Vol.2020-GI-43 No.4
2020/3/13

4人でプレイするBlokusのAIプレイヤの強化学習

大西 紘史[1]　保木 邦仁[1]

概要：4 人でプレイする Blokus は 4 人非ゼロ和完全情報ゲームであり、行動集合の大きさは 10^4 程度に達する。本研究の目的は、4 人でプレイする Blokus の強い人工知能 (AI) プレイヤを開発することである。2 人ゼロ和完全情報ゲームの AI プレイヤの学習アルゴリズムである AlphaZero を修正して、モンテカルロ木探索 (MCTS) と強化学習、深層学習を組合せた手法によって、ランダムな着手から改善していく BlokusZero を提案する。主な修正点はニューラルネットワーク (NN) の構成と、MCTS の探索木である。NN やリプレイバッファの規模を AlphaZero よりも大きく落として BlokusZero の実装を行った。BlokusZero の性能評価を行うために先行研究の BlokusAI である MCTS-MAX^N も実装し、対戦実験を行って勝率を計測した。その結果、ゲーム開始から 4 手は一様ランダム、各プレイヤの MCTS のシミュレーション回数が同じという条件の下で、BlokusZero の勝率は MCTS-MAX^N の勝率を上回った。

1. 研究背景・目的

近年、ゲーム AI プレイヤの開発はかなり発展していて、既に 2 人でプレイする種々のゲームではプロの人間プレイヤに勝つほど強い AI プレイヤが開発されている。このようなゲームでは、自己対戦による強化学習や深層学習が活用されている。例として、DeepStack [1] や AlphaZero [2]、AlphaStar [3] がある。しかし、多人数完全情報ゲームでは強化学習や深層学習を用いた AI プレイヤの開発事例はあまり報告されていなく、特に Blokus での事例は筆者らの知る限りではまだ報告されていない。

本研究の目的は、4 人でプレイする Blokus の強い AI プレイヤを開発することである。Blokus の上級者の棋譜を公開しているウェブサイトなどは存在せず、BlokusAI 開発では棋譜を用いた教師あり学習は困難である。Blokus は完全情報ゲームなので、AlphaZero のように MCTS と強化学習、深層学習を組合せた手法を用いて、ランダムな着手から AI の性能を改善していくのが適していると考えられる。Blokus の既存 AI に MCTS を行うプレイヤがある。本研究では Blokus の強化学習アルゴリズム BlokusZero を提案し、性能を既存手法と比較する。

2. Blokus

Blokus は 20×20 マスの盤と、1 マスの正方形 1〜5 個で構成された 21 種類のピースを用いてプレイされるゲームである。4 人でプレイする場合のルールを説明する。赤、青、緑、黄の 4 色を各プレイヤに 1 色ずつ割り当て、各プレイヤは割り当てられた色の 21 種類のピースを用いる。ピースの種類は図 1 参照。

4 人は行動順を決めて、盤の真上から見て時計回りに盤の隅の席に着く。そして、各プレイヤは順番が回ってくる度にそのプレイヤが所持するピースから 1 つを選び、それを盤にはめるように置く。1 巡目は 4 人ともピースの頂点を盤の隅に合わせて置く。2 巡目以降は、各プレイヤは自身が置いたピース同士の辺が隣接せずに頂点が接するようにピースを置いていく。各プレイヤは設置可能なスペースの確保や、相手への妨害等の戦略を考えながらゲームを進めていく。全員がピースを置けなくなったらゲーム終了となり、最も点数が高いプレイヤの勝ちとなる。

本研究では得点はアドバンスト・スコアというルールで計算する。使用できなかったピース 1 マス分につき 1 点減点となり、一方でピースが全部置けると 15 点加点となる。さらに +15 点のとき、最後に置いたのが 1 マスのピースだった場合はボーナスとして 5 点加点となる。図 2 のようなゲームの例では、赤色のプレイヤは −24 点、青色のプレイヤは +20 点 (最後が 1 マスピースの場合)、緑色のプレイヤは −20 点、黄色のプレイヤは −8 点となり、青色のプレイヤの勝ちとなる。

Blokus は、偶然の要素なし、プレイヤ全員のピース位置を把握可能ならびに勝者が 2 人以上出るという 3 つの特徴を持つ 4 人非ゼロ和確定完全情報ゲームである。プレイ長は最大 84 手である。ピースが 1 つも置かれていない盤へのピースの置き方は 30,433 通りと非常に多い。1 巡目での各プレイヤの着手は、盤の隅にピースを置かなければなら

[1] 電気通信大学
The University of Electro-Communications

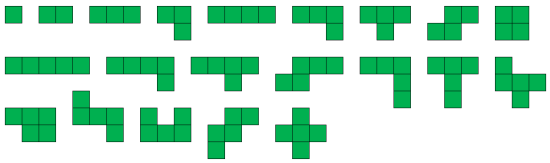
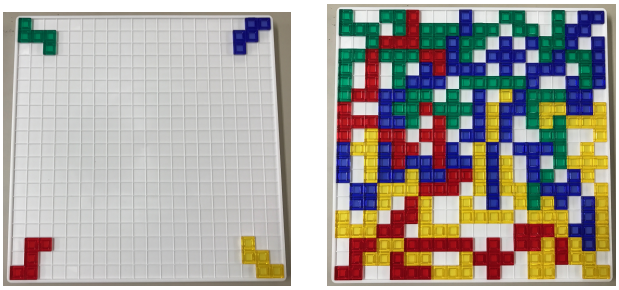


図 1: ピース 21 種類



(a) 開始から 1 巡した直後　　(b) 終了時

図 2: Blokus の盤面の例

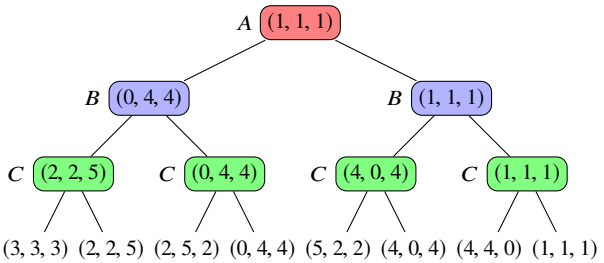


図 3: MAX$^{\mathrm{N}}$ 探索の例 (A はプレイヤ 1 の手番、B はプレイヤ 2 の手番、C はプレイヤ 3 の手番)

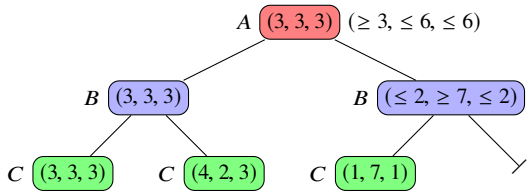


図 4: MAX$^{\mathrm{N}}$ 探索での枝刈りの例 (A はプレイヤ 1 の手番、B はプレイヤ 2 の手番、C はプレイヤ 3 の手番)

ないので合法手数は 58 個である。ゲームが進行し中盤に近づくにつれて、着手可能なピースの頂点数が増加するので合法手数も増加する傾向が見られる。ゲーム中盤を過ぎ終盤になるにつれて、空のマスが減少するので合法手数も減少する傾向が見られる。

3. 先行研究

3.1 多人数完全情報ゲームの木探索アルゴリズム

多人数完全情報ゲームの木探索アルゴリズムは既にいくつか考案されている。本節では、Blokus への適用例が報告されている、MAX$^{\mathrm{N}}$ 探索、Paranoid 探索、Best-Replay 探索 (BRS) の 3 つを説明する。これらは多人数完全情報ゲームの深さ優先探索を効率的に行う。さらに、これらに MCTS を組合せた探索アルゴリズムである MCTS-MAX$^{\mathrm{N}}$、MCTS-Paranoid、MCTS-BRS も説明する。

3.1.1 MAX$^{\mathrm{N}}$ 探索

MAX$^{\mathrm{N}}$ 探索 [4] は、多人数完全情報ゲームの木探索の基本となる探索アルゴリズムである。これは、手番プレイヤが自身の評価値が最も高い行動を選ぶという探索アルゴリズムである。n 人ゲームでの MAX$^{\mathrm{N}}$ 探索において、ある節点 x の評価値ベクトル $\boldsymbol{v}(x)$ は以下のように再帰的に定義される。

(1) x が葉節点のとき
　$v_i(x) = x$ でのプレイヤ $i = 1, \ldots, n$ の評価値
(2) x が葉節点でないとき
　x はプレイヤ k の手番、$C(x)$ を x の子節点集合として、c^* は $v_k(c^*) = \max_{c \in C(x)} v_k(c)$ を満たすとすると
　$\boldsymbol{v}(x) = \boldsymbol{v}(c^*)$

評価値ベクトルはプレイヤ n 人すべての評価値を要素に持つ n 次元ベクトルである。

図 3 は 3 人非ゼロ和ゲームのゲーム木の例で、MAX$^{\mathrm{N}}$ 探索により根節点となるプレイヤ 1 の手番では右の行動を選ぶのが最適ということになる。また、MAX$^{\mathrm{N}}$ 探索は均衡点を見つけることが知られており、評価値ベクトルが $(1, 1, 1)$ となる行動の組はこのゲームでの均衡点である。

MAX$^{\mathrm{N}}$ 探索において枝刈をする手法が考案されている [5]。これは、評価値ベクトルの要素の和に上界や下界がある場合に適用できる。評価値ベクトルの要素の和の上界が 10 で、整数評価値であるゲーム木の枝刈りをする様子を図 4 に示す。プレイヤ 1 の手番の左の行動の評価値が $(3, 3, 3)$ と分かった時点で、評価値ベクトルの和の上界が 10 であることからプレイヤ 1 の手番の評価値ベクトルの各要素の範囲は $(\geq 3, \leq 6, \leq 6)$ となる。プレイヤ 1 の右の子節点の評価値ベクトルの各要素の範囲は $(\leq 2, \geq 7, \leq 2)$ となり、この時点でプレイヤ 1 の手番で選ばれないことが分かるので探索を打ち切ることができる。

3.1.2 Paranoid 探索

Paranoid 探索 [6] では多人数ゲームを 2 人ゼロ和ゲームと仮定し、他プレイヤすべてが結託して自プレイヤ (根節点の手番プレイヤ) を損させるようにしてくると考える。よって、プレイヤすべての評価値からなる評価値ベクトルは不要で、評価値は自プレイヤ 1 人のものだけを考えればよい。すなわち、自プレイヤの手番では評価値が最大の行動を選択し、各他プレイヤの手番では評価値が最小の行動を選択する。これは実質的に Minimax 木探索となるので、効率的な枝刈が可能な $\alpha\beta$ 探索を適用できる。

図 5 は 3 人ゲームの例で、評価値は根節点の手番プレイヤ 1 のものだけを考え、プレイヤ 1 の手番では評価値が最大の行動、プレイヤ 2, 3 の手番では評価値が最小の行動を選ぶ。プレイヤ 1 の手番では左の行動の評価値が −3 とな

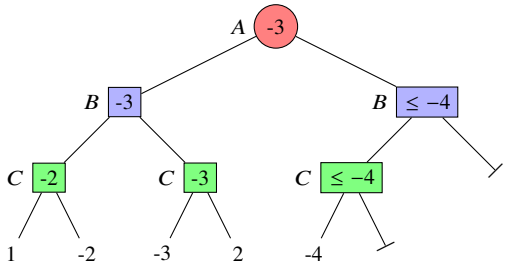


図 5: Paranoid 探索の例 (*A* はプレイヤ 1 の手番、*B* はプレイヤ 2 の手番、*C* はプレイヤ 3 の手番)

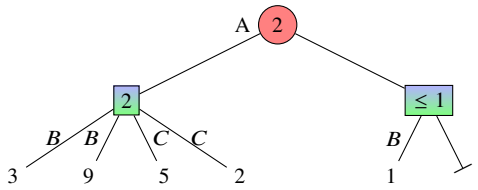


図 6: Best-Reply 探索の例 (*A* はプレイヤ 1 の手番、*B* はプレイヤ 2 の行動、*C* はプレイヤ 3 の行動)

るが、右の行動の評価値が –3 以下だと分かった時点で探索を打ち切る。

3.1.3 Best-Reply 探索

Best-Reply 探索 (BRS) [7] は、Paranoid 探索を改良したものである。他プレイヤすべての手番を深さ 1 つ分にまとめ、他プレイヤの手番では自プレイヤを最も損させられる 1 人の行動を選ぶ。Paranoid 探索で深さ d のゲーム木は深さ $\lceil \frac{2d}{n} \rceil$ となり、Paranoid 探索よりも深くにある自プレイヤの手番を訪問できる。

図 6 は 3 人ゲームの例で、相手であるプレイヤ 2, 3 の手番がまとめられ、プレイヤ 1 の手番の左の子節点では評価値が最小の 2 であるプレイヤ 3 の行動を選ぶ。プレイヤ 1 の右の行動が 2 以下と分かった時点で枝刈りをする。

3.1.4 さらにモンテカルロ法を組合せる手法

モンテカルロ木探索 (MCTS) [8] は、乱数を用いたシミュレーションを利用して探索木を大きくしていく探索アルゴリズムである。木の各節点では、手番プレイヤの番号とプレイヤ 1 から見た報酬の合計、過去の訪問回数が記録される。図 7 の例では、選択、展開、ロールアウト、逆伝播という 4 つのプロセスが繰り返される。アルゴリズム開始時は、探索木は根節点のみである。何らかの手法で子節点を選択してその節点を辿るということを葉節点に辿り着くまで繰り返す。辿り着いた葉節点が終端節点である場合、利得を算出する。一方で、辿り着いた葉節点が未展開の内部節点の場合、その節点の過去の訪問回数が閾値を超えていたら展開する。そこから子節点を選択して、それを探索木に加えて辿る。展開の閾値は基本的に 1 である。ロールアウトを併用する場合、選択された葉節点から何らかの方策でゲーム終了までロールアウトを行って利得を算出する。この利得を報酬と考える。基本的に利得は勝ちなら +1、負けなら 0 のように勝敗を表すものにする。ロールアウトを併用しない場合は、何らかの手法で評価値を算出し、これを報酬と考える。報酬を算出した後は辿ってきた節点を逆に辿っていき、各節点の報酬総和や訪問回数などの統計情報を更新する。

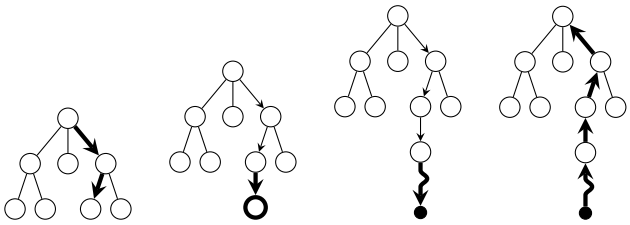


図 7: ロールアウトを併用する MCTS の概要

子節点の選択において、探査と知識利用のバランスを上手く取ることが重要である。このバランスを取るために UCB1 アルゴリズムがしばしば用いられる [9]。手番プレイヤ P の節点 parent において、以下のように子節点を選ぶ。

$$c^* = \arg\max_i \left(\phi_{P=\text{プレイヤ } 1}\left(\frac{w_i}{n_i}\right) + C\sqrt{\frac{\ln n_{\text{parent}}}{n_i}} \right) \quad (3.1)$$

右辺の第 1 項が知識利用、第 2 項が探査に当たる。w_i は子節点 i のプレイヤ 1 の報酬の合計、n_i は i の過去の訪問回数、n_{parent} は parent の過去の訪問回数、C は探査と知識利用のバランスを取るための定数である。$\phi_{\text{condition}}$ は、条件 condition が真のときは恒等写像である。偽のときは、例えば報酬をプレイヤ 1 が勝ちなら 1、負けなら 0 とするならば、$\phi_{P=\text{プレイヤ } 1}(x) = 1 - x$ である。

多人数完全情報ゲームにも、MCTS を適用することは可能である。大きくしていく探索木を MAX$^{\text{N}}$、Paranoid、BRS のゲーム木の構造と同じにすればよい。このようなことを行うのが MCTS-MAX$^{\text{N}}$、MCTS-Paranoid、MCTS-BRS の 3 つの探索アルゴリズムである [10]。ただし、各節点に記録される統計情報が 2 人完全情報ゲームでの MCTS とは異なり、UCB1 アルゴリズムにも変更が必要となる。

MCTS-MAX$^{\text{N}}$ では、プレイヤ 1 だけではなくプレイヤすべての報酬の合計を各節点に記録する。プレイヤ P の手番では、以下のように子節点を選ぶ。

$$c^* = \arg\max_i \left(\frac{w_i^P}{n_i} + C\sqrt{\frac{\ln n_{\text{parent}}}{n_i}} \right) \quad (3.2)$$

w_i^P は子節点 i におけるプレイヤ P の報酬の合計である。また、MCTS-Paranoid と MCTS-BRS では、プレイヤ 1 ではなく根節点の手番プレイヤの報酬の合計を各節点に記録する。プレイヤ P の手番では、以下のように子節点を選ぶ。

$$c^* = \arg\max_i \left(\phi_{P=\text{根節点の手番プレイヤ}}\left(\frac{w_i'}{n_i}\right) + C\sqrt{\frac{\ln n_{\text{parent}}}{n_i}} \right) \quad (3.3)$$

w_i' は子節点 i における根節点の手番プレイヤの報酬の合計

である。

多人数完全情報ゲームにおける UCB1 アルゴリズムでは、Progressive History [11] という改良手法が考案されている。例としてこの手法を MCTS-MAX^N に用いた場合、プレイヤ P の手番では子節点の選択を以下のようにする。

$$c^* = \arg\max_i \left(\frac{w_i^P}{n_i} + C\sqrt{\frac{\ln n_{\text{parent}}}{n_i}} + \frac{s_a}{n_a}\frac{W}{n_i - w_i^P + 1} \right) \tag{3.4}$$

a は子節点 i に対応する行動である。n_a は過去すべてのゲームプレイにおいて行動 a を用いた回数、s_a は過去すべてのゲームプレイにおいて a を用いたときの報酬の合計、W は Progressive History の効力を決める定数である。

3.1.5 性能の比較

以上の計 6 種類の探索アルゴリズムを様々な組合せでプレイヤに割当て、いくつかの多人数完全情報ゲームにおいて対戦実験を行い、それぞれの勝率を求めるという実験結果が報告された [10]。4 人でプレイする Blokus の実験結果について紹介する。

MAX^N、Paranoid、BRS では、あるプレイヤの評価値をそのプレイヤが今までに盤に置いたピースのマス数とした。さらにサイズの大きいピースに対応する子節点を優先的に探索するために、子節点を選ぶ前に Move Ordering を行った。MAX^N、Paranoid、BRS の中では、BRS が最も勝率が高く、次に Paranoid が高かった。また、思考時間を長くするほど、BRS の勝率が高いことが示された。

MCTS-MAX^N、MCTS-Paranoid、MCTS-BRS では、ロールアウトにおいて ϵ-greedy 法を用い、確率 $1-\epsilon$ で使用可能なピースの中でサイズが最大のものをランダムに選んだ。$\epsilon = 0.05$ とし、UCB1 アルゴリズムと Progressive History のパラメータはそれぞれ $C = 0.2, W = 5$ である。MCTS-MAX^N、MCTS-Paranoid、MCTS-BRS の勝率はほぼ互角であった。ただし、思考時間が短いと MCTS-BRS の勝率が若干低くなった。

BRS と MCTS-MAX^N では、思考時間をかけるほど MCTS-MAX^N の勝率が高くなった。

3.2 AlphaZero

AlphaZero は、プロプレイヤの棋譜を用いた教師あり学習は一切行わずに、ランダムプレイヤを開始点とする強化学習で 2 人ゼロ和完全情報ゲームの AI プレイヤを生成する学習アルゴリズムである。2017 年に D. Silver らは、このアルゴリズムを囲碁、将棋、チェスに適用し、それぞれの既存の AI プレイヤの性能を上回ったと報告した。

AlphaZero では、ゲーム木を状態・行動空間で表現する。ここで、状態は駒の配置などで表現される意思決定点、行動は合法手に対応する。終端状態では、ゲームプレイの結果に対応する報酬 v' が与えられる。勝ちなら $v' = 1$、負けなら $v' = 0$、引き分けなら $v' = 0.5$ とする。

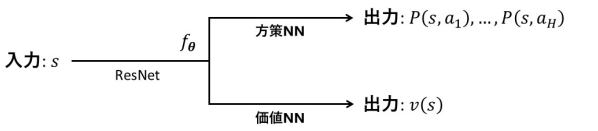


図 8: AlphaZero で用いる CNN 概略図

3.2.1 CNN の構成

AlphaZero では、方策と状態価値の推定を 1 つの畳み込みニューラルネットワーク (CNN) で行う。状態 s を入力し、途中で分岐して行動の確率分布 P と価値 v を推定し出力する CNN, f_θ, を構成する (図 8 参照)。

$$(P, v) = f_\theta(s) \tag{3.5}$$

入力から分岐の間では Residual Network (ResNet) [12] が用いられる。分岐から出力 $P(s,\cdot)$ までを方策ニューラルネットワーク (NN) 、出力 $v(s)$ までを価値 NN と呼ぶ。方策 NN の出力値の個数 H はゲームによって異なる。

CNN に入力する状態の特徴はゲームによって異なるが、状態 s とその過去 1〜7 手前の駒の配置列は共通である。方策 NN の出力にはソフトマックス関数、価値 NN の出力には tanh 関数を作用させているので、$0 < P(s,\cdot) < 1$ かつ $-1 < v(s) < 1$ が成り立つ。価値 $v(s)$ は、状態 s での手番プレイヤのものとする。畳み込み層の直後には Batch Normalization (BN) [13] を用いる。CNN の構成の詳細は以下のようである。

1. ResNet
- 入力
- 第 1 層: 畳み込み層 (フィルタ数 256、サイズ 3×3)、BN、ReLU
- 第 2〜39 層: 残差ブロック 19 個 (畳み込み層のフィルタ数 256、サイズ 3×3)
  分岐 (値を複製)
2. 方策 NN
- 第 40 層: 畳み込み層、BN、ReLU
- 第 41 層: 畳み込み層、ソフトマックスユニット
3. 価値 NN
- 第 40 層: 畳み込み層 (フィルタ数 1、サイズ 1×1)、BN、ReLU
- 第 41 層: 全結合層 (ユニット数 256)、ReLU
- 第 42 層: 全結合層 (ユニット数 1)、tanh ユニット

3.2.2 学習アルゴリズム

自己対戦とミニバッチ学習をそれぞれ繰り返し行うことで、CNN の重み $\boldsymbol{\theta}$ を更新していく。まず、自己対戦は CNN を用いて MCTS を行うプレイヤ同士で行われる。各プレイヤは意思決定点において、その状態を根節点として MCTS を行うことによって行動を決定する。このとき、各状態・行動対 (s,a) の過去の訪問回数 $N(s,a)$、報酬の合計 $W(s,a)$、報酬の平均 $Q(s,a)$、方策 NN の出力 $P(s,a)$、手番プレイヤの番号が記録される。手番プレイヤの番号以外の

初期値は全て 0 である。

AlphaZero の学習アルゴリズムにおける MCTS のシミュレーションを説明する。状態 s での行動は、変形した PUCT アルゴリズム [14] を用いて、以下のように選ぶ。

$$a^* = \arg\max_{a \in \mathcal{A}(s)} (Q(s,a) + U(s,a)) \tag{3.6}$$

$$U(s,a) = C(s)P(s,a)\frac{\sqrt{N(s)}}{1+N(s,a)} \tag{3.7}$$

$$C(s) = \log\frac{1+N(s)+c_{\text{base}}}{c_{\text{base}}} + c_{\text{init}} \tag{3.8}$$

$N(s)$ は状態 s の過去の訪問回数である。$C(s)$ は $N(s)$ が増えるほど大きくなる関数で、c_{base}, c_{init} は定数である。行動を選ぶ前に $\mathcal{A}(s)$ 上で $P(s,a)$ を再正規化しておく。また、多様なゲームプレイを生成できるように、根節点に対応する状態でのみディリクレノイズを用いる。その状態では $P(s,a)$ を再正規化した後、さらに以下のように $P(s,a)$ を更新しておく。

$$P(s,a) \leftarrow (1-\varepsilon)P(s,a) + \varepsilon \text{Dir}(\alpha, s, a) \tag{3.9}$$

ε, α は定数である。α の適切な値はゲームによって異なり、チェス、将棋、囲碁ではそれぞれ 0.3, 0.15, 0.03 としている。ノイズ $\text{Dir}(\alpha, s, a)$ は、次のようにガンマ分布を用いて求める。まず、$\beta = 1$ のガンマ分布の確率密度関数

$$f(x;\alpha) = \frac{x^{\alpha-1}e^{-x}}{\Gamma(\alpha)} \tag{3.10}$$

に従って、$|\mathcal{A}(s)|$ 個の値 $y_1, \ldots, y_{|\mathcal{A}(s)|}$ を生起させる。ただし、$\Gamma(\alpha)$ はガンマ関数である。そして、$\text{Dir}(\alpha, s, a)$ は以下のように計算される。

$$\text{Dir}(\alpha, s, a) = \frac{y_a}{\sum_{i=1}^{|\mathcal{A}(s)|} y_i} \tag{3.11}$$

終端状態に辿り着いたら、前述のように報酬 v' を算出する。一方で、内部節点の葉節点に対応する状態に辿り着いたときは価値 NN で評価し、その出力 $v(s)$ を値域が $(0,1)$ になるように正規化した値を MCTS のシミュレーションの報酬 v' とする。v' を算出した後は、辿ってきた各状態 s の記録を以下のように更新する。

$$N(s,a) \leftarrow N(s,a) + 1 \tag{3.12}$$

$$W(s,a) \leftarrow W(s,a) + \Phi(v') \tag{3.13}$$

$$Q(s,a) \leftarrow \frac{W(s,a)}{N(s,a)} \tag{3.14}$$

Φ は状態を 1 つ辿るたびに変化する関数である。奇数回辿ったときには $\Phi(v') = 1 - v'$ となり、偶数回辿ったときには恒等写像となる。以上のシミュレーションを N_{sim} 回行う。着手は、ゲーム開始から M 手までは根節点に対応する状態 s での $N(s,a)$ に比例した確率で行い、それ以降では $N(s,a)$ が最も多い行動とする。

自己対戦の結果から学習データの組を採取する手法を説明する。プレイ長 K 手のゲーム記録から組 $(s^k, \boldsymbol{\pi}^k, z^k)$ $(k = 1, \ldots, K)$ を採取し、リプレイバッファに登録する。s^k は k 手目の意思決定点に対応する状態、$\boldsymbol{\pi}^k$ は s^k における各合法手のシミュレーションの訪問回数から計算される確率分布、z^k はプレイの勝敗を表す値である。$\boldsymbol{\pi}^k$ は以下のような計算で求める。

$$\pi_a^k = \frac{N(s^k, a)}{\sum_b N(s^k, b)} \tag{3.15}$$

z^k は s^k での手番プレイヤが勝ちなら +1、負けなら −1、引き分けなら 0 とする。$\boldsymbol{\pi}^k, z^k$ はそれぞれ出力 $P(s^k, \cdot), v(s^k)$ の教師信号である。TPU を用いて N_{act} 個の自己対戦を同時に行い、ゲーム記録をリプレイバッファに繰り返し登録する。リプレイバッファのサイズは少なくとも 500,000 プレイ分である。

CNN の学習は、リプレイバッファから組 $(s, \boldsymbol{\pi}, z)$ をランダムに N_{mini} 個採取してミニバッチ 1 つを構成し、ミニバッチ付き確率的勾配降下法によって $\boldsymbol{\theta}$ を更新する。誤差関数は以下である。

$$E = (z-v)^2 - \boldsymbol{\pi}^\top \log P + c\|\boldsymbol{\theta}\|^2 \tag{3.16}$$

c は L2 正規化 (重み減衰) のパラメータである。$\boldsymbol{\theta}$ の更新 1 回を 1 ステップとし、1,000 ステップごとに自己対戦で用いるネットワークの重みを最新にする。700,000 ステップで学習完了となる。

4. 提案手法

4 人でプレイする Blokus の AI を、MCTS と強化学習、深層学習を組合せることで開発する。そこで、AlphaZero の学習アルゴリズムを 4 人非ゼロ和完全情報ゲームである Blokus 用に修正した BlokusZero を提案する。

4.1 CNN の構成

AlphaZero では囲碁と将棋、チェスでの CNN の構成を示していた。Blokus で CNN を用いるために、AlphaZero の CNN の入力と出力の構成を修正する。状態 s を入力として方策と価値を推定し出力する CNN の入力は、表 1 のような構成とする。ネットワークの入力は 20×20 次元 33 チャネルとし、状態 s とその過去 1〜7 手前の各プレイヤのピースの配置と盤上にあるピースの個数を入力する。第 1〜32 チャネルでは 20×20 の格子を Blokus の盤と見立てて、ピースで埋まっているマスを 1、空のマスを 0 とする。第 33 チャネルでは、以下のように正規化した値 d で 20×20 の格子を全て埋める。

$$d = \frac{\text{盤上にあるピースの個数}}{84} \tag{4.1}$$

Blokus の盤へのピースの置き方は 30,433 通りのため、方策 NN の出力値の個数は $H = 30,433$ にする。また、価値

表 1: CNN の入力

チャネル	内容
第 1	状態 s の手番プレイヤのピース配置
第 2	状態 s の次の手番プレイヤのピース配置
第 3	状態 s の 2 番後の手番プレイヤのピース配置
第 4	状態 s の 3 番後の手番プレイヤのピース配置
⋮	⋮
第 29	状態 s の 7 手前の手番プレイヤのピース配置
第 30	状態 s の 7 手前の次の手番プレイヤのピース配置
第 31	状態 s の 7 手前の 2 番後の手番プレイヤのピース配置
第 32	状態 s の 7 手前の 3 番後の手番プレイヤのピース配置
第 33	盤上にあるピースの個数

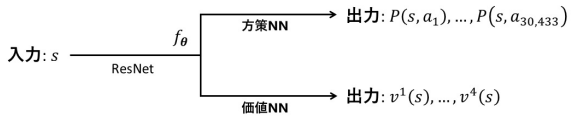


図 9: 本研究で用いる CNN 概略図

NN の出力は 4 次元ベクトル $\boldsymbol{v}(s)$ とし、手番プレイヤ p の価値を $v^p(s)$ とする。本研究で用いる CNN の概略図を図 9 に示す。

3 ヶ月程度で完了できる実験のセットアップを念頭に置き、CNN の構成は以下とする。

- 入力: $20\times20\times33$
- 第 1 層: 畳み込み層 (フィルタ数 128、サイズ 3×3)、BN、ReLU
- 第 2〜21 層: 残差ブロック 10 個 (畳み込み層のフィルタ数 128、サイズ 3×3)
  分岐 (値を複製)

1. 方策 NN
- 第 22 層: 畳み込み層 (フィルタ数 2、サイズ 1×1)、BN、ReLU
- 第 23 層: 全結合層 (ユニット数 30,433)、ソフトマックス関数

2. 価値 NN
- 第 22 層: 畳み込み層 (フィルタ数 4、サイズ 1×1)、BN、ReLU
- 第 23 層: 全結合層 (ユニット数 256)、ReLU
- 第 24 層: 全結合層 (ユニット数 4)、tanh 関数

方策 NN の出力となるソフトマックス関数の手前では畳み込み層を用いず、全結合層を用いる。重みの初期化アルゴリズムは方策 NN の第 23 層と価値 NN の第 24 層を xavier、それ以外を msra とする。重みの数は ResNet が $(3^2\times33\times128+128)+(3^2\times128^2+128)\times10=1,513,984$、方策 NN が $400\times2\times30,433+30,433=24,376,833$、価値 NN が $(400\times4\times256+256)+(256\times4+4)=410,884$ である。

4.2 学習アルゴリズム

AlphaZero の MCTS アルゴリズムは次のように修正する。BlokusZero は $\mathrm{MAX}^{\mathrm{N}}$ の探索木を用いる。Blokus は 4 人非ゼロ和ゲームであるので、各状態にはプレイヤ 4 人それぞれの $W(s,a)$ と $Q(s,a)$ を記録する。以降、W と Q の右上に p を付して該当するプレイヤを示す。MCTS のシミュレーションでは PUCT アルゴリズムを修正し、手番プレイヤ p の状態 s での行動の選択を以下のように行う。

$$a^* = \arg\max_{a\in\mathcal{A}(s)}\left(Q^p(s,a)+U(s,a)\right) \tag{4.2}$$

内部節点の葉節点に対応する状態に辿り着いたときには、4 次元ベクトルの出力 $\boldsymbol{v}(s)$ を 4 人の MCTS のシミュレーションの報酬とする。本研究では報酬を勝ちなら $v=1$、負けなら $v=-1$ とし、正規化は行わない。逆伝播は以下のようになる。

$$W^p(s,a) \leftarrow W^p(s,a)+v^p \quad (p=1,2,3,4) \tag{4.3}$$

$$Q^p(s,a) \leftarrow \frac{W^p(s,a)}{N(s,a)} \quad (p=1,2,3,4) \tag{4.4}$$

価値 NN の出力 $\boldsymbol{v}(s)$ に合わせて、報酬の教師データ z も 4 次元ベクトル $\boldsymbol{z}$ とする。誤差関数は以下のようになる。

$$E=\|\boldsymbol{z}-\boldsymbol{v}\|^2-\boldsymbol{\pi}^\top\log P(s,\cdot)+c\|\boldsymbol{\theta}\|^2 \tag{4.5}$$

価値 NN の誤差関数 (式 (3.15)、(4.5) の第 1 項) の最大値は AlphaZero では 4、BlokusZero では 16 であり、4 倍になっている。よって、方策 NN よりも価値 NN を優先的に学習してしまう可能性がある。$\|\boldsymbol{z}-\boldsymbol{v}\|^2$ に 1/2 や 1/4 を掛けることも考えたが、推考する時間に限りがあったため、本研究ではこのように最も単純な修正を行った。

5. 実験

BlokusZero を実装し、その性能を評価する。そのために Blokus の先行研究のアルゴリズムも実装し、BlokusZero と対戦実験を行う。本章では学習と対戦実験の設定と、結果について説明する。

5.1 設定

5.1.1 学習アルゴリズムの設定

PUCT アルゴリズムの $C(s)$ のパラメータは AlphaZero と同様に $c_{\mathrm{base}}=19,625$, $c_{\mathrm{init}}=1.25$ とする。ディリクレノイズのパラメータをそれぞれ $\varepsilon=0.25$, $\alpha=0.02$ とする。ε は AlphaZero と同じ値だが、Blokus は合法手数が多いことを考慮して α は AlphaZero における囲碁より小さい値にする。$N(s,a)$ に比例した確率で着手を行うのは、ゲーム開始から $M=20$ 手までとする。Blokus のゲームプレイ長は将棋や囲碁に比べると短いので、AlphaZero の $M=30$ 手より小さくする。MCTS のシミュレーションは AlphaZero と同様に $N_{\mathrm{sim}}=800$ 回行う。

CNN の規模に加えてリプレイバッファのサイズも AlphaZero から小さくし、10,000 プレイ分とする。リプレイバッファの初期化ではサイズ上限に達するまで自己対戦が繰り返し行われる。自己対戦では GPU (NVIDIA 社の GeForce GTX 1080 Ti) を 8 個用いて、$N_{\text{act}} = 8$ 個の対戦を同時に行う。自己対戦の終了後、リプレイバッファから $(s, \boldsymbol{\pi}, z)$ の組をランダムに $N_{\text{mini}} = 64$ 個取り出してミニバッチ学習を行う。AlphaZero では $N_{\text{mini}} = 4,096$ だが、これも AlphaZero より小さくする。学習には重み減衰とモメンタムを用い、それぞれのパラメータは AlphaZero と同様に $\lambda = 0.0001$, $\mu = 0.9$ とする。学習を 1,000 回行ったら、リプレイバッファから古い 1,000 プレイ分を捨て、新しい重みを用いて自己対戦を行って 1,000 プレイ分を溜める。このように自己対戦と学習を繰り返し、ネットワークの重みを更新していく。学習率は初期は 0.001 で、40,001 回目の学習からは 0.0001 とする。

5.1.2 対戦実験の設定

BlokusZero と既存の木探索アルゴリズムを行うプレイヤを対戦させて、BlokusZero の性能評価を行う。本研究では 6 種類の木探索アルゴリズムのうち、最も基本的で強い MCTS-MAX$^{\text{N}}$ を用いる。BlokusZero と MCTS-MAX$^{\text{N}}$ を行うプレイヤを複数回対戦させ、それぞれの勝率を計測する。プレイヤ 4 人のうち 1 人を BlokusZero、残り 3 人を MCTS-MAX$^{\text{N}}$ とする。ゲーム開始前に決めるプレイヤの行動順はランダムとする。ゲーム開始から 4 手は、各プレイヤは合法手の中から一様ランダムに着手する。MCTS-MAX$^{\text{N}}$ には先行研究 [10] と同様に Progressive History [11] を適用し、ロールアウトでは ϵ-greedy 法を適用する。パラメータも同様に $\epsilon = 0.05$ とし、UCB1 アルゴリズムと Progressive History のパラメータもそれぞれ $C = 0.2, W = 5$ とする。BlokusZero はディリクレノイズを用いず、最も $N(s, a)$ が大きい行動を着手する。

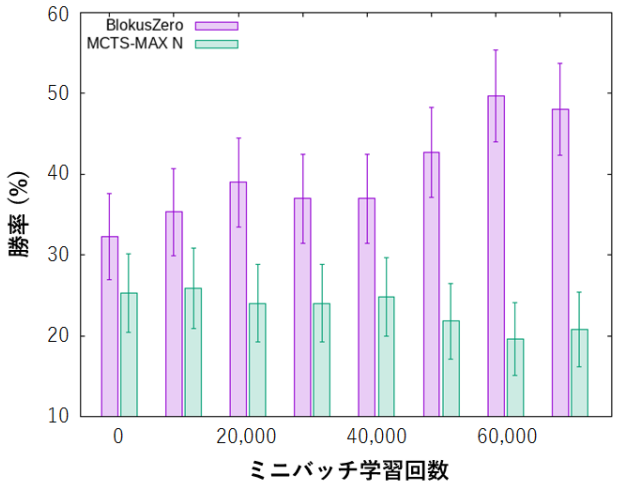


図 10: BlokusZero (シミュレーション回数 800/着手) と MCTS-MAX$^{\text{N}}$ (ロールアウト回数 800/着手) の勝率

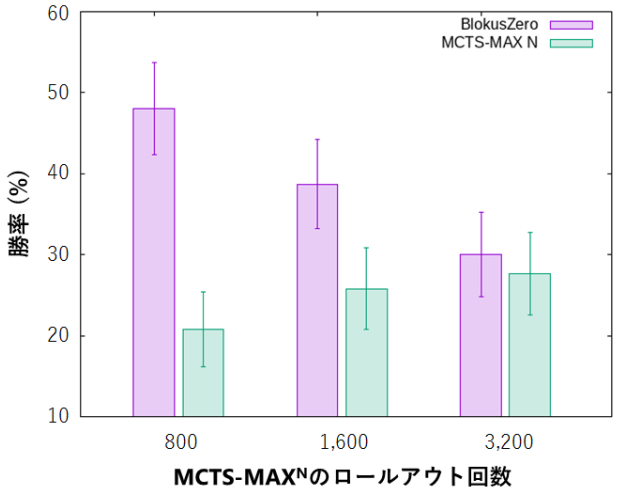


図 11: BlokusZero (シミュレーション回数 800/着手) と MCTS-MAX$^{\text{N}}$ の勝率

5.2 結果

BlokusZero の対戦実験の結果を図 10 に示す。対戦回数は 300 とした。図中のエラーバーは各勝率の 95%信頼区間である。ミニバッチ学習 20,000 回目で BlokusZero は MCTS-MAX$^{\text{N}}$ を有意に勝ち越した。50,000 回目からは BlokusZero の勝率がさらに大きく上がり、60,000 回目では MCTS-MAX$^{\text{N}}$ を大きく上回った。学習率を下げた効果が出たと考えられる。よって、ゲーム開始から 4 手は一様ランダム、BlokusZero と MCTS-MAX$^{\text{N}}$ のシミュレーション回数が同じという条件下で、BlokusZero は MCTS-MAX$^{\text{N}}$ に有意に勝ち越した。

さらに、MCTS-MAX$^{\text{N}}$ のロールアウト回数を 800, 1600, 3200 と変化させて BlokusZero の勝率を計測した (図 11 参照)。対戦回数は 300、ミニバッチ学習の更新回数は 70,000 とした。MCTS-MAX$^{\text{N}}$ のロールアウト回数が BlokusZero のシミュレーション回数の 2 倍のときまでは、BlokusZero の勝率が MCTS-MAX$^{\text{N}}$ を有意に上回った。しかし、4 倍のときには有意な差が見られなかった。

図 12 に、30 ゲーム分で平均をとった方策 NN の出力の性質を示す。ミニバッチ学習の更新回数は 70,000、BlokusZero のシミュレーション回数は 800、MCTS-MAX$^{\text{N}}$ のロールアウト回数は 800 とした。合法手数は、盤上にピースが 16 個程度あるときに最も多くなる傾向がある。このようなときには方策 NN は候補手を上位 100 手にも絞れていない様子が見てとれる。これが、Blokus は有効な手が多いということを意味しているのか、方策 NN の表現力が低いということを意味しているのか、どちらなのかを判断するのは現状では困難である。さらなる調査が望まれる。

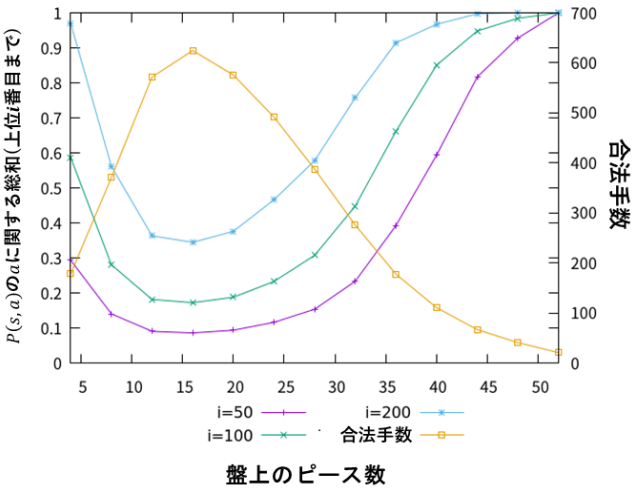


図 12: 方策 NN の出力統計情報

6. まとめ

本研究では、MCTS と強化学習、深層学習を組合せた手法によって、4 人でプレイする Blokus の AI プレイヤを開発した。AlphaZero の学習アルゴリズムを 4 人完全情報ゲームである Blokus に修正した BlokusZero を提案し、CNN の入出力や MCTS に変更を加えた。AlphaZero と比較して CNN やリプレイバッファの規模は大きく落として BlokusZero を実装した。また、BlokusZero の性能評価を行うために MCTS-MAX^N も実装し、対戦実験を行って勝率を計測した。その結果、BlokusZero の勝率は MCTS-MAX^N の勝率を上回った。よって、Blokus において、MCTS を行うプレイヤの性能を改善するという目標は達成できたと考えられる。

BlokusZero の改善案として、AlphaGo Zero [15] のように MCTS において CNN で状態を評価する際に、盤面を回転や反転をさせて状態価値の推定精度を高める方法が考えられる。

謝辞 本研究は JSPS 科研費 JP16K00503, JP18H03347 の助成を受けたものです。

参考文献

[1] M. Moravčík, M. Schmid, N. Burch, V. Lisý, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, and M. Bowling, "DeepStack: Expert-level artificial intelligence in heads-up no-limit poker," *Science*, 2017, Vol. 356, No. 6337, pp. 508-513.

[2] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play," *Science*, 2018, Vol. 362, No. 6419, pp. 1140-1144.

[3] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver, "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature*, 2019, Vol. 575, pp. 350-354.

[4] C. A. Luckhardt and K. B. Irani, "An Algorithmic Solution of N-Person Games," In *Proceedings of The 5th National Conference on Artificial Intelligence (AAAI)*, 1986, Vol. 86, pp. 158-162.

[5] R. E. Korf, "Multi-Player Alpha-Beta Pruning," *Artificial Intelligence*, 1991, Vol. 48, No. 1, pp. 99-111.

[6] N. R. Sturtevant and R. E. Korf, "On Pruning Techniques for Multi-Player Games," In *Proceedings of The 17th National Conference on Artificial Intelligence (AAAI)*, 2000, Vol. 49, pp. 201-207.

[7] M. P. D. Schadd and M. H. M. Winands, "Best-Reply Search for Multi-Player Games," *IEEE Transactions on Computational Intelligence and AI in Games*, 2011, Vol. 3, No. 1, pp. 57-66.

[8] R. Coulom, "Efficient selectivity and backup operators in Monte-Carlo Tree Search," In *Proceedings of International Conference on Computers and Games*, 2007, Vol. 4630 of LNCS, pp. 72-83.

[9] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," *European Conference on Machine Learning*, 2006, Vol. 4212 of LNAI, pp. 282-293.

[10] J. (Pim) A. M. Nijssen and M. H. M. Winands, "An Overview of Search Techniques in Multi-Player Games," *Computer Games Workshop at ECAI*, 2012, pp. 50-61.

[11] J. A. M. Nijssen and M. H. M. Winands, "Enhancements for Multi-Player Monte-Carlo Tree Search," In H. J. van den Herik, H. Iida, and A. Plaat, editors, In *Proceedings of International Conference on Computers and Games*, 2011, Vol. 6515 of LNCS, pp. 238-249.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," In *Proceedings of The 29th IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770-778.

[13] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by ReducingInternal Covariate Shift," In *Proceedings of The 32nd International Conference on Ma-chine Learning*, 2015, pp. 448-456.

[14] C. D. Robin, "Multi-armed bandits with episode context," *Annals of Mathematics and Artificial Intelligence*, 2011, Vol. 61, No. 3, pp. 203-230.

[15] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, 2017, Vol. 550, pp. 354-359.