# モンテカルロ木探索に基づいたトレーディングカードゲームプレイヤー

Nakamura, Kyo / 中村, 杏

(出版者 / Publisher)
法政大学大学院情報科学研究科
(雑誌名 / Journal or Publication Title)
法政大学大学院紀要. 情報科学研究科編 / 法政大学大学院紀要. 情報科学研究科編
(巻 / Volume)
9
(開始ページ / Start Page)
125
(終了ページ / End Page)
130
(発行年 / Year)
2014-03
(URL)
https://doi.org/10.15002/00010533

# モンテカルロ木探索に基づいた トレーディングカードゲームプレイヤー

# Monte-Carlo Tree Search based Player for the Trading Card Game

中村　杏
Kyo Nakamura
法政大学大学院
情報科学研究科情報科学専攻
*E-mail: kyo.nakamura.5h@cis.hosei.ac.jp*

## Abstract

*Artificial intelligence (AI) games have been topic for a long time, even now. However, implementation techniques of AI games for two players: trading card game such as Magic the Gathering, have not been established still. Also, it is hard to say accuracy of the computer player named CPU can reach a high accuracy. I think the trading card games have three problems. First, it's incomplete information game. Second, it's difficult to make the evaluation function, because history is shallow and tactics is complex. Third, it has larger state space than the existing game studied because such as the type of card is enormous. Therefore, this research proposes the implementation of CPU to incorporate the Monte Carlo tree search in trading card games. As mentioned earlier, Monte Carlo tree search is an effective method in a game that is hard to make the evaluation function. By performing the compression of the game attributes, to reduce the state space size, the CPU learns from the data of the competition game results, improves the search speed of the Monte-Carlo Tree Search, and reduces the CPU's think time. As a result, off against the rule-based type CPU and Monte Carlo tree search based CPU, a result of the 2000 races, the winning rate of the Monte Carlo tree search based CPU is 61.7%.*

## 1. 序論

近年，既存のゲームについては徐々にゲーム AI の手法が確立されつつある．しかしながら，Magic the Gathering などに代表される，各自で構築したカードの束，デッキを持ち寄って対戦するゲーム，通称 2 人対戦型トレーディングカードゲームのゲーム AI についての研究は非常に少なく，またそのコンピュータープレイヤー（CPU）の行動精度も高いとは言いがたい．

トレーディングカードゲームはゲーム AI の点から主に 3 つの課題を抱えていると私は考えている．1 つ目に，お互いに持っている情報が全て開示されていない，不完全情報ゲームであること．2 つ目に，日々カードの種類が増え，かつ歴史が浅いため，評価関数が作りづらいこと．最後にカードの種類が膨大であることなどから，状態空間が既存に研究されてきたゲームよりも更に膨大であること．この 3 つである．

これらの課題を解決するため，モンテカルロ木探索を取り入れた CPU の実装を提案する．モンテカルロ木探索はシミュレーションの勝敗を思考の指標として利用するため，評価関数が作りにくいゲームにおいて有効であり，囲碁に応用した研究[1]だけで無く，将棋や麻雀と言ったゲームの CPU においてもその有用性が確かめられつつある探索法である．また，ゲーム属性の圧縮を行うことで，膨大な状態空間への対策を図る．さらに，ゲームの対戦結果のデータを CPU が学習することで，モンテカルロ木探索の探索速度の向上を図り，CPU の思考時間を削減する．

本研究では市販されているようなトレーディングカードゲームに近い，オリジナルの 2 人対戦型トレーディングカードゲームを実装し，その CPU として，モンテカルロ木探索をベースとした CPU を実装する．その後，基本的な CPU として，ルールベース型の CPU と対戦させ，その勝率の推移を見ることで，その有用性を確かめる．

## 2. 関連研究

本節では，まず，2 プレイヤー零和完全情報ゲームにおいて有効である Minimax 法を不確定不完全情報ゲームに応用した*-Minimax 探索について紹介し，次に本研究において用いるモンテカルロ木探索について紹介する．

### 2.1 *-Minimax 探索

不完全情報ゲームのようなノード間の遷移が確率的であるような木で Minimax 探索を行う手法として，Ballard は*-Minimax 探索という手法を提案した[2]．Minimax 木での Min ノードと Max ノードに対して，子ノードへの遷移が確率的であるようなノードを Chance ノードと呼び，Chance ノードを含むようなゲーム木において，Alpha-Beta 探索のような枝刈り探索手法を実現するアルゴリズムである．

近年になってバックギャモンのプレイヤーにそのアルゴリズムを適用する研究が行われ，枝刈りを行わない場合に対して最大で 95%の探索時間を軽減したと報告されている[3]．

### 2.2 モンテカルロ木探索

モンテカルロ木探索において有名なアルゴリズムとして UCT（UCB applied to Trees）[4]がある．これは多腕バンディット問題[5]における効率的な方策である UCB1 を

---

Supervisor: Runhe Huang, Professor

木探索に応用したもので，2006 年に Kocsis らによって提案された．

多腕バンディット問題とは，スロットマシンが複数台あったときにどのスロットマシンが一番報酬が多いのかを探るギャンブラーのモデルである．この問題を解く手法の一つとして UCB（Upper Confidence Bounds）という値を用いる UCB1 という手法がある．

UCB は台の番号を i とし，台 i に関する現状の平均収入を $m_i$，台 i をプレイした回数を $n_i$，台 i の出玉の分散（バラツキ）に相当する量を $V_i$ とすると，次の式で UCB を表すことが出来る．

$$UCB_i = m_i + \sqrt{\left(\frac{V_i}{n_i}\right)\log n} \qquad (1)$$

この UCB と言う値は試行回数に応じた報酬の期待値のようなものであり，UCB が一番高い台を回すということは，全ての台を少しずつ試し，結果が良かった台を回すということになる．

そこで UCT は各ノードに対して UCB を計算し，この値が高いノードの枝を伸ばしていくことで有望な手をより詳しく探索していく手法である．

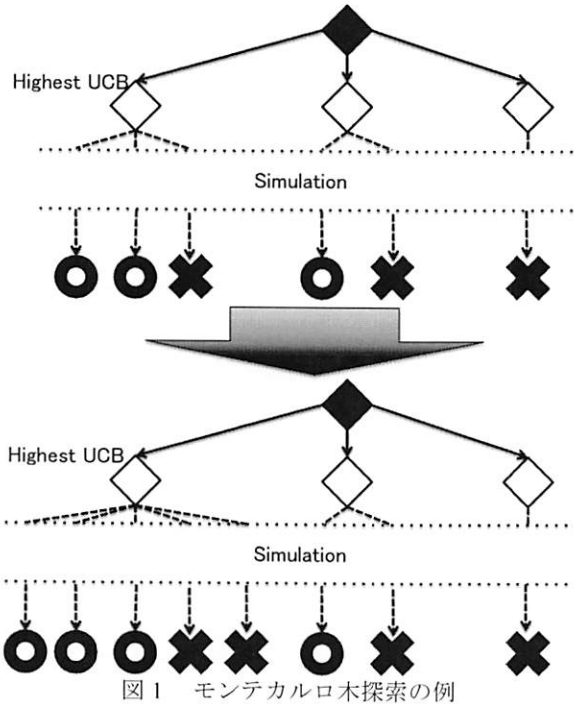

図 1　モンテカルロ木探索の例

上記図 1 において，黒の丸が黒のプレイヤーの勝利であり黒のバツ印が黒のプレイヤーの敗北である．一番左のノードが黒のプレイヤーに対しての UCB 値が高いので，より深く探索していくこととなる．

2008 年 3 月末にこの UCT を使って作られた囲碁の CPU の MoGo が 9 路盤においてプロプレイヤ相手に一勝を収めるという快挙を成し遂げた．

## 3. トレーディングカードゲームのシステム

### 3.1 トレーディングカードゲームの要素

一口にトレーディングカードゲームといってもメーカーにより様々なものがあり，細かいルールの違いは有るものの，分析するとトレーディングカードゲーム特有のゲーム的要素が共通して存在する．

- 2 プレイヤー零和不完全情報ゲーム
- お互いのデッキ（カードの束）は無数のカードから自由に組み合わせ，作ることが出来る
- お互いのプレイヤーが交互にゲームを進めるターン制である
- 自分の手番に複数の行動を組み合わせて 1 手とする

以上に挙げた要素がトレーディングカードゲームに共通する要素である．特にトレーディングカードゲームを特徴付ける要素として，お互いに好きなカードを組み合わせて作ったデッキを使って遊ぶという部分がある．これにより無数のカードによる様々な戦略があり，評価関数を作ることを難しくしている．

本研究ではこれらの要素を持ったオリジナルのトレーディングカードゲームを作成し，実験の題材とする．

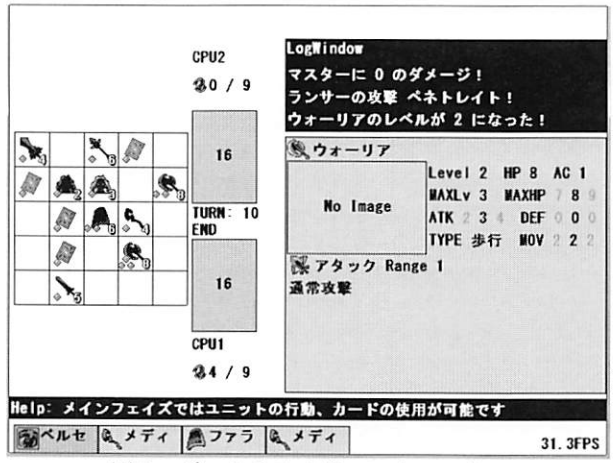

図 2　ゲームのスクリーンショット

### 3.2 トレーディングカードゲームの問題点

本トレーディングカードゲームを CPU 実装の観点から見ると，以下の注目するべき問題点が挙げられる．

- 膨大な状態空間
- 一部の情報がお互いに公開されていない
- 学習に用いる教師が存在しない
- 終局しないと 1 手の価値がわかりづらい
- 自分の手番に数回行動できる
- ランダム手で終局しづらい

以上の点を踏まえ，本研究ではモンテカルロ木探索を用いた CPU を提案する．モンテカルロ木探索では探索によって到達した終局の結果（プレイアウト）を元に現局

面の評価を行うため，評価関数の作りづらさ及び遅れ価値に対して有用性が見込める．

また，膨大な状態空間に対してはゲーム属性の圧縮を試みる．ランダム手で終局しづらい問題に関しては，最低限の評価関数を作成しヒューリスティックに調整すると同時に対戦のリプレイデータ（将棋でいう棋譜）を学習し．学習すればするほど評価関数も成長する強化学習[6]の手法も取り入れる．さらに同様の学習機構をシミュレーションにも適応し，モンテカルロ木探索のランダム手で終局の局面を探索する時の探索効率を上げる．

以上の手法を用いることでトレーディングカードゲームにおいて精度の高い思考を可能とする，モンテカルロ木探索と強化学習を組み合わせたCPUを提案する．

## 4. モンテカルロ木探索CPU実装

### 4.1 全体の流れ

自分のターンになったモンテカルロ木探索CPUは以下の流れに沿って考え，行動する．

i. 自分のターンに行える行動をツリー上に展開したのち，行動の組み合わせをまとめてターンノードとする
ii. ターンノードからモンテカルロ木探索を時間の限り行う
iii. それぞれのターンノードに対してモンテカルロ木探索によるUCB値を算出し，最もUCB値が高かったノードを選ぶ
iv. そのノードの行動の組み合わせを行い，1手とする

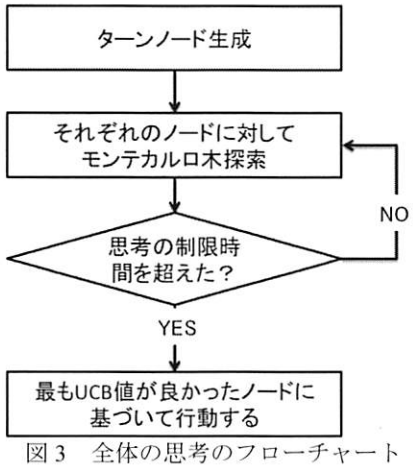


図3　全体の思考のフローチャート

次節より，流れに沿ってそれぞれ説明する．

### 4.2 ターンノード生成

ターンノードとは自分の手番に行える行動の順序を記録した物である．

トレーディングカードゲームにおいては，自分の手番に行える行動は1手のみではなく，盤面の状況が許す限り行動を行うことが出来る．そのため，自分が取れうる行動全ての組み合わせを検討しなくてはならない．

したがって，まず自分の手番になったCPUは現状態から行うことが出来る行動をオーダーノードとして生成する．このオーダーノードは現状態に適応することでゲームのステップが1つ進むこととなる．状態をSとし，ゲームのステップをtと置き，これを数式で表したのが以下の物となる．

$$S_{t+1} = Order(S_t) \tag{2}$$

次にそれぞれのオーダーノードを適応した次状態についても行うことが出来る行動をオーダーノードとして生成し，状態に適応したオーダーノードを親ノード，その状態から生成されたオーダーノードを子ノードとする．ここまでの流れがまとめたのが以下の図4となる．

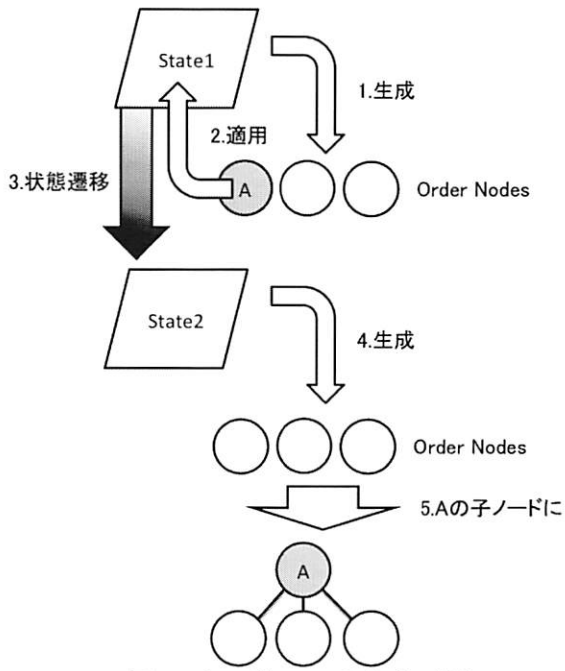


図4　オーダーノード生成の流れ

そして，このサイクルを繰り返し，ツリー上にノードを展開する．このサイクルはオーダーノードが生成されるだけ繰り返し，生成できるオーダーノードがなくなったときに終了する．

ツリー展開の後，それぞれの葉ノードから親ノードをたどってトップノードまで行き，たどった道筋のノードをまとめてターンノードとする．ここまでの流れが以下の図5になる．

また，オーダーノード及びターンノード生成の際にヒューリスティックによる評価関数を用いて，評価点が低いノードは削除することで思考の高速化を図っている．この際に使用される評価関数は，後に述べる強化学習によって，対戦回数を重ねるごとに自動的に更新されていくようになっている．

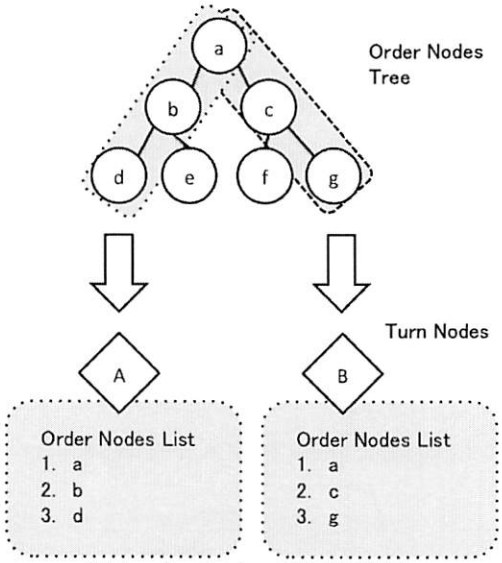

図 5　ターンノード生成の流れ

## 4.3 モンテカルロ木探索

モンテカルロ木探索は以下の流れに沿って進行する．

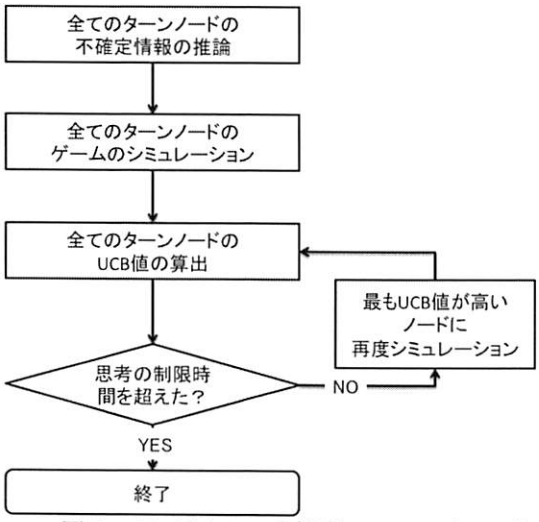

図 6　モンテカルロ木探索のフローチャート

i. 不確定情報の推論
ii. 全てのターンノードに対して 1 回ずつゲームのシミュレーションを行う
iii. それぞれのターンノードについて UCB を算出する
iv. 最も UCB が良い値のノードに対して再度シミュレーションを行う
v. 時間の限り 3 と 4 を繰り返す

### 4.3.1 不完全情報の推論

トレーディングカードゲームのゲーム結果をシミュレーションするためには，お互いの不完全情報を推論する必要がある．したがってまず，このゲームにおける不完全情報である以下の要素を推定する．

- 相手の手札及びデッキの内容
- 自身のデッキの内容

推論はゲームの仕様上あり得る組み合わせを乱数によって幾つか生成し，それらを不完全情報の代わりとすることで行っている．

### 4.3.2 ゲームのシミュレーション

それぞれのターンノードを現状態に適応することで，自分のターン終了時の状態を作り出すことが出来る．そこから相手のターン及び自分のターンの行動を全てランダムに行って，ゲームの決着が付くまでシミュレーションを行う．

ただし，シミュレーションにおける行動の指針を完全にランダムにすると，トレーディングカードゲームでは非情に決着が付きにくくなってしまう．そこで，ここでも評価関数を導入し，行動をランダムに幾つか生成した後，評価関数による篩い分けをすることで，お互いに勝ちに近づける行動を選択できるようにしている．

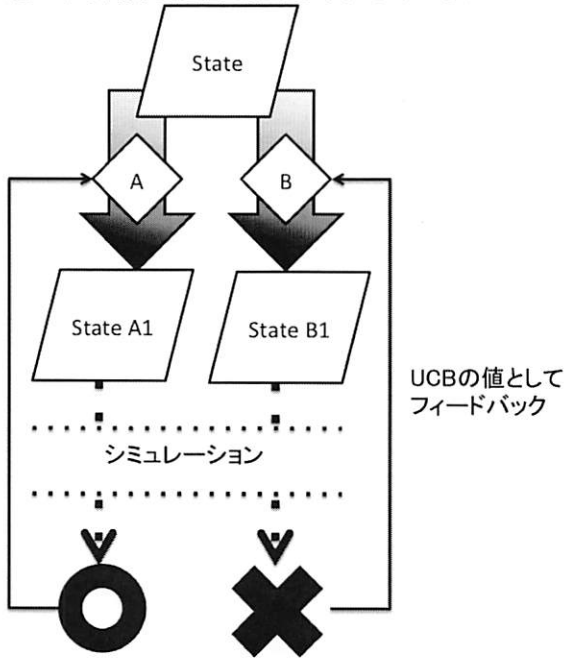

図 7　モンテカルロシミュレーション図解

### 4.3.3 UCB の算出

本ゲームにおける UCB はターンノードの番号を i とし，ターンノード i に関する現状の勝率を $m_i$，ターンノード i のシミュレーション回数を $n_i$ と置くと，以下の数式によって算出される．

$$UCB_i = m_i + \sqrt{\left(\frac{2}{n_i}\right)\log n} \tag{3}$$

上記数式 3 によって算出された UCB 値が最も高かったノードがシミュレーションの結果，勝利への期待値が

最も高いノードであるため，そのノードを更に探索してシミュレーションを重ね，本当に勝てるのかどうかを検証していくこととなる．

### 4.4 学習による効率化

本研究において，CPU の思考時間を削減しつつも精度を高めるために 2 つの学習機構を導入した．

#### 4.4.1 シミュレーション結果の学習

モンテカルロ木探索のシミュレーション結果（プレイアウト）とそのシミュレーションの元になった状態を記憶する．次回以降のシミュレーション時に記憶した状態と同じ状態から始める際はシミュレーションを行わず，前の結果をプレイアウトとして用いることで，他状態のシミュレーションに思考時間を使うことが出来るため，効率と精度の上昇が可能になる．

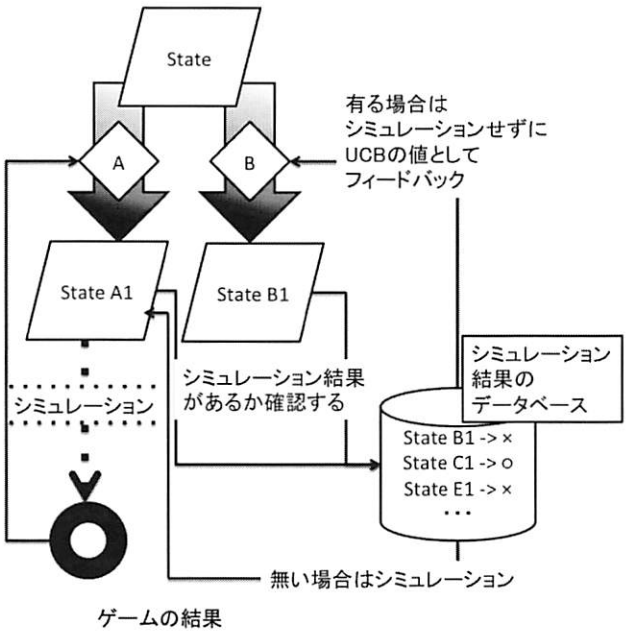


図 8　学習機構図解

さらに実際の対戦結果とその結果に至った過程（リプレイデータ）を記録し，これもシミュレーションの結果として用いることで，更なる効率化を図った．

表 1　ハッシュテーブルの構造

| | 型 | 内容 | |
|---|---|---|---|
| Key | String | 状態を数字の羅列で示した文字列 | |
| Value | int[2] | [0] | 勝利した回数 |
| | | [1] | 探索した回数 |

また，これらの記録にはハッシュテーブルを利用した．キーには状態を数字の羅列で表した文字列データを割り当て，要素にはその状態からのシミュレーションで勝った回数及び探索した回数を整数型の配列で割り当てた．記録の構造にハッシュテーブルを用いることで学習データへの参照の高速化を図っている．

ただし，状態をそのまま記録すると，状態数が膨大なため，記録する際には状態を抽象化し，状態数の圧縮を図った．この抽象化はヒューリスティックに行っており，カードを一枚一枚区別するのではなく，そのカードがゲームにもたらす効果のみで区別することで，記録するカードの種類数を削減した．

また，1 つのターンノードがシミュレーション結果を参照して利用できるのを 1 回に制限し，そのノードから再度シミュレーションを行う必要があった際には，記録から参照せずに通常通りのシミュレーションを行うようにしている．それによって，同じプレイアウトだけを利用しないようにした．

#### 4.4.2 評価関数の強化学習

先のシミュレーション結果の学習でも用いたリプレイデータを評価関数の強化学習に用いることで，行動生成時のふるい分け基準を学習によって更新し，トレーディングカードゲームの評価関数の難しさへの対策を図った．

本研究において，用いた強化学習法は原始モンテカルロ法を応用した物であり，勝利したプレイヤーの行動評価を高く，敗北したプレイヤーの行動評価を低くなるように評価関数を書き換える．評価関数によって得られる行動評価を Q(s, a)で表し（ここで a は状態 s で行う行動である），α を学習による点数とすると以下の式になる．

$$Q(s,a) \leftarrow Q(s,a) + \alpha \tag{4}$$

この式に基づいて書き換えられた評価関数を元に再度対戦を行い，新たなリプレイデータを元に幾度と評価関数を更新することで CPU の思考精度の向上を狙った．このリプレイデータの記録構造は 4.4.1 で用いたハッシュテーブルの構造と似たものとなっており，格納する値の“探索した回数”が“敗北した回数”となっているのが主な違いで，他はほぼ同じ構造となっている．

## 5. 実験と結果

前章にて作成したモンテカルロ木探索 CPU とルールベースのみで作成した CPU を対戦させ，勝率の推移を計測することで，トレーディングカードゲームにおけるモンテカルロ木探索の有用性を検証した．

また，ルールベースに強化学習を組み合わせ，作成した強化学習型 CPU の勝率の推移と比較も行った．実験の条件は以下の通りである．

- お互いに同じデッキを使用する
- デッキのシャッフル及びゲームの先攻後攻はランダムに決定する
- CPU の思考の制限時間は無い
- 対戦回数は 2000 回とする

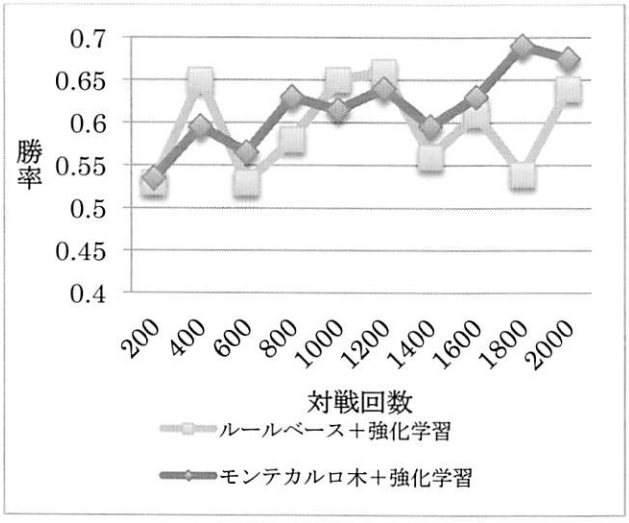


図 9　勝率の推移

表 2　CPU の比較

| CPU の種類 | 2000 戦の平均勝率 |
|---|---|
| モンテカルロ木探索＋強化学習 | 61.7% |
| ルールベース＋強化学習 | 59.5% |

また，実験は以下の実行環境にて行った．

- CPU　Intel Xeon Processor E5520 (2.26 GHz x 2)
- MEM 24GB

上記図 9 はモンテカルロ木探索 CPU と強化学習型 CPU のそれぞれの勝率の推移をグラフで表した物である．縦軸が勝率，横軸が対戦回数となっている．次に，表 2 は今回実装したモンテカルロ木探索 CPU と強化学習型 CPU の対ルールベース CPU 戦を 2000 ゲーム行った際の勝率の比較である．

## 6. 考察

勝率の推移では，モンテカルロ木探索 CPU は最初の 200 ゲームでの勝率が 53%ほどであったのが，2000 ゲーム後には 66%近くであり，ルールベース同士を戦わせた時の勝率を 50%とすると，性能の向上が確認できる．

どちらの CPU にも勝率のぶれが見られるが，これには 2 つの要因が挙げられる．一つ目に試行したゲーム回数が少なく，学習が不十分であること．二つ目にトレーディングカードゲームは運の要素も大きく絡むゲームであること．この二つであると私は考えている．

モンテカルロ木探索 CPU は平均して 61.7%の勝率まで向上することが出来たが，強化学習型 CPU と比較すると勝率においては 2%ほどの上昇にとどまっているため，CPU 自体にもまだまだ改善の余地はあることが分かる．

思考時間については，3 秒前後がターンノード生成部に用いられ，残りがモンテカルロ木探索のシミュレーションに使った時間である．本実験においてはこのシミュレーション部に 15 秒の時間を割り当てた．したがって，本実験のモンテカルロ木探索 CPU の思考時間はおおよそ 20 秒前後となった．

シミュレーションでは，1 秒につき，ゲームの決着までをおおよそ 1 回シミュレートできた．2000 戦の学習によって得られたプレイアウトは実際のゲームの対戦結果から得たプレイアウトを含め，おおよそ 72 万通りであった．学習を重ねるごとに勝率の上昇も見られるため，1 回のシミュレートに時間がかかってしまう場合でも，学習機構を併用することで，トレーディングカードゲームにおけるモンテカルロ木探索に一定の成果が得られることを確認できた．

モンテカルロ木探索はシミュレーション自体には時間がかかるが，今回の実装のようにシミュレーション結果を記録して，シミュレーションの代わりにしたり，またシミュレーション部は並列化が容易なため，並列処理をすることで，さらなる思考時間の短縮と試行精度の向上が見込めると私は考えている．

## 7. 今後の課題

今後の課題としては，ヒューリスティックに調整する部分を出来るだけ少なくすることが挙げられる．

本研究の実装では評価関数部や状態空間の圧縮などにヒューリスティックな調整を行った．しかし，実際のトレーディングカードゲームは実験に用いたゲームよりも更に複雑であり，囲碁と同等かそれ以上にヒューリスティックに評価関数を作ることが難しいだろうと考えられる．

トレーディングカードゲームは日々カードの種類が増えていくため，できる限りヒューリスティックを排除して，自動的にカードを抽象化することが出来るようになれば，カードの種類が増えても，そのままの実装でシミュレーションを回せるため，より汎用性の高い CPU の作成が見込めると私は考えている．

また，本研究の実装ではモンテカルロ木探索のシミュレーションに時間がかかってしまった．現状の思考速度では，実際のゲームに応用するのは困難なため，さらなる思考時間の短縮する方法を模索する必要がある．

## 参　考　文　献

[1] 善添一樹，“モンテカルロ木探索：コンピュータ囲碁に革命を起こした新手法”情報処理，VOL.49, NO6, pp.88-95 , 2008.
[2] BW Ballard. *-Minimax search procedure for trees containing chance nodes. Artificial Intel- ligence, 1983.
[3] T. Hauk, M. Buro, and J. Sch¨afer. Minimax performance in backgammon. Vol. 3846, pp. 51–66, 2004.
[4] L. Kocsis and C. Szepesvari. Bandit based monte-carlo planning. European Conference on Machine Learning, pp. 282–293, 2006.
[5] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. Machine Learning, Vol. 47, No. 2-3, pp. 235–256, 2002.
[6] R.S.Sutton，A.G.Barto，三上貞芳(訳)，皆川雅章(訳)“強化学習”森北出版 (2000)