令和 5 年度

修士学位論文

ゲーム「2048」への Monte-Carlo Softmax 探索の適用

An Application of Monte-Carlo Softmax Search to the 2048 Game

1265112　渡邊 翔太

指導教員　松崎 公紀

2024 年 2 月 28 日

高知工科大学大学院 工学研究科 基盤工学専攻

情報学コース

要 旨

ゲーム「2048」への Monte-Carlo Softmax 探索の適用

渡邊 翔太

近年，AlphaZero の登場により，モンテカルロ木探索（MCTS）とニューラルネットワーク（評価関数）を併用した強化学習が注目されている．AlphaZero の手法は様々なゲームに適用可能な反面，膨大な計算資源が必要であることは大きな問題点である．類似の木探索と評価関数を併用する強化学習手法として，評価関数にポリシー関数を必要としない Monte-Carlo Softmax 探索が提案されている．本研究では，既存の畳み込みニューラルネットワークベースの評価関数と Monte-Carlo Softmax 探索を用いて，確率的一人ゲーム 2048 のコンピュータプレイヤを作成した．また，探索した結果を評価関数の学習データとする，強化学習の実装方法についても検討する．

まず，Monte-Carlo Softmax 探索のアルゴリズムを確率的ゲームに適応させるために，探索の方法を制御する 6 つの規則を設計し，これを選択とバックアップの 2 つに適用した 36 種類のプレイヤを網羅的に評価するための比較実験を行った．結果，最も性能の高かったプレイヤは，1 手あたり 2000 シミュレーションという制限の下で，平均スコア 533 542 を達成した．選択においては非決定的に，バックアップにおいては決定的に評価値を決める規則が効果的であった．

また，強化学習を実装するための実験も行った．2048 のコンピュータプレイヤに関する先行研究によると，TD 学習ベースの強化学習により評価関数を学習させる手法が有効である．本研究では，学習精度を高めるため，木探索を行った結果を用いた強化学習の実現を目指した．しかしながら，単純にモンテカルロ木探索の結果を用いて学習を行うと，探索中の過大評価により評価値が発散するという問題が起こる．本研究では，Q 学習における過大評価の問題を克服する Double Q-learning を参考に，2 つのパラメータセットを用いる学習手

法を設計し，8 通りのハイパーパラメータを持つプレイヤを用いて性能を検証した．結果は，いずれも評価値が安定せず，既存の TD 学習の派生アルゴリズムによる強化学習を越える結果は得られなかった．

キーワード　2048，確率的一人ゲーム，モンテカルロ木探索，Monte-Carlo Softmax 探索 (MCSS)，強化学習，過大評価，Double Q-learning

Abstract

An Application of Monte-Carlo Softmax Search to the 2048 Game

WATANABE, Shota

Recently, the combination of Monte Carlo tree search (MCTS) and neural networks, particularly in the context of evaluation functions, has garnered significant attention following the emergence of AlphaZero. Despite the versatility of AlphaZero's approach across various games, a notable drawback is its substantial computational resource requirement. As an alternative, the Monte-Carlo Softmax search has been proposed, integrating tree search and an evaluation function without the need for a policy function. In this study, we implemented a computer player for the stochastic one-player game 2048, utilizing an established convolutional neural network-based evaluation function and Monte-Carlo Softmax Search. Additionally, we explored the incorporation of reinforcement learning by using the search outcomes as training data for the evaluation function. To adapt the Monte-Carlo Softmax Search algorithm to the stochastic game, six rules governing the search method were devised. Exhaustive experiments were conducted, evaluating 36 different players with variations in the application of these rules to selection and backup. The results indicated that the most successful player achieved an average score of 533 542 under a limit of 2000 simulations per move. Notably, rules involving non-deterministic selection and deterministic backup proved to be effective. Further experiments focused on implementing reinforcement learning. Previous studies on computer players for 2048 have demonstrated the efficacy of TD-based reinforcement

learning in learning the evaluation function. In this study, we aimed to enhance learning accuracy by employing the results of tree search for reinforcement learning. However, a challenge emerged during the learning process using Monte Carlo tree search results: the evaluation values diverged due to overestimation during the search. To address this, we designed a learning method based on Double Q-learning with two sets of parameters, overcoming the overestimation issue in Q-learning. The performance was tested across players with eight different hyperparameters. Unfortunately, the results indicated that none of the evaluation values remained stable, and no improvements were observed beyond reinforcement learning using a derivative algorithm of existing TD learning.

key words 2048, stochastic one-player game, Monte-Carlo Tree Search, Monte-Carlo Softmax Search, Reinforcement Learning, overestimation, Double Q-learning

目次

目次

図目次

表目次

第 1 章

はじめに

本研究では，G.Cirulli が 2014 年に公開した確率的一人ゲーム 2048 を対象とする．近年，AlphaZero の登場により，モンテカルロ木探索（MCTS）とニューラルネットワーク（評価関数）を併用した強化学習が注目されている．AlphaZero の手法は様々なゲームに適用可能な反面，膨大な計算資源が必要であることは大きな問題点である．Antonoglou ら [6] は，AlphaZero を拡張した stochastic MuZero 手法を 2048 に適用し，平均得点約 550 000 を達成している．しかし，その手法は 10^{10} 局面のスケールでニューラルネットワークを学習される必要がある．1 回の学習に著者らの先行研究 [2] と同程度の時間がかかるとすると，それには 150 日の学習が必要であり，これは現実的とは言えない．このことから，より軽量な強化学習手法が有効であることを示すことが必要だと著者らは考える．類似の木探索と評価関数を併用する強化学習手法として，評価関数にポリシー関数を必要としない Monte-Carlo Softmax 探索 (MCSS) [1] が提案されている．本研究では，MCSS と強化学習を併用した軽量なアルゴリズムの設計を目指す.

AlphaZero のアルゴリズムは，バリュー（値）関数とポリシー（方策）関数の 2 つの評価関数を必要とする．一方，2048 においては，バリュー関数のみの学習であっても，TD 学習の派生アルゴリズムにより強いプレイヤができることが先行研究 [2] で示されている．MCSS に基づく強化学習手法 [1] は，AlphaZero と同様に木探索と評価関数を併用する強化学習手法であるが，評価関数にはバリュー関数のみが必要でありポリシー関数を必要としない．そこで本研究では，MCSS に基づく強化学習手法を 2048 に適用し，優れたプレイヤを実現することを目指す.

本研究では大きく分けて以下の 2 点の検証実験を行った.

- MCSS のアルゴリズムを確率的ゲームに適応させるために，探索の方法を制御する 6 つの規則を設計し，これを選択とバックアップの 2 つに適用した 36 種類のプレイヤを網羅的に評価するための比較実験を行った．評価関数には，既に学習済みのパラメータを持つ既存のネットワークを用いた．結果，2048 における木探索の有効な選択・バックアップ規則を得ることができた．
- 2048 において，木探索の結果を用いて学習を行うと，探索中の過大評価により評価値が発散するという問題が起こる．本研究では，Q 学習における過大評価の問題を克服する Double Q-learning を参考に，2 つのパラメータセットを用いる学習手法を設計し，8 通りのハイパーパラメータを持つプレイヤを用いて性能を検証した．結果，今回の方法では過大評価を十分に抑えることはできなかった．

学習なしでの比較実験においては，最も優れたプレイヤは 1 手あたり 2000 シミュレーションという制限の下で，平均スコア 533 542 であった．強化学習の実装に関しては，既存の TD 学習の派生アルゴリズムによる強化学習を越える結果は得られなかった．学習がうまく進まなかった理由について，考察を行う．

本論文の構成は以下の通りである．第 2 章では，ゲーム 2048 のルールを説明した後，2048 の持つ他のゲームと異なる特徴について示す．第 3 章では，MCSS のアルゴリズムとそれに基づく強化学習手法を解説する．第 4 章では，MCSS の 2048 への適用方法として，ランダム性を chance ノードとして含む木に対して探索を行うアルゴリズムを設計し，そのアルゴリズムにおける選択・バックアップの規則（ハイパーパラメータ）を示す．また，強化学習を行う際の問題点である，過大評価という現象について説明し，Q 学習においての過大評価を克服する手法である Double Q-learning についても解説する．第 5 章では，第 4 章で考案した規則についての徹底的な比較を行い，各規則に対して有用性を定義することで，2048 における有効な選択・バックアップの規則を得る．第 6 章では，Double Q-learning の手法を参考に，過大評価を克服する強化学習の実装方法について検討する．第 7 章で本論文をまとめる．

第 2 章

ゲーム 2048

2.1 ルール

ゲーム 2048 は，スライド & マージ型ゲームに分類される確率的一人ゲームである．2048 は，4×4 の盤面でプレイされる．初期状態では，2 つのタイルがランダムに置かれ，それらタイルの値は 2（確率 0.9）か 4（確率 0.1）である．各ターンにおいて，プレイヤは上下左右のいずれかの方向を選択し，全てのタイルを選択した方向に移動する．同じ値をもつ 2 つのタイルが移動方向に衝突すると，それらのタイルは結合して和の値を持つタイルとなり，その和の値が得点に加算される．ここで，結合するタイルは奥側が優先され，一度結合したタイルは同じターンではそれ以上結合しない．したがって，ある行のタイルが 222␣，␣422，2222 であるときに右向きに移動した結果は，それぞれ ␣␣24，␣␣44，␣␣44 となる．どのタイルも移動・結合しない方向を選ぶことはできない．移動・結合の後，新しいタイルが空きマスに 1 つ出現する．新しく出現するタイルの値は初期状態と同様に 2（確率 0.9）か 4（確率 0.1）である．プレイヤがどの方向にも移動・結合できなくなったら終局である．ゲーム 2048 における目標は，上記のルールのもとで移動と結合を繰り返して 2048 の値をもつタイルを作成することである．一方で，コンピュータプレイヤにとってこの目標は容易に達成可能である．そのため，本研究で主に用いるプレイヤの強さを示す指標は，獲得したスコアとする．

2.2　2048 のゲーム木

2048 において，盤面の状態は以下のように分類される：

- Beforestate プレイヤが 4 つの方向から 1 つを選択するフェーズ
- Afterstate 空いているマスにランダムなタイルが追加されるフェーズ

先行研究 [2, 3] では TD 学習に基づく評価関数の学習が行われており，優れた成果が得られている．とくに，評価関数の入力とする局面については，Szubert と Jaśkowski の研究により，学習においてランダム性の影響が小さくなるタイルの出現前（Afterstate 図 2.3）とすることが良いことが知られている．本研究でも，学習する評価関数の入力は Afterstate の盤面とする．

また，この分類に従いゲーム木を定義する．ルート（根）ノードはプレイヤが手を選択する状態であるので，Beforestate に対応する．その子ノードは Afterstate に対応し，更にその子ノードは Beforestate に対応する．従って，2048 のゲーム木は，chance ノードと max ノードに分けられる．

- max ノード プレイヤが報酬を最大化する手を選択する．Beforestate に対応する．
- chance ノード プレイヤが選択不能なランダムな位置にタイルが追加される．Afterstate に対応する．

max ノードは最大で 4 つの子ノードを持ち，chance ノードは最大で 30 個（空きマスが 15 か所の場合）の子ノードを持つ可能性がある．

2.3　評価関数：CNN22

強いプレイヤを開発するためには，優れた探索アルゴリズムと同様に，優れた評価関数が必要である．本研究では，Matsuzaki が設計した畳み込みニューラルネットワーク CNN22[2] を評価関数として使用する．CNN22 は，正規化済みの盤面状態を入力とし，そ

2.3　評価関数：CNN22

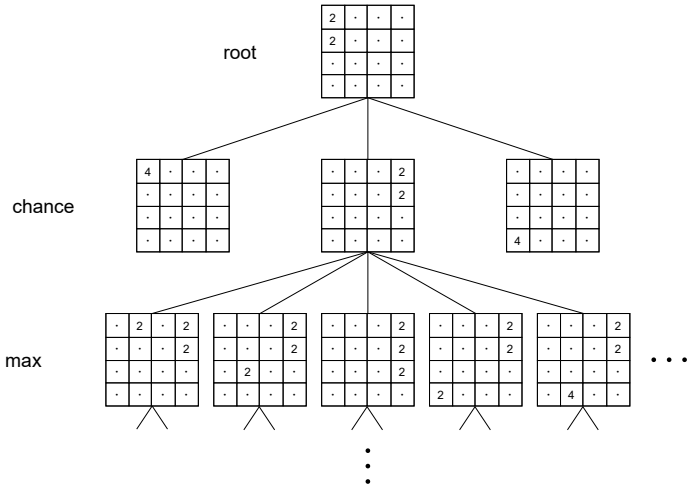

図 2.1　2048 のゲーム木

の評価値を返す.（図 2.2）

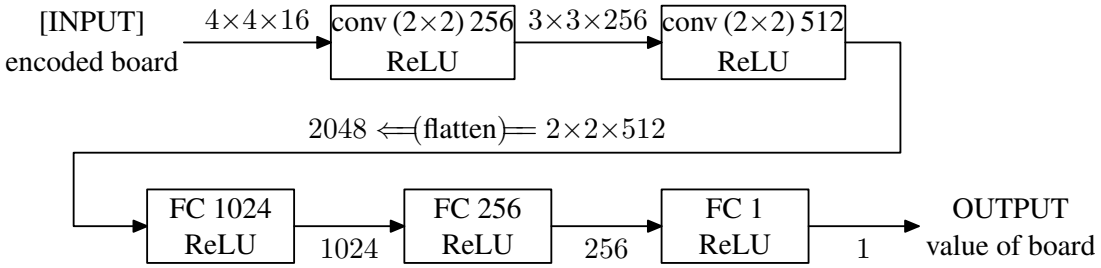

図 2.2　CNN22 の構成 [2]

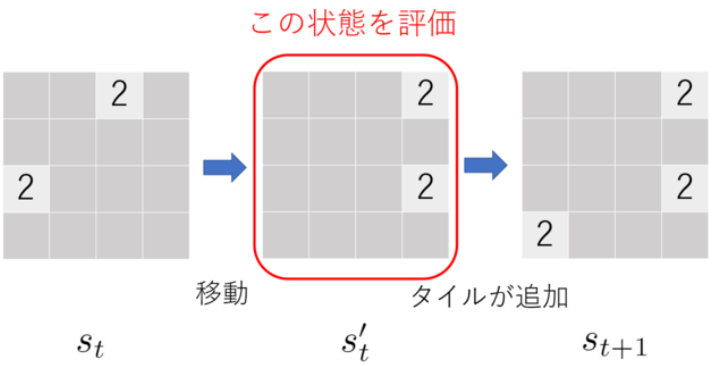

図 2.3　評価関数の入力とする盤面 (Afterstate) の図解

第 3 章

Monte-Carlo Softmax 探索

Monte-Carlo Softmax 探索 (MCSS) は，将棋ソフト「芝浦将棋 Softmax」で用いられている，モンテカルロ木探索（MCTS）をベースとした探索・学習手法である [1]．本章では，一般的な MCTS について説明した後，MCSS のアルゴリズムについて説明する．

3.1 モンテカルロ木探索

モンテカルロ木探索 (MCTS) は，ランダムシミュレーションに基づく確率的選択探索アルゴリズムである．以下の 4 つのステップを 1 シミュレーションとし，これを繰り返すことでゲーム木を成長させる．

- 選択：根から出発し，葉（末端）まで到達する．
- 展開：一定回数その葉を訪れたなら，その葉に子を加えて木を拡張する．
- 評価：選択された葉の評価値を計算する．
- バックアップ：計算された葉の評価値に基づいて，根までの経路上のノードの評価値を更新する．

最も一般的な MCTS のアルゴリズムに UCT[5] がある．UCT における選択ステップでは，子ノードの UCB 値を計算し，最大の UCB 値を持つ子ノードを選択する．評価ステップでは，評価値を得るために，ゲームの終局状態までランダムなゲームプレイを実行する．（プレイアウトと呼ばれる）．この時の評価値は，勝率を表す $[-1, 1]$ または $[0, 1]$ であることが多い．

3.2　Monte-Carlo Softmax 探索

MCTS の派生アルゴリズムである Monte-Carlo Softmax 探索 (MCSS) は，状態の評価値を使ってランダムシミュレーションの質を向上させる．MCSS では，選択とバックアップのステップにおいて，Boltzmann 分布を用いる．p を現在のノード，$x_i \in children(p)$ を p の子ノード，$v(x_i)$ を x_i の評価値，T を温度パラメータとする．このとき，ノード x_i を選択する確率は Boltzmann 分布により次のように与えられる．

$$\pi(x_i) = \frac{\exp(v(x_i)/T)}{\sum_{x_j \in children(p)} \exp(v(x_j)/T)}$$

選択ステップでは，子ノードの評価値による Boltzmann 分布に従う確率で子を選択する．選択ステップで用いる温度パラメータを，選択温度と呼ぶこととする．

図 3.1 は，選択ステップの一例を示したものである．親ノードは値が 0.5，0.45，0.1 の 3 つの子ノードを持っている．選択温度を $T = 0.2$ とすると，3 つの子を選ぶ確率はそれぞれ 0.522, 0.406, 0.070 である．この確率で子ノードを選び，例では 2 番目に良いノードが選ばれている．

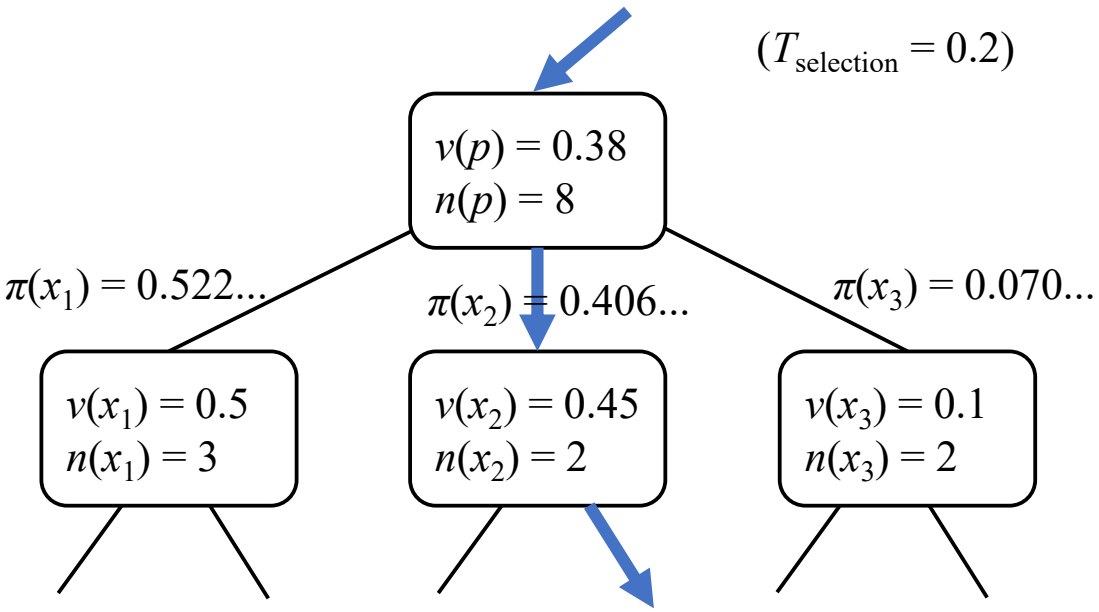

図 3.1　MCSS の選択ステップの例

バックアップステップでは，子ノードの評価値による Boltzmann 分布を重みとして，子ノードの値の重み付き平均を親ノードの値とする．バックアップステップで用いる温度パラメータを，バックアップ温度と呼ぶこととする．

図 3.2 はバックアップステップの例である．まず，2 番目の子の値が 0.33 に更新されたとする．このとき確率を計算しなおすと，0.405，0.335，0.260 となる．この重みを用いて重み付き平均を計算すると 0.34 となり，この値が親ノードの値となる．Boltzmann 分布における温度パラメータを調整することで，探索における「探索と活用」をコントロールできる．

また，評価ステップにおいては，プレイアウトを行う代わりに，評価関数を呼び出して葉の評価値を計算する．評価値による Boltzmann 分布により確率値を与えることで，MCSS の評価関数には局面の値を評価するバリュー（値）関数のみが必要となる．言い換えると，AlphaZero [4] とは異なり，ポリシー（方策）関数は不要である．

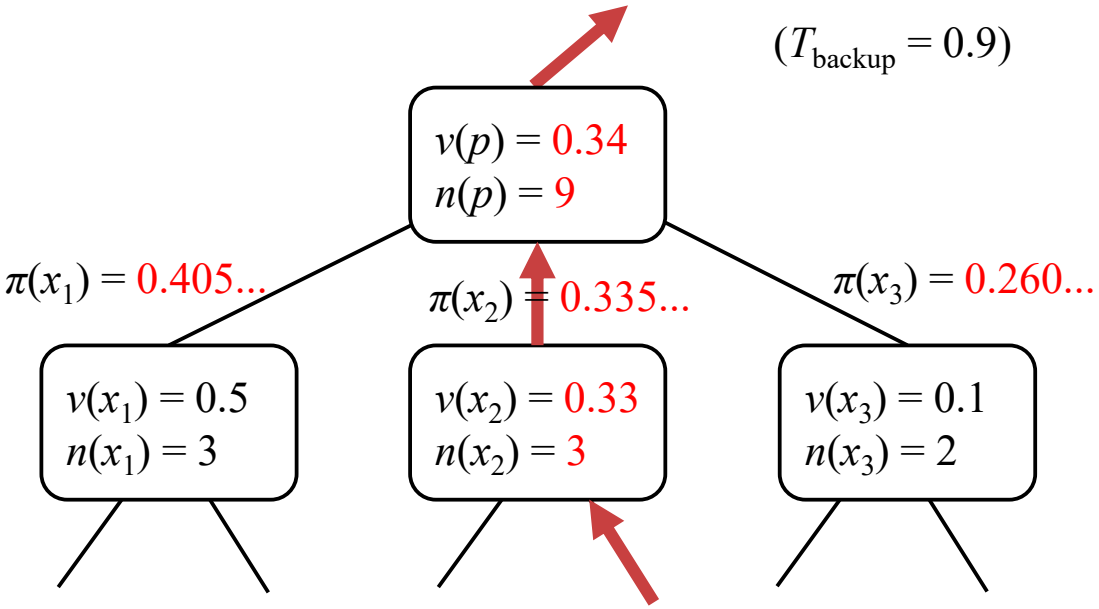

図 3.2　MCSS のバックアップステップの例

3.3　MCSS に基づく強化学習

MCSS を行うと，より深いノードの評価値に基づいて，根に近いノードの値が再計算される．したがって，探索後の内部ノードの値は，評価関数の値に比べてより正しい値を持つと期待される．この差を利用することで，評価関数を強化学習により調整することが可能である．具体的には，MCSS の計算が終わった後で，選択ステップと同様の手法により学習に用いる局面（ノード）をサンプリングする．

第 4 章

2048 への MCSS の適用に向けて

2048 には，一般のゲームと異なる点がいくつかあるため，MCSS を適用する際には専用のアルゴリズムを開発する必要がある．本研究では，以下に紹介するアルゴリズムを設計した．

4.1 確率的一人ゲームへの適用

4.1.1 chance ノードでのアルゴリズム

chance ノードは，ゲーム環境によってランダムにタイルが追加される直前の状態である Afterstate に対応する．Afterstate にある空のセルの数を k とおく．選択ステップでは，新しい 2 のタイルの子ノードを確率 $0.9 \times (1/k)$ で，4 のタイルの子を確率 $0.1 \times (1/k)$ でランダムに選ぶ．バックアップステップでは，子ノードの評価値の加重平均を計算して現ノードの評価値とする．chance ノードでの計算に用いるハイパーパラメータは存在しない．

4.1.2 max ノードでのアルゴリズム

max ノードは，プレイヤが方向を選択する Beforestate に対応する．評価関数とする CNN22 は Beforestate を評価するように訓練していないので（後に行う実験で学習済みのパラメータを用いる），max ノードの最初の訪問で，全ての子ノードを展開し，子ノード毎に CNN22 を 1 回ずつ呼び出す．

また，CNN22 は評価値として 0〜数十万の値を返す．しかし，Boltzmann 分布に含ま

れる exp 関数の引数としてこのような大きな数値を与えることは適切ではない．従って，Boltzmann 分布を求める際には評価値を何らかの方法でマッピングする必要がある．候補として，以下のマッピング方法を考案した．マッピング方法により，温度パラメータ T の操作方法も異なる．なお，式中の n は後述の選択・バックアップの規則によって意味が異なる．

log 評価値 v を $\ln v$ とマッピングし，温度 T を $T = 0.4/2^{\log_{10} n}$ とする．例えば，$n = 1$ で $T = 0.4$, $n = 10$ で $T = 0.2$, $n = 100$ で $T = 0.1$ となるよう設定した．

max 子ノードの最大値 $v_{\max}$ を用いて $v' = v/v_{\max}$ とマッピングする．温度 T を $T = 2.0/2^{\log_{10} n}$ とする．例えば，$n = 1$ で $T = 2.0$, $n = 10$ で $T = 1.0$，$n = 100$ で $T = 0.5$ となるよう設定した．

また，これらのマッピング方法に従って，選択とバックアップにおける規則を 6 種類設計した．

L-log 対象ノードの訪問回数を n として，log マッピングを行う．

G-log 全体のシミュレーション回数を n として，log マッピングを行う．

L-max 対象ノードの訪問回数を n として，max マッピングを行う．

G-max 全体のシミュレーション回数を n として，max マッピングを行う．

ave 温度を $T = \infty$ と固定する．選択の際には等確率で選び，バックアップの際には平均をとる．

best 温度を $T = 0$ と固定する．選択・バックアップの際には最大値を選ぶ．

従って，本研究で用いるプレイヤは，これらの規則を選択・バックアップステップそれぞれに適用した $6 \times 6 = 36$ 種類となる．

4.2　強化学習における問題：過大評価

以下の方法を試したが，いずれも同様に評価値が無限大に発散してしまう問題が発生した．

- 全ての内部ノードの値を学習データとする方法
- 内部ノードのうち，訪問回数が一定を超えるノードの値を学習データとする方法
- 上記の誤差を一定範囲でクリッピングする方法
- 実際に選択した手のみを使って学習させる方法

例として，学習中のノード値の推移を図 4.1 に示す．指数関数的に増えていることがわかる．本研究の MCSS のアルゴリズムでは，max ノードのバックアップステップにおいて，評価値が最大値である子を参照することで評価関数を学習させ続けている．仮説として，評価関数には誤差が少なからずあるため，その影響が学習によって修正されないまま大きくなり続けることが問題点であると考えられる．このような現象は行動と価値の評価に同一の関数を用いる Q 学習では起こりうることであり，過大評価（overestimation）と呼ばれている．この対策として，独立な分布を持つ 2 つの評価関数を用いる Double Q-learning という手法が提案されている [9]．本研究で行う強化学習の実装においては，Double Q-learning のアイデアをもとに過大評価を抑える学習アルゴリズムを考える．

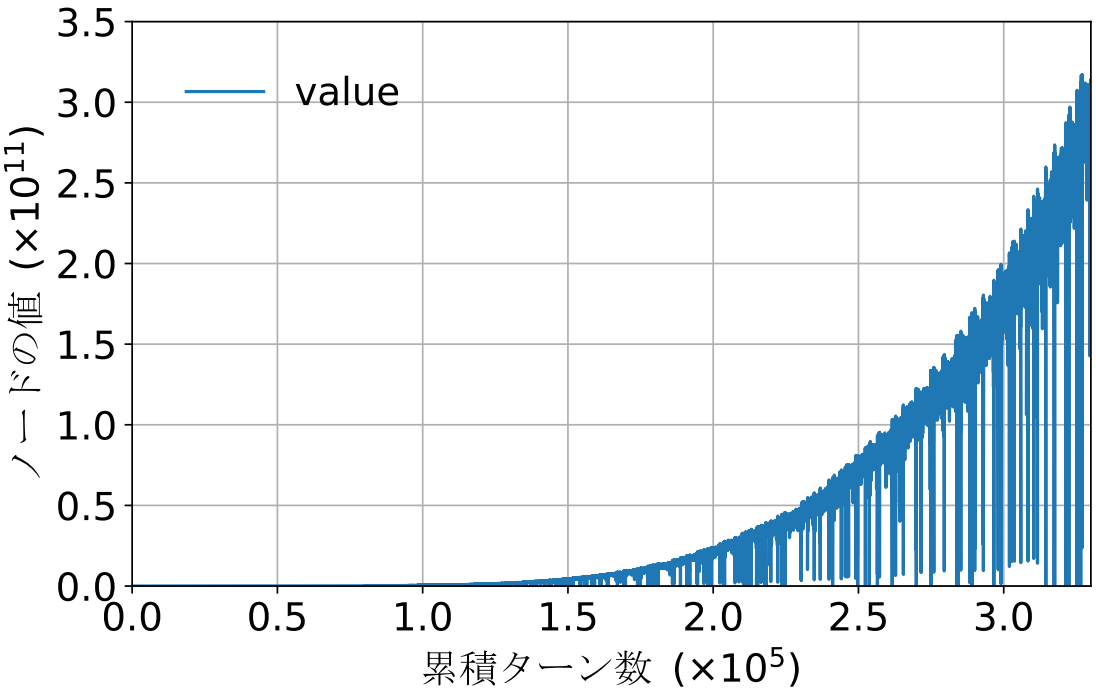

図 4.1　内部ノードの値の変動

第 5 章

選択規則とバックアップ規則の比較

本章では，第 3 章で考案した 6 種類の規則を，選択・バックアップそれぞれに適用したプレイヤを用いた比較実験をお行う．

5.1　36 種類のプレイヤによる規則の比較

6 種類の選択規則とバックアップ規則を比較するため，1 手あたりのシミュレーション回数 $N = 100$ を固定し，36 種類のプレイヤを用いた比較実験を行った．評価関数には，Matsuzaki による TD 学習を 240 時間行った CNN22[2] を用いた．平均スコアについては，5 セットの 50 ゲームにおける平均値と標準偏差を計算した．表 5.1 に実験結果を示す．プレイヤの成績は，選択規則よりもバックアップ規則に大きく依存することが容易にわかる．特に，バックアップ規則に best を用いた場合に，獲得平均スコアが最高値を記録した．（表 5.2）．一方で，ave, L-log, G-log, L-max をバックアップ規則として用いたプレイヤは，シミュレーション数 $N = 100$ であるにも関わらず，貪欲な（ 1 手先読みのみの）TD 学習プレイヤよりも成績が悪かった．

次に，選択規則として最適な規則を明らかにするために，以下の方法で選択規則の有用性を定量化した．まず，各バックアップ規則毎のプレイヤの平均スコアとその中央値を計算した（表 5.2）．そして，これらのバックアップ規則毎の平均スコアに対する，各プレイヤの平均スコア（算術平均，幾何平均）の比率によって，選択規則の有用性を定義した（表 5.3）．上位 2 つの L-max と G-max の差は非常に小さいが，L-max の方が最適なバックアップ手法との組み合わせで良い結果が得られたことから，以下の実験では L-max を選択手法に選

んだ.

L-max　子ノードの最大値 $v_{\max}$ を用いて $v' = v/v_{\max}$ とマッピングした値を評価値とし，対象ノードの訪問回数を n として，温度パラメータを $T = 2.0/2^{\log_{10} n}$ とする.

best　温度を $T = 0$ と固定する．選択・バックアップの際には最大値を選ぶ.

5.2　シミュレーション回数を変更する追加実験

1 手あたり N = 100 のシミュレーションを行った実験結果から，以下の 2 つのプレイヤを選んだ：

Lm-be　選択規則：L-max，バックアップ規則：best

be-be　選択規則：best，バックアップ規則：best

この 2 つのプレイヤを用いて，1 手あたりのシミュレーション回数を 25 回から 2000 回までの範囲で，より長い実験を行った．これまでの実験と同様に，各プログラムを 5×50 ゲーム実行し，平均スコア，最大スコア，n タイルの達成率（$n = 4096, 8192, 16384, 32768$) の 3 つの指標で評価した．平均スコアについては，5 セットの 50 ゲームにおける標準偏差とその平均値を計算した．表 5.4 に結果を示す．また，図 5.1 に，各プレイヤの 1 手あたりのシミュレーション数に対する平均スコアをプロットする．プレイヤ be-be は，$N = 100$ のシミュレーションで最も良い獲得スコアを記録し，それより少ないシミュレーション回数でも良い成績を収めたが，シミュレーション回数に対する性能の向上は小さかった．これは，選択ステップとバックアップステップにおいて決定論的な操作をしていることであると推測される．一方で，プレイヤ Lm-be は，シミュレーション回数に比例して著しい平均スコアの伸びを記録した．2000 回のシミュレーションのもとでの平均スコア 533 542 は，同じ評価関数を用いた expectimax 3-ply search の結果 (平均スコア 394 632[2]) を大きく上回るものであった．この結果は，筆者らの知る限り，PUCT アルゴリズムで 12800 回以上のシミュ

表 5.1　シミュレーション数 $N = 100$ におけるプレイヤの成績

規則		獲得スコア		
選択	バックアップ	平均	± 標準偏差	最大
ave	ave	38 592	± 3 024	133 028
L-log	ave	36 822	± 2 876	80 428
G-log	ave	40 005	± 2 050	80 076
L-max	ave	41 747	± 1 770	122 256
G-max	ave	41 620	± 2 229	123 880
best	ave	36 839	± 1 147	80 152
ave	L-log	44 745	± 5 702	157 056
L-log	L-log	59 842	± 2 810	177 296
G-log	L-log	62 586	± 5 997	269 840
L-max	L-log	66 247	± 4 512	248 424
G-max	L-log	65 873	± 7 636	210 504
best	L-log	52 118	± 3 768	168 132
ave	G-log	60 115	± 5 576	223 640
L-log	G-log	76 504	± 9 771	375 400
G-log	G-log	76 700	± 7 211	331 416
L-max	G-log	93 880	± 4 841	499 584
G-max	G-log	90 848	± 6 455	357 916
best	G-log	71 850	± 4 374	290 964
ave	L-max	93 665	± 14 257	385 920
L-log	L-max	136 006	± 12 909	626 148
G-log	L-max	133 627	± 7 481	734 448
L-max	L-max	156 129	± 17 627	620 564
G-max	L-max	157 096	± 7 042	630 220
best	L-max	98 921	± 4 102	375 876
ave	G-max	152 557	± 4 038	737 616
L-log	G-max	192 647	± 7 156	696 256
G-log	G-max	227 751	± 9 815	795 488
L-max	G-max	215 689	± 16 321	805 424
G-max	G-max	233 091	± 10 091	698 616
best	G-max	137 383	± 5 792	386 340
ave	best	264 972	± 18 360	741 940
L-log	best	284 261	± 12 275	733 284
G-log	best	293 769	± 10 852	743 544
L-max	best	**314 311**	± 24 277	818 008
G-max	best	299 953	± 9 637	630 260
best	best	**361 552**	± 12 842	819 036

レーションを用いた AlphaZero プレイヤー [6] 以外のニューラルネットワークプレイヤより優れている.

表 5.2　バックアップ規則の比較

バックアップ規則	獲得スコア	
	平均値	中央値
ave	39 270	39 298
L-log	58 568	61 214
G-log	78 316	76 602
L-max	129 240	134 816
G-max	193 186	204 168
best	303 136	296 861

表 5.3　選択規則の有用性

	有用性	
選択規則	算術平均	幾何平均
ave	0.8171	0.8128
L-log	0.9873	0.9864
G-log	1.0414	1.0392
L-max	1.1257	1.1239
G-max	1.1260	1.1231
best	0.9024	0.8901

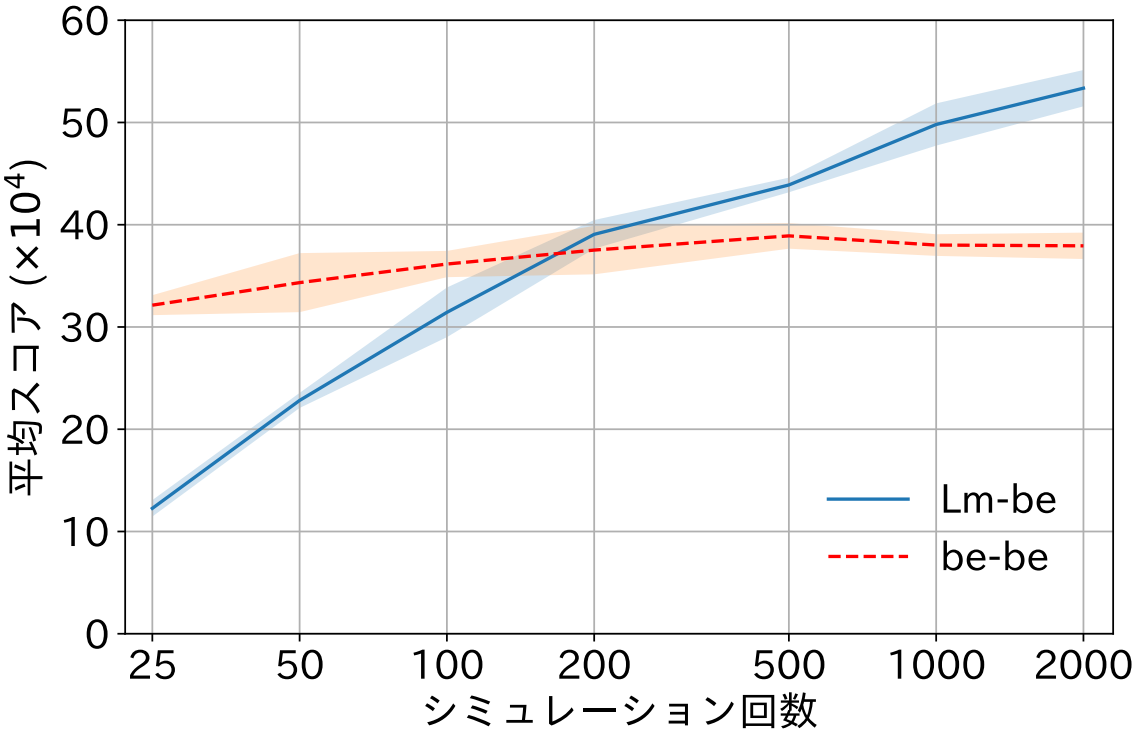

図 5.1　1 手あたりのシミュレーション回数に対する各プレイヤの平均スコア

表 5.4　シミュレーション数を変更した比較実験の結果

(a) 選択 = L-max / バックアップ = best

	獲得スコア		タイル達成率			
N	平均値 ± 標準偏差	最大値	4096	8192	16384	32768
25	122 641 ± 8 044	382 584	83.2%	54.4%	14.4%	0.0%
50	228 185 ± 7 370	626 008	96.0%	86.0%	48.4%	2.8%
100	314 311 ± 24 277	818 008	98.4%	93.6%	72.8%	11.2%
200	390 594 ± 14 003	809 756	99.2%	96.4%	83.2%	25.2%
500	438 882 ± 7 165	833 076	99.6%	99.2%	93.2%	32.4%
1000	498 014 ± 20 658	819 928	99.6%	98.8%	94.8%	46.0%
2000	533 542 ± 17 753	833 008	100.0%	99.6%	96.0%	54.4%

(b) 選択 = best / バックアップ = best

	獲得スコア		タイル達成率			
N	平均値 ± 標準偏差	最大値	4096	8192	16384	32768
25	321 332 ± 9 756	795 648	98.4%	93.6%	74.4%	12.4%
50	343 376 ± 28 865	777 064	98.4%	95.6%	79.2%	16.0%
100	361 552 ± 12 842	819 036	99.6%	97.6%	78.0%	18.4%
200	375 202 ± 23 715	794 852	99.6%	96.8%	82.4%	20.0%
500	389 101 ± 12 543	811 120	100.0%	97.6%	87.2%	20.0%
1000	380 114 ± 10 590	819 848	99.6%	98.0%	86.8%	20.8%
2000	379 369 ± 12 895	795 276	98.8%	97.2%	79.6%	24.4%

第 6 章

過大評価を避ける強化学習手法の検討

本章では，過大評価の影響を抑えつつ，探索した結果から評価関数を学習させる方法について検討する．

6.1 学習アルゴリズムの詳細

使用するプレイヤは以下の方法で探索と学習を行う．なお，本アルゴリズムは Double Q-learning のアイデアをベースとしている．

2 つの評価関数と評価値 葉ノードの評価において，パラメータの異なる評価関数を 2 つ用いる．それぞれ評価関数 A, B とする．それに伴い，探索木中のノードは評価値を 2 つ持つ．それぞれ evA, evB とする．

木の展開と学習方法 シミュレーション中，ノードを選択する際の基準となる評価値は一定のタイミング（ターン毎，もしくはエピソード毎）で 50 %の確率で以下の 2 つの方法のうち 1 つを選択する．

- 子ノードの選択（木の展開）の基準とする評価値: A，学習データとする評価値：B，学習させる評価関数: A
- 子ノードの選択（木の展開）の基準とする評価値: B，学習データとする評価値：A，学習させる評価関数: B

以上に示したように，木の展開の基準とした評価値と，学習させる評価関数，その学習

データとなる評価値は同期しており，常に評価関数 A は評価値 evB を，評価関数 B は評価値 evA を学習する．このように，異なる分布を持つ評価関数同士を学習の際に関与させることで，単一の評価関数による過大評価の影響を抑えることができると期待される．

評価値のバックアップ方法　バックアップの際の評価値の決定方法は 2 通りあり，評価値をバックアップするために最大値をもつ子のインデックス i_{max} を決定する際，もう片方の評価値によって i_{max} を決定する方法と，その評価値自身で決定する方法に分かれる．それぞれ方法 A(Alternate),D(Direct) と名付ける．$children$ を子ノードの集合とおき，子ノードはそれぞれ評価値 evA, evB と遷移時の即時報酬を表す値 r を持つとし，各方法での評価値の更新方法を以下に示す．

方法 A では，以下式で評価値の更新を行う．

$$i_{max} = \arg\max_i(children_i.r + children_i.evB)$$
$$evA \leftarrow children_{i_{max}}.r + children_{i_{max}}.evA$$

なお，evB の更新は式中の evA と evB を逆に読み替えたものである．

方法 D では，以下式で評価値の更新を行う．

$$i_{max} = argmax_i(children_i.r + children_i.evA)$$
$$evA \leftarrow children_{i_{max}}.r + children_{i_{max}}.evA$$

evB の更新方法も同様である．これらの違いは，バックアップの段階で評価関数どうしを互いに関与させるかどうかである．

サンプリングの対象となるノード　ターン終了時（プレイヤが手を選択した際）に，以下に示す一定の基準のいずれかに基づいて学習データのサンプリングを行う．

top5　全ノードのうち，訪問回数が上位 5 つ以内のノードを学習データとする．

child　根ノードの子のうち，最も評価値が高いノードを学習データとする．

これらの方法を用いて，過大評価の影響の抑制を試みる．

6.2　8 種類のプレイヤを用いた学習性能の比較

以下に実験で使用する 8 種類のプレイヤを示す．これらは木の展開と学習方法を変更するタイミング，バックアップ方法，サンプリング方法が異なる．

A-t5-turn　バックアップ方法：A．サンプリング方法：top5，木の展開と学習方法の抽選：ターン終了時

A-t5-ep　バックアップ方法：A．サンプリング方法：top5，木の展開と学習方法の抽選：エピソード終了時

D-t5-turn　バックアップ方法：D．サンプリング方法：top5，木の展開と学習方法の抽選：ターン終了時

D-t5-ep　バックアップ方法：D．サンプリング方法：top5，木の展開と学習方法の抽選：エピソード終了時

A-ch-turn　バックアップ方法：A．サンプリング方法：child，木の展開と学習方法の抽選：ターン終了時

A-ch-ep　バックアップ方法：A．サンプリング方法：child，木の展開と学習方法の抽選：エピソード終了時

D-ch-turn　バックアップ方法：D．サンプリング方法：child，木の展開と学習方法の抽選：ターン終了時

D-ch-ep　バックアップ方法：D．サンプリング方法：child，木の展開と学習方法の抽選：エピソード終了時

結果は，どのプレイヤも評価値の発散により，スコアの上昇は見られなかった．サンプリング方法が同じプレイヤの時間毎の平均スコアを，図 6.1，6.2 にプロットする．また，評価値の発散の様子もプロットする．例として，A-t5-ep のノードの評価値 evA の推移を図 6.3 に示す．指数関数的に増加していることがわかる．

この結果から，次のようなことがわかる．まず，最も発散の速度に影響があったのは，木の展開と学習方法をランダムに再選択するタイミングであった．ターン毎に再選択したプレ

イヤは，エピソード毎に再選択したプレイヤに比べてプログラムの停止が早かった（＝評価値の発散速度が高かった）．また，特にその差はサンプリング方法 ch で顕著である．サンプリング方法 ch では，根ノードの子，つまり木の中で最も浅いノードの中で，最も評価値が高いノードを学習データとする．これは木の中で最も評価値の更新回数が多いノードである確率が高く，その場合過大評価の影響を受けやすいと考えられる．これは仮説だが，同じエピソード中では探索において出現する局面が似通っているため，ターン毎に木の展開と学習方法を切り替えることで評価関数の分布の独立性を保つことができなくなっている可能性が考えられる．また，バックアップ方法は評価値の発散速度やスコアの伸びには影響していないようであった．結果として，今回の方法のみでは完全に評価値の発散を抑えることはできなかった．

6.3　評価値の発散の原因に関する考察

上の実験は，max ノードにおける選択・バックアップの際に最大値を持つ子を参照している点に原因があるという仮説の下で行われた．もしこの他の原因があるとすれば，2048 が持つランダム要素にあると考えられる．本研究の MCSS では，chance ノードにおける選択の際にはタイル追加の確率に従ってランダムに子を展開し，バックアップにおいては子の評価値の加重平均を計算する．これは一見問題がないように思えるが，その先のシミュレーションでは，高い評価値を持つノードが優先的に展開・再評価され，低い評価値を持つノードは軽視される．これにより，実際に近似すべき評価値よりも高い値が評価値として再計算され続けるため，過大評価が発生しているという仮説が考えられる．もしそうであった場合，MCSS のようなヒューリスティックな探索アルゴリズムを 2048 に適用する上で，探索によって計算された内部ノードの値を評価関数に学習させることは適切でない可能性がある．

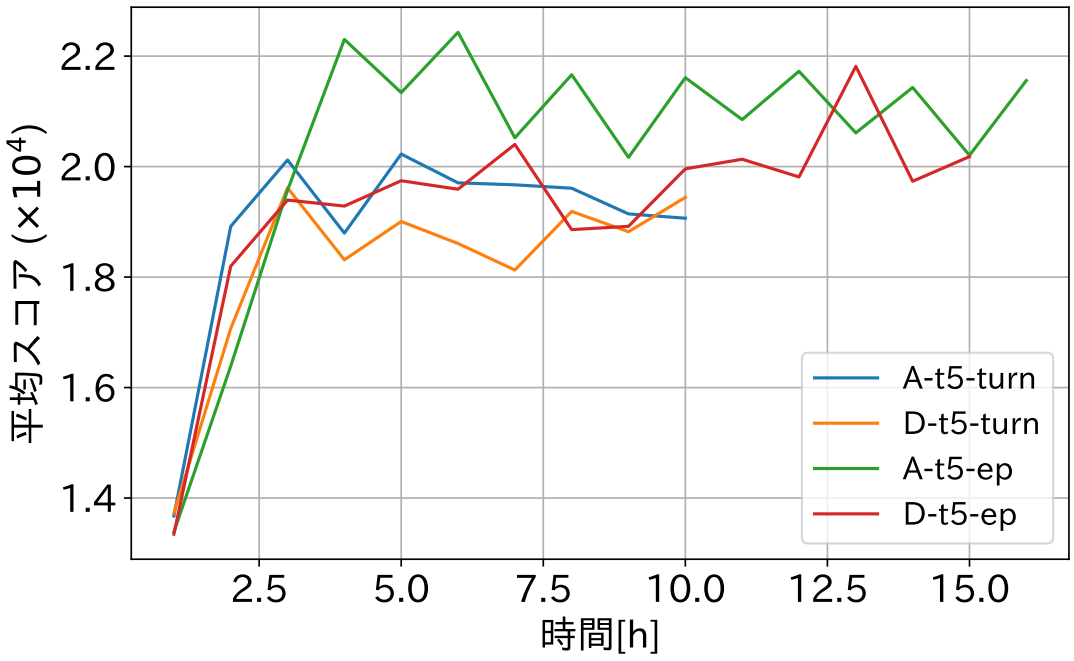

図 6.1　サンプリング方法が t5 のプレイヤの時間毎の平均スコアの推移

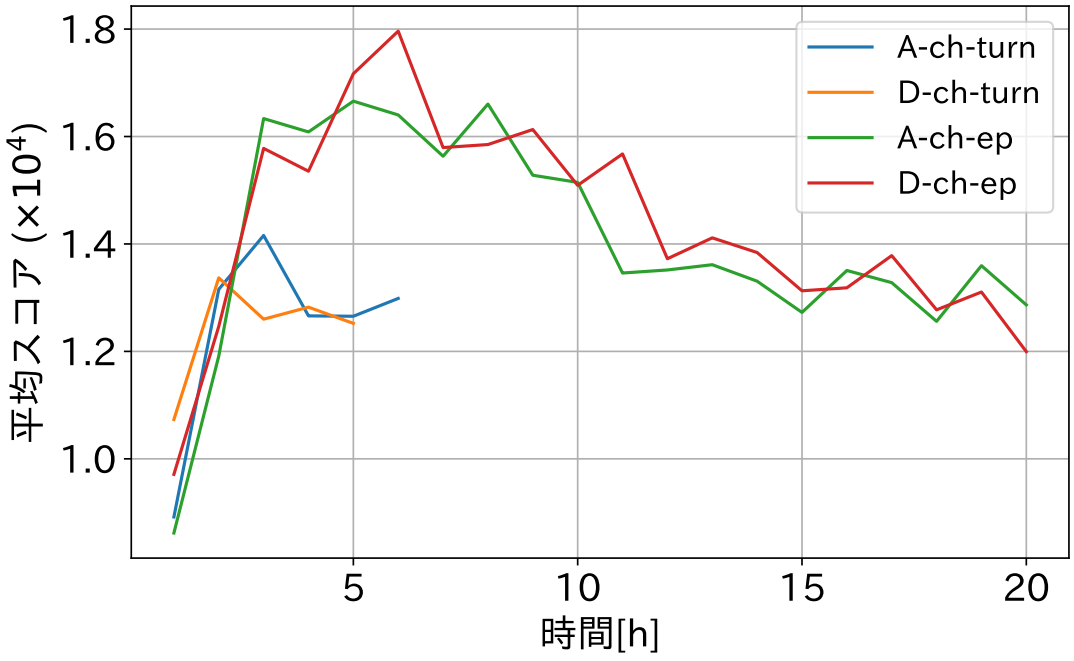

図 6.2　サンプリング方法が ch のプレイヤの時間毎の平均スコアの推移

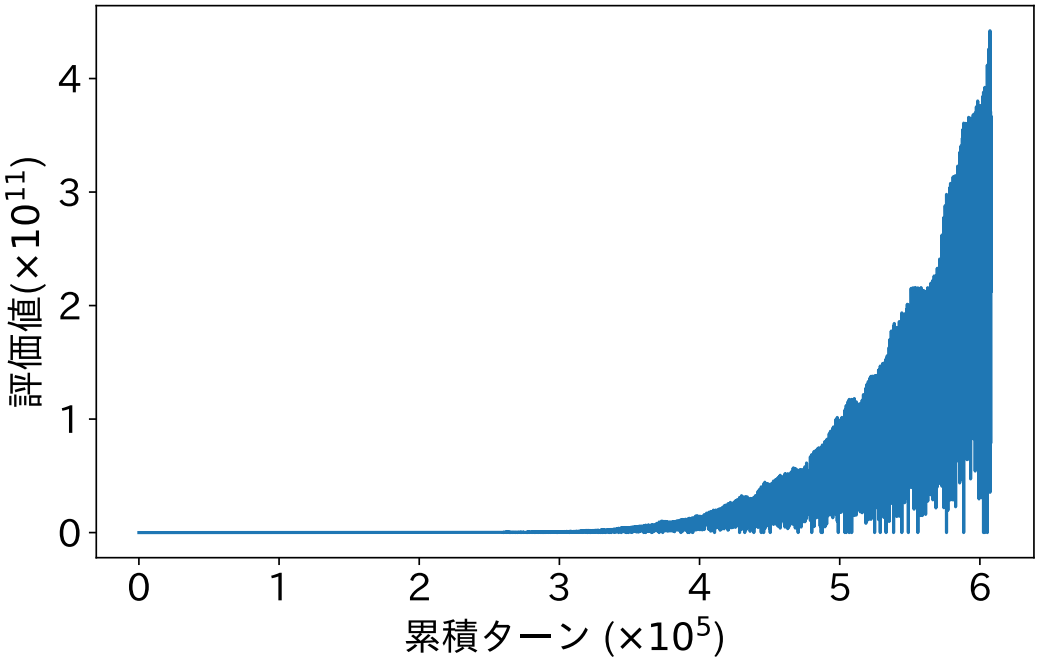

図 6.3　累積ターンに対する局面の評価値の推移

第 7 章

まとめ

本研究では，木探索と強化学習を併用した軽量なアルゴリズムの設計を目指し，既存の CNN ベースの評価関数と Monte-Carlo Softmax 探索を用いて，確率的一人ゲーム 2048 のコンピュータプレイヤを作成した．

まず，Monte-Carlo Softmax 探索のアルゴリズムを確率的ゲームに適応させるために，若干の修正を加えて適用し，探索の方法を制御する 6 つの規則を設計した．これを選択とバックアップの 2 つに適用した 36 種類のプレイヤを網羅的に評価するための比較実験を行った．結果，1 手あたり 2000 シミュレーションという制限の下で，平均スコア 533 542 を達成した．この結果は，同じ評価関数を用いた expectimax 探索を用いるプレイヤよりも優れていた．

また，強化学習を実装するための実験も行った．2048 のコンピュータプレイヤに関する先行研究によると，TD 学習ベースの強化学習により評価関数を学習させる手法が有効である．本研究では，学習精度を高めるため，木探索を行った結果を用いた強化学習の実現を目指した．しかしながら，単純にモンテカルロ木探索の結果を用いて学習を行うと，探索中の過大評価により評価値が発散するという問題が起こってしまう問題が発生した．この問題に対応するべく，本研究では，過大評価の問題を克服する Double Q-Learning を参考に，2 つのパラメータセットを用いる学習手法を設計した．内部ノードの学習による性能向上を目指し，バックアップやサンプリング方法などのパラメータを変更した 8 種類のプレイヤを用いて，学習性能を検証した．結果は，いずれも評価値が安定せず，過大評価の問題を克服することはできなかった．しかし，最初の学習なしの実験においては探索の有効性が示されており，かつノードが評価値と同等の値を持っているため，学習データとして活用できることが

望ましい．そのため，原因を究明した上で学習方法を改良することは今後の課題であるといえる．

謝辞

本研究を行うにあたり，指導教員である高知工科大学情報学群の松崎公紀教授には様々なご指導をいただきました．心より感謝申し上げます．また，本論文の副査を引き受けていただいた吉田真一教授，竹内聖悟講師に心より感謝申し上げます．本当にありがとうございました．

参考文献

[1] 五十嵐治一，森岡祐一，山本一将，MC Softmax 探索における局面評価関数の学習, ゲームプログラミングワークショップ 2018 論文集, pp. 212-219, 2018.

[2] K. Matsuzaki: Developing Value Networks for Game 2048 with Reinforcement Learning, *Journal of Information Processing*, Vol. 29, pp. 336–346, 2021.

[3] M. Szubert and W. Jaskowski: Temporal Difference Learning of N Tuple Networks for the Game 2048, *2014 IEEE Conference on Computational Intelligence and Games*, pp. 1–8, 2014.

[4] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, D. Hassabis: Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm, *arXiv*, Vol. 1712.01815 [cs.AI], 2017.

[5] R. Coulom: Efficient selectivity and backup operators in Monte-Carlo tree search, *in Proceedings of 5th International Conference on Computers and Games (CG 2006)*, 2006, pp. 72–83

[6] I. Antonoglou, J. Schrittwieser, S. Ozair, T. K. Hubert, and D. Silver: Planning in stochastic environments with a learned model, in *The Tenth International Conference on Learning Representations (ICLR 2022)*, 2022.

[7] W. Jaśkowski: Mastering 2048 with delayed temporal coherence learning, multi-stage weight promotion, redundant encoding and carousel shaping, *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 10, no. 1, pp. 3–14, 2018.

[8] H. Guei, L.-P. Chen, and I.-C. Wu: Optimistic temporal difference learning for 2048, *IEEE Transactions on Games*, 2021.

[9] H. van Hasselt, Double Q-learning, *Advances in Neural Information Processing Systems*, 23:2613–2621, 2010.

[10] S. Watanabe and K. Matsuzaki, Enhancement of CNN-based 2048 Player with Monte-Carlo Tree Search, *presented at the 27th Int. Conf. Technol. Appl. Artif. Intell. (TAAI2022)*, Tainan, Taiwan, Dec. 1–3, 2022.