京都将棋の弱解決

塩田 雅弘[1,a)]　伊藤 毅志[1,b)]

概要：京都将棋は 1976 年に田宮克哉によって考案された二人零和有限確定完全情報ゲームである．5×5 マスの盤を使用し，駒を一手ごとに裏返すというルールが特徴である．本論文では京都将棋エンジンを開発し，本将棋で広く使われている強化学習アルゴリズムを京都将棋に適用することで十分に強いプレイヤを作成した．そしてこのプレイヤを用いて証明木を効率良く発見した．この結果，京都将棋を弱解決し，本ゲームの初期局面が先手必勝であることを示した．

Kyoto Shogi Is Weakly Solved

Masahiro Shioda[1,a)]　Takeshi Ito[1,b)]

Abstract: Kyoto shogi is a two-player zero-sum game invented by Tamiya Katsuya in 1976. It is played on a 5×5 board and each player has to flip the piece moved after each turn, which characterizes this game. In this research, we developed a kyoto shogi engine and then applied the reinforcement learning algorithm that is widely used in standard shogi to kyoto shogi to make a strong player. We found a proof tree effectively using this engine. As a result, we we weakly solved kyoto shogi and showed that the first player wins at the initial position.

1. はじめに

将棋や囲碁などの二人零和有限確定完全情報ゲームでは，すべての局面で解 (先手必勝・後手必勝・引き分け) が存在する．このようなゲームを解決することはゲームプログラミングやゲーム情報学における目標の一つである．ゲームの解決は以下の 3 つのレベルに分類される [1].

(1) 超弱解決：初期局面の勝敗は解明されているが，具体的な手順はわからない．
(2) 弱解決：初期局面の勝敗が解明されていて，プレイヤの手順もわかっている．
(3) 強解決：初期局面から到達可能な局面すべての勝敗と手順がわかっている．

将棋の初期局面から到達可能な局面数は 10^{68}〜10^{69} 程度 [2] であり，これを解決することは現実的でない．将棋には現行のルールを簡略したいくつかの派生ゲームが存在し，そのうちの 1 つであるどうぶつしょうぎは総局面数が少なく，後退解析によって強解決されている [3]．どうぶつしょうぎよりも複雑な将棋類として，京都将棋があり，その総局面数はどうぶつしょうぎよりも十分大きいため後退解析は困難であると考えられる．

一方近年のコンピュータ将棋では，自己対戦棋譜を利用して評価関数を学習する手法が確立し，プレイヤの棋力は大幅に向上している [4]．これらの手法は汎用性があるため，京都将棋にも適用可能である．本論文では，後退解析が難しい京都将棋に対して，十分に強いプレイヤを作成してゲームを弱解決する方法とその結果について記述する．

2. 関連研究

2.1 どうぶつしょうぎの強解決

どうぶつしょうぎは 2008 年に女流棋士の北尾まどか初段によって考案された将棋の派生ゲームである．縦 3 マス横 4 マスの盤面と 4 種類の駒 (ライオン，ぞう，きりん，ひよこ) を使用する，本将棋に比べて非常に簡潔なゲームとなっている．田中は後退解析という手法を用いてどうぶつ

[1] 電気通信大学大学院情報理工学研究科
Graduate School of Informatics and Engineering, The University of Electro-Communications
[a)] shioda@minerva.cs.uec.ac.jp
[b)] taito@mbc.nifty.com

© 2021 Information Processing Society of Japan

しょうぎを強解決し，初期局面から到達可能な約 2.5×10^8 局面すべてのゲーム値を求めた [3]．これにより，初期局面は 78 手で後手勝ちであることがわかっている．

2.2 強化学習アルゴリズム

強いプレイヤを作成するためには評価関数の精度を向上させることが不可欠となる．近年のコンピュータ将棋では，自己対戦棋譜を用いた学習手法が確立したことで，人間の棋譜を用いるよりも遥かに多くの教師局面を利用できるようになりプレイヤの棋力は飛躍的に向上した．この強化学習アルゴリズムでは，まずプレイヤ同士で自己対戦を行い棋譜から教師局面を生成する．探索深さや探索局面数を制限して自己対戦を行うことで大量の教師局面を生成することができる．また棋譜の偏りを減らす目的で，ランダムに指し手を選択したり，自己対戦の開始局面を複数用意するといった工夫がなされることもある．次に先述の自己対戦によって生成された教師局面を用いて，評価関数のパラメータを更新する．パラメータの更新にはミニバッチ勾配降下法 (SGD) が使われる．目的関数には 2 値分類の交差エントロピーを使用する．

$$H(m,q) = -m\log q - (1-m)\log(1-q) \quad (1)$$

$$m = (1-\lambda)t + \lambda p \quad (2)$$

ここで $p \in [0,1]$ は教師の評価値から算出した勝率，$q \in [0,1]$ は静止探索の評価値から算出した勝率，$t \in \{0,1\}$ は自己対戦におけるその局面の勝敗 (勝ちのとき 1，負けのとき 0) である．$\lambda \in [0,1]$ は教師の評価値 p と実際の勝敗 t の重要度を制御するパラメータであり，コンピュータ将棋では瀧澤によって導入された手法である [5]．更新したパラメータを用いて，再度自己対戦による教師局面の生成とパラメータの調整を繰り返すことで評価関数の精度の向上を図る．

3. 京都将棋

京都将棋は 1976 年に田宮克哉によって考案された将棋の派生である．5×5 マスの盤を使用し，図 1 に示す初期局面から互いに 1 手ずつ進めることで進行する．ゲームの目標は，本将棋と同様に相手の玉将を詰ますことである．使用する駒は以下の 5 種類であり，玉将以外の駒は動かすごとに必ず裏返す必要がある．

- 玉将
- 表が香車，裏がと金
- 表が銀将，裏が角行
- 表が金将，裏が桂馬
- 表が飛車，裏が歩兵

持ち駒を打つ場合には，駒の表裏を選択して打つことができる．また本将棋では禁じ手となる二歩，打ち歩詰め，行き所のない駒，連続王手の千日手はいずれも反則とはならない．千日手と連続王手の千日手は共に引き分けとなる．その他のルールは本将棋と同様である．

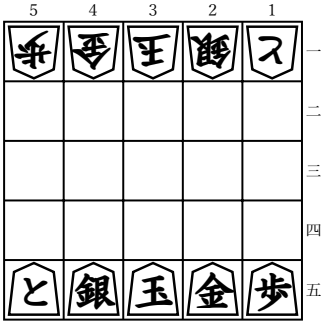

図 1 京都将棋の初期局面
Fig. 1 Initial position of kyoto shogi.

表 1 京都将棋における持ち駒総数ごとの局面数
Table 1 Number of positions per total number of pieces in hand in kyoto shogi

持ち駒総数	局面数
0	48,585,426,075,648,000
1	12,146,356,518,912,000
2	1,339,671,674,880,000
3	84,349,698,048,000
4	3,313,271,808,000
5	83,239,833,600
6	1,311,552,000
7	11,923,200
8	48,600
合計	62,159,201,802,653,400

京都将棋の総局面数を大雑把に見積もるために，ルールを一切考慮せずに可能な駒の配置の組み合わせを考えると表 1 のようになる．この合計値は，どうぶつしょうぎの場合 (約 1.6×10^9) に比べて十分に大きく，京都将棋では後退解析は困難であると考えられる．

4. 解析の概要

本章では京都将棋を弱解決する具体的な方法について述べる．事前に Fairy-Stockfish[*1] を用いて京都将棋を対局させたところ先手番が大きく勝ち越したことから，初期局面は先手必勝であることが予測される．はじめに本論文で使用する証明木を構築するアルゴリズムについて述べ，さらに証明木を効率良く発見するために行ったエンジンの開発と評価関数の学習について説明する．

4.1 証明木

ある局面のゲーム値を証明するのに最低限必要な部分木を証明木と呼ぶ [6]．例えばある局面が先手必勝であることを示したい場合，以降の局面で後手がどのように指し手

[*1] https://github.com/ianfab/Fairy-Stockfish (参照 2021.10.04)

© 2021 Information Processing Society of Japan

Algorithm 1: 先手必勝を示す証明木を構築するアルゴリズム．S_{w} は後手番の局面集合．child(s) は局面 s のすべての子局面を生成する関数．nextMove(s) は局面 s における指し手を 1 つ返す関数．doMove(s, m) は局面 s から指し手 m で進めた局面を返す関数．

```
1  function proofTree(S_w):
2      while S_w が空でない do
3          s_w ← S_w.pop()
4          foreach s_b ∈ child(s_w) do
5              if s_b が引き分けまたは詰み局面 then
6                  return false
7              end
8              move ← nextMove(s_b)
9              s'_w ← doMove(s_b, move)
10             if s'_w が引き分け局面 then
11                 return false
12             end
13             if s'_w で後手玉が詰んでいない then
14                 S_w.push(s'_w)
15             end
16         end
17     end
18     return true
19 end function
```

を選択しても，先手番には少なくとも 1 つの必勝手が存在することを示せばよい．本論文ではこの事実に注目した．

Algorithm 1 は先手必勝を示す証明木を構築するアルゴリズムである．はじめに後手番の局面集合 S_{w} を用意し，S_{w} から局面を 1 つ取り出す (3 行目)．次に取り出した局面 s_{w} に対してすべての子局面を生成する (4 行目)．各子局面 s_{b} からさらに 1 手指し進め (8–9 行目)，指し進めた局面 s'_{w} が終局していない場合はこれを S_{w} に追加する (14 行目)．以上の操作を S_{w} が空になるまで繰り返す．このアルゴリズムが真を返すとき，先手必勝を示す証明木を発見したといえる．

Algorithm 1 において証明木を効率良く見つけるためには，8 行目の関数 nextMove で悪手を選択することは避けなければならない．本論文では，エンジンの開発と評価関数の学習を行い，十分に強いプレイヤを作成することによってこの問題の解決を図る．作成したプレイヤで各局面を思考させ，その最善手を 8 行目の変数 $move$ に代入する．そして proofTree が真を返すとき，後手がどのように指し手を選んでも先手に必勝手が存在するため，先手必勝であることを証明できる．

4.2 エンジンの開発

やねうら王[*2]は第 29 回世界コンピュータ将棋選手権の優勝プログラムであり，最も強い将棋エンジンの 1 つである．本論文では，やねうら王をベースとする京都将棋エンジンの開発を行った[*3]．開発言語は C++であり，探索部はやねうら王に実装されているものを改変せずに使用した．ソースコードの改変を行った箇所を以下に列挙する．

- ビットボード
- 指し手生成
- 局面の更新処理
- 1 手詰め判定 (局面を動かさずに 1 手詰めの存在を判定する関数)

4.3 評価関数の学習

評価関数の構造は，那須によって考案された NNUE 評価関数 [7] を使用した．二駒関係 KP を改変した特徴量を入力とする全結合ニューラルネットワークであり，中間層は $256 \times 2, 32, 32$ ノードからなる．ネットワークの層が浅く CPU で高速に動作する点が特長である．

2.2 節の手法は汎用性があるため，やねうら王に実装されている NNUE とその学習部を京都将棋用に変更したものを用いてこの手法を適用した．評価関数パラメータのすべての値を 0 とした状態から学習を 30 回実施した．自己対戦時の探索深さは 9 とし，ハイパーパラメータは 1 回あたりの生成する教師局面数を 10^7，ミニバッチサイズを 10^5，学習率を 0.1 とした．式 (2) の λ の値は，学習 1–5 回目は 0，6–10 回目は 0.1，11–20 回目は 0.33，21 回目以降は 0.5 とした．

5. 結果

初期局面から☗ 1 四歩成と進めた局面 (図 2) を Algorithm 1 の集合 S_{w} に与えた．図 2 を根節点とする証明木を発見することができれば，初期局面は☗ 1 四歩成で先手必勝であることが示せる．Algorithm 1 中の 8 行目では，開発エンジンおよび学習した評価関数を用いて局面を思考させ，探索結果から求まる最善手を選択した．エンジン探索部では前向き枝刈り手法やハッシュ表を用いているが，指し手の選択にのみ探索結果を利用するのであればアルゴリズムの正しさに影響は及ぼさない．

メモリ 32GB の Ryzen9 5950X マシン 1 台を用いたところ，およそ 11 時間で証明木を発見することに成功した．これにより，京都将棋は先手必勝のゲームであることが示された．証明木の総節点数は 54,728,872 個であり，そのうち終端節点 (合法手が 0 の局面) は 18,443,888 個であった．また終局までの最長手数は初期局面から数えて 57 手であり，この手順は付録 A.1 に記載した．

本論文では初手☗ 1 四歩成以外の指し手を調べていないほか，手順中の先手の指し手が常に最善である保証もな

[*2] https://github.com/yaneurao/YaneuraOu (参照 2021.10.04)

[*3] ソースコードは https://github.com/msioo/kyotoshogi で公開している

© 2021 Information Processing Society of Japan

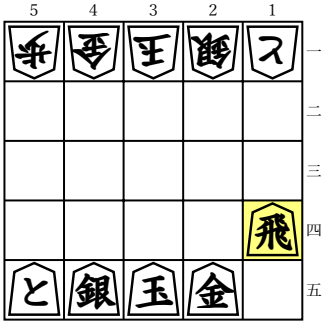

図 2　初期局面で 1 五の歩兵を 1 四に動かした局面 (後手番)
Fig. 2　Position after move from 15-pawn to 14 at the initial position.

いため，最短手順は未だ解明されていないことに留意されたい．

6. まとめと課題

本論文では京都将棋の弱解決を行った．具体的には，やねうら王をベースとした京都将棋エンジンを開発し，強化学習の手法を適用して評価関数の学習を行うことで十分に強いプレイヤを作成した．そしてこのプレイヤを利用することで，初期局面で先手が必勝となる手順を効率良く発見した．これにより京都将棋が先手必勝であることを示した．本論文の手法は作成したプレイヤの探索結果に依存して証明木を構築しているため，指し手の選択を誤ると証明木の発見に必要以上の時間を要する場合や発見に失敗する場合が想定され，これらは今後の課題である．

参考文献

[1] Allis, L. V.: Searching for solutions in games and artificial intelligence, PhD Thesis, Univ. of Limburg (1994).
[2] 篠田正人：将棋における実現可能局面数について，ゲームプログラミングワークショップ 2008 論文集，Vol. 2008, No. 11, pp. 116–119 (2008).
[3] 田中哲朗：「どうぶつしょうぎ」 の完全解析， Vol. 2009-GI-22, No. 3, pp. 1–8 (2009).
[4] Kaneko, T. and Takizawa, T.: Computer Shogi Tournaments and Techniques, *IEEE Transactions on Games*, Vol. 11, No. 3, pp. 267–274 (2019).
[5] 瀧澤　誠，伊藤毅志：進化し続けるコンピュータ将棋：2. elmo の開発と技術-第 27 回世界コンピュータ将棋選手権優勝プログラムインタビューから，情報処理，Vol. 59, No. 2, pp. 153–156 (2018).
[6] 岸本章宏：チェッカー解明秘話，情報処理，Vol. 48, No. 11, pp. 1257–1263 (2007).
[7] 那須　悠：高速に差分計算可能なニューラルネットワーク型将棋評価関数，`https://github.com/ynasu87/nnue/blob/master/docs/nnue.pdf` (2018).

付　　録

A.1 証明木

本論文で発見した先手の必勝手順の一部を示す．図 2 の後手の各合法手に対する先手の応手は以下の通りである．カッコ内の数字は初期局面から終局に至るまでの手数を示している．

☖ **1** 二と成	☗ 4 四と成 (45 手)
☖ **2** 二と成	☗ 4 四と成 (57 手) ∗ 手順 (a)
☖ **1** 二銀成	☗ 4 四銀成 (23 手)
☖ **2** 二銀成	☗ 4 四銀成 (57 手) ∗ 手順 (b)
☖ **3** 二銀成	☗ 1 一飛成 (23 手)
☖ **2** 二玉	☗ 4 四銀成 (21 手)
☖ **3** 二玉	☗ 1 一飛成 (29 手)
☖ **4** 二玉	☗ 4 四と成 (21 手)
☖ **3** 二金成	☗ 5 四と成 (25 手)
☖ **4** 二金成	☗ 1 一飛成 (23 手)
☖ **5** 二金成	☗ 1 一飛成 (23 手)
☖ **5** 二歩成	☗ 1 一飛成 (55 手)

手順 (a). 初手から

☗ 1 四歩成	☖ 2 二と成	☗ 4 四と成	☖ 3 二銀成
☗ 3 四金成	☖ 5 二歩成	☗ 2 二桂成	☖同　玉
☗ 2 五香	☖ 3 一玉	☗ 1 二飛成	☖ 2 一金
☗ 3 四銀成	☖ 1 二金成	☗同角成	☖ 4 二金成
☗ 2 二金	☖ 4 一玉	☗ 3 二金成	☖ 3 四飛
☗ 4 五玉	☖ 4 四飛成	☗同　玉	☖ 5 四桂成
☗ 3 四玉	☖ 4 二飛成	☗ 2 三香成	☖ 5 二玉
☗ 2 四角	☖ 5 三玉	☗ 1 一銀成	☖ 4 三歩成
☗ 5 一飛	☖ 5 二と	☗同飛成	☖同　玉
☗ 4 四と	☖同金成	☗同角成	☖ 3 三歩
☗同角成	☖同飛成	☗同　玉	☖ 1 一角
☗ 3 四玉	☖ 3 三と	☗同銀成	☖同角成
☗同　玉	☖ 4 二銀	☗ 3 四玉	☖ 5 一玉
☗ 2 一飛	☖ 5 二玉	☗ 5 四香	☖ 5 三銀成
☗ 4 三銀	まで 57 手で先手の勝ち		

手順 (b). 初手から

☗ 1 四歩成	☖ 2 二銀成	☗ 4 四銀成	☖ 2 一と成
☗ 2 二角成	☖同香成	☗ 4 五と成	☖ 5 二歩成
☗ 3 四金成	☖ 2 一と成	☗ 4 一香成	☖同　玉
☗ 3 三桂	☖ 3 二玉	☗ 2 一桂成	☖ 5 三角
☗ 2 五玉	☖ 4 三玉	☗ 4 一角	☖ 3 二飛成
☗ 4 五香	☖ 5 四玉	☗ 3 二角成	☖ 4 五玉
☗ 4 三飛	☖ 4 四と	☗ 5 三飛成	☖ 3 三と
☗ 2 三角打	☖ 3 二と成	☗同角成	☖ 3 四と成
☗同飛成	☖ 1 三角	☗ 2 四と	☖ 4 二金
☗ 1 三と成	☖ 3 二金成	☗ 3 五と	☖ 5 四玉
☗ 5 二歩成	☖ 4 三玉	☗ 3 二飛成	☖ 5 三角
☗ 2 二角	☖ 3 二玉	☗ 3 三歩成	☖ 4 二玉
☗ 4 五と成	☖ 4 四角成	☗同香成	☖ 5 五飛
☗ 4 五と成	☖同飛成	☗ 3 一銀	☖ 5 二玉
☗ 4 二金	まで 57 手で先手の勝ち		

© 2021 Information Processing Society of Japan