情報処理学会論文誌 Vol.0 No.0 1–8 (??? 1959)

ボードゲーム「オストル」の省メモリな完全解析

滝沢 拓己[1,a)]

受付日 xxxx年0月xx日, 採録日 xxxx年0月xx日

概要：「オストル (Ostle)」は雅ゲームスが 2017 年に発表したボードゲームである．自分の駒を動かして相手の駒を押し出すという単純なコンセプトだが，直前の盤面に戻る手を禁止するなどのルールにより高いゲーム性を持つ．このゲームは，無限ループによる実質的な引き分けは存在するものの，二人零和確定完全情報ゲームに属しており，後退解析 (retrograde analysis) により全ての局面の解析解を求められうるゲームである．本研究では後退解析に基づくプログラムを実装し，初期局面が引き分けであることを確かめた．さらに，勝利まで 147 手かかる局面が存在することや，任意の局面に到達するまでには最大 26 手かかることといった，興味深い性質を発見した．また本研究の解析プログラムの実装において，「オストル」のルールに根差した考察に加え，完備辞書 (fully indexable dictionary) を利用してメモリ消費量を削減したことが重要な役割を果たした．実装は GitHub で公開されている．
https://github.com/eukaryo/ostle_solver

キーワード：オストル，完全解析，後退解析，完備辞書，簡潔ビットベクトル

Complete Analysis of Board Game “Ostle” with Low Memory Consumption

Hiroki Takizawa[1,a)]

Received: xx xx, xxxx, Accepted: xx xx, xxxx

Abstract: “Ostle” is a board game released by Miyabi Games in 2017. The concept of the game is simple: move your pieces to push out your opponent’s pieces, but the rules, such as the prohibition of returning moves to the previous board, make the game highly challenging. “Ostle” belongs to perfect-information two-person zero-sum games. Therefore, from a theoretical viewpoint, all states in “Ostle” can be decided as winning, losing, or draw (including infinite loops). In this study, we implemented a program based on retrograde analysis and confirmed that the initial state is draw. In addition, we also discovered some interesting properties, such as the existence of a state that requires 147 moves to win, and that up to 26 moves are required to reach an arbitrary state from the initial state. In order to reduce memory consumption, we gave “Ostle”-specific considerations and utilized fully indexable dictionary. The source code is available on GitHub.
https://github.com/eukaryo/ostle_solver

Keywords: Ostle, retrograde analysis, fully indexable dictionary, succinct bitvector

1. はじめに

計算機によってボードゲームを解析することは，チャールズ・バベッジが 1864 年に自伝 [2] のなかで構想を記して以来，人類の目標のひとつであり続けている．近年における著名な成功例としては，2007 年にチェッカー [9] が完全解析されたことが挙げられる．また他方では，様々なボードゲームが新たに提案され，広く楽しまれている．例えば 2008 年に日本女子プロ将棋協会は「どうぶつしょうぎ」というボードゲームを子どもや初心者に向けて発表したが，これの完全解析 [17] が成されている．この研究は将棋と情報科学を結ぶ接点のひとつとなっいる．

[1] 東京大学大学院 新領域創成科学研究科
Graduate School of Frontier Sciences, The University of Tokyo, Kashiwa, Chiba 277–0882, Japan
[a)] takizawa_hiroki_17@stu–cbms.k.u–tokyo.ac.jp
c3375c15@gmail.com
https://github.com/eukaryo/ostle_solver

「オストル (Ostle)」は雅ゲームス[*1]が 2017 年に発表したボードゲームである．将棋やチェスに似たボードゲームで，相手の駒を場外に押し出すというシンプルなルールで人気がある．「オストル」を遊ぶためのキットが様々な店舗で販売されている．また公式サイトでは「オストル」をプレイするための pdf が無料で公開されており，印刷してハサミで駒を切り出せば遊ぶことができる．

ところで，ゲームの完全解析と呼ばれる研究は，実際には詳細さに応じて以下のように分類できると知られている．[1], [6]

(1) 初期局面の勝敗の厳密解を求める．
(2) (1) に加えて，計算機が現実的な対局時間の内で最善手を求めるための具体的な方法を与える．
(3) 対局中に出現しうるあらゆる局面について，その勝敗の厳密解を求める．
(4) (3) に加えて，勝利なら最短・敗北なら最長の，終局までに要する手数を求める．

チェッカーの完全解析 [9] では，大規模な終盤のデータベースを用意することで，(2) の意味での完全解析を達成していた．

本研究では，「オストル」に関して，前述の分類における (4) の意味での完全解析を達成した．加えて我々は本研究において，メモリ消費量を少なく抑え，かつ解析中のデータをストレージに退避させたり [8] もせずに解析を完遂するため，様々な工夫を行った．「オストル」のルールに根差した考察に加えて，後退解析 (retrograde analysis)[12] や完備辞書 (fully indexable dictionary)[7] という技術を利用した．完備辞書は文字列検索 [16] やバイオインフォマティクス [5] といった分野で用いられている既知の技術だが，本研究は異なる用途への興味深い応用例として位置付けられるであろう．

1.1 関連研究

ゲームの完全解析とは意味合いが異なるが，計算機がゲームにおいて人類に勝利することもまた科学者の目指すところである．端緒は 1950 年にクロード・シャノンが具体的なチェスプログラムの構成方法を示した [10] ことであろう．近年の著名な例としては，チェス [4]，囲碁 [11]，ある種の[*2]ポーカー [3] において，計算機が人間のトッププレイヤーに勝利したことが報告されている．

これらは計算機の能力の高さを示しているだけではなく，その研究過程で情報処理技術に関する新たな知見をもたらしてもいる．例えばコンピュータ将棋の研究のなかで，複数の将棋ソフトによる合議制の有効性が発見された．[13], [14] 合議制は「あから 2010」というシステムに採用され，耐障害性の向上や棋風の変化をもたらし，清水市代・女流王将（当時）への勝利に貢献したとされる．[19] その後も合議制の有効性の詳細な評価や原理の解明のために研究が続けられている．[15]

1.2 「オストル」のルール

「オストル」のルールは公式サイト[*3]で分かりやすく図解されているが，本項でも説明することにする．

「オストル」のゲーム盤は 5 × 5 マスから成る．2 人のプレイヤーが盤の最上段と最下段に各々の 5 個の駒を置いて，中央に穴を置いた状態を初期局面とする．

プレイヤーは手番において自分の駒もしくは穴を上下左右の 4 方向に 1 マスだけ動かすことができる．穴を動かせるのは，動かす先のマスが空白である場合のみである．一方で駒を動かすときには，移動先のマスに別の駒があっても押しのけて移動することができる．

押しのけられた駒は同じ方向に移動する．この「押しのけられて移動する」処理は必要に応じて何回でも再帰的に行われる．最終的に，盤外か穴のマスに到達した駒はゲームから除外される．敢えて自分の駒を盤外や穴に向けて移動させることも許される．相手の駒が残り 3 つになれば勝ちであり，自分の駒が残り 3 つになれば負けである．

いかなる盤面も，その 2 手前の盤面と完全に同一になってはいけない．そうする指し手は禁じ手とされる．

2. 解析手法

2.1 準備

2.1.1 用語

以降，「盤面」という単語は「盤上の駒と穴の配置」を意味することにする．「オストル」では直前の盤面に戻す指し手が禁じ手であるため，盤面が完全に同一であって禁じ手が異なるようなケースは区別して後退解析しなければいけない．そのため，「『現在の盤面』に『禁じ手が存在するか』の情報と，存在するならその『禁じ手となる指し手（あるいは直前の盤面など，禁じ手を特定できる情報）』を加えたデータ」のことを「局面」と呼ぶことにする．

また，手番側が即座に勝利できる盤面のことを「即勝利可能盤面」と呼ぶことにする．

2.1.2 数字 25 個フォーマット

「オストル」の盤面を書き表す方法として，数字 25 個からなるフォーマットを以下の通りに定義する．

- 「オストル」の 5 * 5 マスを flatten した 25 マスを考え，各マスの状態に (‘0’,‘1’,‘2’,‘3’) の 4 文字を割り当て，25 文字の文字列に変換する．
- 具体的には，空白マスなら ‘0’，手番側の駒なら ‘1’，相手の駒なら ‘2’，穴なら ‘3’，と割り当てる．

[*1] https://twitter.com/MiyabiGames
https://gamemarket.jp/booth/2009
[*2] six-player no-limit Texas hold’em poker
[*3] https://gamemarket.jp/game/76310

- その数字 25 文字の文字列に，‘-’ を好きなだけ挿入してもよいこととする．解釈の際には ‘-’ は単に無視される．

例えば初期局面の盤面は 11111-00000-00300-00000-22222 と表せる．（対称性により他の表記も可能である．例えば 10002-10002-10302-10002-10002 など）

2.2 解析全体の流れ

まず「オストル」の盤面としてありうる盤面を全列挙した．その後，各盤面に対してありうる禁じ手を全列挙して，ありうる局面の全列挙とした．次に，何らかの盤面から到達可能な局面のみを「非自明な局面」として，これを全列挙した．そして，後退解析のためのゲームグラフ（局面をノードとして指し手をエッジとするグラフ）を陽に構築せずに後退解析を行った．加えて後退解析とは別に，初期局面から全ての局面に向けて幅優先探索を行い，各局面が初期局面から最短何手で到達できるか調べた．列挙された全局面が初期局面から到達可能であるかどうかも同時に調べた．

2.3 盤面の全列挙

2.3.1 定義

- 「即勝利可能盤面」までを含み，勝利する手を指した後の盤面は含めない．
- 「即勝利可能盤面」でも，勝利する手を指さなくてもよい．言い換えると，どこかで勝利を見逃さないと決して辿り着けないような盤面も全列挙に含める．
- 対称な盤面は同一視する．（水平反転・垂直反転・行列転置の 3 操作を行うか行わないかで，合計最大 8 通りの対称性がある）

2.3.2 考察 1

空白 25 マスに対して『最初にどこかに穴 1 個を配置して，その後残り 24 マスに手番側のコマ 5(or 4) 個と相手のコマ 5(or 4) 個を配置する』という操作を行うと任意の盤面を構成できる．これを文字通り全通り実行して全列挙することを考える．このとき対称な盤面をあとで同一視するのならば，穴の配置箇所として考慮すべき場所は，上三角行列に含まれ，かつ左側 3 列に含まれる 6 マス (0-origin で座標を書くなら ((0,0),(0,1),(0,2),(1,1),(1,2),(2,2))，数字 25 個フォーマットで書くなら 33300-03300-00300-00000-00000) だけでよい．なぜなら，その他のマスに穴を配置した場合，その後でコマをどのように配置しようとも，結果生じる盤面と対称な盤面が，列挙した盤面のなかに必ず存在するからである．

2.3.3 考察 2

もう一つの考察として，上記の 6 マスのうち異なるマスに穴を配置した 2 つの盤面同士は決して対称の関係にならない．そしてコマの数が異なる盤面同士も決して対称の関係にならない．ゆえに，『穴の位置とコマの数』の場合分け（24 通り）によって，全盤面を網羅しつつ相互排他的に分割できるといえる．すると，対称な盤面の検出もその場合分けの内側同士でだけ考慮すればよくなる．

Algorithm 1 対称な盤面を同一視する

Require: p:盤面
Require: $H(p)$:盤面を引数とし，水平反転させる関数
Require: $V(p)$:盤面を引数とし，垂直反転させる関数
Require: $T(p)$:盤面を引数とし，行列転置させる関数
Require: $C(p)$:盤面を引数とし，整数を返す単射な関数

```
a ← p
for i ← [1, 7] do
    b ← a
    if (i & 1) ≠ 0 then
        b ← H(b)
    end if
    if (i & 2) ≠ 0 then
        b ← V(b)
    end if
    if (i & 4) ≠ 0 then
        b ← T(b)
    end if
    if C(a) < C(b) then
        a ← b
    end if
end for
return a
```

2.3.4 対称な盤面を同一視する方法について

全ての盤面から整数への単射な関数は簡単に用意できる．（例えば，穴の位置を 5bit で表し，手番側と相手の駒の配置を各々 25bit で表せば，任意の盤面を 55bit で表せる）そのため，全ての盤面に順序を定めることができる．そのうえで，引数盤面と対称な盤面たちのうち順序が最も若い盤面を代表盤面として返す関数を用意すれば，対称な盤面を同一視できる．Alogtithm 1 はその関数の例である．このアイデアは本研究に新規なものではなく広く知られており，他のボードゲームの完全解析を行った研究 [18] やオセロソフトの実装などで用いられている．

2.3.5 手順

以上の考察により，盤面の全列挙の計算中に余分に必要になるメモリ量を削減できた．基本戦略は『対称性を考慮せず全列挙したのち，対称な盤面を同一視する関数を作用させ，ソートしてから隣接する重複要素を削除する』というものだが，対称性を考慮しない全列挙を 24 通りの全部一気にではなく一つづつ行うことができたからである．（全部一気に行えば，最終的に得られる盤面数の 2 倍程度の盤面を一時的に保持する必要が生じる）

2.4 非自明な局面の全列挙

2.4.1 考察 1

「即勝利可能盤面」においては，勝利する指し手が禁じ手になっていることは決してない．なぜなら，勝利する指し手とは相手の駒を押し出す手であるが，一般に，駒を押し出す手は明らかに禁じ手になりえないからである．結論として，「即勝利可能盤面」が内包する局面は全て即勝利可能な局面である．

2.4.2 考察 2

任意の盤面における合法手の数はたかだか 24 通りである．（着手位置 6 箇所，4 方向）ゆえに，各盤面はたかだか 25 個の局面を内包している．そのため，後段で後退解析を行う際に，結果となる終局までの手数を格納する配列は，盤面数 * 25 要素あれば十分である．しかし，（後述するが）盤面数が 2.74 G 通りあるため，仮に 16bit 整数の配列を盤面数 * 25 要素だけ確保すると，それだけで 137 GB になる．このメモリ消費を削減するために，「非自明な局面」だけを列挙し，その要素数の配列だけを確保することにした．その配列への添字アクセスには完備辞書 [7] が有効である．この点については 2.5 節で詳述した．

2.4.3 考察 3

前述の通り，各盤面はたかだか 25 個の局面を内包している．そのうえで，それら 25 個の多くは到達不可能な局面であると推測される．なぜなら，ある盤面 p 上の合法手 m について，m が禁じ手となるような局面に到達可能であることは，p から m を指して遷移した先の盤面 p' から 1 手で p に戻るような指し手 m' が存在することと同値だが，そのようなケースは割合としては少ないと考えられるからである．（この推測が正しかったことは 3.2 節で判明する）そこでまず，「非自明な局面」を以下のように定義した．

2.4.4 後退解析における「非自明な局面」の定義

- 「局面 p' が非自明である」 $\Longleftrightarrow$ 「ある盤面 p と p 上の合法手 m の組が存在して，p にて m を指すことによって p' へ遷移でき，かつ，p' を盤面として見たとき即勝利可能ではない」
- 後退解析のための全列挙においては，「即勝利可能盤面」が内包する局面は全て自明であるとして，列挙の対象外とする．

2.4.5 手順

Algorithm 3 で示した手順により非自明な局面を列挙した．いくつかの初歩的な関数については擬似コードの記述を割愛したが，ソースコードが GitHub で公開されている．(https://github.com/eukaryo/ostle_solver)

2.4.6 合法手の全列挙について

合法手の全列挙は単純なので擬似コードを割愛したが，要請される条件と注意点が以下の通りに存在する．

- 引数の盤面 p が同一ならば，返り値（指し手のリスト）は常に完全に同一順序でなければならない．なぜなら，その指し手の添字番号を用いて局面の添字番号を計算するからである．
- 複数の異なる指し手によって同一の盤面に遷移することがある．（例：初期局面で，端の駒を左・右・下に動かす手）しかし，これらを同一視する必要はなく，単純に各々異なる指し手として出力しても問題ない．なぜなら，そのような指し手は駒を押し出す手であり，ゆえに決して禁じ手にならないからである．言い換えると，複数の指し手によって同一の盤面に遷移するケースにおいて，それらの指し手を同一視しなくても，禁じ手の数がたかだか 1 つであることに変わりはない．この注意点に関しては 2.6 節での議論に用いられる．

Algorithm 2 禁じ手の有無を調べる（通し番号を返す）

Require: p:現在の盤面
Require: q:直前の盤面
Require: $U(p)$:Algorithm 1 (対称な盤面を同一視する関数)
Require: $G(p)$:盤面を引数とし，全合法手のリストを返す関数
Require: $D(p,m)$:盤面と手を引数とし，遷移後の盤面を返す関数
Ensure: 返り値は [0,24] の整数であり，禁じ手が存在しないなら 0 を，存在するなら種類に応じて異なる値を返す．

```
M ← G(U(p)) { 合法手を全列挙する．}
for i ∈ [0, len(M)) do
    r ← D(U(p), M[i]) { 盤面 p にて i 番目の指し手を指す }
    if U(r) = U(q) then
        return i + 1
    end if
end for
return 0
```

Algorithm 3 非自明な局面を全列挙する

Require: P:全盤面のリスト
Require: $U(p)$:Algorithm 1 (対称な盤面を同一視する関数)
Require: $F(p,p')$:Algorithm 2 (禁じ手の有無を調べる関数)
Require: $G(p)$:盤面を引数とし，全合法手のリストを返す関数
Require: $D(p,m)$:盤面と手を引数とし，遷移後の盤面を返す関数
Require: $B(p,P)$:引数盤面 p の，P における添字番号を返す関数
Require: $M(p)$:引数盤面 p が「即勝利可能盤面」かを返す関数

```
V ← bitvector(len(P) * 25)
V ← 0 { 全ビットを 0 で初期化する．}
for i ∈ [0, len(P)) do
    M ← G(P[i]) { 盤面 p の合法手を全列挙する．}
    for m ∈ M do
        p' ← U(D(m, p)) {p で m を指すことで p' に遷移する．}
        if M(p') is False then
            V[B(p'|P) × 25 + F(p', P[i])] ← 1 {p' の禁じ手の有無を調べて局面を確定し，対応する V 上のビットを立てる．}
        end if
    end for
end for
return V
```

2.5 「非自明な局面」の配列と完備辞書について

Algorithm 3 の返り値は，全盤面数 * 25 bit のビットベクトル V であり，「非自明な局面」に対応する位置のビットが立っている．立っているビットの総数が「非自明な局面」の総数であるから，その要素数の 16bit 整数配列 S を用意して，後退解析の結果を S に格納することにした．

V 上で i 番目に位置する局面にアクセスするとき，それの S 上での添字番号を求める必要がある．その答えは V 上の範囲 $[0, i)$ のなかで立っているビットの数に等しい．このように，ビットベクトルで先頭から i 番目までに立っているビットの数を求めるクエリは「ビットベクトルへの rank クエリ」と呼ばれている．ビットベクトルが変更されない前提のもとで，完備辞書 [7] という補助データ構造を用意すれば，rank クエリを定数時間で処理できることが知られている．完備辞書の空間計算量は $o(|V|)$ にできるという利点も知られている．とはいえ，今回は「オストル」の完全解析についてしか考えないので，($|V|$ を無限大に飛ばしたときの漸近的な挙動であるところの）計算量理論は重視せずともよい．今回は計算速度とのバランスを考えつつ実装し，結果として完備辞書の大きさは V の大きさの 3 割程度に収まった．

2.6 後退解析

後退解析をナイーブに実装する場合は，局面をノードとして指し手をエッジとするグラフを初めに構築する．しかし今回は，そのグラフを陽に持つことなく後退解析する方法を採用した．すなわち，「全ての可能な盤面 $p \in P$ について 1 手先読みして，盤面 p が内包する局面たちの解析解が新たに得られたならそれを記録する（または，決着までの手数として前回と異なる値が得られたなら更新する）」という操作を反復的に繰り返し，全く記録・更新されなくなったら終了とした．

解析の具体的な手順は以下の通りであった．盤面 p に着目したとして，まず（禁じ手のルールは一旦無視して）p 上の合法手を全列挙した．次に，各合法手で盤面を p から p' に遷移させ，局面 (p', p) について後退解析による解が得られているか（ないしは p' が即勝利可能盤面であるか）を調べた．（この際に前述した完備辞書ベースの添字アクセスを用いた）次に，それらの結果をもとに，盤面 p が内包する全局面の解析解を求めた．

例えば勝利が確定している指し手が 2 つ以上あった場合，盤面 p が内包する全ての局面の勝利が確定する．なぜなら，2 つの指し手のどちらかが禁じ手だとしてももう片方の指し手で勝利できるからである．あるいは例えば，ただ 1 つの指し手 m で勝利が確定していて，残り全ての指し手で敗北が確定している場合，m が禁じ手である局面の敗北が確定し，残り全ての局面の勝利が確定する．（その他のケースについての説明は割愛するが，興味のある読者は公開されているソースコードを参照して欲しい）

この議論では，全列挙された合法手のうち禁じ手がたかだか 1 つであることが前提とされている．しかし，2.4.6 節で述べたとおり，複数の異なる指し手が同じ盤面に遷移する場合でもこの前提は保たれるため，それらの指し手を同一視する必要はない．このことにより合法手を全列挙する関数を簡潔かつ高速にすることが可能となった．

2.7 初期局面からの幅優先探索

前述した「非自明な局面」の全列挙では，何らかの盤面からの遷移先でありうる局面すべてを列挙した．この方法によって得られる局面の集合には，初期局面から到達可能な全ての局面が含まれる．しかし，初期局面から到達不可能な局面が含まれているかどうかは不明であった．

この問いを解決するため，初期局面から幅優先探索を行い，全列挙した「非自明な局面」の全てに到達可能であることを確かめた．また同時に，初期局面から最短何手で各局面に到達できるかを調べた．

この幅優先探索の際には，「即勝利可能盤面」が内包する局面も非自明な局面として扱うことにした．

2.8 計算環境

本研究の解析には Amazon EC2 の c5.9xlarge インスタンスを用いた．具体的なスペックは，Intel(R) Xeon(R) Platinum 8124M CPU @ 3.00GHz, 18 Cores, 2 Threads per core, 72 GB Memory であった．

3. 解析結果

3.1 盤面の全列挙

盤面を全列挙した結果，合計 2,735,147,685 通りの盤面が得られた．内訳は表 1 に示した通りであった．ただし，これは初期局面から到達不可能な盤面を少数ながら含む．例えば，盤面 00000-00000-00001-11112-22223 はいかなる盤面からも合法手によって到達できない．

これらの盤面のうち，「即勝利可能盤面」は 399,102,582 通り (14.5916 %) であった．

3.2 後退解析のための，非自明な局面の全列挙

合計 2,735,147,685 通りの盤面が得られたため，局面数の上界は 68,378,692,125 (= 2,735,147,685 * 25) 通りとなる．このうち非自明な局面は 11,148,725,918 通り (16.3044 %) であった．ただし，ここでは「即勝利可能盤面」に含まれる局面は列挙の対象外とした．

3.3 後退解析

後退解析の結果は表 2 に示した通りであった．表にある局面数を全て足し合わせると 11,148,725,918 となり，非自明な局面の数と一致する．初期局面は引き分けであった．

表 1 列挙された盤面数

穴の位置	手番側の駒数	相手の駒数	盤面数
(0,0)	5	5	247,127,256
(0,1)	5	5	494,236,512
(0,2)	5	5	247,127,256
(1,1)	5	5	247,127,256
(1,2)	5	5	247,127,256
(2,2)	5	5	61,788,564
(0,0)	5	4	82,378,152
(0,1)	5	4	164,745,504
(0,2)	5	4	82,378,152
(1,1)	5	4	82,378,152
(1,2)	5	4	82,378,152
(2,2)	5	4	20,598,588
(0,0)	4	5	82,378,152
(0,1)	4	5	164,745,504
(0,2)	4	5	82,378,152
(1,1)	4	5	82,378,152
(1,2)	4	5	82,378,152
(2,2)	4	5	20,598,588
(0,0)	4	4	25,744,590
(0,1)	4	4	51,482,970
(0,2)	4	4	25,744,590
(1,1)	4	4	25,744,590
(1,2)	4	4	25,744,590
(2,2)	4	4	6,438,855

手番側の駒数	相手の駒数	盤面数
5	5	1,544,534,100
5	4	514,856,700
4	5	514,856,700
4	4	160,900,185

計算には 2.8 節に記述した計算環境を用いて，完了まで約 35 時間を要した．計算結果をファイル出力すると 97.4 GB となった．

注意点として，表 2 における「手数」は即勝利可能盤面に到達するまでの手数であるから，偶数だと勝利を，奇数だと敗北を意味する．これは例えば詰将棋において「n 手詰め」の n が奇数だと勝利を意味するのと反対になっている．表 2 の「手数」に 1 を足せば実際に勝利 (敗北) するまでの手数となり，偶奇と勝敗の関係性が詰将棋等のそれと一致する．

3.3.1 勝利まで 147 手かかる局面

表 2 によると，勝利までの手数が最長の局面は勝利まで 147 手かかり，それは 7 局面存在する．具体的には，2 つの盤面 00100-11200-00001-31022-20020 と 00012-00222-10010-01010-00032 がその「147 手詰め盤面」であることがわかった．この 2 つの盤面は，禁じ手がない局面において双方とも 147 手詰めであった．残りの 5 局面は，この 2 つの盤面に禁じ手がある局面のなかの 5 通りであった．

表 2 「即勝利可能盤面」に到達するまでの手数とそのような局面の数．手数 inf は引き分けを意味する．

手数	局面数	手数	局面数	手数	局面数
inf	339,367,091	50	9,994,880	100	27,311
1	577,327,477	51	8,667,693	101	22,936
2	1,208,259,074	52	7,940,165	102	21,612
3	250,385,204	53	6,893,605	103	18,273
4	514,915,495	54	6,314,317	104	15,740
5	294,380,826	55	5,502,897	105	13,176
6	569,040,388	56	5,039,556	106	10,996
7	352,821,271	57	4,382,022	107	9,486
8	559,455,180	58	3,998,228	108	7,740
9	379,808,723	59	3,470,949	109	7,290
10	535,425,081	60	3,163,370	110	5,481
11	379,563,356	61	2,764,844	111	5,015
12	462,977,806	62	2,521,503	112	3,885
13	352,330,833	63	2,195,886	113	3,741
14	395,732,654	64	2,010,460	114	2,885
15	318,063,321	65	1,759,413	115	2,593
16	338,626,104	66	1,595,483	116	1,785
17	281,546,559	67	1,400,525	117	1,447
18	287,320,623	68	1,267,104	118	1,005
19	244,575,415	69	1,108,248	119	771
20	241,883,809	70	1,006,232	120	690
21	208,979,928	71	878,861	121	425
22	202,402,295	72	801,288	122	433
23	176,160,115	73	700,241	123	282
24	168,414,137	74	649,879	124	387
25	146,956,825	75	561,353	125	308
26	139,319,551	76	528,175	126	321
27	121,669,807	77	449,395	127	212
28	114,485,394	78	434,621	128	226
29	99,874,325	79	361,217	129	205
30	93,577,296	80	352,810	130	179
31	81,527,252	81	291,770	131	211
32	76,116,163	82	285,312	132	73
33	66,122,478	83	233,005	133	113
34	61,530,140	84	229,210	134	44
35	53,329,309	85	186,949	135	99
36	49,530,287	86	179,324	136	68
37	42,847,011	87	144,595	137	146
38	39,660,828	88	138,428	138	74
39	34,312,611	89	113,388	139	106
40	31,679,932	90	108,165	140	44
41	27,408,599	91	86,970	141	40
42	25,272,339	92	83,620	142	16
43	21,858,443	93	67,110	143	7
44	20,072,851	94	65,247	144	9
45	17,341,986	95	53,164	145	4
46	15,898,815	96	51,284	146	7
47	13,731,415	97	40,837		
48	12,600,718	98	38,653		
49	10,908,242	99	30,388		

表 3 初期局面から各局面に到達するまでの最短手数とそのような局面の数．手数 0 は初期局面そのものを意味する．

手数	局面数	手数	局面数
0	1	14	411,886,389
1	9	15	767,525,717
2	102	16	1,262,744,615
3	954	17	1,851,900,832
4	6,329	18	2,259,589,185
5	33,052	19	2,356,709,939
6	147,620	20	1,884,609,912
7	556,811	21	1,172,437,043
8	1,863,530	22	475,193,903
9	5,542,830	23	113,051,575
10	15,200,179	24	9,503,831
11	38,307,337	25	115,519
12	91,419,758	26	97
13	201,637,267		

3.4 初期局面からの幅優先探索

3.4.1 非自明な局面の全列挙

初期局面からの幅優先探索のために非自明な局面の全列挙を行う際には，「即勝利可能盤面」に含まれる局面も列挙対象とする必要があった．その結果得られた非自明な局面は 12,919,984,336 通り (18.8948 %) であった．

（ちなみに前述の通り，後退解析の際には「即勝利可能盤面」に含まれる局面は対象外としていたが，その結果得られた非自明な局面は 11,148,725,918 通り (16.3044 %) であった．）

3.4.2 探索結果

探索結果は表 3 の通りであった．初期局面からその局面に到達するまで必要な最短手数が最も多い局面では，到達までに 26 手かかることがわかった．また，表 3 の局面数の総和が 12,919,984,336 であることから，先に全列挙した「非自明な局面」は全て初期局面から到達可能であることが確かめられた．計算には 2.8 節に記述した計算環境を用いて，完了まで約 5 時間を要した．計算結果結果をファイル出力すると 114 GB となった．

4. 結論

本研究では，「オストル」の全局面の勝敗および最善を尽くした場合の手数を求める完全解析を行った．結果として，初期局面が引き分けであることに加えて，勝利まで最長で 147 手かかる局面が存在することや，26 手以内で任意の局面に到達できることといった，興味深い性質を発見した．本研究の解析結果の出力ファイルをもとに，任意の局面で最善の指し手を瞬時に出力するプログラムを作ることは容易である．初期局面は引き分けであるから，そのプログラムは人間が間違えない限り決して負けず，引き分けの手を選び続けるはずである．そのうえで，相手の人間が間違えやすい局面に誘導するプログラムを作ることは今後の課題となるだろう．

世界には様々な未解析のボードゲームがあるが，計算機の能力は限られている．そのため，より多様なゲームをより詳細なレベルで完全解析するためには，メモリ消費量や計算時間を削減する工夫をゲームごとに考案することが不可欠である．本研究では様々な考察と完備辞書の利用などに取り組み，結果として，現在では PC としても容易に入手可能な安価な計算環境で解析を行うことができた．本論文および公開されているソースコードが，ボードゲームの完全解析の事例報告のひとつとして，将来にわたり様々なゲームの解析を試みる人々の助けになれば幸いである．

謝辞 本研究ならびに学業全般にわたって多大なご指導ご鞭撻を頂いたことに関して，筆者が所属する東京大学大学院新領域創成科学研究科メディカル情報生命専攻浅井研究室の皆様に深く感謝します．また，私に「オストル」を紹介して頂いたことについて，筆者の友人であり現在は同専攻笠原研究室の博士課程に在籍している横山稔之氏にも深く感謝します．

参考文献

[1] ALLIS, L. V.: Searching for Solutions in Games and Artificial Intelligence, *Ph.D. Thesis, Department of Computer Science, University of Limburg* (1994).

[2] Babbage, C.: *Passages from the Life of a Philosopher*, London: Longman (1864).

[3] Brown, N. and Sandholm, T.: Superhuman AI for multiplayer poker, *Science*, Vol. 365, No. 6456, pp. 885–890 (online), DOI: 10.1126/science.aay2400 (2019).

[4] Campbell, M., Hoane, A. J. and Hsu, F.-h.: Deep Blue, *Artif. Intell.*, Vol. 134, No. 1–2, p. 57–83 (online), DOI: 10.1016/S0004-3702(01)00129-1 (2002).

[5] Conway, T. C. and Bromage, A. J.: Succinct data structures for assembling large genomes, *Bioinformatics*, Vol. 27, No. 4, pp. 479–486 (online), DOI: 10.1093/bioinformatics/btq697 (2011).

[6] Gévay, G. E. and Danner, G.: Calculating ultra-strong and extended solutions for Nine Men's Morris, Morabaraba, and Lasker Morris, *IEEE Transactions on Computational Intelligence and AI in Games*, Vol. 8, No. 3, pp. 256–267 (2015).

[7] Jacobson, G. J.: Succinct static data structures, *Ph.D. Thesis, Carnegie Mellon University* (1988).

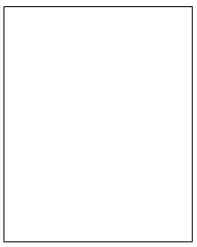
[8] Romein, J. and Bal, H.: Solving awari with parallel retrograde analysis, *Computer*, Vol. 36, No. 10, pp. 26–33 (online), DOI: 10.1109/MC.2003.1236468 (2003).

[9] 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).

[10] Shannon, C. E.: XXII. Programming a computer for playing chess, *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, Vol. 41, No. 314, pp. 256–275 (online), DOI: 10.1080/14786445008521796 (1950).

[11] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou,

I., Panneershelvam, V., Lanctot, M. et al.: Mastering the game of Go with deep neural networks and tree search, *nature*, Vol. 529, No. 7587, pp. 484–489 (2016).
[12] Thompson, K.: Retrograde analysis of certain endgames., *J. Int. Comput. Games Assoc.*, Vol. 9, No. 3, pp. 131–139 (1986).
[13] 伊藤毅志：コンピュータ将棋の新しい波：4. 合議アルゴリズム「文殊」 単純多数決で勝率を上げる新技術，情報処理，Vol. 50, No. 9, pp. 887–894 (2009).
[14] 伊藤毅志，小幡拓弥，杉山卓弥，保木邦仁：将棋における合議アルゴリズム——多数決による手の選択，情報処理学会論文誌，Vol. 52, No. 11, pp. 3030–3037 (2011).
[15] 曾根彰吾， 長束薫，由井薗隆也，飯田弘之：同一チェスプログラムグループによる合議制における投票結果の分散パターンの調査，ゲームプログラミングワークショップ 2014 論文集，Vol. 2014, pp. 75–81 (2014).
[16] 岡野原大輔：高速文字列解析の世界: データ圧縮・全文検索・テキストマイニング，岩波書店 (2012).
[17] 田中哲朗：「どうぶつしょうぎ」の完全解析，情報処理学会研究報告. GI, [ゲーム情報学]， Vol. 22, pp. C1–C8 (2009).
[18] 田中哲朗：ボードゲーム「シンペイ」の完全解析，情報処理学会論文誌， Vol. 48, No. 11, pp. 3470–3476 (2007).
[19] 保木邦仁，金子知適，横山大作，小幡拓弥， 山下宏：あから 2010 勝利への道：2. あから 2010 のシステム設計と操作概要，情報処理， Vol. 52, No. 2, pp. 162–169 (2011).



滝沢 拓己 （学生会員）

2017 年東京大学理学部生物化学科卒業．2019 年同大学大学院新領域創成科学研究科メディカル情報生命専攻修士課程修了．現在は同博士課程に在籍中．