動的に行動を変化させる対戦型テトリスAI

瀧澤, 和也 / Takizawa, Kazuya

(出版者 / Publisher)
法政大学大学院理工学研究科
(雑誌名 / Journal or Publication Title)
法政大学大学院紀要. 理工学研究科編
(巻 / Volume)
65
(開始ページ / Start Page)
1
(終了ページ / End Page)
8
(発行年 / Year)
2024-03-24
(URL)
https://doi.org/10.15002/00030747

動的に行動を変化させる対戦型テトリス AI

Competitive Tetris AI with dynamically changing behavior

瀧澤和也
Kazuya Takizawa
指導教員　平原誠

法政大学大学院理工学研究科応用情報工学専攻修士課

When playing competitive Tetris, players observe their opponents' boards and adjust their actions accordingly. In contrast, traditional Tetris AI models rely solely on the player's own board state for decision-making. This study aimed to develop a Tetris AI that dynamically adjusts its gameplay strategy by utilizing a weight-generating neural network. The designed neural network generates weights for a function that evaluates the AI's board using both its own board and the opponent's board as input, enabling it to adapt its behavior based on the evolving game state. By integrating information from both sides of the gameplay, our AI represents a departure from conventional Tetris AI approaches, thereby encouraging more adaptive and competitive gameplay strategies.

Key Words：*Tetris, Genetic Algorithm, Neural Network*

1.　はじめに

人が対戦型のテトリスをプレイする際には，対戦相手の状況を見ながら自身の行動を変えている．しかし，現在作成されている多くの対戦型テトリス AI は，一人用のテトリス AI[1][2][3]と同じで対戦相手の盤面を参照せずに自身の盤面の状況のみで次の行動を決定している．対戦相手の状況に応じて行動を変更する AI[4]もあるが，これは攻撃時と防御時の 2 つの行動を切り替えているだけにすぎず，対応できる状況が限られてしまう．

そこで，本研究では盤面を評価する関数の重みを，対戦中に動的に変更することで動的に行動を変更するテトリス AI を作成することを試みた．

2.　従来のテトリス AI

従来のテトリス AI のプレイ手順を図 1 に示す．テトリス AI はまず現在の自身の盤面に操作ミノ（現在操作しているテトリミノ）が設置可能な盤面をすべて列挙する．これらの盤面を盤面評価関数で評価し，評価値が最大となった盤面になるように操作ミノを設置する．これを繰り返してテトリスをプレイする．

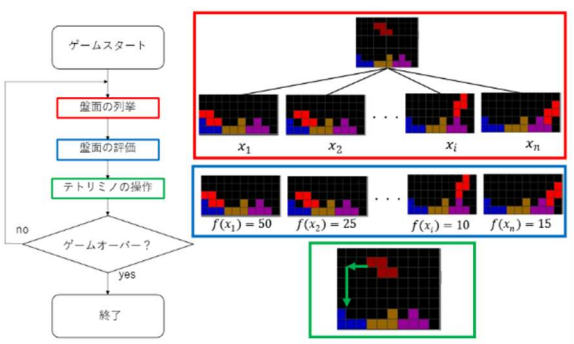

図 1　テトリス AI のプレイ手順

3. 提案手法

3.1 盤面評価関数

盤面評価関数とは，盤面の特徴ベクトルと学習などで得られる重みベクトルとの内積で盤面の評価を数値化するための関数である．プレイ時には評価値が最も高い盤面になるようにプレイするので，重みベクトルが変化するとAIの挙動は変化する．

本研究では9つの特徴量を用いるため，盤面xにおけるi番目の特徴量を$f_i(x)(i=1,\dots,9)$とし，重みをw_iとすると盤面評価関数$f(x)$は

$$f(x) = w_1 \times f_1(x) + w_2 \times f_2(x) + \cdots + w_9 \times f_9(x) = \sum_{i=1}^{9} w_i f_i(x)$$

と表せる．

以下に9つの特徴の説明を示す．

$f_1(x)$：最大の高さ

$f_1(x)$は，盤面内の最も高いブロックの高さを返す．「最大の高さ」が大きいとゲームオーバーしやすくなるので，これを低くすることが大切である．図 6 は最大の高さが11となる盤面の一例である．

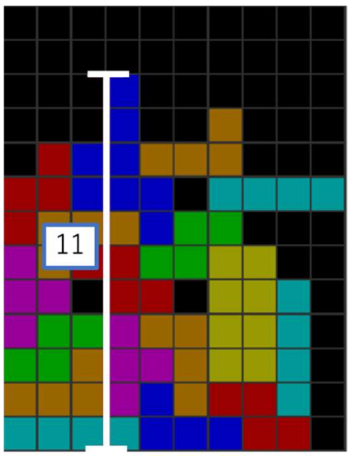

図2　$f_1(x)$が11となる盤面xの一例

$f_2(x)$：穴の上のブロック数

$f_2(x)$は，穴の上にあるブロックの総数を返す．穴の上にブロックがあるとその穴を消去するために上のブロックを消去する必要があり，有効に使える盤面が狭くなってしまう．図3は穴の上のブロック数が4となる盤面の一例である．

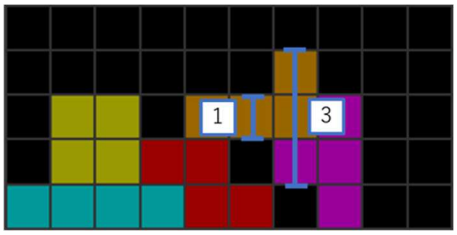

図3　$f_2(x)$が4となる盤面xの一例

$f_3(x)$：穴の深さ

$f_3(x)$は，縦方向につながった穴の深さを返す．「穴の深さ」が大きいと同時消しができるので相手にお邪魔ブロックを効率的に送ることができる．図4は穴の深さが4となる盤面の一例である．

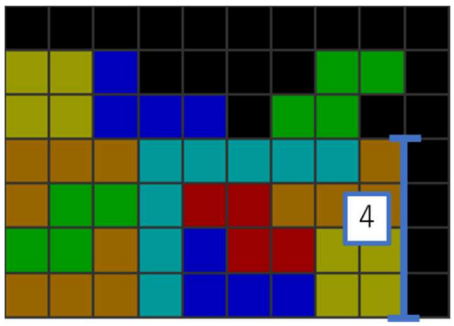

図4　$f_3(x)$が4となる盤面xの一例

$f_4(x)$：水平方向の変動

$f_4(x)$は，盤面を水平方向に見ていったときにブロックの有無が変化した回数を返す．「水平方向の変動」が大きいと凸凹した盤面といえるので，テトリミノを置きづらくなってしまう．なお，壁から連続しているブロックの変動をカウントすると壁に沿って置くことを悪く評価してしまうので壁から連続しているブロックはカウントしない．図5は水平方向の変動が12となる盤面の一例である．

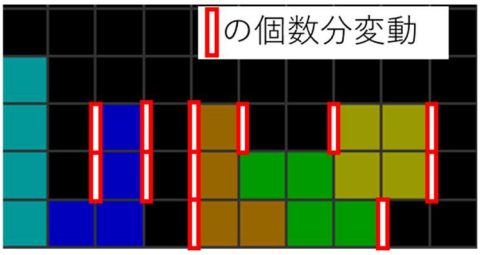

図5　$f_4(x)$が12となる盤面xの一例

$f_5(x)$： 垂直方向の変動

$f_5(x)$は，盤面を垂直方向に見ていったときにブロックの有無が変化した回数を返す．「垂直方向の変動」が大きいと穴の多い盤面といえるので「穴の上のブロックの数$f_2(x)$」の時と同様に有効に使える盤面が狭くなってしまう．この特徴量も「水平方向の変動$f_4(x)$」と同様の理由で地面から連続しているブロックはカウントしない．図6は垂直方向の変動が 12 となる盤面の一例である．

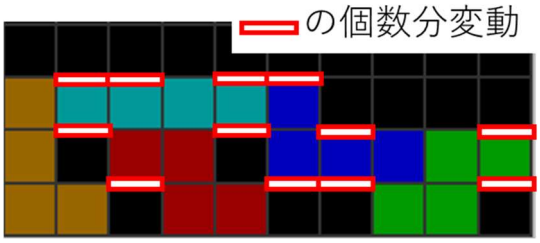

図 6　$f_5(x)$が 12 となる盤面xの一例

$f_6(x)$： 凹凸

$f_6(x)$は，盤面の中で隣り合う列同士の高低差の総数を返す．隣り合う列同士の高低差が大きいとテトリミノを置きづらくなってしまう．図 7 は凹凸が 10 となる盤面の一例である．

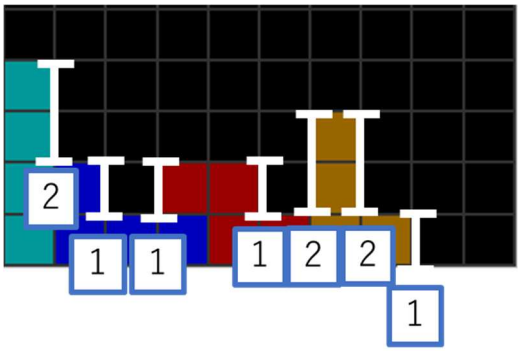

図 7 $f_6(x)$が 10 となる盤面xの一例

$f_7(x)$： 送信ライン数

$f_7(x)$は，操作後に相手に送ることが出来るお邪魔ブロックのライン数を返す．送信ライン数は，表 1 の規則によって自身の消去ライン数に応じて決定される．

表 1　送信ライン数の規則

消去ライン数	送信ライン数
1	0
2	1
3	2
4	4

$f_8(x)$： ホールドボーナス

$f_8(x)$は，I ミノをホールドしている際に 1 を返す．縦方向につながった穴があるときに I ミノを使うと同時に多くのラインを消去することができるので I ミノをホールドしているときにボーナスを付与した．

$f_9(x)$： 操作フレーム

$f_9(x)$は，操作にかかるフレーム数を返す．ミノを動かすときや設置したとき，ラインを消去したときにゲーム内のフレームを消費する．この規則を表 2 に示す．

表 2　操作フレーム

操作	消費フレーム数
移動、回転	1
設置	7
1 ライン消し	35
2 ライン消し	40
3 ライン消し	40
4 ライン消し	45

3.2　重み生成ニューラルネットワーク

重み生成ニューラルネットワーク（重み生成 NN）は，自身と相手の状況を入力とし，自身の盤面評価関数の重みベクトルを出力するニューラルネットワークである．これを用いることで，状況に応じて行動を変化させるテトリス AI が構築できると考えた．

図 8 に重み生成 NN の構造を示す．重み生成 NN は 3 層のニューラルネットワークになっており，入力には自身と相手の状況を用いた．具体的には，特徴量のf_1（最大の高さ），f_3（穴の深さ）と，敵から送られてくる攻撃の情報(予約お邪魔ブロック数)の 3 つを自身だけでなく相手についても用意したので，入力は計6つとなってい

る．f_1はどれだけゲームオーバーしやすいかの情報，f_3はどの程度の攻撃が送れるかの情報，予約お邪魔ブロック数はどの程度の攻撃が送られてくるかの情報になる．入力を増やしすぎると学習が難しくなることからこの 3 つを入力とした．中間層の素子数は 10 とした．出力層には 9 つの素子があり，この出力を自身の盤面評価関数の重みベクトル$w_i(i = 1, \dots, 9)$としている．

NN の学習は次に説明する遺伝的アルゴリズム(GA)によって行った．重み生成 NN は，AI がテトリミノを設置するたびに重みベクトルを出力して自身の盤面評価関数に適用する．

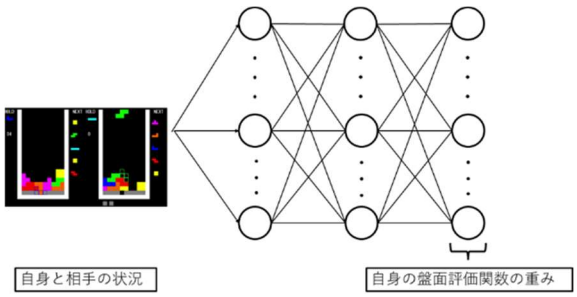

図 8　重み生成 NN の構造

3.3　GA による学習

重み生成 NN の学習には GA を用いた．GA とは生物の進化の過程を模して，交叉，突然変異，淘汰を繰り返し行う最適化手法である．本研究では，個体を重み生成 NN の重みとし，集団サイズは 20 とした．図 9 に GA の流れを示す．

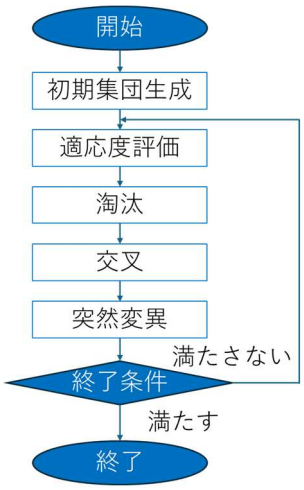

図 9　GA の流れ

まず，初期集団を作成する．本研究では，一般的な GA でも用いられるように各個体の遺伝子は，−1.0~1.0の間でランダムに決定した．

次に淘汰を行うために各個体がどれだけ優れているかを示す適応度を評価する．本研究では各個体を重み生成 NN に適用して，実際に相手 AI と対戦を行ったときの勝率を適応度とした．ここで相手 AI は重み生成 NN を用いずに，ゲームの最初から最後まで同じ盤面評価関数の重みベクトルを用いる AI であり，この重みベクトルは適応度を一定時間での送信ライン数とする GA を用いて，事前に学習したものである．なお，テトリスは降ってくるミノがランダムで決まるため偶然勝利することがある．そのため対戦回数は 20 回とし，その際の勝率を適応度とする．

次に適応度に応じて次の世代に残す個体と残さない個体を選択する．本研究ではエリート選択と呼ばれる適応度が高い個体を残す方法で淘汰を行い，14 個体を次世代に引き継いだ．

次に行う交叉とは，現世代の個体を用いて次世代の個体を生み出す操作である．本研究では$blx-\alpha$交叉と呼ばれる方法で交叉を行った．一つの遺伝子に対する$blx-\alpha$交叉の例を図 10 に示す．この例では，親 1 の遺伝子が−1，親 2 の遺伝子が 3 でαは 0.5 としている．$blx-\alpha$交叉は 2 つの親から 1 つの子を生成する方法で，各親の各遺伝子をもとに子の遺伝子を決定する．両親の遺伝子の値を大きい方からx, yとすると，子の遺伝子は$y-\alpha(x-y)$から$x+\alpha(x-y)$の間の値をランダムに取る．この操作を全遺伝子について行うことで新たな個体を生成する．

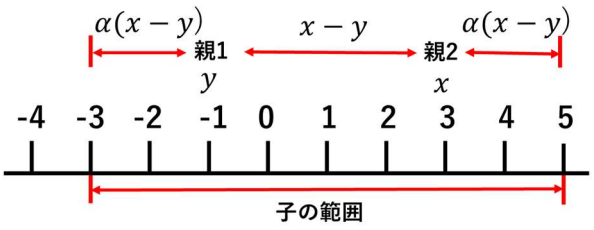

図 10　$blx-\alpha$交叉の例

突然変異とは，ある確率で個体の一部を変更することであり，局所最適解からの脱出を期待できる．本研究では，0.2 の確率で突然変異する個体を決定し，その個体の遺伝子それぞれを（1 −勝率）の確率で−1.0~1.0の間のランダ

ムな値に変更するように設定した．

3.4　提案手法 1

前述の方法で 1500 世代まで学習させた．最良個体の勝率の推移を図 11 に示す．

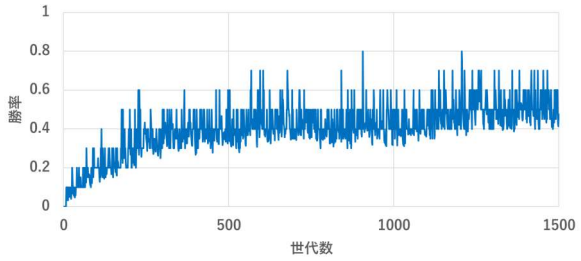

図 11　提案手法 1 の学習推移

より厳密に勝率を測るために，1500 世代時点の最良個体を用いて相手 AI と 100 試合した結果，勝率は 35%であった．

3.5　提案手法 2

提案手法 1 の方法では学習の最初から強い AI と対戦を行って適応度を計測している．この方法では，各個体の適応度に差ができにくく，良い個体であっても淘汰されてしまう可能性があると考えた．そこで，提案手法 2 では相手 AI の盤面評価関数の重みに対し正規乱数を用いて，段階的に相手を強くすることでこの問題の解決を試みた．具体的には，平均値を相手 AI の重みベクトルとする正規乱数で，標準偏差を学習が進むに連れて小さくしていった．学習の流れを図 12 に示す．

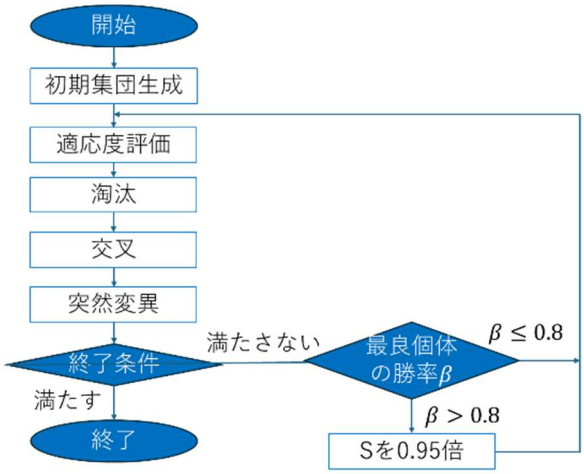

図 12　提案手法 2 の学習方法

提案手法 1 との相違点は，評価の際に相手 AI の重みベクトルに正規乱数を用いている点である．相手 AI の重みは，−10000~10000の値をとるので学習の最初は標準偏差 S を 1000 とし，各世代で最良個体の勝率βが 80%を超えるたびに 0.95 倍した．これにより，学習が進むに連れて相手 AI は段階的に強くなっていく．提案手法 2 の学習時の最良個体の勝率(左軸)と，標準偏差(右軸)の推移を図 13 に示す．

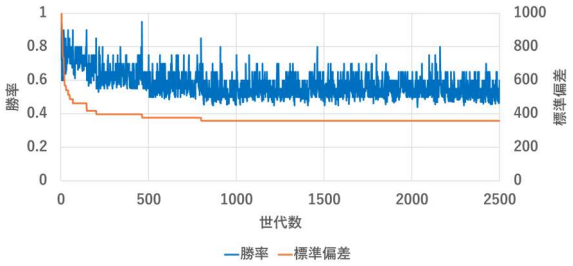

図 13　提案手法 2 の学習推移

学習の序盤は，順調に標準偏差が小さくなっているが，800 世代ほどで標準偏差が 358 に到達し，以降は勝率が 80%を超えることなく停滞してしまっている．なお，2500 世代の学習には 479.1 時間かかった．2500 世代時点の最良個体を用いて相手 AI(S=0)と 100 試合した結果，勝率は 8%であった．

3.6　提案手法 3

提案手法 2 では，勝率が 8%となったがこれは局所解に陥り学習が停滞してしまったためだと考えられる．学習回数を増やすことで局所解からの脱出を期待できるが，学習に非常に時間がかかってしまう．そこで提案手法 3 では 1 世代あたりの試合数を半分の 10 試合に減らし，標準偏差を下げる基準に一つ前の最良個体の勝率と現世代の最良個体の勝率の平均を用いることで，評価の精度を保ちながら学習時間の短縮を試みた．提案手法 3 の学習時の最良個体の勝率(左軸)と，標準偏差(右軸)の推移を図 14 に示す．

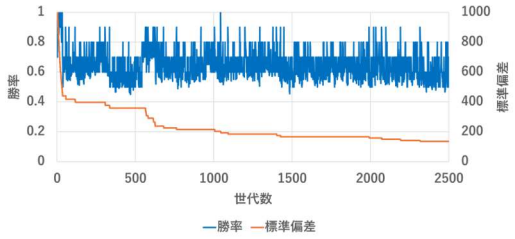

図 14　提案手法 3 の学習推移

提案手法2に比べて，標準偏差が順調に小さくなっているが，1000 世代付近からは，勝率が 80%を超えるのに多くの世代を要するようになっている．2500 世代到達時点で標準偏差は，135 であった．学習の所要時間は，291.1 時間であり，学習時間を約 0.6 倍に短縮することができた．2500 世代時点の最良個体を用いて相手 AI(S=0)と 100 試合した結果，勝率は 24%であった．

3.7　提案手法 4

提案手法3では，世代が進むに連れて学習の進みが遅くなった．これは，個体の類似度が高くなり，新しい解が生まれにくくなったためだと考えられる．提案手法4では標準偏差が小さくなるたびに，適応度が上位の2個体のみを残して他の個体を−1.0~1.0の間でランダムに初期化した．これにより，提案手法 3 の問題が解決できると考えた．提案手法 4 の学習時の最良個体の勝率(左軸)と，標準偏差の推移(右軸)を図 15 に示す．

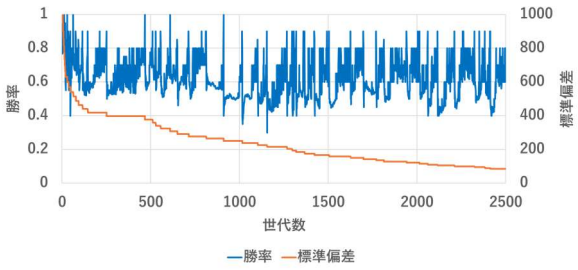

図 15　提案手法 4 の学習推移

提案手法3と比べて，世代数が進んでも標準偏差が下がる間隔は大きくならなかった．2500 世代到達時点で標準偏差は 85 であった．2500 世代時点の最良個体を用いて相手 AI(S=0)と 100 試合した結果，勝率は 37%であった．

ここで，提案手法4の学習がうまく行っているのか判断するために，500 世代ごとにその時点での最良個体と標準偏差 0～300 の相手で 100 試合ずつ対戦を行った．この結果を図 16 に示す．なお，点線は学習時に到達していた標準偏差を表している．図 15 の 1500 世代時点を見ると，学習時には標準偏差 200 の相手に対して，勝率が 80%を超えた上で，より強い相手である標準偏差 200 未満の相手と対戦して学習していたのにもかかわらず，図 16 の 1500 世代時点を見ると 100 試合した時の勝率は 50%ほどになってしまっていることがわかる．

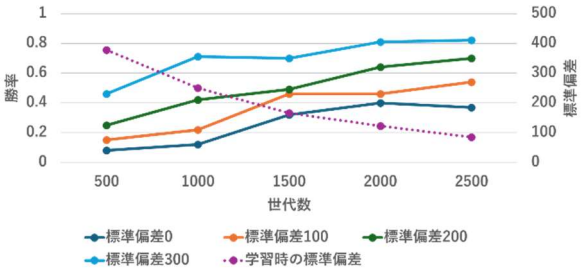

図 16　提案手法 4 の 500 世代ごとに
100 試合した時の勝率

3.8　提案手法 5

提案手法 4 では，学習時に 10 試合あたりの勝率が 80%を超えていたのにも関わらず，学習後に 100 試合行いより厳密に勝率を測ってみると 50%程度であったという問題があった．これは，テトリスのランダム性や，相手 AI の重みに正規乱数を用いていることにより，偶然勝率が高くなってしまうことがあるためだと考えた．そこで，提案手法 5 では 1 世代あたりの試合数を 20 に増やしたうえで，標準偏差を下げる基準に現世代と前世代の各最良個体の勝率の平均を用いた．これで，より厳密に学習中の個体の評価ができると考えた．提案手法5の学習時の最良個体の勝率(左軸)と，標準偏差(右軸)の推移を図 17 に，500 世代ごとにその時点での最良個体と標準偏差 0～300 の相手で 100 試合ずつ対戦した結果を図 18 に示す．

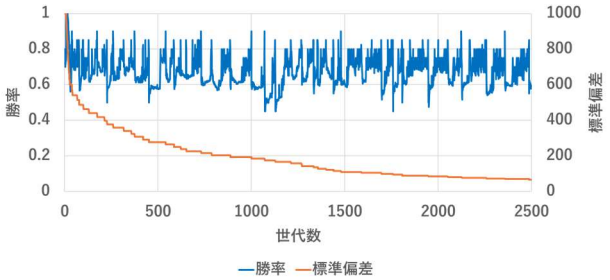

図 17　提案手法 5 の学習推移

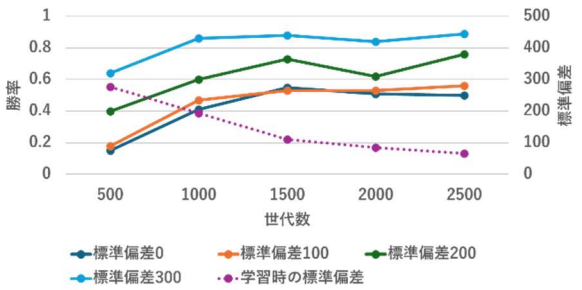

図 18　提案手法 5 の 500 世代ごとに
100 試合した時の勝率

提案手法 5 では, 2500 世代到達時点で標準偏差は 65 であった. 2500 世代時点の最良個体を用いて相手 AI(S=0)と 100 試合した結果, 勝率は 50%であった. 図 17 の 1000 世代時点を見ると, 学習時には標準偏差が 200 の相手に対して, 勝率が 80%を超えているが, 図 18 の 1000 世代時点を見ると 100 試合したときの勝率は 60%程になってしまっていることがわかる.

3.9　各提案手法の結果まとめ

各提案手法で最終世代の最良個体を用いて, 相手 AI(S=0)と 100 試合した際の勝率を表 3 に示す.

表 3　各手法の結果

提案手法	勝率
提案手法 1	35%
提案手法 2	8%
提案手法 3	24%
提案手法 4	37%
提案手法 5	50%

4.　考察

4.1　学習時と学習後の勝率について

図 16 と図 18 からわかるように学習時に勝率が 80%を超えている相手に対しても, 同じ個体を用いてより厳密に勝率を計測すると 80%を大きく下回ることがあった. これは, テトリスのランダム性や, 相手 AI の重みに正規乱数を用いていることにより, 勝率が 80%を超える実力がなくても, 偶然勝率が 80%を超える個体が現れてしまっているからだと考えた. ここで, 学習の際の勝率の変化を見てみる. 例として, 提案手法 4 の 500 世代から 700 世代の標準偏差と勝率の推移を図 19 に示す.

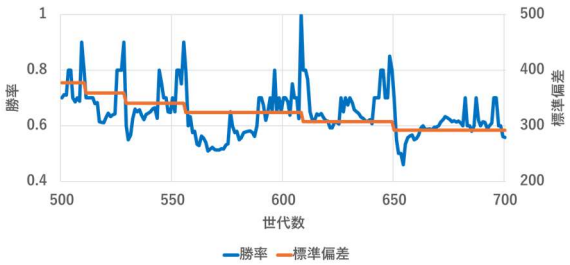

図 19　提案手法 4 の 500 世代から
700 世代までの学習の推移

これを見ると標準偏差が下がるたびに勝率が 60%以下に大きく落ちてしまっていることがわかる. この段階では標準偏差はおよそ 20 ずつ小さくなっており, 標準偏差 20 の違いで勝率が 20%ほど変わってしまっていることになる. 実際に, 標準偏差 20 の差でどの程度勝率が変わるか検証するために, 提案手法 4 の 555 世代時点の最良個体を用いて標準偏差を 320 および 340 とする相手と, 1000 試合ずつ行った. 標準偏差 320 に対しては勝率が 55.8%で標準偏差 340 の相手に対しては, 勝率が 61.4%であった. この結果により, 標準偏差が 20 下がるぐらいでは勝率が 20%も変わるほど強さに差がないことがわかる. 以上のことより, 対戦回数 10 回では偶然勝率が 80%を超えることが多々あったことが推測できる. また, ある個体の厳密な勝率を 60%と仮定すると, 10 試合したときその個体が A 回以上勝つ確率 P(A)は,

$$P(A) = \sum_{k=A}^{10} \binom{10}{k} (0.6)^k (0.4)^{10-k}$$

と表せ, P(8)は 0.1673 と求められる. これは十分に起こりうる確率だと言える. さらに, GA では, 学習が進むに連れて個体の類似度が高くなっていく. 仮に, 20 個体中 10 個体の厳密な勝率が 60%になったとすると, そのうち 1 個体以上が 8 回以上勝つ確率は

$$1 - \{1 - P(8)\}^{10} \approx 0.84$$

となり, 非常に起こりやすいことだと言える. 世代が進む

に連れて，勝率が上がっていっているのは，学習がうまくいっている以外にもそのような理由が考えられ，これによって実際に勝率が 80%を超えていなくても学習が進んでいったのだと考えられる．

4.2　学習方法の検討

前節から学習の際に偶然勝率が大きくなってしまうことがあると分かった．これを防ぐ方法はいくつか考えられる．1 つ目は，標準偏差を下げる基準の勝率を高く設定することである．仮に 90%と設定すると，P(9)は 0.046 となり十分に小さくなる．2 つ目は，勝率の平均をとる世代数を多くすることである．これを多くすることで，数世代にわたり連続で勝率が大きくならないといけないため，偶然の確率は小さくなる．3 つ目は，集団サイズを小さくすることである．前節でも述べたようにGAでは学習が進むと，個体の類似度が高くなり，どの個体もある程度の勝率になる．最良個体の勝率を参照するので，偶然 1 個体でも勝率が高くなれば良い．個体数が多いとこれが起こりやすくなってしまうため，GA の性能を落とさない程度に小さくすることが必要である．

5.　今後の課題

本研究で不十分であった点を今後の課題として 4 つ提案する．1つ目は，重み生成NNへの入力の吟味である．本研究では，筆者自身の考えをもとに6つの入力を決定したが，ほかの有効な特徴を入力に用いることで性能の向上が期待できる．2 つ目は，学習方法の改善である．これは前章で述べたような方法を用いることで，改善できると考えられる．3 つ目は，試行回数を増やすことである．実行時間の関係で各手法 1 度ずつしか試行できなかったので，試行回数を増やし平均的な性能をもってモデルを評価する必要がある．4 つ目は，対戦時の入力と出力を計測することである．これにより，どのような状況でどのように行動を変化させればよいか検討することができ，実際の対戦型テトリスに生かすこともできると考えられる．

謝辞

本研究を進めるにあたり，熱心なご指導と適切なご助言を賜りました修士論文指導教員の平原誠准教授に心から感謝の意を表します．

参考文献

[1] 宮崎真奈美，荒川正幹，“ニューラルネットワークと遺伝的アルゴリズムを用いたテトリスコントローラの開発”，情報処理学会第 74 回全国大会講演論文集，pp.539–540, 2012.

[2] Amine Boumaza, “On the evolution of artificial Tetris players”, The IEEE Symposium on Computational Intelligence and Games, pp. 387-393, 2009.

[3] Christophe Thiery, Bruno Scherrer, “Building Controllers for Tetris”, International Computer Games Association Journal, vol.32, pp.3-11, 2009.

[4] 瀧澤和也，平原誠，”相手の盤面状況を考慮する対戦型テトリス AI”，日本知能情報ファジィ学会第 80 回知的システム研究会, pp.35-37, 2021.