ゴブレットゴブラーズの必勝法

宮城県仙台第三高等学校

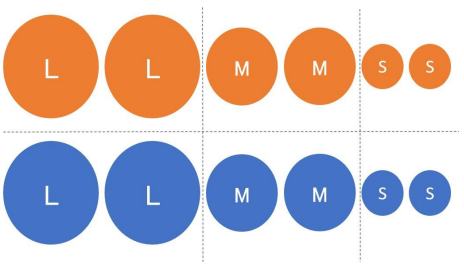
私たちは,二人零和無限確定完全情報ゲームのゴブレットゴブラーズの必勝法が自分たちで求められるのか興味を持ち,研究を始めた.必勝法は後退解析と呼ばれる解析方法を用いることで求められるのではないかと仮説を立て,実際にアルゴリズムを設計しC言語で実装した.当初はプログラムの実行所要時間が100年を超えていたが,3進数の考え方を用いた独自のデータ形式の利用やアルゴリズムの改良を繰り返し,30分で全ての処理が終わるまでに高速化を果たした.また,実行結果からゴブレットゴブラーズは先手必勝であることが判明した.

1　背景

ゴブレットゴブラーズはblue orangeが販売する二人用ボードゲームであり,ゲーム理論では二人零和無限確定完全情報ゲームに分類される.二人零和有限確定完全情報ゲームに分類されるゲームは,必ず先手必勝,後手必勝,引き分けのいずれかであることが知られている(ツェルメロの定理).そこで私たちは,二人零和無限確定完全情報ゲームのゴブレットゴブラーズの必勝法が自分たちで求められるのか興味を持ち,研究を始めた.

2　ゴブレットゴブラーズ

ゴブレットゴブラーズは3×3のマスの上で三つのサイズのコマを交互に動かして遊ぶゲームである.お互いに三つのサイズ(以下S,M,Lで表記)のコマを二つずつ所持しており,互いのコマはオレンジ色と青色に区分されている[資料1].



資料1

3　ルール

ルールは以下の通りである.

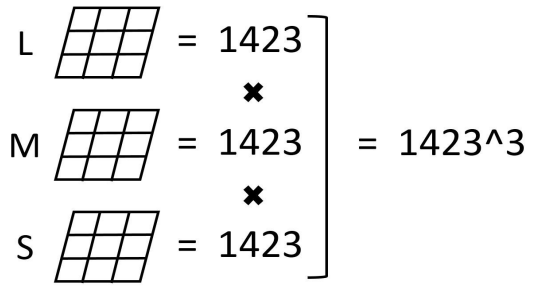
- 自分のコマを盤面の空いている場所に置くことができる.
- 自分のコマより小さいコマに被せて隠すことができる.
- 自分のコマの上に自分のコマを被せることもできる.
- 持ち駒のほか,既に盤面に置いた「見えている自分のコマ」を動かすこともできる.

4　総盤面数

盤面の領域をサイズごとに分割する,他領域内のコマ同士は互いに干渉せず領域のサイズとコマの数が領域によって変わらないことから,このゲームの領域ごとの局面数は9マスに青のコマ二つとオレンジのコマ二つを並べる時の並べ方の通り数と同じである.この値は盤面上にある青のコマの数をn,盤面上にあるオレンジのコマの数をmと置くと下の計算式で表すことができる.

$$\sum_{n=0}^{2}\sum_{m=0}^{2}\frac{9!}{n!m!(9-n-m)!!}$$

この式を計算すると,領域ごとの局面数は1423だとわかる.それぞれの領域は互いに干渉しないので,このゲームの総局面数は1423の3乗で2881473967とわかる資料2.



資料2

5　解析方法

ボードゲームの完全解析にとられる手法はいくつかあるが,今回は後退解析と呼ばれる解析方法を用いて解析を進めた.後退解析を選んだ理由としては,ミニマックス法やGlundy数を用いた解析法よりも理論が単純であるため,プログラミングをしたことがなかった私たちでも実現可能だと考えたためである.後退解析のアルゴリズムを以下に示す.

1. 勝負のついた局面の集合から開始する.

2. 勝負のついた局面の1手前の局面を求める .
   → (手番プレイヤーの) 負け局面から1手前の局面を (手番プレイヤーの) 勝ち局面とする.
   → 勝ち局面から 1手前の局面の勝敗が未確定のとき,そこから可能な手がすべて勝ち局面に遷移するときは負け局面とする.

3. 操作を繰り返して,勝ち,負け局面の集合がそれ以上増えなくなったら終了とする.

後退解析には遷移先の局面の解析結果(以下状態値と表記)が必要である.そのため解析済みの局面の集合と,それぞれの局面に対応した状態値の保存が必須である.

6　局面の表現方法

このゲームにおいて,勝敗に関係するのは見えているコマだけである.そのため,マスごとに見えているコマが誰のものかを探索する処理が必要となる.さらに,後退解析のアルゴリズムで最も多く呼び出されるのも,マスごとに見えているコマが誰のものかを探索する処理である.そこで,本研究では三振数を用いた独自の表現方法で局面を表すことで処理の高速化を図った.コマをサイズとプレイヤーごとに分割し,それぞれに固有値を割り振る資料3.Sサイズは1,Mサイズは3,Lサイズは9.自分のコマなら符号は正,相手のコマなら符号は負である.

	青	橙
L	+9	−9
M	+3	−3
S	+1	−1

資料3

9個のマスそれぞれで置かれているコマの固有値を加算すると,加算した値の符号がそのマスにおかれているコマの中で最も大きい(見えている)コマのプレイヤーの符号と合致する.これはn桁の3進数の最小値がn-1桁の3進数の最大値よりも大きいという性質を利用したものである.この表現方法を用いることで,最大三回のアクセスと比較の処理のループを,三回の加算の処理に置き換えることができる.一般に比較と

加算では,加算の方が処理が速くなることが多い.さらに,遷移可能な局面を探索する処理においても,マスごとの固有値の和の絶対値と動かすコマの固有値の絶対値の大小を比較することで,比較的容易に遷移可能な局面を算出することができる.この表現方法を用いた場合,この表現方法を用いない場合に比べ約10倍の高速化が見られる.

７　状態値の保存方方式

この表現方法は後退解析の状態値の保存に適さない.そのため一意で連続した番号を全局面に割り振り,総局面数と同じサイズの一次元リストを用いて,状態値の保存を行った.その際,番号と局面を相互に変換するアルゴリズムを用いることで局面の番号を探索する過程を省略している.このアルゴリズムを用いることで,メモリに保存するデータに局面の詳細がなくとも番号から局面を復元することが可能であるため,必要なメモリ量が大きく削減されることとなった.

８　プログラム

今回のプログラムは,総局面の列挙と勝敗の確認を行なったのちその結果に基づいて後退解析を行ったが,そのうち総局面の勝敗確認にかかった時間は並列化ありでは1秒以下,並列化なしでは約30秒であった.プログラムはC言語で記述,gccコンパイラを用いてコンパイルを行った.コンパイル時に-O3オプションで最適化を行った.

９　結果

約30分でこのゲームは先手必勝であるという結果が得られた.一手目に限った話で言えば,S,Lのコマはどのマスに置いても必勝法に繋がるが,Mのコマはどのマスに置いても相手が最善手を打ち続けたら必ず自分が負けるということが分かった.

１０.１　並列化による改善

プログラムにOpenMPによる並列化の処理を書き加えた場合,処理時間は10分以下に縮まった.しかしパラメーターの組み合わせを十数回試したが処理時間が9分以下になることはなかった.これは今回用いた並列化ではスレッドの同期を行わない動的なスレッド管理であったため,データの割り振りによるオーバーヘッドが発生しやすいというところに起因していると予測した.また並列化は保守性が低下する要因になりうるため,並列化を用いて得られた結果は並列化を用いないで得られた結果と照合して保守性が保たれているかのチェックを行なった.結果50回並列化を用いたプログラミングを実行しその全てにおいて出力が変わらないことを確認できた.このことから,並列化を用いた場合も保守性が保たれる確証を得た.また最適化を行わない場合はOpenMPによる並列化を施したプログラムはコンパイル不可,コンパイル時の最適化レベルが-O3以下の場合には実行時にOpenMPの関数でエラーが確認された(コンパイルは可能).これは,OpenMPの環境設定時に発生したコンパイルエラー(ファイルの不足)を解決するために,いくつかの設定ファイルを推奨されない方法(Pathを手動で設定)で移動,改変したことが原因だと考えられる.

１０.２　今後の展望

私たちは今回の研究で得られた知見の中で最も新規性,汎用性に優れているものは三振数を用いた盤面の表現方法であると考えた.相互変換が可能であり,片方の変換だけなら必要な処理が加算のみであるという特性と,データの次元を落としながら,一列に並んだ要素の中で最

も端にある要素を判別できるという特性を活かせるような使い方を考えていきたい.

１１　参考文献

田中哲郎,"「どうぶつしょうぎ」の完全解析",
情報処理学会研究報告：ゲーム情報学(GI), vol. 22, no. 3, pp. 1-8, 2009.