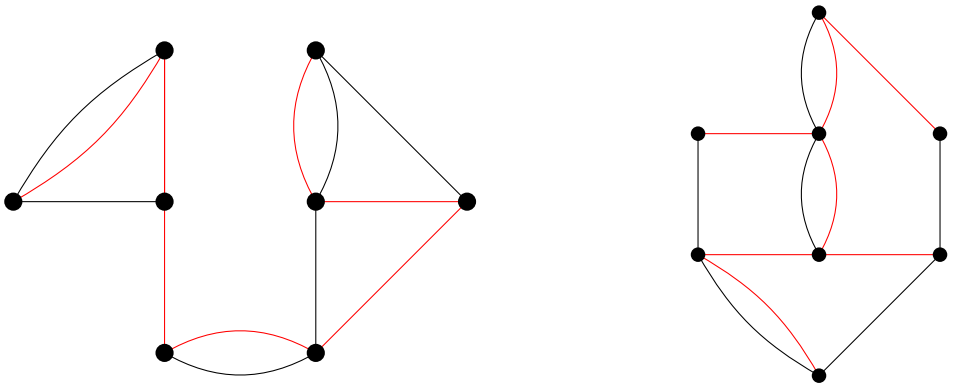


図 6.1: 局面 G

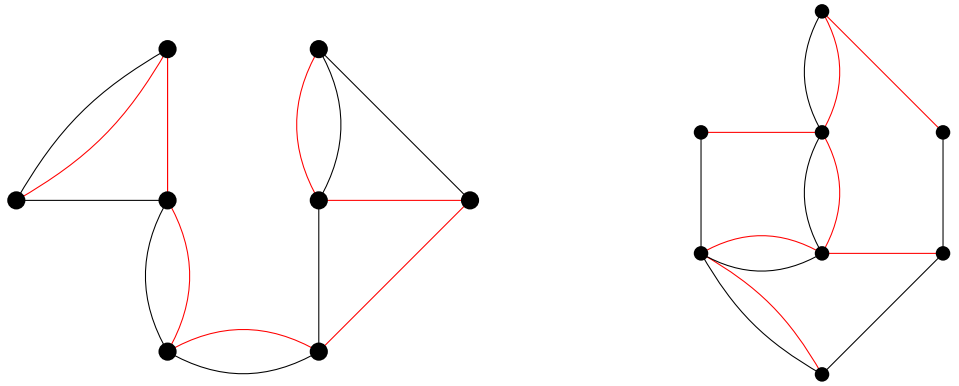
図 6.1 に対応した左対局者グラフ $LG(G)$ と右対局者グラフ $RG(G)$ を作成する．これらは生成分のみで構成されていることがわかる．



(a) $LG(G)$　　**(b)** $RG(G)$

図 6.2: 局面 G に対応する対局者グラフ

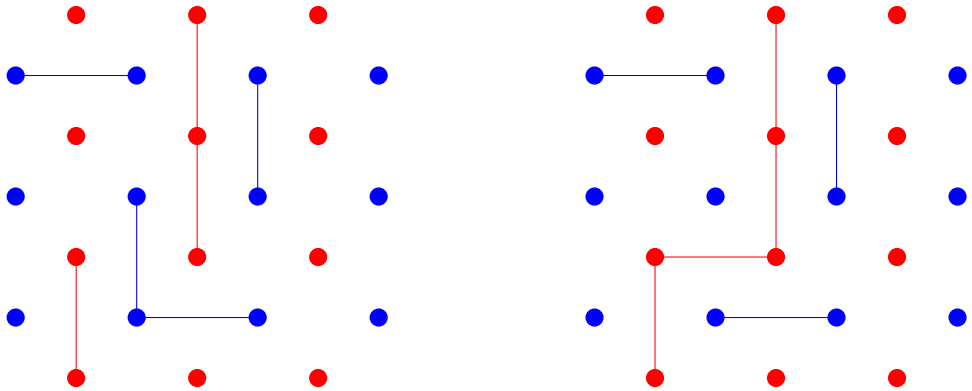
これらのグラフにおいて手順 3 に従い，二重辺に変更することで 2 つの辺素な全域木を取ることができる一重辺を選択し，それを二重辺に変更する．この例では，$LG(G), RG(G)$ ともにそのような辺が存在する．その選択した辺に対応する橋ユニットを架橋した局面を G^{L_1}, G^{R_1} とすると，各対局者グラフは次の図のようになる．



(a) $LG(G^{L_1})$　　**(b)** $RG(G^{R_1})$

図 6.3: 一重辺を二重辺に変更したグラフ

手順 4 に従い選択した辺に対応する橋ユニットを架橋した局面 G^{L_1}, G^{R_1} は次図のようになる.

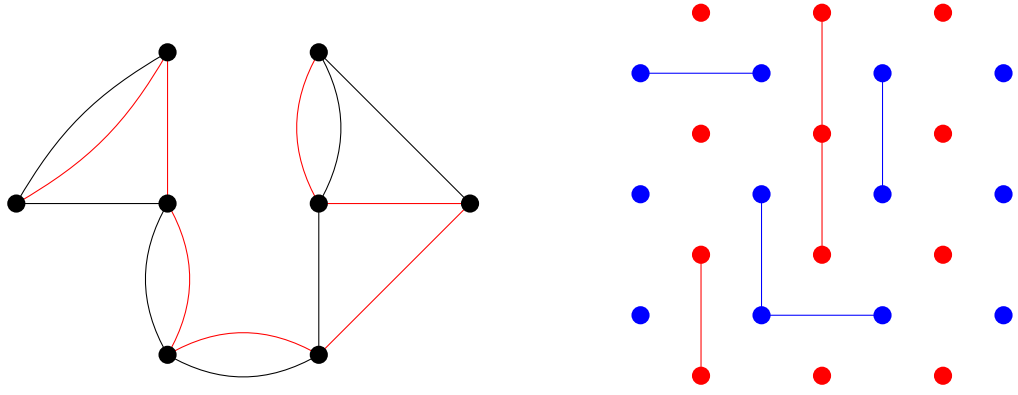


(a) 局面 G^{L_1}　　**(b)** 局面 G^{R_1}

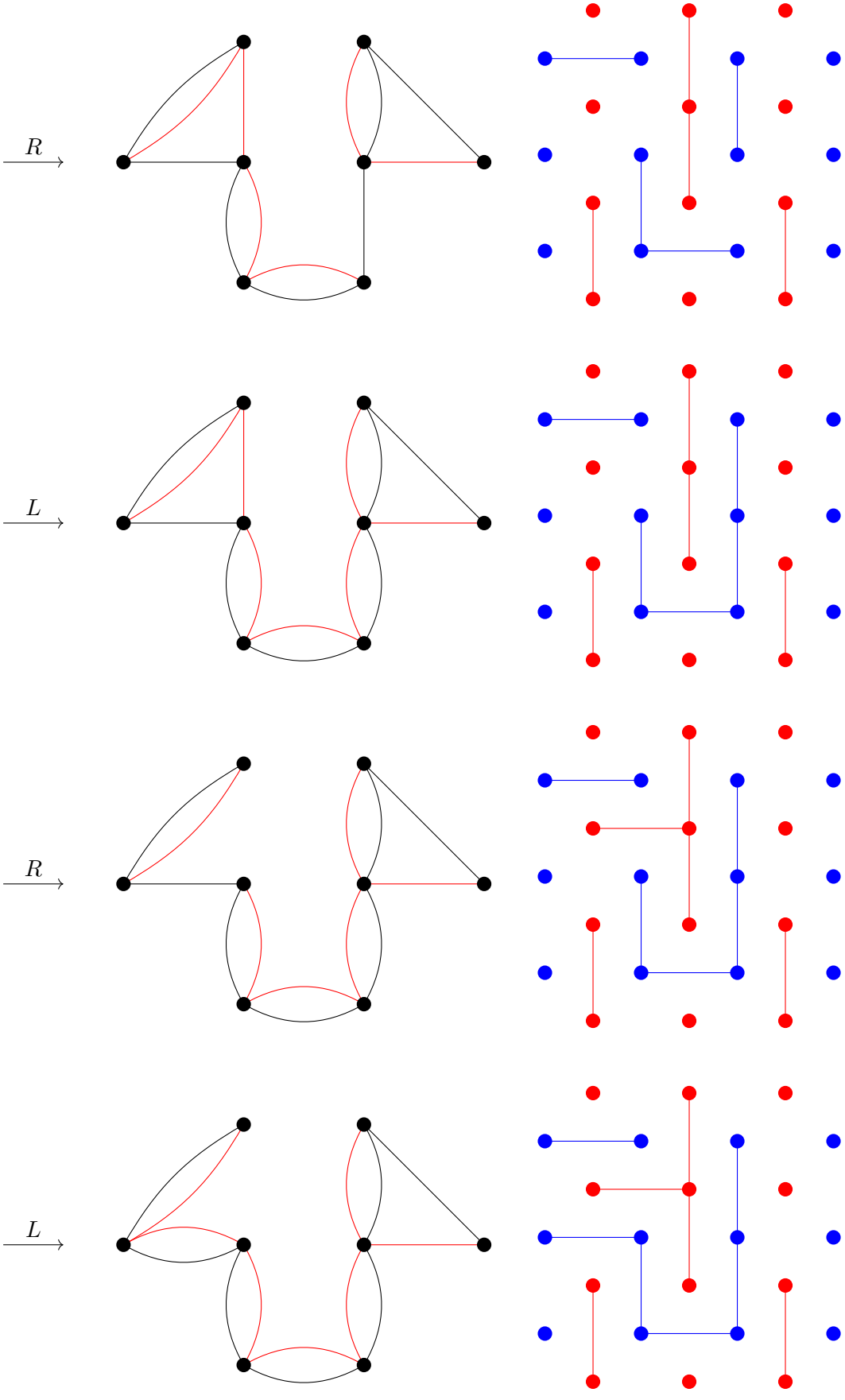
図 6.4: ゲームの局面に対応する対局者グラフ

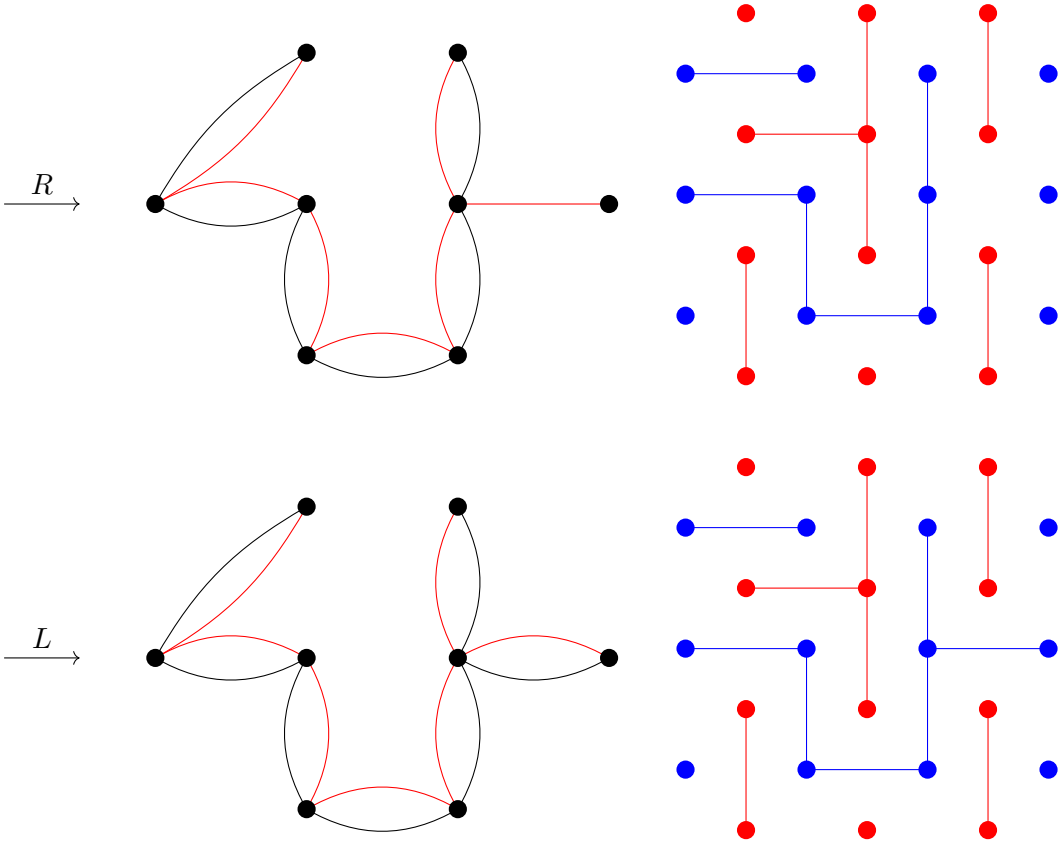
$RG(G^{R_1})$ は既に勝利局面となっていることから右対局者は勝利している.

ここからは G^{L_1} からの局面について着目し，随意全域木戦略に基づいた実際のゲームの局面と対応する左対局者グラフの流れを以下に示す.



(a) 左対局者グラフ $LG(G^{L_1})$　　**(b)** 局面 G^{L_1}


R
L
R
L



以上の流れの結果左対局者は勝利している．

この例では局面 G から両対局者ともに先手であれば随意全域木戦略を実行し勝利することができたことから必勝戦略として機能している．

第 7 章　選択肢の優劣判別方法の予想

随意全域木戦略を実行している際，複数の選択肢が生じる例がある．そこで選択肢の優劣を判別するための関数を導入し，判別方法を提案する．ここでいう優劣を定義するために組合せゲーム理論 [1] を用いる．

7.1　選択肢の優劣判別方法の予想

ここで，判別に用いる関数を以下のように定義する．

定義 7.1 (優劣判別関数 $b_L(G), b_R(G)$)
局面 G において，関数 $b_L(G), b_R(G)$ を以下のように定義する．

$$b_L(G) = n \overset{\text{def}}{\iff} \text{左が橋ユニットを最小}n\text{本架橋すれば勝利局面}$$
$$b_R(G) = n \overset{\text{def}}{\iff} \text{右が橋ユニットを最小}n\text{本架橋すれば勝利局面}$$

ただし，何本架橋しても勝利局面とならないとき $n = \infty$ とする．

優劣判別関数を用いることで，任意の局面に対して各対局者の選択肢の組合せゲーム理論における比較演算に従い定義される優劣が以下のように判別できると予想する．

予想 7.2 (優位な選択肢の判別方法)
局面 G 中の支柱間 e_i と e_j に橋ユニットを架橋する左選択肢 G_i^L と G_j^L に対して，以下が成立する．

- $b_L(G_i^L) < b_L(G_j^L)$ のとき: $G_i^L > G_j^L$ となる
- $b_L(G_i^L) = b_L(G_j^L)$ かつ $b_L(G_i^R) < b_L(G_j^R)$ のとき: $G_i^L > G_j^L$ となる

同様に右選択肢 G_i^R と G_j^R に対して，以下が成立する．

- $b_R(G_i^R) < b_R(G_j^R)$ のとき: $G_i^R < G_j^R$ となる
- $b_R(G_i^R) = b_R(G_j^R)$ かつ $b_R(G_i^L) < b_R(G_j^L)$ のとき: $G_i^R < G_j^R$ となる

また，選択肢の等価性も以下のように判別できると予想する．

予想 7.3 (等価な選択肢の判別方法)
左選択肢 G^L 中に $b_L(G^L) = 0$ が存在しないとき，局面 G 中の支柱間 e_i と e_j に橋ユニットを架橋する左選択肢 G_i^L と G_j^L は，以下が成立する．

- $b_L(G_i^L) = b_L(G_j^L)$ かつ $b_L(G_i^R) = b_L(G_j^R)$ のとき: $G_i^L = G_j^L$ となる

右選択肢に対しても対局者を反対に考えることで同様の性質が成立する．

7.2　提案予想の検証

予想 7.2,7.3 を実際の局面を例に CGSuite を用いて検証を行う．

以下にデータを収集した局面の例を紹介し，支柱間にラベルを付ける．端支柱同士の間に橋ユニットを架橋することは何もせず相手に手番を渡すのと同等であると考え，端支柱同士の間は考慮しない．

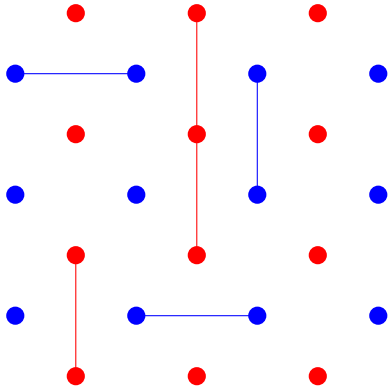


図 7.1: 局面の例

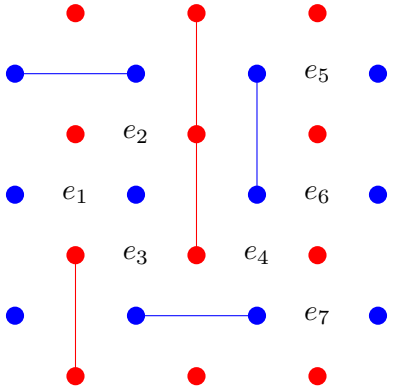


図 7.2: 支柱間のラベル付け

左選択肢の優劣を判別する．図 7.2 を参照して $b_L(G_i^L), b_L(G_i^R)$ を求め，以下にまとめる．

辺 e_i	$b_L(G_i^L)$	$b_L(G_i^R)$
e_1	2	3
e_2	2	3
e_3	2	∞
e_4	3	3
e_5	3	3
e_6	3	3
e_7	2	4

表 7.1: 辺 e_i に対応する $b_L(G_i^L), b_L(G_i^R)$

辺 e_i	$b_L(G_i^L)$	$b_L(G_i^R)$
e_3	2	∞
e_7	2	4
e_1	2	3
e_2	2	3
e_4	3	3
e_5	3	3
e_6	3	3

表 7.2: 表 7.1 をソートしたもの

以上の結果から予想 7.2,7.3 に従って左選択肢の優劣を考えると，

$$G_3^L > G_7^L > G_1^L = G_2^L > G_4^L = G_5^L = G_6^L$$

という予想をすることができ，これは実際の局面値の比較と一致している．

右選択肢の優劣を判別する．図 7.2 を参照して $b_R(G_i^R), b_R(G_i^L)$ を求め，以下にまとめる．

辺 e_i	$b_R(G_i^R)$	$b_R(G_i^L)$
e_1	1	1
e_2	1	1
e_3	0	2
e_4	1	1
e_5	1	1
e_6	1	1
e_7	1	1

表 7.3: 辺 e_i に対応する $b_R(G_i^R), b_R(G_i^L)$

辺 e_i	$b_R(G_i^R)$	$b_R(G_i^L)$
e_3	0	2
e_1	1	1
e_2	1	1
e_4	1	1
e_5	1	1
e_6	1	1
e_7	1	1

表 7.4: 表 7.3 をソートしたもの

以上の結果から予想 7.2,7.3 に従って右選択肢の優劣を考えると，

$$G_3^L < G_1^L, G_2^L, G_4^L, G_5^L, G_6^L, G_7^L$$

という予想をすることができ，これは実際の局面値の比較と一致している．

第 8 章　今後の課題

随意全域木戦略を提案したが，この戦略が必勝戦略であることや予想した各種性質の検証はできていない．また選択肢の優劣判別方法も予想したが十分な検証ができていない．これらの検証は今後の課題である．

CGSuite での「橋を架けろ」の実装において勝利局面の判定アルゴリズムの最適化が不十分であると考える．それにより，局面値を計算できる局面が限定されてしまい，データが十分に収集できなかった．アルゴリズムの最適化とさらなるデータの収集が今後の課題である．

謝辞

本研究を進めていくにあたりご指導いただきました，草刈圭一朗教授に心より感謝申し上げます．また，日頃からお世話になっております草刈研究室の皆様に感謝いたします．

参考文献

[1] M.H.Albert, R.J.Nowakowski, D.Wolfe(川辺治之訳), 組合せゲーム理論入門: 勝利の方程式, 共立出版, 2011

[2] R.J.Wilson(西関隆夫, 西関裕子訳), グラフ理論入門 (原書第 4 版), 近代科学社, 2001

[3] D.B.West, Introduction to Graph Theory (2nd Edition), Pearson Education,Inc, 2001