

卒業論文

# ゲーム「2048」のプレイヤーについて

08-152021 金澤望生

指導教員 山口和紀 教授

2018 年 1 月

東京大学教養学部学際科学科総合情報学コース



## 概要

インターネットブラウザやスマートフォン上で遊ぶことのできるパズルゲーム「2048」をプレイする AI の改良を行った．改良には盤面上で最も大きな数のタイルが隅にあることを重視する独自のヒューリスティック「corner bonus」を使用した．(仮)

キーワード ゲーム AI , 機械学習



# 目次

第 1 章	導入	1
1.1	モチベーション	1
1.2	2048 について	1
1.3	2048 のルール	2
第 2 章	先行研究の紹介	3
2.1	Szubert & Jaskowski (2014)	3
2.2	Wu et al. (2014)	5
2.3	Oka & Matsuzaki (2016)	7
2.4	Yeh et al. (2016)	10
2.5	Jaskowski (2017)	11
第 3 章	本研究のアイデア	17
3.1	人間による 2048 の方策	17
3.2	corner bonus	17
第 4 章	提案と実装	18
4.1	corner bonus の 2048 プレイヤへの導入	18
第 5 章	実験	19
5.1	予備実験	19
5.2	CB を適用する箇所の検討	19
5.3	CB を変更する箇所の検討	19
第 6 章	考察と結論	20
	謝辞	21
	参考文献	22
	付録 A	23



# 第 1 章

## 導入

モチベーションや 2048 の基本ルール・指標について説明します。

### 1.1 モチベーション

「2048」は、インターネットブラウザおよびスマートフォンなどの端末でプレイすることができるパズルゲームである。非常にシンプルで誰もが直感的にルールを理解できるのに対し、ゲームを理解したり安定して勝ち続けたりすることは非常に難しい 2048 は、その絶妙なバランスが評価されて非常に多くの人々が遊ぶようになった。同時に 2048 はゲーム AI 研究の対象としても活発に取り上げられるようになり、近年では強化学習を用いて訓練した非常に強いプレイヤーが複数の研究者から発表されている。

これらのプレイヤーの実装手法は汎用性が高く、他のゲーム（特に 2048 に類似したゲーム）や課題に対しても適用することが可能な手法が多く用いられている反面、人間が 2048 をプレイする際に用いている経験的な知識・方策は明示的に組み込まれていないことが多い。一部の先行研究においては特徴量として人間が注目する指標を組み込んでいるものの、これらの指標も強化学習によって訓練されるため、経験的な知識が学習によって獲得されるということはない。

そこで、本研究においては、これまでに発表されてきた強化学習で訓練を行う 2048 プレイヤーに対して、人間による経験的な知識をトップダウン型で組み込むことにより、計算機によってのみ訓練されたプレイヤーよりも強いプレイヤーを実装し、新たな側面から 2048 の自動プレイヤーに改良の余地があることを示すことを目的とする。

### 1.2 2048 について

2048 が Gabriele Cirulli によって GitHub 上に公開されたのは 2014 年 3 月のことである。Cirulli は 2048 を開発していた当時、「1024」および「2048」<sup>\*1</sup>というゲームに熱中していた。これらのゲームはどちらも Asher Vollmer によるゲーム「Threes」にルールがよく似ている類

---

<sup>\*1</sup> Cirulli 自身が発表した 2048 とは似ているが、ルールが異なる別のゲームである

## 2 第1章 導入

似作品であるが、Cirulli は当時 Threes の存在については認識していなかった。Cirulli はこれらの Threes に端を発するゲームを改良するべく新たなゲームを開発し、ついに「2048」として公開することとなった。ここで挙げた 1024, Cirulli が参考にした 2048, そして Cirulli 自身が発表した 2048 は全て Threes の類似作品であるものの、これらの作品群の中では Cirulli による 2048 が最も人気を集めており、全世界で 2300 万人以上の人々が 2048<sup>\*2</sup>で遊んだとされる。

2048 はシンプルで直感的にルールを理解できる一方で、マスターすることが非常に難しいゲームであることが特徴となっており、このために多くのプレイヤーを熱中させていると同時にゲーム AI 研究の題材としても多く取り上げられていると考えられる。また、ゲームとしての 2048 について着目すると、2048 については以下のような特徴を挙げることができる：

### 完全情報ゲームである

2048 は単一の状態から始まり、1 人のプレイヤーが自分の手番で何かしらの行動を起こすことによって状態が遷移し、さらにプレイヤーはゲームの状態を全て把握することができる。したがって、2048 は完全情報ゲームであると言える。

### 非決定論的ゲームである

2048 は 1 人のプレイヤーによってのみプレイされるゲームであるが、状態の変化の際には乱数によって決定される要素があるため、プレイヤーは次の状態を知ることができない。したがって、2048 は非決定論的 (nondeterministic) ゲームである。ただし、プレイヤーは次の状態がいくつあり、その状態にどれぐらいの確率で遷移するのか自分で計算することはできる。

### 明確な終局が予め設定されていない

チェスや将棋といったゲームについては、どのような状態に到達するとゲームが終了するのか規定されており、プレイヤーはお互いに相手を終局に向かわせることを念頭にゲームをプレイする。一方で、2048 には次の状態に遷移することが不可能になったらゲームが終了するというルールはあるものの、その状態に到達するまでは無限にゲームを進めることができる。すなわち、プレイヤーが熟達すればするほど長い時間ゲームを進めることができ、その分プレイヤーの優秀さを示す指標も良くなる。この特徴により、2048 は強化学習やゲーム AI のベンチマークとしてよく機能していると考えられる。

## 1.3 2048 のルール

---

<sup>\*2</sup> なお、ここから単に 2048 と表記した場合、それは Cirulli が開発した 2048 のことを指す



## 第 2 章

# 先行研究の紹介

既存研究が使用している手法とプレイヤーの成績について説明します。(編注・提出原稿では以下は削除)なお, Jaskowski は Jaśkowski が正しい表記ですが, 差し当たり Jaskowski と表記し, 最後にまとめて正しい表記に置換します。

### 2.1 Szubert & Jaskowski (2014)

Szubert & Jaskowski は, TD 学習を用いたプレイヤーの訓練と  $n$  タプルネットワークを用いた価値関数の表現を組み合わせることによって, 人間の知識やゲーム木探索を使用しないで十分強い 2048 プレイヤを実装することに成功した。

#### 2.1.1 TD 学習

TD 学習の「TD」とは temporal difference の略であり, すなわち状態間における価値の差分を学習することによって学習器の訓練を行う手法である。2048 にあてはめると, とある盤面  $s'$  の価値と, その盤面の 1 プレイ後の盤面  $s'_{next}$  の価値の差分を取り, これを現状態  $s$  について  $s'$  に足し込んでいくことで訓練を行うことになる。TD 学習にはさまざまな派生があるが, Szubert & Jaskowski が使用している TD(0) 学習は以下の式によって表現される:

$$V(s) \leftarrow V(s) + \alpha(r + V(s') - V(s))$$

この式において,  $V$  は価値関数,  $\alpha$  は学習率,  $r$  は報酬である。学習率は計算された差分を価値関数の更新にどれほど反映するかを決定するパラメータである。

TD 学習は Tesauro によるバックギャモンへの適用でよく知られるようになり, 碁やオセロ, チェスにおけるゲーム AI の方策決定の手法として用いられるようになった。

#### 2.1.2 $n$ タプルネットワーク

TD 学習によって盤面の評価とその学習を行うことができるが, 盤面と評価値をどのように結びつけるかが問題になる。まず, 2048 で有り得るすべての盤面に対して評価値を与える 1

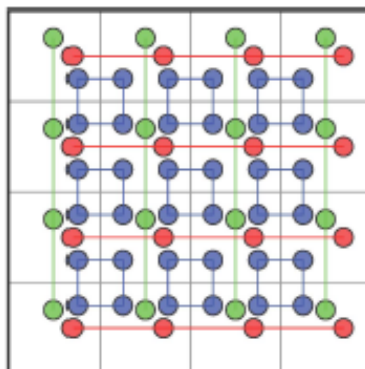


図 2.1. n タプルネットワークの例

対 1 対応のルックアップテーブル (LUT) を作成することを考えると, 2048 で有り得る盤面の数は  $(4 \times 4)^{18} \approx 4.7 \times 10^{21}$  と膨大な数になり, このような LUT を計算機上で実装することは現実的に不可能である。

そこで, 一部のマスの組み合わせによる「タプル」というクラスターを作成し, さらに複数のタプルを組み合わせることで盤面を表現する手法「n タプルネットワーク」を 2048 に導入することが, Szubert & Jaskowski によって提案された。たとえば, 図 2.1 のような n タプルネットワークを実装した場合, 1 つのゲーム内で保持すべき重みの数は 860625 であり, 全ての有り得る盤面に対する LUT を保持するのに対して非常に少なくて済む。

n タプルネットワークは Bledsoe & Browning (1959) によりパターン認識に用いられたのが最初の採用例である。ゲーム AI の分野では Jaskowski (2014) によってオセロに適用され, 一定の成果が得られた。

### 2.1.3 結果と課題

本手法をもとに行った実験のうち, 最も良い勝率を達成したプレイヤーを用いて 10 万ゲーム中の成績を検証したところ, 勝率は 0.9781 であり, 平均スコアは 100,178 であった。1 ゲーム中に達成されたスコアで最も良かったのは 261,526 であった。Szubert & Jaskowski による新たな手法は探索ベースの手法よりも大幅に高速で, かつ成績が良かった。

しかしながら, この手法では「常に 2048-tile を生成すること」よりも「時々 16384-tile を生成すること」を重視しているため, 勝率は必ずしも 100% を達成できていない。また, 人間の知識を一切導入していないため, 最も大きな数のタイルが盤面上の端に配置されないなど, 人間の直感的な戦略とは反しているといったデメリットがあった。

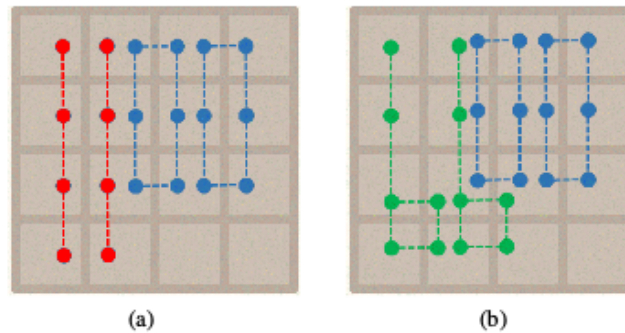


図 2.2. Szubert & Jaskowski (a) と Wu (b) が提唱した  $n$  タプルネットワーク

## 2.2 Wu et al. (2014)

Wu は, Szubert & Jaskowski の手法を改良し, 木探索を用いた先読みと組み合わせることによってさらに良いプレイヤーを実装することに成功した.

### 2.2.1 $n$ タプルネットワークの配置の改善

Wu は, Szubert & Jaskowski が考案した  $n$  タプルネットワークのうち, 直線型で 4 タプルとして配置していたタプルを, 図 2.2 (b) のように柄杓型の 6 タプルに変更した. これによって増える重みの数は約 2 倍程度であったが, この変更によって Szubert & Jaskowski のものよりも飛躍的に良い成績を得ることができた. なお, なぜこのようなタプルの配置が最善だと判断したのかについて, Wu は論文において言及していない.

### 2.2.2 Multi-Stage TD 学習の導入

Multi-Stage TD 学習 (MS-TD 学習) とは, ゲームの局面に応じて異なる価値関数を保持することによって, よりそれぞれの局面に対して適切な重みを学習させることを目的とした手法である. Wu は学習のプロセスを 3 つのステージに分割し, ゲームプレイも同様に 3 つのステージに分割して行うようにした. Wu の提唱した 2048 における MS-TD 学習は, 以下のよう手順で学習を行う. なお, 「 $T_{16k}$ 」とは「そのゲーム中で初めて 16384-tile を生成することに成功した時」, 「 $T_{16+8k}$ 」とは「そのゲーム中で初めて 16384-tile を生成した後に, 初めて 8192-tile を生成することに成功した時」のことを示す.

1. 第 1 ステージにおいては, 初期盤面からゲームを始めて, 価値関数が十分飽和するまで学習を行う. このステージで学習された価値関数の重みのことを「Stage-1 価値関数」と呼ぶことにする. また, 学習ゲーム中に  $T_{16k}$  を達成したなら, その時の盤面を全て保存しておく.

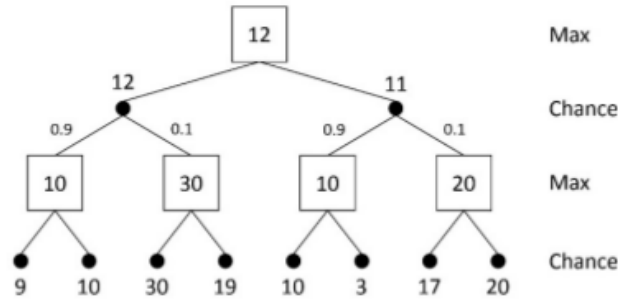


図 2.3. Expectimax 木の例 (Wu et al. (2015))

2. 第2ステージにおいては、第1ステージで保存した盤面からゲームを始めて、TD 学習を行う。このステージで学習された価値関数の重みのことを「Stage-2 価値関数」と呼ぶことにする。また、学習ゲーム中に  $T_{16+8k}$  を達成したなら、その時の盤面を全て保存しておく。
3. 第3ステージにおいては、第2ステージで保存した盤面からゲームを始めて、TD 学習を行う。このステージで学習された価値関数の重みのことを「Stage-3 価値関数」と呼ぶことにする。

その後、以下のような手順でゲームプレイを行う。

1. 盤面が  $T_{16k}$  を達成するまでは、Stage-1 価値関数を用いてゲームプレイを行う。
2. 盤面が  $T_{16k}$  を達成してから  $T_{16+8k}$  を達成するまでは、Stage-2 価値関数を用いてゲームプレイを行う。
3. 盤面が  $T_{16+8k}$  を達成してからは、Stage-3 価値関数を用いてゲームプレイを行う。

### 2.2.3 Expectimax 木探索

Szubert & Jaskowski (2014) によって、木探索ベースの 2048 プレイヤは強化学習で訓練したプレイヤに劣ることが示されたが、Wu は強化学習によるプレイヤに対して Expectimax 木探索を補助的に組み合わせることによって、さらにプレイヤの性能を高めようと試みた。

Expectimax 木探索には、Max ノードと Chance ノードという2種類のノードがあり、それぞれのノードの値は子ノードから決定される。Max ノードの値は、子ノードのうち最も大きな値のノードの値となる。例えば図 2.3 の根ノードの値は 12 であるが、これは子ノードの「12」と「11」のうち最大の値である 12 を取ったものである。一方で Chance ノードの値は子ノードの期待値となる。例えば図 2.3 の根ノードの子ノードの1つである「12」というノードは、0.9 の確率で 10 となるノードと 0.1 の確率で 3 となるノードの期待値、すなわち  $0.9 \times 10 + 0.1 \times 3 = 12$  によって 12 という値が決定する。

Wu の提案においては、Max ノードの値はプレイヤがアクションを選択して遷移を行った後

の盤面，Chance ノードは遷移を行った後にランダムタイルを発生させた後の盤面が与える評価値となる．例えば，ある盤面  $s$ （アクション選択と遷移が終わった直後の盤面とする）の評価値を深さ 3 の Expectimax 木探索を用いて求めたい時，図 hogehego のような探索木が考えられる． $s$  の盤面が分かれば，その子ノードであるランダムタイル生成後の盤面，さらにその子ノードである遷移後の盤面を求めることができる．さらに，葉ノードにあたる Chance ノードの値は価値関数が与える評価値とすることで，各ノードの値を求めることができ，最終的に根ノード，すなわち評価値を求めたい盤面  $s$  の評価値も求められるということになる．

（図 2.4. 何かいい感じの 2048 の探索木を自分で描画して貼る）

Expectimax 木探索を用いて先読みをすることで，将来の盤面を予想してより良いアクションを選択できるようになった．これはスコアの面での成績が良くなることに繋がる一方で，maxTile=2048 を達成する割合，すなわち勝率の向上にも大きな影響をもたらした．

## 2.2.4 結果と課題

Wu によるプレイヤーは Szubert & Jaskowski のものに比べて著しく良い成績を達成した．まず Szubert & Jaskowski が達成できなかった 32768-tile の生成に成功し，10.9% の確率で 32768-tile を生成できるようになった．勝率に関しては 1 を達成，すなわち 2048-tile は 100% の確率で生成できるようになり，平均スコアは 328,946，最大スコアは 605,752 を記録した．これは当時としては 1 つの例外<sup>\*1</sup>を除き，計算機による 2048 プレイヤの中で最も優れた成績であった．

一方で， $n$  タプルネットワークの形状変更や MS-TD 学習のステージングについては，この研究で行なわれた調整についてこれといった根拠が述べられておらず，依然として改良の余地は残していた．特に  $n$  タプルネットワークの形状変更については，次の Oka & Matsuzaki の研究で詳しく検討されることとなった．

## 2.3 Oka & Matsuzaki (2016)

Szubert & Jaskowski は当初図 2.1 のような  $n$  タプルネットワークを考案していたが，この  $n$  タプルネットワークによる学習器はあまり性能が高くなく，平均スコアは 10 万ゲーム学習した時点で 5 万～6 万程度にとどまっていた．そこで，図 2.2 (a) のような  $n$  タプルネットワークへの改良が行われ，その結果平均スコア・勝率ともに改善することができた．さらに Wu は Szubert & Jaskowski の考案した  $n$  タプルネットワークを改良し，成績を伸ばすことができた．

しかしながら，ここまでは「タプルのサイズを大きくすると学習器の成績も良くなる」という大まかな関係しかわかっておらず，成績を最善にする  $n$  タプルネットワークはどのような形状になるのか厳密には検討されていなかった．これを厳密に検討したのが Oka & Matsuzaki

<sup>\*1</sup> Xiao による深深度先読みと人間による調整を行った評価関数を用いたプレイヤーがこれにあたるが，Wu のものよりも 100 倍遅い：<https://www.youtube.com/watch?v=JQut67u8LIg>

表 2.1. 形成されうる  $n$  タブルの数

$N$	3	4	5	6	7	8	9	10	11	12	13
All	77	252	567	1051	1465	1674	1465	1051	567	252	77
Connected	8	17	33	68	119	195	261	300	257	169	66

















$N$	4 best tuples				4 worst tuples			
	$\langle 6-01 \rangle$	$\langle 6-02 \rangle$	$\langle 6-03 \rangle$	$\langle 6-04 \rangle$	$\langle 6-65 \rangle$	$\langle 6-66 \rangle$	$\langle 6-67 \rangle$	$\langle 6-68 \rangle$
6								
	31,161	24,530	22,207	20,576	9,644	9,642	9,563	9,052
	$\langle 7-001 \rangle$	$\langle 7-002 \rangle$	$\langle 7-003 \rangle$	$\langle 7-004 \rangle$	$\langle 7-116 \rangle$	$\langle 7-117 \rangle$	$\langle 7-118 \rangle$	$\langle 7-119 \rangle$
7								
	32,900	23,504	23,483	23,338	8,543	8,204	7,918	7,683

図 2.4. 最も優れている / 劣っている上位 4 つの 6 タブル・7 タブル (Oka &amp; Matsuzaki (2016))

である。

### 2.3.1 $n$ タブル単体の性能の評価

Oka & Matsuzaki は、まず  $n$  タブル単体の性能を網羅的に評価することを目指した。2048 の盤面上で  $n$  個のタイルを選んで形成されうる  $n$  タブルの数は、表 2.1 の通りとなっている。

ここで、Connected とはタブル上の全てのタイルが 1 個以上の他のタブルの接していることを指す。Oka & Matsuzaki は Connected タブルのうち、十分性能が良く、なおかつ現実的に  $n$  タブルネットワークとして運用することができる  $N = 6$  と  $N = 7$  の場合のみ検討することとした。彼らは次に、形成されうる Connected な 6 タブル 68 個の中からランダムに 10 個のタブルを選んで 100 万ゲームの学習を行う実験を 680 回<sup>\*2</sup>繰り返し、各タブルの性能を比較可能な数値として算出した。7 タブルについても同様な実験を行った。

その結果、6 タブルと 7 タブルのうち最も優れた 4 つのタブルと最も劣った 4 つのタブルが、図 2.4 の通り明らかになった。

### 2.3.2 $n$ タブルネットワークに組み込む $n$ タブルの数の検討

6 タブルおよび 7 タブルの性能が判明したので、性能が良い任意の数のタブルを組み合わせることで  $n$  タブルネットワークを作成することが可能になった。Szubert & Jaskowski および Wu は 4 個のタブルを組み合わせることで  $n$  タブルネットワークを作成していたが、Oka & Matsuzaki

<sup>\*2</sup>  $680 \times 10 = 6800$  より、全てのタブルが 100 回は選ばれるようにするため

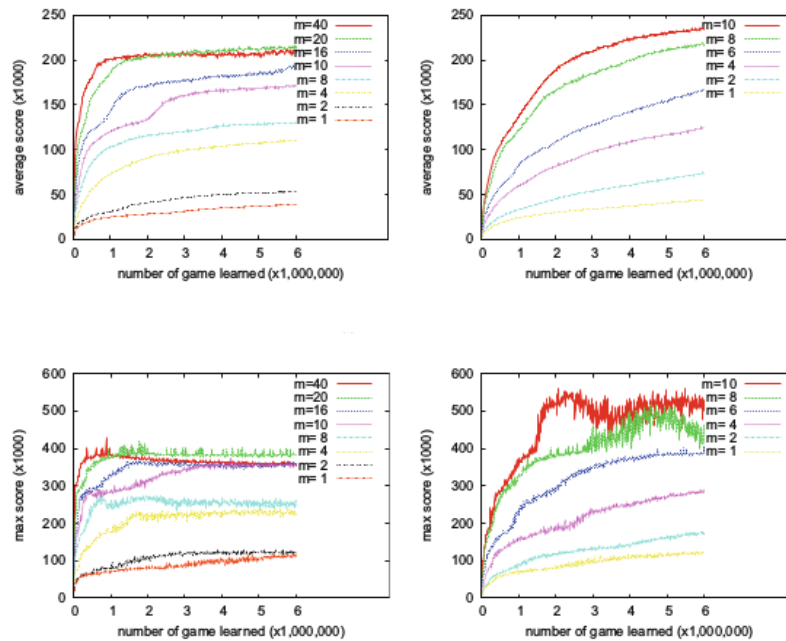


図 2.5. 成績上位タプル  $m$  個を組み合わせたタプルネットワークの実験 (Oka & Matsuzaki (2016))

は 5 個以上のタプルを組み合わせることを検討した．すなわち， $n$  タプルの数が多くなればなるほど  $n$  タプルネットワーク全体としての性能も上がると思われるが，最も性能が良くなるタプルの数は何個かを明らかにすることである．

実験においては，6 タプルの場合最大で上位 45 個，7 タプルの場合最大で上位 10 個のタプルから  $n$  タプルネットワークを作成することとした．各  $n$  タプルネットワークに対して，600 万ゲーム学習を行わせ，10,000 ゲームごとに平均スコアと最大スコアを記録した．その結果が図 2.5 である．なお， $m$  は実験を行うにあたって上位  $m$  個の  $n$  タプルを採用して  $n$  タプルネットワークを生成したことを示す．

上記の通り，実験群の中では 10 個の 7 タプルを組み合わせたタプルネットワークが最も良い成績を収めることがわかった．なお，グラフからも読み取れるが，7 タプルによるネットワークに関しては 600 万ゲーム学習しても平均スコア・最高スコアが収束しない．そのためさらに組み合わせるタプルを増やすとさらに成績が良くなることが考えられるが，技術上の制約により  $m = 11$  以上は実現できなかった<sup>\*3</sup>．

<sup>\*3</sup> 7 タプルを組み合わせるタプルネットワークを作成する場合，重みを保持するために 1 個のタプルにつき 1GB のメモリを消費しなければならない．Oka & Matsuzaki が実験を行った環境はメモリが 12GB しか確保できなかったため， $m = 11$  以上は実験を行うことが困難だったのではないかと考えられる．また，この研究においてゲーム木探索を組み込むことができなかったのもメモリの制約が原因ではないかと考えられる

### 2.3.3 結果と課題

Oka & Matsuzaki の研究における最も良い  $n$  タプルネットワークは、勝率が 0.9850、平均スコアが 234,136、最高スコアが 504,660 という成績を残した。これはゲーム木探索を用いない計算機を用いた 2048 プレイヤとしては最も良い成績であった。しかしながら、ゲーム木探索を用いなかったことによりゲーム序盤では安定性を欠いてしまい、 $W_u$  が達成した勝率 1 を割り込む形となってしまったし、単純に 2048 プレイヤとしての成績を比べると  $W_u$  のものよりも平均スコア・最大スコアともに低くなっている。一方で、ゲーム木探索を採用しなかったことはプログラムの高速化という点では利があり、1 秒あたり 88000 手の遷移を行うことができるプログラムとなった。これは  $W_u$  の研究と比べて約 290 倍高速である。

## 2.4 Yeh et al. (2016)

Yeh は Wu et al. (2014) の研究にも参画していた共同研究者で、 $W_u$  の研究の流れを引き継ぐような形で 2048 プレイヤの改良を行った。 $W_u$  の研究で用いた MS-TD 学習の考え方、改良された  $n$  タプルネットワーク、Expectimax 木探索は引き続き用いられているため、この研究で新たに加わった要素のみを述べる。

### 2.4.1 新たな特徴の追加

Yeh は、 $n$  タプルネットワークによる学習の他に、盤面に現れた特徴を評価関数として用いることとした。盤面の評価値には従来の  $n$  タプルネットワークによる評価値と新たに採用した特徴による評価関数の和を用い、これを TD 学習で訓練した。なお、採用した特徴は以下のつである。

1. 大きな数が書かれたタイルの個数
2. 何も書かれていないタイルの個数
3. 異なる数が書かれた (distinct) タイルの個数
4. マージすることができるタイルのペアの組数
5. 書かれているタイルの数が  $(n, 2n)$  の形になっているタイルのペアの個数

### 2.4.2 MS-TD 学習のステージングの改良

$W_u$  は MS-TD 学習を行うにあたり 3 ステージに分けて訓練・ゲームプレイを行っていたが、このステージの分け方をさらに細かくした。Yeh によってテストされたステージングは以下の通りである。なお、 $T_{x+y+zk}$  という表記は、「盤面に初めて  $1000x, 1000y, 1000z$  のタイルが同時に現れた時」のことを指す。各タイルの数は百の位で切り捨てられて表現されている。

1. 4 ステージ： $T_{8k}, T_{16k}, T_{16+8k}$  を境界としてステージを分ける



2. 4 ステージ :  $T_{16k}, T_{16+8k}, T_{16+8+4k}$  を境界としてステージを分ける
3. 5 ステージ :  $T_{16k}, T_{16+8k}, T_{16+8+4k}, T_{16+8+4+2k}$  を境界としてステージを分ける
4. 6 ステージ :  $T_{16k}, T_{16+8k}, T_{16+8+4k}, T_{16+8+4+2k}, T_{16+8+4+2+1k}$  を境界としてステージを分ける

これらのステージングを、上から順に戦略 1 から戦略 4 とする。まず戦略 1 をテストした際、 $T_{8k}$  を境界としてステージを分けることは意味がないことがわかったため、戦略 2 以降では  $T_{16k}$  以降のステージングのみを考えることとした。戦略 2 から戦略 4 をテストした結果、平均スコアでは戦略 4 が最も優秀であることがわかったが、最高スコアと 32678-tile の到達率では戦略 3 が最も優秀であった。各戦略ごとの成績は表 hogehoge の通りである。

(表 hogehoge をここに書きます)

以上の結果より、Yeh は戦略 3 が最も優れたステージングだと判断し、これ以降は戦略 3 を用いて実験を行った。

### 2.4.3 学習率の調整

TD 学習をより正確に行うために、まず学習率  $\alpha = 0.0025$  で TD 学習を行った後、価値関数に大きな改善が見られなくなったと判断したら、学習率  $\alpha = 0.00025$  に変更して学習を続行させるようにした。

### 2.4.4 $TD(\lambda)$ の使用

Szubert & Jaskowski および Wu は TD 学習において  $TD(0)$  を使用していたが、Yeh は  $TD(0.5)$  を用いて 5 ステップの報酬を平均化することとした。すなわち、

(ここに数式を書きます)

で表現される  $R_t^\lambda$  を用いて TD 学習を行うようにした。

### 2.4.5 結果と課題

上記の改良を重ねた結果、Yeh は平均スコア 443, 526, 最大スコア 793, 835 と非常に良い成績の学習器を訓練することに成功した。なお、32678-tile への到達率は 31.75%, 1 秒あたりの遷移速度は 500 手であった。さらに、検証ゲーム中のある 1 ゲームにおいて 65536-tile を生成することにも成功した。これは強化学習ベースの 2048 プレイヤでは当時として最も強く、Xiao のよる深深度探索ベースのものよりも平均スコアでは優っており、Yeh の方が 125 倍速いプログラムであった。

## 2.5 Jaskowski (2017)

Jaskowski は、Yeh までの 2048 プレイヤの改良の流れを受け継ぎながら、これまで用いられてきた TD 学習や訓練・検証のステージングなどに対して改良を加えつつ、新たな手法を加

えることでさらに優秀なプレイヤーの実装を目指した。

### 2.5.1 自動的な学習率決定アルゴリズムの適用

Jaskowski は  $TD(\lambda)$  を使用する点は Yeh の研究を継承したが、 $TD$  学習で最も重要なパラメタである学習率を決定するにあたり、新たな手法を提案した。すなわち、Yeh の研究では手動で学習率が設定されていたのに対して、Jaskowski はこれまでに提案されてきたオンライン適応学習による学習率決定のアルゴリズムを 2048 にも適用することを試みた。

Bagheri et al. による Connect 4 への適用の研究において、これらの学習率決定アルゴリズムはいずれも標準的な  $TD$  学習よりも良い成績を残すことが明らかとなったが、一方で長期的に見るとアルゴリズム間の差はそこまで大きいものにはならなかった。そこで、Jaskowski は最もシンプルな Temporal Coherence ( $TC(\lambda)$ ) と、より先進的でチューニングの必要がない Autostep を選択して実験することにした。

#### $TC(\lambda)$

これまでの  $TD(\lambda)$  では実験者が学習率  $\alpha$  を手動で決定していたが、 $TC(\lambda)$  においては、手動で決定される  $\alpha$  の代わりに以下のパラメータを用いて学習率  $\gamma$  を決定する。

- メタ学習率  $\beta$  : 実験者が手動で設定するパラメータ
- 累積誤差関数  $E(s)$  : これまで  $TD$  学習で価値関数  $V(s)$  の更新を行う際に用いてきた誤差  $\delta_t = r_{t+1} + V(s_{t+1}) - V(s_t)$  を、 $V(s)$  の更新を行う際に  $E(s)$  に対して加算する。盤面  $s$  は価値関数と同様に、 $n$  タプルネットワークを用いて近似される
- 累積絶対誤差関数  $A(s)$  :  $V(s)$  の更新を行う際に、 $\delta_t$  の絶対値を  $A(s)$  に対して加算する。盤面  $s$  は価値関数と同様に、 $n$  タプルネットワークを用いて近似される
- 適応学習率  $\alpha$  :  $E(s)$  と  $A(s)$  の値によって自動的に決定されるパラメータ

適応学習率  $\alpha$  は以下の式で求められる。

$$\alpha = \begin{cases} \frac{|E_i[s'_k]|}{A_i[s'_k]}, & \text{if } A_i[s'_k] \neq 0 \\ 1 & \text{otherwise} \end{cases}$$

また、学習率  $\gamma$  は以下の式で求められる。

$$\gamma = \frac{\alpha \times \beta}{m} \quad (m \text{ は } n \text{ タプルネットワークに格納されているタプルの数})$$

このような学習率の求め方をすることによって、累積誤差と累積絶対誤差の値が近い場合は学習率が大きく、逆に累積誤差と累積絶対誤差の差が大きくなると学習率は小さくなる。ある盤面  $s$  に対して累積誤差  $E(s)$  と累積絶対誤差  $A(s)$  の差が大きいということは、 $s$  に対して負の値の  $\delta_t$  が加算されることが多くなっているということであり、すなわちそれはこの盤面に対しての価値関数の評価が上手く行っていないということであるから、学習率を低くして価値

関数の更新を穏やかに行うのは理にかなっていると考えられる。

### Autostep

Autostep は Mahmood によって提案されたアルゴリズムで、Sutton による IDBD の拡張にあたる。Autostep では、ステップサイズの上限を定め、もしステップサイズが大きくなりすぎていることが検知されたらステップサイズ値を小さくするという方法で、IDBD を改良したものである。Autostep では 3 種類のパラメータを設定する必要があり、Jaskowski の研究では、このうち初期ステップサイズ  $\alpha_{init} = 1.0$ 、割引率  $\tau = 0.0001$  は固定され、メタ学習率  $\mu$  の値をいくつか選択して実験が行われた。

### 結果

Jaskowski は、前述の 2 つの自動的な学習率決定アルゴリズムに加え、従来の TD 学習<sup>\*4</sup>を採用した合計 3 種類の手法で、1 手先読み（すなわち、先読みを行わない）と 3 手先読みの 2 つの条件で実験を行った。その結果、全ての学習率決定手法のうち、1 手先読みの場合は  $TC(\lambda)$  で  $\beta = 1.0$  の時に平均スコアが最も良くなり、3 手先読みの場合は  $TC(\lambda)$  で  $\beta = 0.5$  の時に平均スコアが最も良くなった。1 手先読み・3 手先読みの場合ともに、 $TD(\lambda)$  および Autostep はどのように学習率・メタ学習率を設定しても  $TC(\lambda)$  に優る平均スコアを達成することはできなかった。

Jaskowski は、Autostep は定常的な教師データが存在する作業に特化しているため、訓練データが定常的ではない 2048 にはあまり適していなかったと考えている。また、 $TC(\lambda)$  および Autostep 両者に共通する欠点として、何かしらの近似した盤面に対応する重みを保持する必要があることから、 $TD(\lambda)$  よりも多くの n タプルネットワークを保持しなければならないため、計算時間が長くなってしまうことを指摘している。

## 2.5.2 Multi-Stage Weight Promotion

Jaskowski は Wu が提唱した MS-TD 学習の考え方を概ね継承したが、ステージングと重みの選択において細かい改良を加えた。

### 新しいステージング

Wu では 3 ステージ、Yeh では 5 ステージに分けることが最善であるとされていたステージングについて、Jaskowski は  $2^g$  ステージに分け（ $g$  は正の整数）、また各ステージの「長さ」 $l$  を  $l = 2^{15+1-g}$  と定義した。例えば  $g = 4$  の時、長さ  $l = 2^{12} = 4096$  である。この場合、以下のようなステージングがなされる。

1. ステージ 1：初期盤面から  $T_{4096}$  まで
2. ステージ 2： $T_{4096}$  から  $T_{8192}$  まで

---

<sup>\*4</sup> なお、 $TC(\lambda)$  および  $TD(\lambda)$  においては、 $\lambda = 0.5$  とした

## 14 第2章 先行研究の紹介

3. ステージ 3 :  $T_{8192}$  から  $T_{8192+4096}$  まで
4. ステージ 4 :  $T_{8192+4096}$  から  $T_{16384}$  まで
5. ...
6. ステージ 16 :  $T_{32768+16384+8192+4096}$  から  $T_{65536}$  まで

なお,  $TC(\lambda)$  で導入した累積誤差関数  $E$  と累積絶対誤差関数  $A$  についても同様にステージングを行って重みを保持する.

### Weight Promotion

Wu および Yeh のステージ分けにおいては, 新しいステージの学習を始める際に再び何も学習していない状態から  $n$  タプルネットワークの訓練を行わなければならなかったため, 汎化性能が著しく損なわれていた. この汎化性能の弱体化はステージ数が多くなればなるほど進むため, Yeh はステージを多くすれば多くするほど学習器の性能は悪くなることを指摘しており, その結果 5 ステージに分けるのが最も良いと考えたのである. この問題を「ステージ数を多くしない」以外の方法で解決するために, Jaskowski は各ステージの初期状態の評価値を, 前のステージの同じ盤面から引き継ぐことにした. Jaskowski はこの方策を Weight Promotion と呼び, 上記の多段ステージ分けとともに提案した.

### 結果

Jaskowski は, 以下の 6 つの条件について, 1 手先読みと 3 手先読みの 2 つの条件を掛けあわせて合計 12 の実験を行った. なお, WP は Weight Promotion を適用したことを示し, 6. 以外の 5 つの実験においては全て共通の 5 つの 6 タプルによる実験を行った.

1. ステージ分けを行わない (純然たる  $TC(\lambda)$ )
2.  $g = 4$
3.  $g = 3, WP$
4.  $g = 4, WP$
5.  $g = 5, WP$
6. 5 つの 7 タプルによる  $TC(\lambda)$  で, ステージ分けを行わない

この結果, 1 手先読みでは 6. の 7 タプルによる  $TC(\lambda)$  が最も良い成績を収め, 1. に関しても 2. と 5. の実験には成績が優るなど, ステージ分けに対しては顕著な効果が見られなかった. 一方 3 手先読みでは 4. の実験が最も良い成績を収め, ステージ分けを行わない 1. と 6. の実験よりも高い平均スコアを得た. また学習プロセスに注目すると, ステージ分けを行う実験では平均スコアが単調増加していたのに対し, ステージ分けを行わない場合は平均スコアが逆に悪くなっていることが確認された.

以上の結果より, Jaskowski はステージ分けを行わない学習器は 1 手先読みの条件に対して過学習が起きてしまうが, ステージ分けを行う学習器は過学習に耐性があり, 成績の悪化は起こらなくなると結論付けた. なお, Weight Promotion については 1 手先読みに対しても 3 手

先読みに対しても適用することでより良い成績が得られることが確認された。

### 2.5.3 Carousel Shaping と余剰タブルの追加

#### Carousel Shaping

前節のステージ分けにおける改良を行った際、従来の学習器は 1 手先読みの条件の下で過学習を行ってしまい、3 手先読みでは学習を重ねるに従って成績を落としてしまうという状況が確認された。これは従来の Multi-Stage 学習ではゲームで重要になる終盤のステージでの学習時間が少なくなってしまうことが原因であると考えた Jaskowski は、終盤のステージにおける学習時間を確保するため、Carousel Shaping という新たな Multi-Stage 学習の手法を提案した。Carousel Shaping の疑似コードは以下の通りである。

---

#### Algorithm 1 Carousel Shaping

---

```

1: function CAROUSELSHAPING
2:    $stage \leftarrow 1$ 
3:    $initstates[x] = \emptyset$  for  $x \in \{1 \dots 2^g\}$ 
4:   while not enough learning do
5:     if  $stage = 1$  then
6:        $s \leftarrow \text{INITIALSTATE}()$ 
7:     else
8:        $s \leftarrow \text{RANDOMCHOICE}(initstates[stage])$ 
9:        $\text{LEARNFROMEPISODE}(s) \triangleright \text{Updates } initstates$ 
10:     $stage \leftarrow stage + 1$ 
11:    if  $stage > 2^g$  or  $initstates[stage] = \emptyset$  then
12:       $stage \leftarrow 1$ 
13: end function

```

---

Carousel Shaping (CS) では、明確に学習を行うステージを分けるのではなく、1 回のゲームの中で複数のステージにまたがった学習を可能にする。ステージ 1 の初期盤面から学習を始め、もし途中でステージ分けの境界にあたる盤面に到達したならその都度 *initstates* に登録しつつ<sup>\*5</sup>学習を行う。学習が 1 ゲーム分終わったら次のステージに進み、そのステージの初期盤面を取得し、学習を行う。もしステージが  $2^g$  を超えるか、そのステージの初期盤面がまだ 1 つも記録されていなかった場合、ステージ 1 に戻って学習を行う。このように、CS はその名 (Carousel=回転木馬) の通り学習するステージをローテーションさせながら学習を行う。

---

<sup>\*5</sup> *initstates* は各ステージの初期盤面を保持するデータ構造で、各ステージごとに最近到達された初期盤面 1000 個を保持する

### 余剰タプルの追加

Jaskowski が導入した最後の手法が余剰タプルの追加 (Redundant Encoding) である。これまで使用してきた 5 個の 6 タプルに加え、より小さい 2 個の 3 タプルと 5 個の 4 タプルをタプルネットワークに加えて学習を行う。ここで追加したタプルは既に使用しているタプルの形状の中に含まれているため、一見すると無意味なタプルではあるが、追加することでより速く汎化することが確認された。ただし、サイズが小さいとはいえ 7 個のタプルを追加するため、学習時間は非常に長くなる。

### 2.5.4 結果

以上の改良を加え、さらに Expectimax 木探索による探索を先読みする手数ではなく 1 手あたり 1000 ミリ秒まで許容したところ、Jaskowski による 2048 プレイヤは平均スコア 609,104 を記録した。これは先行研究のいずれよりも優れた成績である。またこの研究における最も優秀なプレイヤは 1 秒に 1 手のペースでアクションを選択するため、1 秒あたり 500 手進めることができる Yeh のものより表面上は遅くなるが、仮に 1 秒に上限 1000 手のペースでアクションを選択するよう調整した場合は平均スコアが 527,099 となり、Yeh の 443,526 を上回る。すなわち、学習のみならず検証時のプレイにおいても Yeh のプレイヤよりも性能が高いということになる。

## 第 3 章

# 本研究のアイデア

本研究で導入しようとしている手法のアイデアについて説明します。(編注) 以下では本研究で導入するアイデアのことを仮に「corner bonus」と表記しますが、この表記は後ほど別の表現に置き換えることになるかと思います

### 3.1 人間による 2048 の方策

一般的に人間が 2048 をプレイする時に採用する方策・知識を説明します

### 3.2 corner bonus

上記の中でも特に「最も大きな数のタイルは隅に置くべきである」とする方策 ,corner bonus を説明します

## 第 4 章

# 提案と実装

### 4.1 corner bonus の 2048 プレイヤへの導入

corner bonus を 2048 プレイヤへ導入することを提案します．具体的には評価値に CB を掛け合わせるとか



## 第 5 章

# 実験

提案したアイデアの実験結果と既存研究の実験結果を比較します．比較対象は Szubert？

### 5.1 予備実験

CB ってホントに意味あるのかというところを明確にするために  $CB=1.1 \sim 1.5$  で適用して実験を行います 1.5 では意味がないけど 1.1 とか 1.2 なら意味ありそうだぞということを言いたい

### 5.2 CB を適用する箇所の検討

学習中・検証中・どちらにも CB を適用した結果を比較しますどっちもが一番意味ありそうということを知りたいです

### 5.3 CB を変更する箇所の検討

スコア 80000 ぐらいで CB を 1.1 に変更するといいぞってことを言いたい

## 第 6 章

# 考察と結論

実験結果をもとに，結果の考察を行い，本研究をまとめます．

# 謝辞

謝辞を書きます。

## 参考文献

[1]

[2]

## 付録 A

表やプログラムリストの掲載が必要になったらここに掲載します。