Vol.2017-GI-37 No.15
2017/3/7

リミテッドTRAXのゲーム値を求める試み

桑原 和人[1,a)]　保木 邦仁[1,b)]

概要：TRAX は二人零和完全情報ゲームであり，これの盤面の大きさを 8×8 に制限したものはリミテッド TRAX と呼ばれる．我々はこれまでに 6×6 TRAX のゲーム値が引き分けという結果を報告している．本研究では，(1) ルール上許されるタイル配置の総数を粗く見積もり，合法局面の数を他のゲームと比較し，(2)4×4 から 7×7 までの TRAX を解き，8×8 TRAX のゲーム値を求めるのに要する時間を見積もり，(3)8×8 TRAX を解くことを試みる．

Kazuto Kuwabara[1,a)]　Kunihito Hoki[1,b)]

Abstract: TRAX is a two-player zero-sum perfect-information game, and the game with small sized board of 8×8 is called limited TRAX. So far we have reported that the game value of 6×6 TRAX is draw. In this study, (1) we roughly estimate the number of tile configurations and compare it with the number of legal positions of other games, (2) by solving from 4×4 to 7×7 TRAX, we estimate a duration of time to solve 8×8 TRAX, and (3) we attempt solving 8×8 TRAX.

1. はじめに

TRAX は [*1]，1980 年にニュージーランド人の David Smith 氏が考案した二人零和確定完全情報ゲームである．他のボードゲームとは異なり，盤面の大きさには基本的に制限がないという特徴がある．

TRAX にはリミテッドトラックスと呼ばれる盤面の大きさを制限してプレイする遊び方が存在する．リミテッドトラックスでは盤面の縦幅，横幅が 8 より大きくなるような場所にはタイルを配置することが出来ない．盤面が全て埋まった時点で決着が着いていなかった場合は引き分けとなる．本論文ではこれを 8×8 TRAX と呼ぶ．

本研究は以前報告した研究の続きである．本報告では，論文の自己完結性をある程度保つため，ゲームのルールと探索の方法に関して前報告と同じ説明を繰り返した．前報告との違いは，ゲームのルールに関しては盤面サイズが 7×7 以下の TRAX のビクトリーラインの定義である．探索の方法に関してはスレッド並列化を行う点に違いがある．

[1] 電気通信大学
The University of Electro-Communications, Chofu, Tokyo 182–8585, Japan
[a)] kaz03301@gmail.com
[b)] k.hoki@uec.ac.jp
[*1] Official TRAX Website: http://www.traxgame.com/ (last access, 2016)

図 1　Trax で使用するタイル

2. TRAX のルール

TRAX のルールについて説明する [1].

2.1 ゲームの進行

TRAX は，タイルを 1 枚ずつ交互に並べていく 2 人用対戦ゲームである．並べるタイルは図 1 に示されるように 6 種類ある．先手が白，後手が赤のプレイヤである．タイルを配置する際，赤のラインは赤，白のラインは白につながるようにタイルを配置しなければならない．自分の色にも相手の色にもつなぐことができる．

例えば図 2 の左の盤面は，3 手までタイルを配置した状態であるが，次の 4 手目を配置する場合右下のように異なる色のラインがつながる手は配置することができない．

2.2 連鎖ルール

TRAX では，通常，先手・後手は交互に 1 枚ずつタイルを配置していく．しかし，連鎖ルールが適用される場合は，例外的に 1 ターンで 2 枚以上のタイルが配置される．連鎖の発生条件は，タイルを 1 枚置いた後，タイルがない場所に集まる同色ラインの数が 2 になることである．この

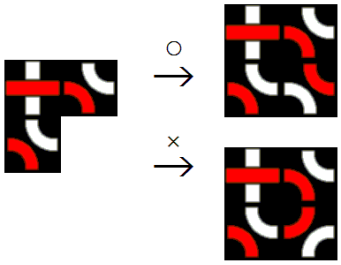

図 2　不正なタイルの配置

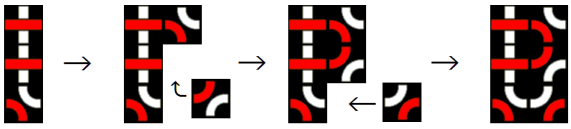

図 3　連鎖が 2 回発生し，計 3 枚のタイルが 1 ターンで配置された例

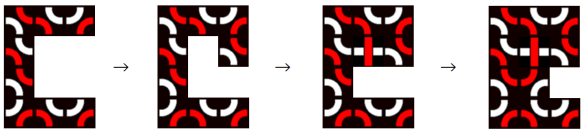

図 4　3 本の白色ラインが 2 度の連鎖発生後に 1 箇所に集まった例

図 5　赤のループと白のビクトリーラインの一例

ような空きスペースは自動的に埋められる．連鎖ルールに従ってタイルを配置した後，さらに連鎖発生条件が満たされた場合にも同じように空きスペースが自動的に埋められる (図 3 参照)．

また，ある空きスペースに同色のラインが 3 本以上集まったときは，そのターンのプレイヤの手は無効となり，ターン最初の盤面に巻き戻される (図 4 参照)．

2.3　勝利条件

ループかビクトリーラインが形成されるとゲーム終了となる．ループとは，両端がつながったラインである (図 5 左参照)．ビクトリーラインとは，盤面の最上端と最下端を縦断，または最左端と最右端を横断する，長さ 8 列以上のラインである (図 5 右参照)．白のループまたはビクトリーラインが形成されると白の勝ち，赤のループまたはビクトリーラインが形成されると赤の勝ちとなる．手番プレイヤでない方のプレイヤの勝利条件が満たされる場合もある．また，白と赤両方が同時に勝利条件を満たしたときは手番プレイヤの勝ちとなる．

2.4　8×8 TRAX

TRAX にはタイルの枚数に制限がない．一方，8×8 TRAX ではタイルを配置できるスペースが制限される．ルールは通常の TRAX に準ずるが，場に配置できるタイルは縦横 8 列までとなる．64 枚すべてのタイルを配置しても勝負がつかなければ引き分けとなる．

本研究では，8×8 よりサイズの小さい TRAX も扱う．また，3 本以上の同色のラインが集まる空きスペースができるような手は非合法手として扱い，空きスペースが残っていて合法手がない場合には手番プレイヤの負けとする．

通常の TRAX では，ビクトリーラインの長さは 8 列以上である．したがって，盤面サイズが 7×7 以下の TRAX ではこれが作られることがない．そこで，n が 7 以下の $n \times n$ TRAX ではビクトリーラインの長さを n 列とした．

3.　合法局面数の見積もりと他のゲームとの比較

$n \times n$ TRAX においてルール上許されるタイル配置の総数を粗く見積もり，合法局面数を他のゲームと比較する．この見積もりでは，平行移動・回転・反転操作により複数生成される配置の重複を排除して数える．

3.1　方法

$n \times n$ TRAX のタイル配置の総数を見積もる．これを見積もる方法は，[2] のモンテカルロ法による合法局面数の見積もりに基づく．

(1) $n \times n$ の盤面の各マスについて，タイルを置かない，もしくは 6 種類のタイルのいずれかを置くという 7 通りの選択肢を考える．そして，7 通りの内一つを一様ランダムに選択する．これにより，$7^{n \times n}$ 通りの配置から無作為に 1 つの配置が選択される．

(2) 盤面が以下の条件を全て満たすとき $z = 1$，そうでないとき $z = 0$ として，期待値 $\bar{z}$ を求める．

条件 1　島が 2 つ以上存在しない

条件 2　島が左上に寄っている

条件 3　同色ラインが 2 以上集まった空きマスが存在しない

条件 4　同色のラインが繋がっている

島とは，1 つ以上タイルから成るものであり，上下左右に隣接している 2 つのタイルを地続きとする．また，条件 2 により平行移動による配置の重複が排除される．なお，ゲーム開始点から作ることのできない配置は本方法では排除されない．

(3) $7^{n \times n}\bar{z}/8$ を見積もり結果とする．8 で割ることにより，回転・反転による配置の重複数をやや大きめに見積もって，これらの配置を排除する．

盤面サイズ 3×3 から 5×5 における見積もり結果を表 1 に示す．この方法では，盤面のサイズが 5×5 以上において条件 4 を満たす盤面が出現しなかったため，期待値 $\bar{z}$ を求めることが出来なかった．そこで，z を以下のようして近似した．

- 条件 1 から 3 のいずれかを満たさなければ $z = 0$

表 1 $n \times n$ TRAX のタイル配置の総数の見積もり (方法 1)

3×3	4×4	5×5
3.4×10^3	8.3×10^5	—

表 2 $n \times n$ TRAX のタイル配置の総数の見積もり (方法 2)

3×3	4×4	5×5
3.1×10^3	8.1×10^5	9.3×10^8
6×6	7×7	8×8
2.8×10^{12}	1.6×10^{15}	2.6×10^{18}

表 3 他のゲームの合法局面数

Tic Tac Toe [2]	10^3
チェッカー [2]	10^{18}
オセロ [2]	10^{28}
チェス [2]	10^{50}
将棋 [4]	10^{70}
囲碁 (19 路)[5]	10^{170}

- 条件 1 から 3 を全て満たす場合は，タイルが上下左右に隣接している箇所の数を h とした時，$z = (1/2)^h$

島が 2 つのタイルから成る場合，$h = 1$ である．そして，$z = 1/2$ は同色ラインが繋がる確率を与える．なぜならば，ある辺のラインの色は 6 種類中 3 種類のタイルが白，他の 3 種類が赤になるからである．h が 2 以上の場合，$z = (1/2)^h$ は同色ラインが繋がる確率を正しく見積もらないが，簡易な近似としてこれを採用する．z を近似した場合の見積もり結果を表 2 に示す．

3.2 8 × 8TRAX の合法局面数

他のゲームの局面数を表 3 に示す．局面数とは，ルール上許される駒や石の配置と手番プレイヤの組み合わせの数のことであり，過去の手順は考慮しない．表 3 より，8×8 TRAX のルール上許されるタイルの配置の総数は 2.6×10^{18} と粗く見積もられる．これを 2 倍すると手番プレイヤの組み合わせをやや多めに考慮したことになるであろう．したがって，局面数は 5×10^{18} 程度である．これはチェッカーと同程度である．

チェッカーは，ゲーム進行方向に勝敗を分析する証明数探索と，ゲーム進行方向とは逆順に勝敗を分析してエンドゲームデータベースを構築する後退解析の組み合わせにより解かれた [3]．チェッカーよりも合法局面数の多い二人完全情報ゲームが解かれたという報告は著者の知る限りない．

証明数探索は TRAX においても有効ではないかと考えられる．実際，以前の研究では，深さを閾値とした深さ優先探索の反復深化よりも証明数探索を深さ優先で行う df-pn 探索の方が格段に効率が良いことが示された．しかし，ゲーム進行と共にタイル数が増えていく TRAX ではエンドゲームデータベースの構築は非常に困難だと考えられる．

4. 探索プログラムの実装

AND/OR 木を探索してゲーム局面の勝敗を求める [6]．AND/OR 木は Min-Max 木の特殊形であり，各プレイヤの利得は 0 と 1 の 2 値で表される．

4.1 AND/OR 木

AND/OR 木は，AND 節点と OR 節点から構成され，根節点を現在の局面とする根つき木である．各節点は各ゲーム局面に対応し，節点間の枝はゲームの進行に対応する．

OR 節点では，利得を 1 にしたい方のプレイヤが手番を持つ．AND 節点では利得を 0 にしたい方のプレイヤが手番を持つ．各節点は 1，0，不明のいずれかの値を持つ．

子節点を持たない節点を先端節点と呼ぶ．ゲーム終了条件を満たした先端節点を終端節点と呼び，利得がそのまま終端節点の値となる．

非終端節点を内部節点と呼ぶ．OR 内部節点の値は，値が 1 の子節点が 1 つでも存在すれば 1，すべての子節点の値が 0 なら 0，それらでなければ不明となる．AND 内部節点の値は，値が 0 の子節点が 1 つでも存在すれば 0，すべての子節点の値が 1 なら 1，それらでなければ不明となる．

4.2 トランスポジションテーブル

本研究では，任意の 2 局面に対して，タイルの配置と手番が同じであれば，その局面に到達する手順が異なっていたとしても同じ局面とみなす．そして，そのような複数の局面を 1 つの節点に対応させる．したがって，木ではなく有向非巡回グラフ (DAG) の探索を行うこととなる．なお，TRAX では 1 ターンあたり 1 枚以上のタイルが配置され，配置したタイルが消滅することもないため巡回は存在しない．

DAG を探索すると，同一節点を 2 回以上訪問するようなことが頻繁に起こる．そこで，探索の効率化をはかるため，トランスポジションテーブル (以下 TT) を利用する．TT とは探索した局面の結果を保存するハッシュ表である．ある局面を探索しようとした時，その局面の結果が TT に保存されていればこれを参照するだけで探索を終了出来るため，探索時間が短縮される．2 局面の比較やハッシュキーには，手の乱数値の排他的論理和をとった 64 ビットのゾブリストハッシュ法を使用する [7]．また，ハッシュ表はチェーンハッシュ法を用いて実装した．

探索の効率化のため，回転，反転，平行移動して等しくなる局面を同一局面とみなす．このために，ゾブリストハッシュキーを生成する際には，回転および反転された $n \times n$ TRAX の盤面を，$n \times n$ の平面に再配置する．この時に，全てのタイルは形を変えずに左上に寄せる．このようにして生成された 8 つの 64 ビットハッシュキーのうち，最も値

が小さいものを節点のキーとする．

深さ優先探索では，訪問した節点の情報を TT に保存する．本研究では，これらの情報の TT への保存に加えて探索経路上の各深さの節点に対する全子節点のゾブリストハッシュキーを TT とは別に保持する．探索中，子節点の情報は保持されているゾブリストハッシュキーを用いて TT から取得する．

TT のエントリが全て埋まってしまった時にガベージコレクション (GC) を行い，空きエントリを確保する．ある局面を根とする部分木のサイズが大きいほど，その局面情報は価値が高いと考えられる．なぜならば，部分木のサイズが大きいほど再計算に時間がかかるからである．したがって，エントリを捨てる時は部分木のサイズが小さいようなエントリから捨てていくのが良い．

本研究の実装は，small Tree GC に基づく [8]．TT のエントリにある節点の情報を登録する際，この節点以下の探索に要した訪問節点数を保存する．TT のエントリが全て埋まったら，次の操作を行う．

(1) 閾値 θ を 1 に設定する
(2) 訪問節点数が θ より小さいエントリを全て空にする
(3) TT の空き容量が一定の割合 r を超えていたら GC を終了する
(4) そうでなければ θ を 1 増やし，手順 2 へ戻る

本研究では r は 0.3 に設定した．

4.3 深さ優先証明数探索 (df-pn 探索)

Df-pn 探索は Allis らの証明数探索を深さ優先で探索する方法である [9]．300 手詰め以上の詰将棋を全て解いたことで有用性が示された [10], [11]．証明数探索で次の最有力節点を選ぶときに前回と同じ経路をたどったとすると，根まで戻って潜るという操作は無駄である．Df-pn 探索では，最有力節点が探索中の節点の下にあるときは戻らずそのまま探索を行うことが出来る．

Df-pn 探索では，探索打ち切りの条件に用いられる閾値として深さ優先探索の探索深さは用いない．証明数と反証数を閾値に用いて打ち切り条件を設定し，各節点を訪問する順番を制御する．

本研究では，df-pn 探索を複数スレッドで並列に行うよう実装した．実装は，[12] に基づいている．ただし，あるスレッドが探索中の節点を証明または反証したことを別スレッドに知らせ，その節点の探索を打ち切るという機能を [12] では提唱しているが，この機能による効率の改善が見られなかったため，本研究ではこの機能を省いた．

4.4 深さを制限した浅い AND/OR 木探索の併用

1，2 手でゲームが終了するような局面に対しては，TT を用いたり証明数，反証数を用いたりすることによる探索の効率化はあまり望めない．そのため，df-pn 探索の内部節点で浅い深さ 2 の AND/OR 木探索を行う．

この内部節点 v を根とした浅い深さ 2 の AND/OR 木探索により，v の値が判明した場合，この結果を TT に保存する．ただし，GC で用いられる情報である部分木の訪問回数は 0 として保存する．ここで，この浅い探索で訪問した節点情報は TT に登録しない．浅い AND/OR 木探索が終了しても v の値が判明しない場合，浅い AND/OR 木探索によって値が判明しなかった子節点を用いて df-pn 探索を継続する．これにより，TT エントリへの登録回数を減らし，GC の発生回数が削減される．

このように，浅い AND/OR 木探索を df-pn 探索に併用する方法は先行研究にもみられる [13]．この先行研究では，詰将棋において df-pn 探索が効率化され，探索節点数および探索時間が削減されることを見出した．なお，この先行研究では新規節点のみに対して浅い AND/OR 木探索を行うことを目指したが，本研究ではこれを訪れた全ての df-pn 探索の内部節点に対し行った．

4.5 疑似コード

Df-pn 探索の疑似コードを次に示し，疑似コード内で現れる関数について説明する．

```
int Df-pn(node R){
    R.φ = ∞; R.δ = ∞;
    parallel-for (each t of threads) MID(R);

    RetrieveProofandDisproofNumbers(R, φ, δ, vpn);
    if(φ = ∞) return proven;
    else return disproven;
}

void MID(node N){
    if(IsTerminal(N)){
        Evaluate(N);
        SaveProofandDisproofNumbers(N, N.φ, N.δ);
        return;
    }

    moves = GenerateMoves(N);
    result = AndOrSearchThenDiscardDisproven(N, DEPTH,
                                                moves);
    if(result = proven){ //proven
        N.φ = 0; N.δ = ∞;
        SaveProofandDisproofNumbers(N, N.φ, N.δ);
        return;
    }
    if(moves.num = 0){ //disproven
        N.φ = ∞; N.δ = 0;
        SaveProofandDisproofNumbers(N, N.φ, N.δ);
        return;
    }

    children = GenerateChildren(N, moves);
    Mark(N);
    while(N.φ > ΔMin(children) && N.δ > ΦSum(children)){
        node C_best = SelectChild(children, N.φ_c, δ_2);
```

```
        C_best.φ = N.δ + φ_c - φSum(N);
        C_best.δ = min(N.φ, δ_2+1);
        MID(C_best);
    }
    UnMark(N);

    N.φ = ΔMin(N); N.δ = φSum(N);
    SaveProofandDisproofNumbers(N, N.φ, N.δ);
}

node SelectChild(node[] children, int &φ_c,int &δ_2){
    δ_c = φ_c = ∞;
    for(each child C of children){
        RetrieveProofandDisproofNumbers(C, φ, δ, vpn);
        δ += vpn;
        if(δ < δ_c){
            C_best = C;
            δ_2 = δ_c; φ_c = φ; δ_c = δ;
        }
        else if(δ < δ_2) δ_2 = δ;
        if(φ = ∞) return C_best;
    }
    return C_best;
}

int ΔMin(node[] children){
    delta_min = ∞;
    for(each node C of children){
        RetrieveProofandDisproofNumbers(C, φ, δ, vpn);
        δ += vpn;
        delta_min = min(delta_min,δ);
    }
    return delta_min;
}

int ΦSum(node[] children){
    phi_sum = 0;
    for(each node C of children){
        RetrieveProofandDisproofNumbers(C, φ, δ, vpn);
        phi_sum += φ;
    }
    return phi_sum;
}

node[] GenerateChildren(node N, move[] moves){
    node[] children;
    for(each m of moves){
        node C = MakeMove(N, m);
        C.key = GenerateUniqueKey(C);
        children.Add(C);
    }
    return children;
}
```

Df-pn

証明数・反証数の閾値を ∞ に初期化し，各スレッドで並列に再帰関数 MID を呼ぶ．この際，TT は全スレッドで共有する．探索が終了したら，根節点のゲームの値を返す．

MID

対象のノードの証明数・反証数が閾値を超えるまで深さ優先探索を行う再帰関数．

IsTerminal

終端節点かどうかを判定する．

Evaluate

終端節点の勝敗を判定する．手番プレイヤの勝ちならば，$N.\phi = 0$, $N.\delta = \infty$，手番プレイヤの負けならば，$N.\phi = \infty$, $N.\delta = 0$ となる．

GenerateMoves

対象の節点に対応する局面の擬合法手を全て生成する．3 本以上の同色のラインが集まる空きスペースができる非合法手も含む．

AndOrSearchThenDiscardDisproven

浅い AND/OR 木探索を行い，その結果を返す．また，この探索によって負けとなることが判明した手や非合法手を削除する．

GenerateChildren

対象の節点の子節点を全て生成する．子節点のゾブリストハッシュキーの生成もここで行われる．

MakeMove

対象の節点から与えられた手を打った後の子節点を返す．

GenerateUniqueKey

節点に対応する局面から，64 ビットのゾブリストハッシュキーを生成する．回転・反転・平行移動して等しくなる局面を同一局面とみなし，それらに対しては同一のハッシュキーが生成される．

Mark・UnMark

TT に保持される節点情報は，その節点を探索中のスレッドの数を含む．この値を 1 増やす，または 1 減らす．

RetrieveProofandDisproofNumbers

TT を参照し，節点の情報を取り出す．vpn は仮想的な証明数・反証数を表す．vpn の値は，この節点を探索中のスレッド数とする．複数のスレッドが同じ子節点を選択しないようにするために vpn は使われる．

SaveProofandDisproofNumbers

節点の情報を TT に保存する．

ΔMin

TT に保持されている全子節点の δ の最小値を返す．

ΦSum

TT に保持されている全子節点の ϕ の和を返す．

SelectChild

δ の最も小さい子節点を選択する．さらに，閾値の計算に必要な ϕ_c(選択した子節点の ϕ) と δ_2(子節点の中で 2 番目に小さい δ) を求める．

表 4 4×4 から 7×7 TRAX，先手勝ちと引き分けの利得を 1 とした場合．根節点の値はすべて 1.

	探索節点数	時間 (s)	GC 発生回数
4	16,004	0	0
5	337,034	6	0
6	17,186,949	505	0
7	910,806,368	45,237	0

表 5 4×4 から 7×7 TRAX，先手勝ちと引き分けの利得を 0 とした場合．根節点の値はすべて 0.

	探索節点数	時間 (s)	GC 発生回数
4	26,856	0	0
5	797,902	14	0
6	43,343,665	1,197	0
7	7,678,241,792	438,930	11

5. 実験

盤面サイズが 4×4 から 7×7 までの TRAX を解き，8×8 TRAX のゲーム値を求めるのに要する時間を見積もる．そして，8×8 TRAX を解くことを試みる．

$n \times n$ TRAX では引き分けが存在するため，結果は勝ち，負け，引き分けの 3 通りになる．AND/OR 木探索は利得が 3 通りある場合を扱うことが出来ないので，先手勝ちもしくは引き分けの利得を 1，先手負けの利得を 0 とした探索と，先手勝ちの利得を 1，先手負けと引き分けの利得を 0 とした探索の両方を行う．前者の探索で値が 1 になった節点は勝ちか引き分け，値が 0 になった節点は負けである．一方，後者の探索で値が 1 になった節点は勝ち，値が 0 になった節点は負けか引き分けである．

実験に使用した CPU は Intel Xeon X5690x2 12 コア，メモリは 24GB である．TT には 700,000,000 エントリー (約 20GB) を確保した．

5.1 4×4 から 7×7 TRAX の探索結果

4×4 から 7×7 TRAX は引き分けという結果が得られた．表 4，表 5 に実行時間と GC 発生回数を示す．引き分けの利得を 0 とした場合の 7×7 TRAX のみ GC が発生し，その回数は 11 回であった．

図 6 に実行時間をまとめる．引き分けの利得を 1 とした場合の 8×8 TRAX の結果は約 2 ヶ月で得られると見込まれる．また，引き分けの利得を 0 とした場合は約 2 年で得られると見込まれる．ただ，GC 発生回数は 8×8 TRAX ではさらに増えるであろうから，これらは楽観的な見込みと言えよう．

5.2 リミテッド TRAX を解く試み

リミテッド TRAX を解くことを試みる．2 手先の局面を列挙し，その局面を根節点として df-pn 探索を行う．回転・反転・平行移動を考慮すると，1 手先の局面数は 2，2 手先

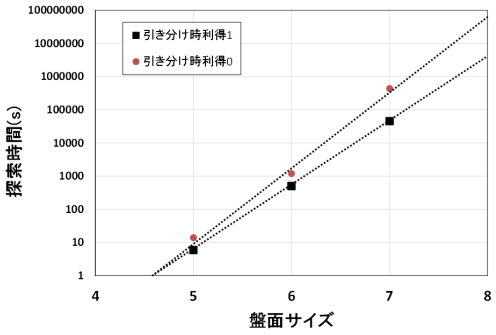

図 6 4×4 から 7×7 TRAX を解くのに要した時間

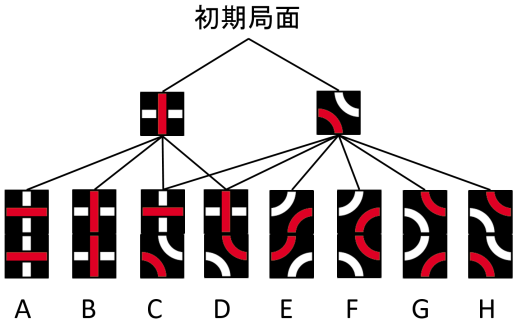

図 7 根節点 (白手番) 付近の DAG

の局面数は 8 となる．それらを図 7 に示す．2 手先の局面 G は明らかに先手 (白) 勝ちである．Df-pn 探索プログラムは，局面 C と E に関しては先手勝ちか引き分けという結果を出力した．局面 C に関しては 2 時間 (GC 0 回)，E に関しては 14 日 (GC 22 回) を要した．他の 2 手先の局面に関しては現在探索中である．

6. おわりに

本研究では，8×8 TRAX のルール上許されるタイル配置の総数を粗く見積もり，合法局面数を他のゲームと比較した．モンテカルロ法により粗く見積もられた局面数は 5×10^{18} 程度であり，これはチェッカーと同程度であった．また，4×4 から 7×7 までの TRAX を解き，8×8 TRAX のゲーム値を求めるのに要する時間を見積もり，これが現実的な時間で終わるのではないかという感触を得た．

8×8 TRAX の 2 手先の局面は 8 個あり，そのうち 1 つは先手勝ち，2 つは先手勝ちか引き分けである．なお，本研究では 3 本以上の同色のラインが集まる空きスペースができるような手は非合法手として扱い，空きスペースが残っていて合法手がない場合には手番プレイヤの負けとした．

参考文献

[1] Bailey, D.: *Trax Strategy For Beginners*, D.G. Bailey, New Zealand, second edition (1997).
[2] Allis, L. V. et al.: *Searching for solutions in games and artificial intelligence*, Ponsen & Looijen (1994).
[3] Schaeffer, J., Burch, N., Björnsson, Y., Kishimoto, A.,

Müller, M., Lake, R., Lu, P. and Sutphen, S.: Checkers is solved, *science*, Vol. 317, No. 5844, pp. 1518–1522 (2007).
[4] 篠田正人：将棋における実現可能局面数について，ゲームプログラミングワークショップ 2008 論文集， Vol. 2008, No. 11, pp. 116–119 (2008).
[5] Tromp, J.: Number of legal Go positions, `http://tromp.github.io/go/legal.html` (2016).
[6] 小谷善行，岸本章宏，柴原一友， 鈴木豪：ゲーム計算メカニズム: 将棋・囲碁・オセロ・チェスのプログラムはどう動く，コロナ社 (2010).
[7] Zobrist, A. L.: A new hashing method with application for game playing, *ICCA Journal*, Vol. 13, No. 2, pp. 69–73 (1970).
[8] Nagai, A.: A New Depth-First-Search Algorithm for AND/OR Trees, Master's thesis, The University of Tokyo, Tokyo, Japan (1999).
[9] Allis, L. V., van der Meulen, M. and Van Den Herik, H. J.: Proof-number search, *Artificial Intelligence*, Vol. 66, No. 1, pp. 91–124 (1994).
[10] 長井歩， 今井浩：df-pn アルゴリズムの詰将棋を解くプログラムへの応用，情報処理学会論文誌， Vol. 43, No. 6, pp. 1769–1777 (2002).
[11] Kishimoto, A., Winands, M. H., Müller, M. and Saito, J.-T.: Game-tree search using proof numbers: The first twenty years, *ICGA Journal*, Vol. 35, No. 3, pp. 131–156 (2012).
[12] Kaneko, T.: Parallel Depth First Proof Number Search. (2010).
[13] 金子知適，田中哲朗，山口和紀， 川合慧：新規節点で固定深さの探索を併用する df-pn アルゴリズム，第 10 回ゲーム・プログラミングワークショップ，pp. 1–8 (2005).