修　　士　　論　　文

Arimaa 評価関数における比較学習

Comparison Training of Arimaa Evaluation Functions

指導教員　　**鶴岡 慶雅 准教授**

東京大学工学系研究科
電気系工学専攻

氏 名　　**37-146434 川上裕生**

提 出 日　　**平成 28 年 2 月 4 日**

概 要

ゲーム AI の研究は人工知能研究の一分野として古くから行われており、本稿では Arimaa というゲームを対象とした。Arimaa は 2002 年にコンピュータにとって難しくなるようなゲームとして開発されたゲームである。Arimaa の最大の特徴として、1 度の手番に駒を 4 回（4 ステップ分）動かせるため各ターンの合法手の数が非常に多いということがあげられる。そのため Arimaa では探索を行なうコストが非常に高くなってしまう。Arimaa の AI の多くはアルファ・ベータ探索と評価関数を利用して指し手を決定しており、評価関数の精度を向上することができれば深い探索を行なうことが難しい Arimaa においても強いプレイヤが作成できると考えられる。

本稿では、Arimaa の評価関数を比較学習で学習することを目指す。比較学習は、上級者の棋譜を利用し上級者の指し手を真似るようにして学習を行なう手法であり、チェスや将棋においてよい結果を示している。比較学習を行なう際には、簡単な先読みを行なうことで精度が向上することが知られているが、Arimaa において 2 手、3 手先を読むことですら時間がかかってしまうことが問題点としてあげられる。その問題の解決として、駒の取り合いに限定した静止探索を利用することで、高速に先読みを行なうことを試みた。

Arimaa での比較学習を行った結果、静止探索を利用することで学習の効果が大きく高まることが判明し、手調整で作成された評価関数を利用したプログラム “Faerie” に対して 72.1%、静止探索なしで学習した評価関数に対して 71.3%の勝率を残した。静止探索を利用することで、棋譜の指し手の意味をうまく理解して学習することができたと考えられる。

目 次

図目次

表 目 次

第1章　はじめに

1.1　背景

ゲーム AI は人工知能研究の題材のひとつとして古くから研究が行われている。ゲームを研究題材として用いる理由は、ルールが明確であること、勝敗により性能の評価が行い易いこと、人間にとって面白いものであることなどがあげられる。

機械学習を用いてコンピュータゲームの性能を向上させる研究はゲーム AI の研究の初期の頃から行われてきている [2]。機械学習を用いることで、そのゲームに対する深い知識を持たない人でもゲーム AI の作成が可能になり、手作業でヒューリスティックを組み込む必要もなくなるため手間も削減できる。また、機械学習により人間がうまく説明できないような “直感”をコンピュータに理解させることや、まだ人間が理解していないゲームの深い知識を得ることが期待できる。

最初のゲーム AI の研究は Strachey が 1952 年に作成したチェッカープログラムである [3]。チェッカーはコンピュータにとって扱いやすい比較的シンプルなゲームであったが、コンピュータは徐々により複雑なゲームもプレイできるようになっていった。1997 年にはチェスプログラム Deep Blue が当時の世界チャンピオンであったガルリ・カスパロフに勝利を収め、将棋においてもコンピュータが人間のトップに匹敵するとの研究結果がでている [4]。その結果、チェスや将棋などよりもコンピュータにとって難しいゲームの研究が近年では盛んに行われるようになっている。

チェスや将棋などのプログラムでは、現在の局面がどちらのプレイヤがどの程度有利なのか形勢判断を行なうために、評価関数を利用している。評価関数はかつては人手で調整されることが多かったが、プログラムの製作者にそのゲームに対する深い知識が必要となり、評価値の計算に用いられる膨大な数の特徴量を調整する必要があるため、コストが非常に大きかった。そのため、近年では機械学習を用いて自動的にパラメータを調整する手法がよく用いられている。その中でも評価関数を学習する手法の一つである比較学習はチェス、将棋において良い性能を示している。比較学習は上級者の棋譜データさえあれば利用可能であり、他のゲームにおいても利用しやすく、応用することが可能だと考えられる。

本論文では Arimaa というゲームを対象とする。 Arimaa はチェス・将棋・囲碁などと同じ、二人零和有限確定完全情報ゲームに分類されるゲームである。 Arimaa は 2002 年に Omar Syed 氏が、AI にとっては難しいものの、ルール自体は子供でも理解できるというコンセプトで作成したゲームである。

2004 年以降毎年 1 回、 Arimaa チャレンジとして、コンピュータ対人間の対戦が行われている。2015 年 4 月に行われた Arimaa チャレンジにおいて、Arimaa プログラム Sharp が 7 勝 2 敗で初めて人間のトッププレイヤに対して勝利した。しかし、Arimaa は人間にとっても新しいゲームでこれま

でに蓄積されてきた知識が少なく、競技人口も少ないため、チェスや将棋のトッププレイヤよりも Arimaa のトッププレイヤはレベルが低いと考えられるため、人間にも AI にも向上の余地は多く残されていると思われる。また、この Sharp というプログラムは、Arimaa 上級者のアドバイスを利用してムーブオーダリングや評価関数の調整を繰り返すことで実力を向上させたプログラムであり、人工知能や機械学習の研究の対象としての Arimaa はまだ研究する意義があるということができる。

Arimaa が AI にとって難しい理由を説明する。まず一つ目に、一ターンごとの合法手の数が非常に多いことがあげられる。 Arimaa では一ターンに 4 回の移動を行うことができ、そのため一ターンに平均して 16,000 通りの合法手が存在する [5]。類似した他のゲームにおける一ターンの合法手の数の平均は、チェスでは 35 通り、将棋では 80 通り [6]、囲碁では 250 通り程度であり、 Arimaa の合法手の数は桁違いに多いということが分かる。合法手の数が多いことによりゲーム木のサイズが非常に大きくなるため、単純な力任せの探索方法では深い探索を行うことができない。二つ目に、駒の初期配置をプレイヤーが自由に決められるということがあげられる。ゲーム開始時に、先手番のプレイヤーが手前二段に好きなように駒を配置したあとに、後手番のプレイヤーも駒を配置する。この駒の初期配置の方法だけでも 4.2×10^{15} 通り存在し、このゲームを複雑にする一因となっている。また、初期配置が一つに定まっていないため、チェスや将棋などのゲーム AI で行われている定跡の利用が難しくなっている。これらの要因により、 Arimaa はコンピュータにとって難しいゲームになっている。

Arimaa の AI の多くは、チェスや将棋の AI と同様にアルファ・ベータ探索と評価関数を用いて指し手を決定している。しかし、Arimaa では合法手が非常に多いため、チェスや将棋のような深い探索を行なうことが事実上不可能である。そこで、深い探索を行わなくても良い指し手を選択するために、評価関数を改善することを考える。理論上は、完全な評価関数を作成することができれば探索を行わなくても最善手を選択することが可能である。

AI にとって難易度の高いゲームである Arimaa の研究を行なうことにより、今後更に難易度が高いゲーム、果ては現実世界の課題についても解決可能な AI の作成の足掛かりになることが期待される。

1.2 本研究の目的と貢献

本研究では、比較学習 [7] を用いて Arimaa の評価関数を学習することを目指す。比較学習は、上級者の指した指し手のあとの局面の評価値が他の指し手のあとの局面の評価値よりも高くなることを利用し学習を行なう学習手法であり、ある局面が具体的に何点の評価値であるのかというデータは必要としない。そのため、上級者の棋譜のデータさえあれば学習を行なうことができ、大量の教師データを集めることが容易であるという利点がある。チェスや将棋では、比較学習を利用することで有用な評価関数を学習することに成功している [8][9]。そのため、Arimaa においても比較学習をうまく利用することで、有用な評価関数の学習が可能なのではないかと考えられる。

Arimaa における比較学習の問題点の一つに、Arimaa 上級者のレベルがチェスや将棋の上級者のレベルよりも低く、学習に利用できる棋譜の質が低い点がある。人間のトッププレイヤに勝つことを目指す際には、Arimaa 上級者のレベルが低いということはコンピュータ側にとって有利な条件にもなるが、棋譜を利用した学習を行なう際には学習時に棋譜中に登場する指し手が最善手であるとは

限らない局面が多くなることを意味しており、うまく学習ができなくなる可能性がある。将棋において、金子の研究 [10] では、プロ棋士の棋譜、コンピュータ将棋の棋譜、アマチュアの棋譜をそれぞれ教師データとして利用したときの違いが考察されている。棋譜の質が低いアマチュアの棋譜を利用して学習を行った場合でも、プロ棋士の棋譜を利用して学習を行ったものには及ばないものの、有用な評価関数を学習することができており、棋譜の質がそれほど高くない場合であっても、比較学習を利用することは可能であると考えられる。

また、Arimaa では各ターンの合法手が非常に多いため、探索を行なうことのコストが非常に高いという問題点がある。探索のコストが非常に高く、学習時に探索を行なうことが難しい Arimaa において、評価関数の精度を高めることができるのかについて調査した。

1.3 本論文の構成

第 1 章で本研究の背景、目的を述べる。第 2 章で Arimaa を含むゲーム AI でよく利用される手法、また Arimaa のルールやその難しさについて述べる。第 3 章で評価関数の学習に関する関連研究を述べる。第 4 章で本研究で行った比較学習の内容と、評価実験の結果を示す。最後に第 5 章でまとめを行なう。

第2章　ゲームAI

2.1　ゲームAI

2.1.1　ゲーム木

ゲームのある局面をノード、選択される指し手をエッジとした木構造で、ゲームの進行を表すことができる。これをゲーム木と言う。ゲーム木を探索することで、先の展開を読むことが可能になる。代表的な手法として、Minimax 探索が存在する。

ゲームの複雑さを示す指標の一つとして、ゲーム木の大きさがあげられる。ゲーム木が大きくなるほど、コンピュータにとって難しくなることが多い。例えばゲーム木の大きさが 10^{30} 程度と比較的小さいチェッカーでは、両プレイヤが最善手を指し続けた場合に引き分けになることが既に示されている [11]。Arimaa のゲーム木の大きさは 10^{400} 程度であり [12]、チェスの 10^{120}、将棋の 10^{220} と比べて非常に大きい。

代表的なゲーム木探索の手法の一つとして Minimax 探索があげられる。Minimax 探索は、自分も相手も常に最善手を指すという前提で探索を行っていくアルゴリズムである。Minimax 探索の様子を図 2.1 に示す。評価値が大きいほど自分にとって有利な局面であることを示しており、自分の手番 (max node) では評価値が大きい方を、相手の手番 (min node) では評価値が小さい方が選択される。

Minimax 探索では、不要必要に関係なく全てのノードを探索している。探索を行わなくても結果が変化しない部分の計算を省くことで、Minimax 探索の高速化を行ったアルゴリズムがアルファ・ベータ探索である。アルファ・ベータ探索を用いることにより、Minimax 探索よりも高速に探索を行なうことが可能になる。

アルファ・ベータ探索の様子を図 2.2 に示す。図では左のエッジから探索を行っているとする。評価値が?となっている局面が探索が行われなかった局面である。深さ 2 で左から 2 番目の max ノードが探索されるときには、親ノードにあたる深さ 1 で一番左の min ノードが 3 以下の値しかとらないことが分かっており、かつ自分の子ノードに 5 という評価値が現れたことから、自分が 5 以上の評価値にしかなりえないということが判断できる。そのため、自分の子ノードをこれ以上探索する必要がなくなり、枝刈りが生じる。同じような考え方により、右側のノードでも枝刈りが発生する。

アルファ・ベータ探索では既に探索された部分の木の評価値を利用して枝刈りを行なうため、ゲーム木のどの部分から探索を行なうのかによって枝刈りの頻度が大きく変化する。各ノードにおける子ノードの数（branching factor）が b 個存在するゲーム木を深さ d で探索する場合の探索する必要のあるノード数を考える。悪い手から順番に探索が行われた場合、全く枝刈りが発生せず最も効率の悪い結果となり、b^d 個のノードを探索する必要がある。しかし、最もいい手を最初に探索できた場

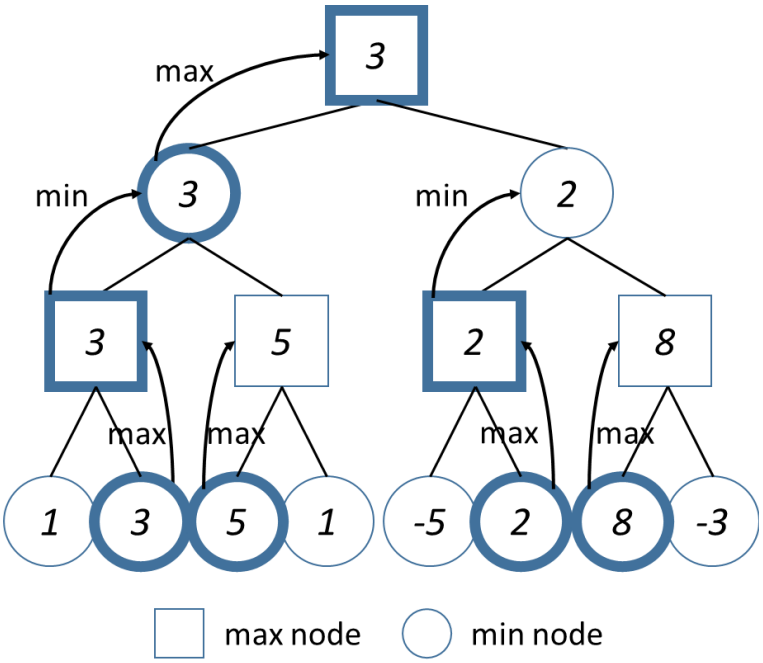

図 2.1. Minimax 探索

合、最も効率の良い結果となり、$b^{\lceil d/2 \rceil} + b^{\lfloor d/2 \rfloor} - 1$ 個のノードを探索すればよい。

そのため、探索する前に探索を行なう指し手の順番をうまく決めることで、探索のコストを大きく減らすことができる。この探索前に行われる探索順序の並び替えのことをムーブオーダリングと呼ぶ。ムーブオーダリングの手法としては、PV ムーブ、キラームーブ、ハッシュムーブ、ヒストリーヒューリスティックなどが一般的なムーブオーダリングの手法としてよく用いられている。

2.1.2 評価関数

ゲームが終局となるゲーム木の末端まで、全ての局面を探索することができれば、そのゲームの勝敗を完全に予想することが可能になる。しかし、ゲームが終了するまで読み切ることは現実的な時間では不可能であることが多い。そのため、ゲームが終了していない局面についても有利・不利の判断を行いながら、探索をする必要がある。局面の有利・不利を数値化したものを評価値と呼び、評価値を計算するための関数を評価関数を呼ぶ。完璧な評価関数が分かれば、最善手が何であるかわかることになるため探索を行う必要がなくなる。しかし、完全な評価関数を作成することも多くの場合不可能であるため、探索と評価関数を組み合わせて利用する必要がある。精度の高い評価関数の計算には時間がかかるため、探索できるノード数が減少してしまう。そのため、評価関数の精度と探索の深さのバランスをとる必要がある。実際には、短時間で計算できるような評価関数を用いて、

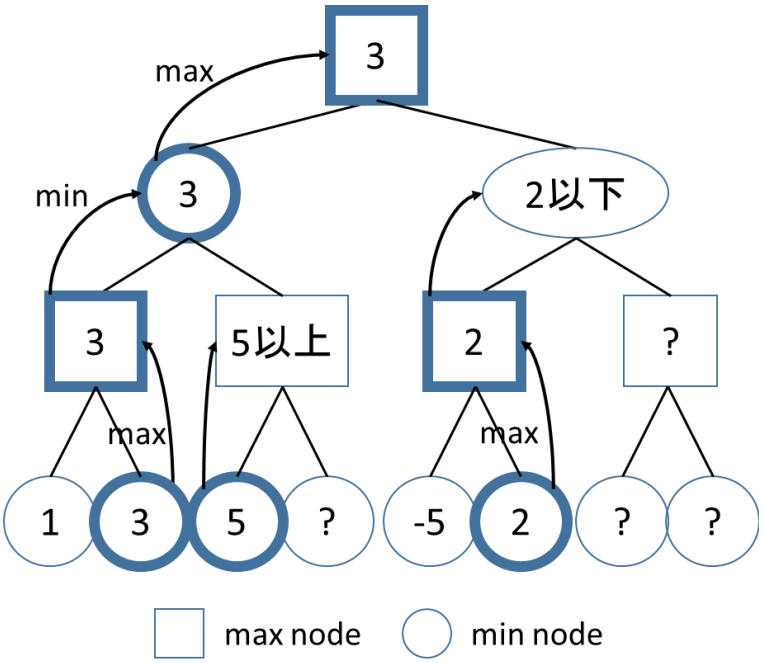

図 2.2. アルファ・ベータ探索

長時間かけて探索を行うことがよく行われる。

評価関数は、探索中に何度も呼び出されるため、精度が高いことと共に、計算速度が速いことも求められる。そのため、局面の特徴ベクトルと重みベクトルの線型和の形でよく表される。単純な線型和であるため評価関数の表現力に限界はあるものの、探索と組み合わせて利用したときの精度が重要であるため、評価関数単体での精度に拘ることで評価関数の計算に時間をかけてしまい探索にかけられる時間が減ってしまっては本末転倒であり、線型和の評価関数が良いとされている。

局面からどのように特徴量を抽出するかは、多くの特徴量の中から学習に有用な特徴を選択するような研究も行われている [13] が、ゲームに関する知識を持った人の手によって選択されることが多い。有用な特徴量を利用することにより、線型和であっても精度の高い評価が可能になる。

重みベクトルは、機械学習を利用して調整されることが多い。重みベクトルを調整するための機械学習の手法としては、比較学習や強化学習がよく知られている。比較学習については 3.3 章で詳しく述べる。

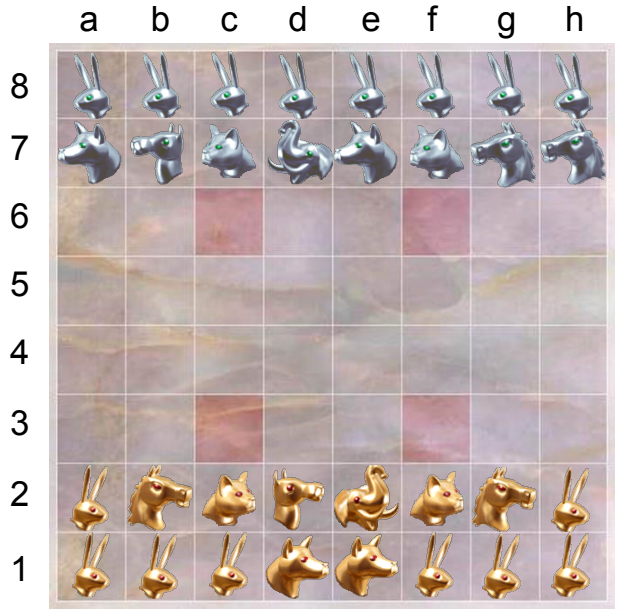

図 2.3. Arimaa の駒の初期配置の一例 [1]

2.2 Arimaa

2.2.1 Arimaa のルール

Arimaa では、8 × 8 のマス目を持った盤を用いる。駒はプレイヤーごとに象とラクダが各 1 個、馬と犬と猫が各 2 個、ウサギが 8 個である。盤のサイズがチェスと同じであり、また駒の種類と数もチェスと同じであるため、チェスの駒と盤を流用できるようになっている。駒の強さは強い方から順に象、ラクダ、馬、犬、猫、ウサギの順になっており、これは後に述べる PUSH や PULL、 FREEZE に影響する。8 個のウサギのうちいずれか 1 個を反対側の一番奥の段まで先に到達させたプレイヤーの勝利となる。

駒の初期配置

Arimaa ではチェスや将棋などとは異なり、駒の初期配置は決まっておらず、プレイヤーが最初に決める必要がある。まず最初に先手のプレイヤーが手前二段に自分の駒を好きなように配置する。先手が全ての駒を配置したら、後手も同様に自分の駒を配置する。その後は交互に手番を消費して、駒を動かしていく。駒の初期配置の一例を図 2.3 に示す。

駒の動き

プレイヤーは一度の手番において最大 4 ステップまで動かすことができる。自分の駒を一度動かす行動は 1 ステップ分である。各駒は隣接する上下左右の空きマスに移動することができる。しかしウサギは後ろに移動することはできない。一度の手番で一つの駒を何度も動かすこともできるし、複数の駒を動かすこともできる。しかし、自分よりも強い相手の駒が隣接しており、かつ味方の駒が隣接していない場合、その駒は FREEZE し動かすことができない。4 ステップ全てを消費せずにパスすることも可能であるが、最低でも 1 ステップは消費しなければならない。

Arimaa には PUSH や PULL という特別な動きが存在し、これにより相手の駒も動かすことができる。 PUSH は、自分の駒と隣接している相手の駒を任意の方向の空きマスに移動させ、相手の駒が移動する前の位置に自分の駒を移動させる。 PULL は相手の駒と隣接している自分の駒を移動させた後に、隣接していた相手の駒を自分の駒が移動する前の位置に移動させる。 PUSH や PULL は自分の駒と相手の駒の移動で計 2 ステップ消費したことになる。PUSH や PULL は自分より弱い駒に対してのみ可能であり、自分と同じ強さの駒や自分より強い駒に対しては行うことができない。FREEZE している自分の駒を動かして PUSH や PULL を行なうことはできないが、FREEZE している相手の駒を PUSH 、 PULL することは可能である。

トラップ

c3, c6, f3, f6 の 4 つのマスにはトラップが存在する。トラップ上にいる駒は、味方の駒が隣接していない場合ゲームから除外される。PUSH や PULL を用いて相手の駒をトラップに落とすことができ、そのようにして相手の駒を減らしていくことがこのゲームの序盤、中盤における基本的な戦術となる [14]。

PUSH や PULL を利用して駒を取る例を 2.4 に示す。c6 のトラップ上には後手のウサギが存在するが、今の状態では後手の犬がトラップの隣にいるためウサギの駒は取られることはない。しかし、先手の象が後手の犬を PUSH することにより、c6 の後手のウサギの駒に隣接する味方の駒がいなくなり、後手のウサギは取られることになる。

同様に f6 のトラップ上には後手の馬が存在するが、今の状態では後手の猫がトラップの隣にいるため馬の駒は取られることはない。後手の馬の駒が先手のウサギの駒を PULL しトラップの存在する f6 の位置に移動させることで、後手は先手のウサギを取ることができる。

特殊な状況におけるルール

ウサギが最奥の段に到達する以外に、特殊な状況において勝敗が決まる場合が存在する。

- 両方のプレイヤがウサギを全て失ってしまった場合、その勝負は引き分けとなる。
- 千日手を防ぐため、手番終了時の駒の配置が 3 回同じになった場合は、そのプレイヤの負けとなる。
- 動かせる駒が一つもない場合は、そのプレイヤの負けとなる。

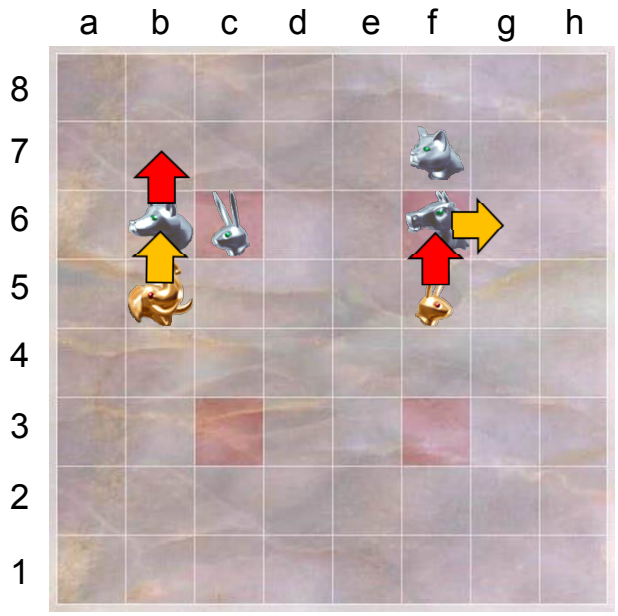

図 2.4. トラップを利用して駒を取る例

2.2.2 Arimaa の難しさ

Arimaa はチェス、将棋、囲碁などと同じく、二人零和有限確定完全情報ゲームに分類されるゲームである。

二人 プレイヤの人数が二人であること。三人以上のゲームでは対戦相手一人を不利にすることで自分が有利になるとは限らないため、難易度が高くなりやすい。

零和 プレイヤ全員の利得の合計が 0 であること。それぞれの利得を、勝ちなら+1、負けなら-1、引き分けなら 0 のように考えるとプレイヤ全員の利得の合計が 0 になる。零和でないゲームでは、プレイヤ全員が勝利したり、プレイヤ全員が敗北することがあるため、零和ゲームよりも難しくなることが多い。

有限 合法手の数や局面の状態数の数が有限であること。有限でないゲームでは、局面を探索していくことが困難になる。

確定 確率に左右される要素が存在せず、指し手に対して次状態が一意に定まること。サイコロを利用するようなゲームは不確定ゲームとなる。

完全情報 局面の状態を全て把握することができること。相手の手札が見えないカードゲームなどは、不完全情報ゲームとなる。不完全情報ゲームは見えない情報の予測を行なう必要が出てくるため、難易度が高くなりやすい。

二人零和有限確定完全情報ゲームは、ゲーム理論の中では最も単純な部類のゲームに分類されるものであり、古くから研究が行われている。しかし、Arimaa を上手にプレイすることは決して簡単ではない。Arimaa を上手にプレイすることが難しい理由を順に説明する。

2.2.2.1 合法手の数の多さ

まず一ターンごとの合法手の数が非常に多いことがあげられる。 Arimaa では一ターンに 4 回の移動を行うことができ、そのため一ターンに平均して約 16,000 通りの合法手が存在する [5]。チェスでは 35 通り、将棋では 80 通り [6]、囲碁では 250 通り程度であり、Arimaa の合法手の数は非常に大きい。このことにより、Arimaa では 1 手深く読もうとするごとに 16,000 倍の数の局面を調べる必要が出てくる。そのため、浅い探索でも探索に時間がかかり、深い探索を行なうことは現実的な時間では不可能であり、数手先の局面を予測することすら難しい。またチェスにおいては終盤になればなるほど駒の数が減少して合法手の数が減少してくるものの、Arimaa に関しては合法手の数はほぼ一定であり、終盤になれば探索が楽になるというようなことは発生しない。

人間は、知識や経験を利用したり、ある駒を取るなどの短期的な目標を設置したりすることで、考える必要のある指し手とそうでない指し手を分類し、指し手の候補を 1 ターンに数手程度にまで一瞬で絞ってしまうことでこの問題を解決している。しかし、コンピュータに同じことを行わせることは容易ではない。

2.2.2.2 駒の初期配置

Arimaa ではプレイヤが初期配置を選択することができ、そのパターンは約 4.2×10^{15} 通り存在している。チェスや将棋では、コンピュータに序盤の定跡を覚えさせることで上級者と同じ指し手が指せるようになり、序盤の実力の向上を果たしている。しかし、Arimaa ではコンピュータに序盤の定跡を覚えさせることはまず不可能であり、定跡に頼らずに序盤を指していく必要がある。人間は序盤の定跡を覚えておけば、少し異なる似た局面に遭遇したときでもその違いが意味が指し手に影響の与える違いなのかそうでないのかを判断することで、うまく対応することができるが、コンピュータにその判断を行わせることはやはり容易ではない。

2.2.2.3 ゲーム木の大きさ

ゲーム木の大きさはゲームの複雑さを表す指標の一つとして考えることができる。初期配置の方法 4.2×10^{15} 通り、各ターンの合法手の数が約 16,000 通り、1 試合の平均ターン数が約 92 ターン [5] であることから、Arimaa のゲーム木の大きさは

$$4.2 \times 10^{15} \times 16,000^{92} \approx 1.8 \times 10^{402} \tag{2.1}$$

と求めることができる。他のゲームとの比較を表 2.1 に示す。ゲーム木の大きさから、Arimaa がチェス、将棋、囲碁よりも複雑なゲームであるということができる。

表 2.1. ゲーム木の大きさ

ゲーム	合法手の数	平均ターン数	ゲーム木の大きさ
チェス	35	80	10^{123}
将棋	80	115	10^{220}
囲碁	250	150	10^{360}
Arimaa	16,000	92	10^{402}

2.2.2.4 戦略の必要性

Arimaa においては合法手の多さにより数手先の局面を探索することが困難である。そのため、深い探索を行なうことで長期的な戦略を考えるということができない。また、チェスや将棋においては、相手のキング（王将）を取るという明確なゴールが設定されているのに対し、Arimaa の勝利条件は 8 体いるウサギのうちいずか 1 体を最も奥の段まで到達させるというものであるため、明確な目標となる駒が存在せず、争いが発生する地点を考えることが難しい。相手の駒を取るということに関しても、トラップの周囲の駒というのが非常に重要な問題となってくるため、多数の駒の位置関係、強さの関係を考慮し、トラップを支配することができるかを検討する必要がある。

そのため、Arimaa は長期的な戦略が必要となってくるゲームであり、コンピュータにとって苦手なゲームであるということができる。

2.2.3 Arimaa の基本的な戦術

Arimaa の初期配置では、強力な駒を中央や前段に、ウサギの駒を端や後段に置くのが一般的である。最初のうちは相手も味方も駒が多く存在するため、ウサギの駒が前進してゴールを目指そうとしても、そもそも相手の駒が邪魔になって進むことができないためである。

そのため、Arimaa では序盤から中盤にかけてはトラップをめぐる攻防が行われる。トラップを利用して相手の駒を減らすことで、ウサギが奥の段に到達する隙間を作り出すことを目指す。ここで最も重要となってくる駒は象である。象は PUSH されることも PULL されることもないため相手に取られる心配も少なく、またトラップの周囲にいればトラップ一つを安全な状態に保つことができる。そのため、象の駒を前線に送り出し、相手側にある二つのトラップの攻防に参戦させるのが一般的である。象単体、象と馬、象とラクダなどで相手側のトラップを攻撃しつつ、残りの駒で相手の攻撃から自分側のトラップを守るというのが一般的な展開である。

象は相手に PUSH されることも PULL されることもないものの、常に自由に動き回ることができるわけではない。相手の象の動きを封じるための手段がいくつか存在する。

一つは図 2.5 の赤枠で囲った部分のような Phalanx と呼ばれる形がある。味方の駒を固めて配置し、PUSH される場所をなくしてしまうことで、先手の象の移動を妨げることができている。

他には、図 2.6 に示す Frame と呼ばれる状態や図 2.7 に示す Hostage と呼ばれる状態がある。これらの状態では、象の駒が動くことは可能であるものの、象が動いてしまった場合自分の駒を失う結果になってしまうため、象の動きに制限がかかった状態であるといえる。

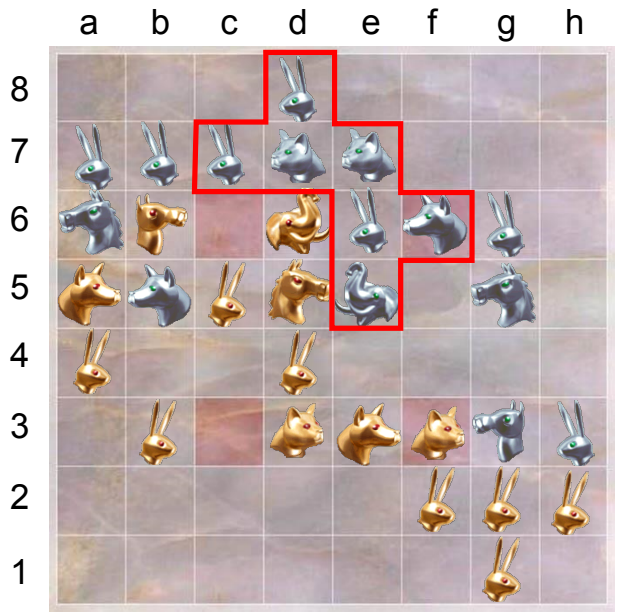

図 2.5. Phalanx

また象の移動可能な範囲が少なくなる一例として、図 2.8 のような状態があげられる。この局面では、先手の象を動かすとトラップにかかってしまうため、このままでは動かすことが不可能になってしまっている。これは Elephant Blockades と呼ばれる、相手の象の周囲を自分の駒で固めることにより、相手の象が隣接している駒を PUSH することも PULL することもできない状態にする戦術である。

他にも中央二列に自分の駒を並べることにより、相手の象を左右のどちらかのトラップ周辺にしか移動できなくするなどの戦術が考えられる。

トラップをめぐるの攻防が行われて、両者の駒が減ってきたらウサギの出番である。序盤はウサギは前進してもトラップによって捕まえられてしまうだけで終わってしまうが、駒が少なくなり相手の守りに隙が生じたらウサギが前進してゴールを目指すことになる。端からウサギが前進していくことが多いが、中央からゴールをすることもある。

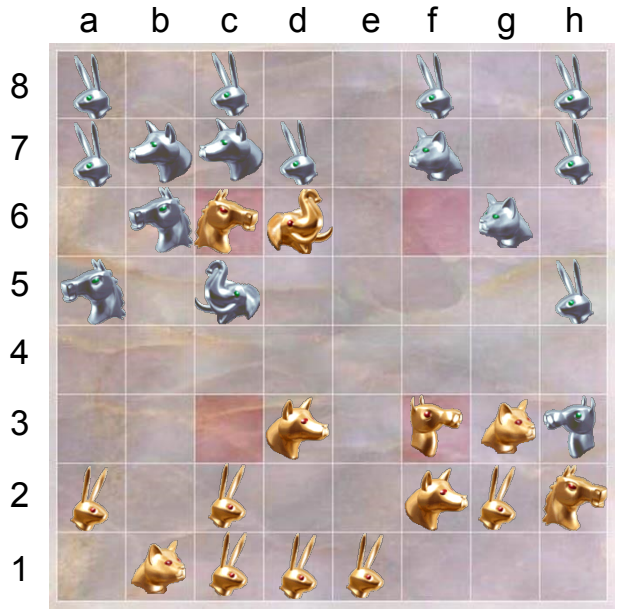

図 2.6. Frame

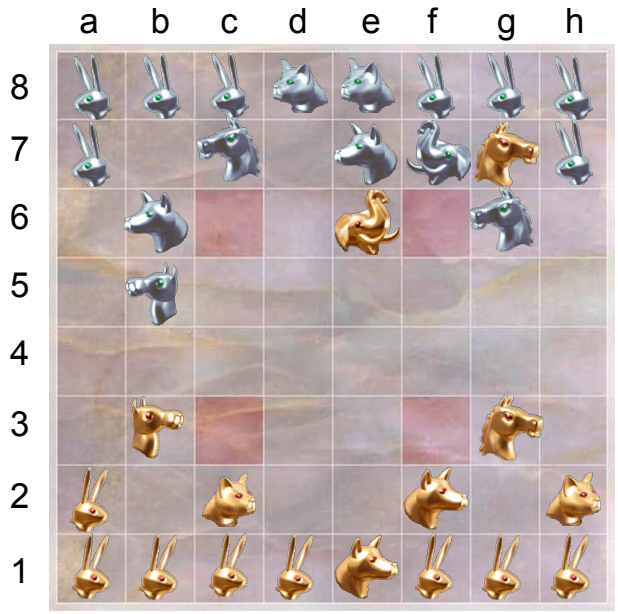

図 2.7. Hostage

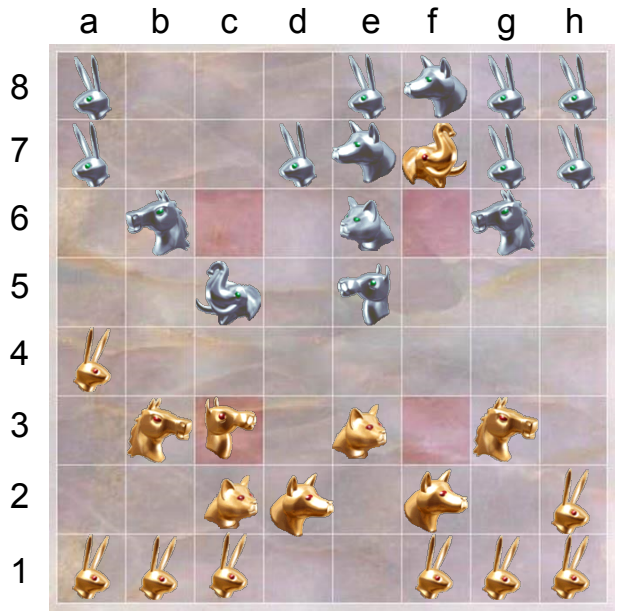

図 2.8. Elephant Blockades

第3章　関連研究

3.1　教師あり学習

教師あり学習とは、機械学習の手法の一つであり、正解ラベルがつけられたデータを教師データとして利用することで、教師データに含まれていない未知の入力に対しても正答率が高くなるように学習を行う手法である。将棋プログラム Bonanza [9] が成功を収めて以来、将棋の評価関数の学習は教師あり学習で行われるようになっている。教師データに含まれていないデータに対する性能を汎化性能と言う。教師あり学習では教師データに対して過剰に適合してしまい、未知のデータに対する性能が低下してしまう過学習が起こることがある。そのため学習後は教師データとは異なるデータを用いて汎化性能を評価する。

3.1.1　パーセプトロン

パーセプトロンは教師あり学習において重要な手法の一つである [15]。図 3.1 に示されるような、正例と負例の二値がラベル付けされた教師データから正例・負例を分類する分離平面を学習する場合を考える。教師データの例から作成された教師ベクトル $\mathbf{x}$ に対しての出力 y は重みベクトル $\mathbf{w}$ を用いて以下のように表される。

$$y = f(\mathbf{w} \cdot \mathbf{x}) \tag{3.1}$$

ここで $f(x)$ は以下の式で表される。

$$f(x) = \begin{cases} 1, & x \geq 0 \\ -1, & x < 0 \end{cases} \tag{3.2}$$

出力 y が例についたラベル t と異なっていた場合、重みベクトルを以下のように更新する。

$$\mathbf{w} \leftarrow \mathbf{w} + c(t - y)\mathbf{x} \tag{3.3}$$

c は学習率と呼ばれる、小さな正の値が用いられる。

パーセプトロンではこの更新を繰り返すことでデータを分類する重みベクトルを学習する。パーセプトロンは正例と負例が線形分離可能である場合は必ず収束し、それらを分離する超平面を見つけることが可能である。

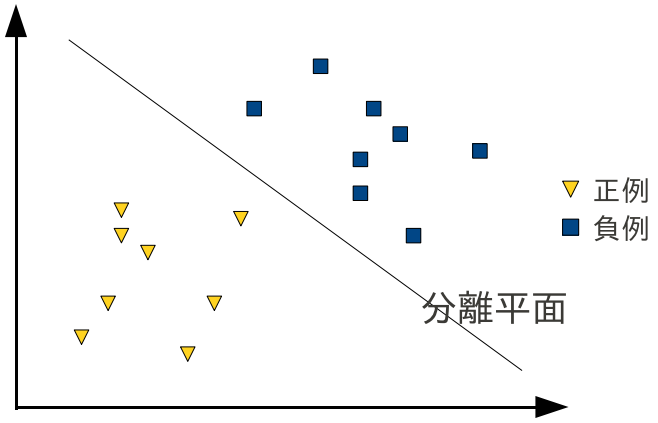

図 3.1. 二次元平面上での二値分類問題の例

3.1.2 平均化パーセプトロン

パーセプトロンは図 3.2 のように、データが線形分離可能でない場合収束しない。その欠点を改良したものとして平均化パーセプトロン [16] がある。平均化パーセプトロンでは、通常のパーセプトロンと同じ学習を行いつつ過去の重みベクトルの和と重みベクトルの更新回数を記憶しておく。全ての入力データを読みこんだら過去の重みベクトルの平均を計算し、それを最終的な重みベクトルとする。こうすることにより、収束性の問題を解決し、過学習を防ぐことができる。求めたい重みベクトルの平均 $\frac{1}{T}\sum_{t=0}^{T}\mathbf{w}_t$ は、t 回目の更新後の重みベクトルを $\mathbf{w}_t$、t 回目の更新で加算される更新重みベクトルを $\mathbf{u}_t$ を用いて、

$$\frac{1}{T}\sum_{t=1}^{T}\mathbf{w}_t = \frac{1}{T}\sum_{t=1}^{T-1}(T-t)\mathbf{u}_t \tag{3.4}$$

$$= \mathbf{w}_T - \frac{1}{T}\sum_{t=1}^{T-1}t\mathbf{u}_t \tag{3.5}$$

と表すことができるため、更新の重み付き和である $\sum_{t=1}^{T-1}t\mathbf{u}_t$ を保持しておくことで実装が簡単になる。

3.1.3 マージンパーセプトロン

正解データが誤っていた場合だけでなく、$\mathbf{w}\cdot\mathbf{x}$ が一定の値以下であったときにも更新を行う手法がある [17]。これをマージンパーセプトロンと言う。これにより、識別平面とデータとの距離で表されるマージンを大きくすることができ、汎化性能を高めることができる。

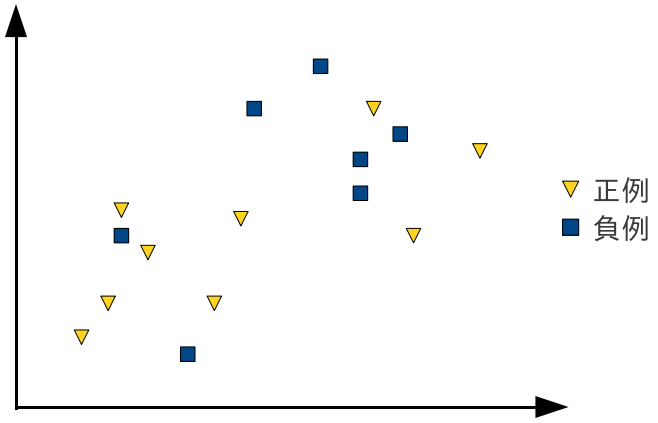

図 3.2. 線形分離不可能な二値分類問題の例

3.2 静止探索

静止探索は、通常の探索の後に評価値が大きく変動しそうな指し手に限って探索を行い、得られた局面の評価値を最終的な評価値として利用することで、局面の評価値を安定させる手法である [18]。チェス、将棋などのゲームでは、駒の奪い合いや、駒を成る手が生じると大きく評価値が変動する。そのため、数手先で大きく評価値が変動するにも関わらず、水平線効果により正確な評価値を求めることができないという状況が発生する。探索する指し手を限定して、駒の取り合いなどのない “静止した局面”になるまで探索を行なうことで、水平線効果を防ぎ、短時間で安定した評価値を得ることが可能になる。

Arimaa における静止探索は、Wu の研究 [12] や、ネット上で公開されている Arimaa プログラムの一つである OpFor [19] などで実装が行われているものの、定量的な評価は行われていない。静止探索を行なう際には、どのような指し手を探索の対象に含めるかが問題となる。 Wu の研究では駒を取る手と駒を取られるのを防ぐ手、 OpFor では駒を取る手を静止探索に利用している。Arimaa では、水平線効果により、数手先で両方の駒を取られてしまうにも関わらず、本来取られずに済む自分の弱い駒をトラップの隣に移動させることで、強い駒が取られるのを遅らせるような手を選択してしまうことが考えられる。静止探索を行なうことで、取られてしまう駒を意味もなく増やしてしまうような手が選択されることを防ぐことが期待される。

3.3 比較学習

Arimaa に類似したゲームであるチェスや将棋の評価関数学習手法としてよく用いられている、比較学習 [7] の研究について述べる。比較学習は上級者の棋譜を利用した評価関数の学習手法であり、プログラムが上級者の指し手を真似するように重みベクトルが調整される。

チェスにおける比較学習では、棋譜中の局面から棋譜の指し手を一手指した局面と、他の手を一手指した局面を比較してもうまくいかないことが明らかになっている。一手動かした局面で学習を行なうというのは、指し手の表面的な情報しか捉えておらず、指し手の目的、狙いというものがプログラムに理解できていないと考えられる。プログラムに指し手の目的、狙いを理解させるために、静止探索や浅い探索を行なうことで、有用な評価関数の学習に成功した [8]。

将棋では、 Bonanza [9] が評価関数をプロ棋士の棋譜から学習することに成功して以来、教師あり学習がよく用いられている。Bonanza で用いられている学習手法は、比較学習をベースとした手法である。数式で表すと以下の誤差関数 J を最小化する特徴ベクトル $\mathbf{v}$ の最適化問題となる。

$$J(P_0, ..., \mathbf{v}) = \sum_{i=0}^{N-1} \sum_{m=1}^{M} T[\xi(p_m, \mathbf{v}) - \xi(p_{m=0}, \mathbf{v})] \tag{3.6}$$

ただし N は学習データの局面数、P_i は局面、M はその局面における合法手の数、ξ は局面の評価関数、T は評価値の差を棋譜の指し手との一致度に変換する関数であり、シグモイド関数などが用いられる。p_m は局面 P の指し手 m の後の局面を探索して得られた最善応手手順の末端局面である。$m = 0$ は棋譜の指し手である。これに、過学習を防ぐための正則化項を加えたものを目的関数として、最急降下法により目的関数 J を最小化することで重みベクトルの学習を行う。この手法は俗に “Bonanza Method” と呼ばれる。

将棋プログラム「激指」[20] では比較学習と平均化マージンパーセプトロンを利用した手法が用いられている [21]。この手法では、ある局面 P に対する指し手 m のあとの最善応手手順を探索し、得られた末端局面を利用した学習を行う。その局面に対する評価値を返す関数を $\xi(p_m, \mathbf{v})$ とし、全ての合法手の集合を M とする。まず棋譜の指し手が他の指し手と比べて相対的に高く評価されているかどうかを確認し、棋譜の指し手より評価値が高い、もしくはある margin を持って評価値が近いものを発見する。具体的には局面 P における棋譜の指し手 $m = 0$ に対して

$$M' = \{m \in M | \xi(p_m, \mathbf{v}) + \text{margin} > \xi(p_{m=0}, \mathbf{v})\} \tag{3.7}$$

となる指し手の集合 M' を求める。M' に含まれている指し手は棋譜の指し手と比較して評価値が十分小さくはないということになる。

全ての $m \in M'$ の特徴ベクトルについて棋譜の指し手の特徴ベクトルとの差を取り、その平均を重みベクトル $\mathbf{v}$ に加える。式で表すと式 (3.8) のようになる。

$$\mathbf{v} \leftarrow \mathbf{v} + \frac{1}{|M'|} \sum_{m \in M'} (\phi(p_{m=0}) - \phi(p_m)) \tag{3.8}$$

$\phi(P)$ は、局面 P の特徴ベクトルである。それを全ての局面に対して繰り返し、最後に学習途中に現れた重みベクトルの平均を取り、それを最終的な重みベクトルとしている。

3.4 Arimaa プログラム

3.4.1 Arimaa におけるアルファ・ベータ探索

Arimaa では 2.1.1 章で述べたアルファ・ベータ探索がよく用いられている。チェスや将棋で行われているものと同じ手法であるが、 Arimaa では 1 手を 4 ステップと分けて考えることもできるため、4 ステップをまとめた 1 ターンの動きを 1 つの指し手として木を作成するターンベース探索と、各ステップの動きを 1 つの指し手と考えて木を作成するステップベース探索の二つが考えられる。

ターンベース探索を行なうと、各局面での合法手の数は多くなるが、木の深さは浅くなる。逆にステップベースを行なうと、各局面での合法手の数は少なくなるが、木の深さは深くなる。探索が終了したときに得られる結果は両方共同じになるが、探索の速さや途中で探索を打ち切った場合の結果が変わってくる。ターンベース探索では 4 ステップ分の動きを考えるため、探索前に指し手の評価を行い良さそうな順に並び替える作業である Move Ordering が行い易く、比較的高速に探索を行える。ステップベース探索では、1 ステップの動きとしては良くないものの、後のステップの布石になるような動きを Move Ordering で発見するのは難しく、Move Ordering の精度が落ちやすい。合法手が多く枝刈りが頻繁に起こる Arimaa では Move Ordering は重要であり、棋譜から Move Ordering を学習した研究がいくつか存在する [12][22]。ターンベース探索では、2 ターンの探索を行ったあと 3 ターンの探索が終わる前に探索が打ち切られた場合、2 ターンで探索を行った場合の結果を返すことしかできない。一方ステップベース探索では、探索が途中で打ち切られた場合でも、例えば 2 ターンと 2 ステップ先の局面の評価値で判断することが可能になる。ステップベースとターンベースの両方に利点・欠点が存在するため、上位の Arimaa プログラムはこの二つを組み合わせて探索を行っている [23]。

3.4.2 Arimaa におけるモンテカルロ木探索

ランダムシミュレーションを利用するモンテカルロ法と木探索を組み合わせた手法であるモンテカルロ木探索という探索手法が存在する。この手法は囲碁でよく用いられている手法であり、2016 年 1 月に初めて人間のプロ棋士に勝利した “AlphaGo”[24] にも利用されているなど、非常に良い結果残している手法である。このモンテカルロ木探索を Arimaa で利用した研究も存在する [25][26] が、アルファ・ベータ探索に匹敵する結果は残していない。この理由としては、囲碁では石の価値というものを計算することが難しいものの、 Arimaa は囲碁とは異なり駒の価値というものを計算しやすく、評価関数が囲碁よりも比較的簡単に作成が可能であるということや、Arimaa では合法手の数が膨大すぎる影響でランダムシミュレーションがある程度収束するまでに必要な回数が多くなってしまうことなどがあげられる。

3.4.3 Arimaa プログラム “Sharp”

2015 年の Arimaa チャレンジで人間のトッププレイヤ相手に初めて勝ち越したプログラムである Sharp は、自己対戦や浅い探索での対戦を繰り返し、Arimaa 上級者の助けを借りながら手作業で Move

Ordering の調整、評価関数の調整を行なうことで、強いプログラムとなった [23]。Sharp の弱点として序盤の弱さが上げられており、序盤の評価関数を調整することが難しいということが分かる。機械学習を用いて自動的に評価関数を調節することで、これらの問題を解決できる可能性があるのではないかと考察している。

3.5 Arimaa の評価関数の学習

Arimaa の評価関数を機械学習を用いて行った研究を 3 つ紹介する。

まず一つ目に強化学習による Arimaa の評価関数の学習である [12]。この研究では、 TD, TD-Leaf, Rootstrap, Treestrap の 4 つの学習手法を用いて評価関数を学習している。強化学習は、棋譜などの教師データを用いて良い指し手を教えることで学習していく手法とは異なり、自分自身が行った行動とその結果得られる報酬から自動的に学習を行っていく手法である。

TD は、ある状態での評価値とその状態の後の状態での評価値との差を利用して評価関数を学習する手法である [27]。状態 s_t から次状態 s_{t+1} に遷移する場合、評価関数の重みベクトルを θ、状態 s_t での評価値を $H_\theta(s_t)$ とおくと、TD 誤差 δ_t は

$$\delta_t = H_\theta(s_{t+1}) - H_\theta(s_t) \tag{3.9}$$

と表すことができる。この TD 誤差を最小化するように評価関数の学習を行っていく。実験では TD(λ) と呼ばれる学習手法を用いており、その更新式はパラメータ $\lambda \in [0,1]$ を用いて

$$H_\theta(s_i) \overset{\theta}{\leftarrow} H_\theta(s_i) + \sum_{j=i+1}^{n} \lambda^{i-j-1}\delta_j \tag{3.10}$$

と表される。

TD-Leaf は TD の亜種の一つであり、ある状態から探索したときの末端局面の評価値を用いて更新を行なう学習手法である [28]。状態 s から深さ D で探索して得られた末端局面の評価値を $H_\theta^D(s)$ とすると、TD-Leaf の更新式は

$$H_\theta^D(s_i) \overset{\theta}{\leftarrow} H_\theta^D(s_i) + \sum_{j=i+1}^{n} \lambda^{i-j-1}\delta_j \tag{3.11}$$

となる。ただし

$$\delta_i = H_\theta^D(s_i) - H_\theta^D(s_{i-1}) \tag{3.12}$$

である。

Rootstrap は探索のルート局面と末端局面の評価値を近づけるように学習する手法である [29]。更新式は

$$H_\theta(s) \overset{\theta}{\leftarrow} H_\theta^D(s) \tag{3.13}$$

となる。

Treestrap は探索中に出現した全ての局面に対して末端局面に評価値を近づけるように学習する手法である [29]。学習時に Minimax 探索を用いる場合とアルファ・ベータ探索を用いる場合で更新式が異なるが、論文中ではアルファ・ベータ探索が用いられている。更新式は

$$H_\theta(s') \stackrel{\theta}{\leftarrow} H_\theta^{D-d(s')}(s') \tag{3.14}$$

となる。ただし、s' は探索中に出現する末端局面でない全ての局面、$d(s')$ は探索木における s' の深さである。

対戦実験において人間が手でつけた重みベクトルを持つ評価関数を利用したものと比較してどれも有意に勝率が向上しており、特に Rootstrap を用いて学習したものが最もいい結果を残している。強化学習を利用することで評価関数を改善できることが分かる。

二つ目に、将棋プログラム Bonanza [9] で用いられている比較学習を Arimaa に適応したもの研究がある [30]。Bonanza の学習手法は 3.3 で説明した通りである。しかしこの論文では、駒の位置に対する重みを学習したにとどまっており、Arimaa の評価関数に有効であると考えられる他の特徴量を使用していない。そのため、プログラムの強さも、既存のプログラムより劣るものとなっている。特徴量を追加して学習を行なうことで、有用な評価関数の学習ができたのではないかと考えられる。

三つ目に、Hřebejk らの研究 [31] では、TD 誤差学習と比較学習を組み合わせつつ、線形計画法で解ける問題として定式化することで、深さが 1 より大きい探索を行わずに有用な評価関数の学習に成功している。まず以下のように a,b,c,d を定義する。

$$a = |f(\text{current}) - f(\text{played})| \tag{3.15}$$

$$b = |f(\text{current}) - f(\text{best})| \tag{3.16}$$

$$c = f(\text{best}) - f(\text{played}) \tag{3.17}$$

$$d = f(\text{played}) - \text{average} \tag{3.18}$$

$f(\text{current})$ は教師データの局面の評価値、$f(\text{played})$ は教師データの指し手を指した局面の評価値、$f(\text{best})$ は最も評価値が高い指し手を指した局面の評価値、average は全指し手の評価値の平均である。a,b,c,d を用いて各教師データの局面に対する誤差関数を以下のように定める。

$$l(p) = \alpha a + \beta \max(a, b) + \gamma c + \delta(0, G - d) \tag{3.19}$$

α,β,γ,δ,G はそれぞれ学習のパラメータである。第一項と第二項は TD 誤差学習、第三項は比較学習のような動作を行なう。第四項は評価値のスケーリングに用いる。最終的な誤差関数は各局面について $l(p)$ を足しあわせ、正則化項を加えた

$$L = \sum_{p \in \text{training positions}} l(p) + \rho R \tag{3.20}$$

のような形となる。正則化項である R は各重みベクトルの要素 w_i の絶対値の和である。

$$R = \sum_i |w_i| \tag{3.21}$$

このようにして定式化した誤差関数を線形計画問題に変形し定式化することで学習を行っている。これは学習時に深さが 2 手以上の探索を行わないため、評価関数の重みベクトルが変化しても各局面の評価値を調べることが容易なため可能である。例えば第一項である αa は、各局面 i に対して変数に a_i,a'_i を導入し、制約として

$$(x_{j,\text{current}} - y_{j,\text{played}}) \tag{3.22}$$

を用いる。線形計画法を用いることによって誤差関数の最適化にかかる時間を減らし、有用な評価関数の学習に成功している。

第4章　Arimaa の評価関数の比較学習

本研究では、 Arimaa の実用に足る評価関数を棋譜から学習することを目指す。

4.1　予備実験

Arimaa において将棋と同様の手法で評価関数の学習を行おうとした場合に問題点として、将棋では浅い探索を非常に短い時間で行うことができるが、 Arimaa では浅い探索を行うことにすら時間がかかってしまうことが挙げられる。例えば今回実験を行った “Faerie” [1] という Arimaa プログラムでは、次の自分の手番の指し手まで（3 手番先）を読むのにだいたい 1 秒から 2 秒程度の時間を要する。

学習時に探索を行わない場合と、探索を行う場合の 2 つの手法で学習を行った。探索を行わない場合は、棋譜の指し手を指した局面の特徴量と、ランダムな合法手を指した局面の特徴量の比較を行った。この場合は、探索を行っていないため多くの指し手について評価値の比較を行うことができる。探索を行う場合は、全ての指し手について探索を行うのではなく、現状の評価関数で最善であると考えられた手のみに絞った。棋譜の指し手を指さずに深さ d の探索を行って得られた最善応手手順の末端局面の特徴量と、棋譜の指し手を指した後の局面を深さ $d-1$ の探索を行って得られた最善応手手順の末端局面の特徴量を比較し、学習を行う。学習の手順を図 4.1、図 4.2 に示す。

4.1.1　評価

4.1.1.1　評価設定

インターネット上で公開されている Arimaa プログラムである Faerie を用いて実験を行った。 Faerie は C 言語で書かれた Arimaa のサンプルプログラムとして公開されているプログラムであり、先行研究でもよく用いられている [25][26][30]。そのため、実験で用いるのに適したプログラムであると判断した。Faerie はステップベースのアルファ・ベータ探索と評価関数を用いた標準的な Arimaa プログラムである。今回の実験では、手番の途中のステップで探索を打ち切るということをせずに、常にターンの終わりまで探索を行った。評価関数は人間が手でつけた特徴量・重みベクトルが利用されている。静止探索は行われていない。ハッシュテーブルは実装されている。ムーブオーダリングに関しては、キラームーブ、ハッシュムーブの実装が行われている。駒の初期配置に関しては、先手の場合は 1 パターン、後手の場合でも 2 パターンしか選択することができない。

学習用の棋譜としては、 Arimaa のウェブサイト [1] からダウンロードできる棋譜約 30 万局 (2016 年 2 月時点) のうち、2002 年から 2014 年 6 月までの棋譜を利用し、その中で対戦者の Rating が 2,200 を超えている棋譜 2,321 局 (186,775 局面) を利用した。棋譜には “待った” などが含まれている場合があるが、その局面は学習から除外して行った。

学習の手法は 3.3 章で述べた平均化マージンパーセプトロンによる学習手法をベースとした。マージンは 10 として学習を行った。探索を行わない場合は、棋譜中の各局面について棋譜の指し手とは別に 1,000 手を比較対象として用い、186,775 局面を学習に用いた。静止探索を行なう場合は 100,000 局面を学習に用いた。重みベクトルの初期値として、 Faerie で利用されている重みを用いた。

学習した評価関数は棋譜との指し手の一致率と、対戦実験によって性能を測定した。対戦実験では、評価関数を学習していないオリジナルの Faerie 、探索を行わずに学習した評価関数、静止探索を用いて学習した評価関数のそれぞれを対戦させた。2002 年から 2014 年 6 月までの棋譜のうち、対戦者の Rating が 2,000 を超えていて、学習データとして利用していない棋譜から、重複のないように初期局面を選び出し、先手後手を入れ替えて対戦を行った。対戦時は探索の深さを 2 に固定した。

棋譜との指し手の一致率はテストデータ用の棋譜として 2014 年 7 月から 8 月の棋譜から、その中で Rating が 2,200 を超えている棋譜 126 局 (8,819 ターン) を利用した。学習した評価関数で探索を行い得られた最善手と、棋譜中の指し手との比較を行った。 Arimaa では一度の手番において最大 4 ステップ分の動きが可能であり、異なる指し手が 4 ステップ中 3 ステップは同じ動きをしていたり、異なる指し手であっても手番終了時の局面が同一であったりすることが考えられる。そのため、各手番において同じステップが出現する回数と、手番終了時の局面が一致する回数を調査した。

4.1.1.2 特徴量

評価関数の特徴量は、 Faerie で用いられていた特徴量をそのまま採用した。Faerie の特徴量は大きく分けて「 trap に関する特徴量」、「駒の価値に関する特徴量」、「ウサギに関する特徴量」の 3 つから成り立っている。

Trap に関する特徴量

- 各 trap に隣接している先手の駒の強さの合計
- 各 trap に隣接している後手の駒の強さの合計
- 各 trap に隣接している駒のうち最も強い駒を持つプレイヤー

駒の強さは象から順に 6, 5, 4, 3, 2, 1 とする。

駒の価値に関する特徴量 Fairie の駒の価値に関する評価関数は、駒の種類をベースとして、その駒に対して各フラグが立っているかどうかを利用して重みを計算しており、特徴量の線形和としては設計されていない。そのため、学習を行う際には “フラグ 1, 2, 4 が成立している象” の重みといったように特徴量を書き換えている。

各フラグを以下に示す

表 4.1. Faerie と同じ特徴量のみで学習した場合

評価関数	勝率	一致率	
		ターン	ステップ
Faerie	50%	7.8%	21.8%
探索なし 186,775 局面 × 1,000 手 10 イテレーション	17% (34 - 166)	1.0%	11.6%
探索あり 10,000 局面 1 イテレーション	45.5% (91 - 109)	7.8%	21.5%
探索あり 10,000 局面 2 イテレーション	48.5% (97 - 103)	7.6%	21.2%
with search 30,000 局面 1 イテレーション	48.5% (197- 203)	7.3%	21.7%

- その駒が FREEZE しているか
- 隣接する空きマスがあるか
- Trap に隣接している場合、他に同じ trap に隣接する味方の駒が他にいるか
- Trap から二マス離れた地点にいる場合、 trap の隣が空きマスか
- Trap 上にいる場合、複数の敵の駒に隣接されているか
- 犬や猫の駒の場合、その駒のいる段が 3 段目以降か

ウサギに関する特徴量

- 各ウサギのいる段
- 七段目にいるウサギの前と左右が空きマスか、味方の駒か、それともそれ以外か

4.1.2 実験結果

探索を行わない場合は学習のイテレーションを 10 回、探索を行った場合ではイテレーションを 1 回回した。学習時間は探索を行わない場合で約 8 時間、探索を行った場合 10,000 局面で約 14 時間、30,000 局面で 41 時間かかった。

実験の結果を表 4.1 に示す。探索を行わない場合は勝率・一致率ともに低い結果となった。教師データの数を多く用意できるものの、1 手先の評価のみしか学習されていないため、探索を行う際に適した評価関数が学習できてないのだろう。探索を行った場合では、元の評価関数と精度が殆ど変化していない。これは、探索時間との兼合いで学習に用いることができる局面数が少ないため、重みベクトルが大きく更新されることがなかったからだろう。

この実験においては、特徴量を追加することで性能が向上するのではなく、学習による重みベクトルの調整のみで評価関数の精度の向上が起こることの期待して Faerie で用いられている特徴量と同じ特徴量を用いて実験を行った。特徴量を追加して学習を行い性能が向上した場合、特徴量が増えて表現力が増えた影響で精度が向上したのか、それとも学習の効果が出て性能が向上したのか判断できないためである。しかし、Faerie では人手で調整しやすい特徴量を局面から抽出しており、機械学習を行なうのに適しているかどうかは考慮されておらず、利用している特徴量の種類も限られている。特徴量もスパースなものが多く、ほとんど出現しない特徴量も存在した。そのため精度の高い評価関数を作るためには、機械学習が行いやすいように更に特徴量を増やす必要があると考えられる。例えば二駒間の位置関係のような特徴量は、将棋やチェスの評価関数の特徴量としてよく用いられているが、要素数が非常に多く手動で調整するのは困難である。このように、手動で調整するのが困難な特徴量を学習することができれば、機械学習によって強いプログラムを作成することが可能になる。

4.2 静止探索

4.1 章で行った実験から、学習時に探索を行おうとすると、学習に利用できる局面が非常に少なくなってしまい、学習を行うことが難しいことが確認できた。そこで、静止探索を利用することで、探索を行うのに近い効果を得ることを目指す。

静止探索では、評価値の変動が大きい指し手に絞って探索を行なう必要がある。Arimaa における静止探索では、1 手の指し手の中にも、評価値を大きく変動させる静止探索に利用すべきステップと評価値を殆ど変動させないステップが混在していることが考えられる。そのため、今回は駒を取ることに関連しそうなステップのみを生成し、それらのステップを組み合わせて駒を取る手を生成し、探索に用いた。生成した各ステップの駒の動きを以下に示す。

- 相手の駒を PUSH または PULL することでトラップ上に移動させる動き
- 相手の駒を PUSH または PULL することでトラップに隣接するマスに移動させる動き
- トラップに隣接している相手の駒に、その駒よりも強い自分の駒で隣接する動き
- トラップ上にいる相手の駒を守っているトラップに隣接している相手の駒を PUSH または PULL する動き

こうすることにより、全ての指し手を生成してそこから駒を取る指し手を探すよりも速く駒を取る指し手が生成でき、かつ駒を取ることに無関係なステップの生成を抑えることができる。

表 4.2. 静止探索の有無による、先の局面との評価値の差の比較

	評価値の差の平均
静止探索なし	524
静止探索あり	621

4.2.1 評価設定

静止探索の評価を行なうために、数手先の評価値との比較を行った。静止探索を行なって得られる局面の評価値が、静止探索を行わない場合と比較して、未来の局面との評価値の差が小さくなっていることを確かめる。現在の局面から探索を行わずに得られる評価値を V_0、現在の局面から静止探索を行って得られる評価値を V_q、棋譜から得られる 2 手先の局面の評価値を V_0'、2 手先の局面から静止探索を行って得られる評価値を V_q' とする。このとき、静止探索を行わない場合の評価値の変動 $|V_0 - V_0'|$ と、静止探索を行った場合の評価値の変動 $|V_q - V_q'|$ を調べた。静止探索を行った場合の評価値の変動が小さければ、静止探索を行なうことで未来の局面の評価値を正確に予測できるようになったことが示される。

評価に用いる棋譜は、指し手の一致率を計測するのに用いた Rating が 2,200 を超えている棋譜 126 局を利用した。評価関数として、Faerie で元々利用されている評価関数をそのまま利用した。また、静止探索で勝敗を読みきってしまう局面は計算から除外した。

4.2.2 実験結果

実験結果を図 4.2 に示す。

静止探索を利用した場合のほうが評価値の差が大きくなってしまっている。この原因として、実際に駒を取るために必要なステップ以外の部分のステップをうまく考慮できていないことが考えられる。駒を取る以外のステップにおいても評価値は変動するため、その部分の考慮がうまくできていないと考えられる。また、実験では “Faerie” の評価関数を利用したため、静止探索の効果をうまく表現できない評価関数になっていた可能性もある。

4.3 特徴量を追加し、静止探索を利用した場合

4.1 章で行った実験から、Faerie で利用されている特徴量のみでは、うまく評価関数の学習が行うことができないということが確認できた。そのため、学習に利用する特徴量を追加した。

学習時に探索を行わない場合と、1 手探索＋静止探索を行なう場合の 2 つの手法で学習を行った。探索を行わない場合は、棋譜の指し手を指した局面の特徴量と、ランダムな合法手を指した局面の特徴量の比較による学習を行った。この場合は探索を行っていないため、多くの局面との評価値の比較を行なうことができる。静止探索を行なう場合は、棋譜の指し手を指した局面から静止探索を行っ

た局面と、1 手探索を行い得られた局面から静止探索を行った局面の比較による学習を行った。静止探索は最大 2 手である。学習の手順を図 4.3、図 4.4 に示す。

4.3.1 評価設定

4.1 章で行った実験と基本的に同一の設定である。しかし、Faerie で元々利用されていた特徴量に加えて、 Wu の研究 [12] を参考に、以下の特徴量を追加した。

駒の分類

全ての駒を以下の 12 種類に分類し、他の特徴量を求める際に利用した。ウサギ以外の駒は、自分より強い相手の駒の数（0-6）に応じて 7 種類、ウサギの駒は、相手のウサギ以外の駒の数と相手のウサギの駒の数を利用し、5 種類に分類した。

駒の価値

Arimaa においては、ウサギ以外の駒の動きは同じであり、駒の強さも相対的なものでしかない。そのため、相手の駒に何が残っているかに応じて、自分の駒の価値は大きく変動し、“犬の駒の価値”のように、駒の種類でそのまま価値を定めるのは適切でないと考えられる。

ウサギ以外の駒は、自分より強い相手の駒がいくつあるか（0-6）の 7 通りと、同じ強さの相手の駒がいくつかあるか（0-2）の 3 通りの組み合わせで 21 通りとした。ウサギの駒は、自分より強い相手の駒がいくつかあるか（0-8）の 9 通りと、相手のウサギの駒がいくつあるか（0-8）の 9 通りの組み合わせで 81 通りとした。

残っているウサギの数

ウサギの駒は最初は 8 体いるものの、自分のウサギが 0 体になってしまうと、勝つことはほぼ不可能になってしまう。そのため、残っているウサギの数が少なくなればなるほどウサギの価値は上がると考えられる。A そのためウサギの数（0-8）を特徴量に利用した。

駒の種類とその駒のいる段、マス

12 種類に分類したそれぞれの駒が、いる段、マスを特徴量として利用した。マスについては左右は線対称で等しくなるようにした。駒の存在するマスが決まればその駒のいる段も一意に定まるため冗長にも思える特徴量ではあるが、各駒のいるマスというのは教師データ中に出現する頻度も少ないため、うまく学習できないことも考えられる。しかし比較的出現頻度の高い各駒のいる段も特徴量として加えることで、教師データの数が少なくなりやすいという問題点を緩和することができる。

表 4.3. 勝率と一致率

評価関数	勝率		一致率	
	vs Faerie	vs 静止探索なし	ターン	ステップ
Faerie	50.0%		7.8%	21.8%
静止探索なし	56.7% (565 - 433)	50.0%	8.1%	22.2%
静止探索あり	72.1% (1440 - 558)	71.3%(1425 - 573)	9.2%	23.2%

トラップのコントロール

トラップの周囲にいる 4 マスにいる駒の数とそれぞれの強さを特徴量として利用した。また、自分より強い相手の駒が存在しない、押されることも引かれることもない駒がその中に含まれているかどうかを特徴量として利用した。

あと何ステップでゴールできるか

ウサギの駒があと少なくとも何ステップあればゴールできるかを調べ、特徴量として利用した（最大連続 9 ステップ）。ウサギの駒を動かすステップ以外に、味方の FREEZE している駒に隣接するステップ、ウサギの前方にいる駒を動かすステップを考慮して求めた。

最も強いの駒の移動可能範囲

自分より強い相手の駒が存在しない駒（大抵の場合象の駒）が、自由に動き回ることができない場合、トラップをめぐる攻防に大きな制約がかかってしまう。そのため、その最も強い駒が 3 ステップあった場合に移動可能なマスの数、4 ステップあった場合に移動可能なマスの数を特徴量として利用した。

4.3.2　実験結果

学習データ 186,775 局面に対して学習のイテレーションを静止探索を行う場合、静止探索を行わない場合に対してそれぞれ 5 回行った。学習時間は静止探索を行わない場合で 20 日、静止探索を行った場合で 30 日ほどであった。CPU は Intel Xeon CPU X5680 3.33GHz を用いた。

まず、学習回数に対する勝率、一致率の変化を調べた。Faerie に対する 200 戦の勝率を図 4.5、図 4.6、学習した評価関数のテストデータに対する一致率の変化を図 4.3.2、図 4.3.2、図 4.3.2、図 4.3.2 に示す。

実験の結果を表 4.3 に示す。静止探索なしで学習した評価関数、静止探索を用いて学習した評価関数は共に Faerie に対して有意に勝ち越している。棋譜を利用して学習を行なうことで、 Arimaa の有用な評価関数を学習することができたと言える。

静止探索なしで学習した評価関数は、テストデータとの一致率において Faerie と大きな差は見られなかった。一般にゲーム AI の分野においては、棋譜との一致率が向上すれば強さも同時に向上することが多いが、強さは向上したが棋譜との一致率は向上しない（あるいはその逆で一致率は向上したが強さが向上しない）ということがしばしば発生する。その理由としては

- 最善手を指す確率は変化しなかったものの負けに直結するような大きなミスが減少したから
- 最善手と評価値がほぼ変わらない次善手が存在するから
- 一致率を見るための棋譜データにおいて指されている手がそもそも最善手ではなかったから

などの理由が考えられる。実際に AI が指した指し手や棋譜の指し手を確認すればどの理由で強さの向上が見られたのか確認することはできるが、筆者の Arimaa の実力が低いため判断ができなかった。

静止探索ありで学習した評価関数は、Faerie に対しても、静止探索なしで学習した評価関数に対しても大きく勝ち越しており、学習時に静止探索を行なうことが非常に効果的であったことが分かる。Arimaa においては 1 手先まで読むというのが 4 ステップ分の探索を行なうことと同義であるため、静止探索なしの方においてもある種の浅い探索を行っているということができるが、1 手先まで読むだけでは不十分であったと考えられる。一致率に関しても、Faerie、静止探索なしよりもはるかに高い値を示しており、上級者が指す指し手を真似ることに成功しており、その結果強さが向上していることが分かる。静止探索なしではこのような一致率の向上は見られなかったことから、1 手分先読みするだけでは上級者の指し手をうまく学習することができておらず、指し手の意味を理解できていなかったと考えられる。静止探索を利用することで、ある程度高速に先読みを実現することができ、それによって棋譜の指し手の意味を理解して学習を行なうことができ、強さが向上したと考えられる。

4.3.3 考察

今回の実験では静止探索を加えることによって学習の精度が大きく向上した。この結果は、“相手の駒を取る” という行為が Arimaa における短期的な戦術として有用であったためと考えれる。しかし、“相手の駒を取る” といったような分かりやすい「静止探索に利用すべき手」が分からない（もしくは存在しない）ゲームに対してどのように応用すべきかは定かではない。

また、今回の実験では相手の駒を取ることに関連するステップのみを生成し、相手の駒を取る指し手を生成して静止探索を行なうことを目指したが、FREEZE している味方の駒に隣接して移動可能な状態にする、移動の妨げとなっている自分の駒を移動するなどのステップを行わなければ相手の駒を取れない状況が存在するため、相手の駒を取る手の全てを生成しているわけではない。しかし、相手の駒を取る手の全てを生成すると、最初の 2 ステップで相手の駒を取ることが可能な場合に残りの 2 ステップのパターンが膨大になってしまい、静止探索の効率が悪化してしまうなどの問題点が考えられる。また、静止探索中に味方の駒が取られることを防ぐ手を生成するためには、“味方の駒が取られることを防ぐ手”であるかどうかを識別する方法が必要となってくる。これには相手の手番に駒が取る手が存在するかどうか、指し手の生成を行って確認する必要があるため、計算に時間がかかると思われる。

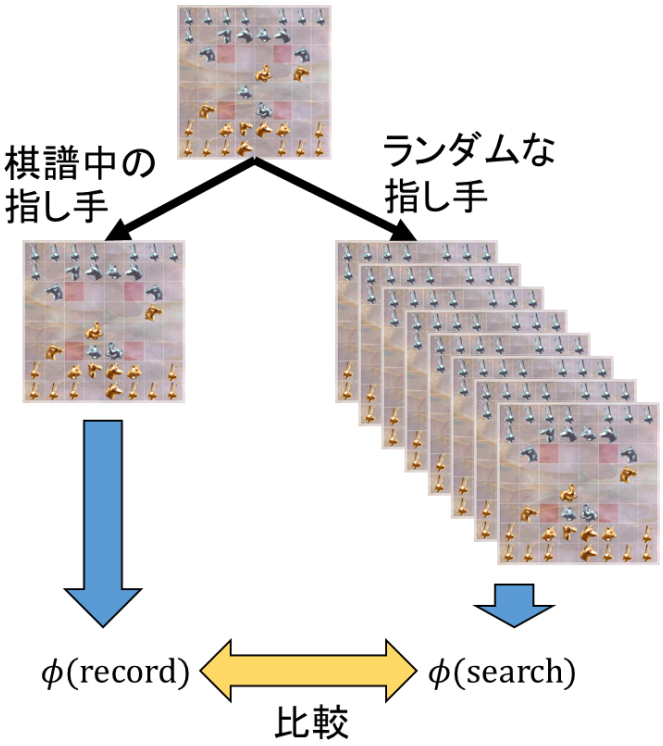

図 4.1. 探索を行わない学習

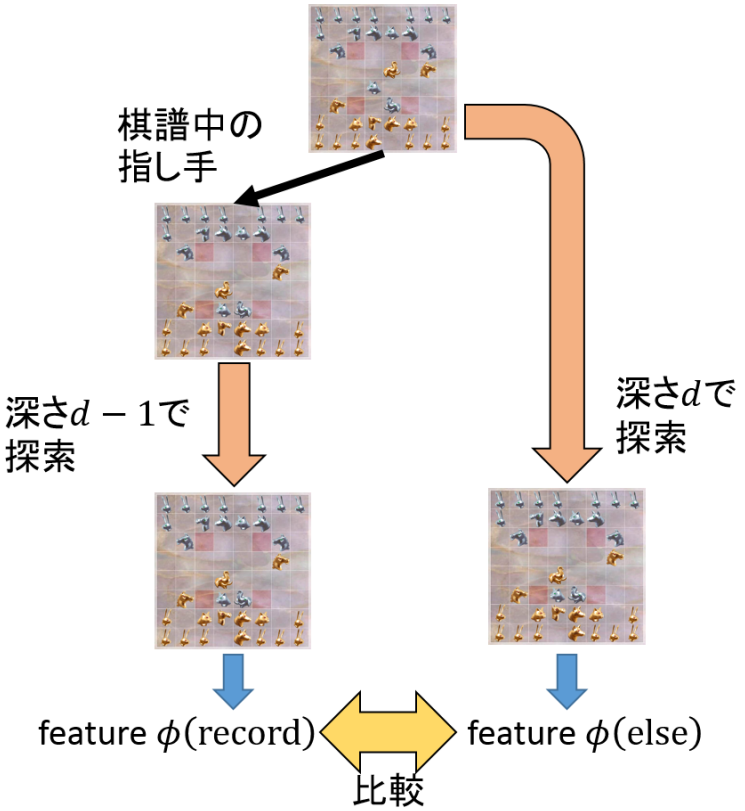

図 4.2. 探索を用いた学習

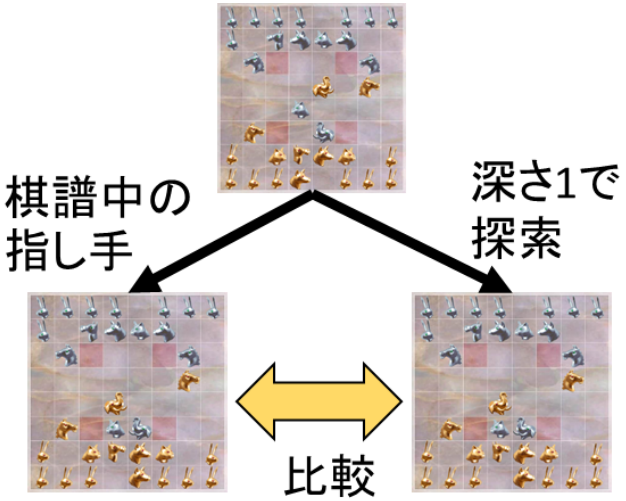

図 4.3. 静止探索を用いない学習

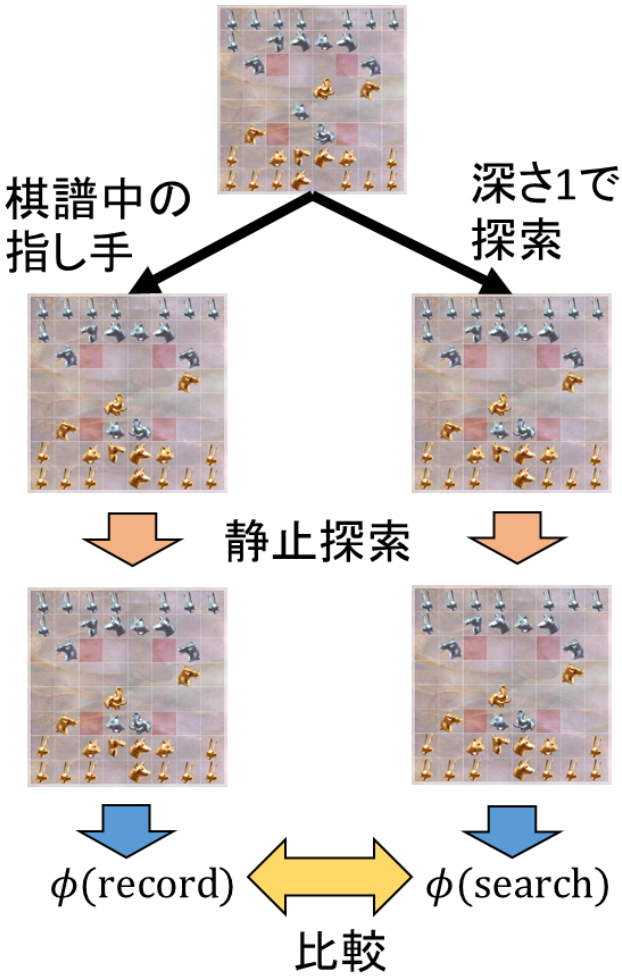

図 4.4. 静止探索を用いた学習

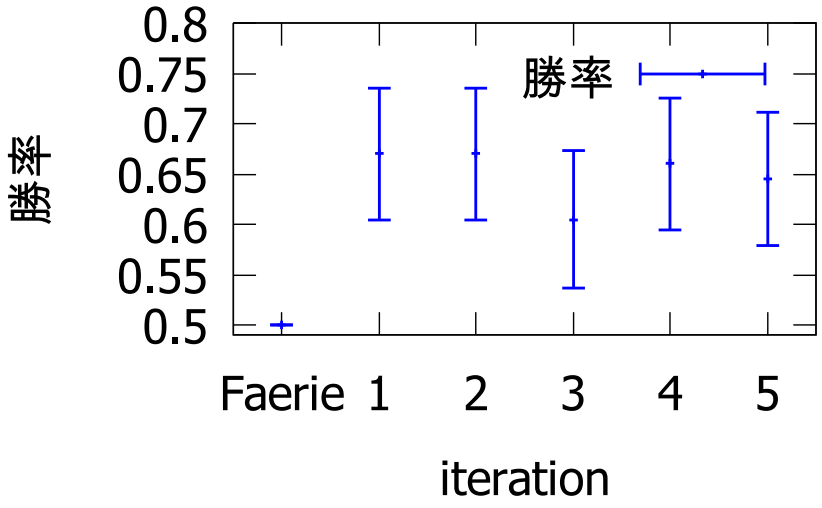

図 4.5. Faerie に対する勝率：静止探索なし

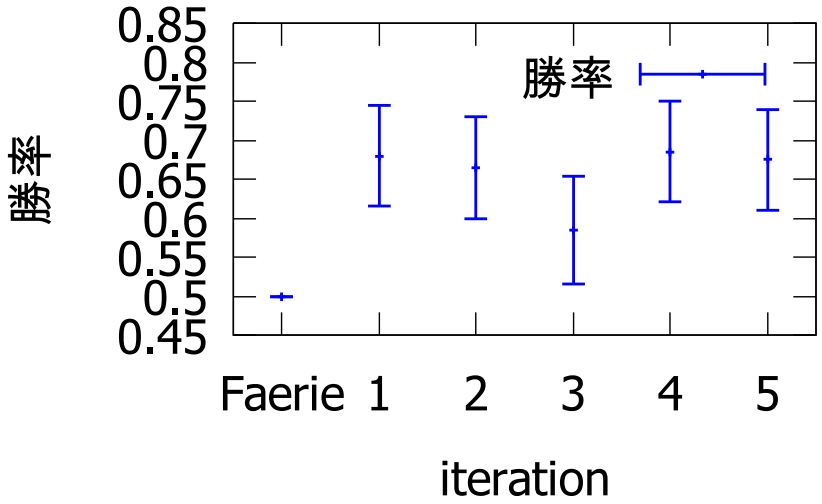

図 4.6. Faerie に対する勝率：静止探索あり

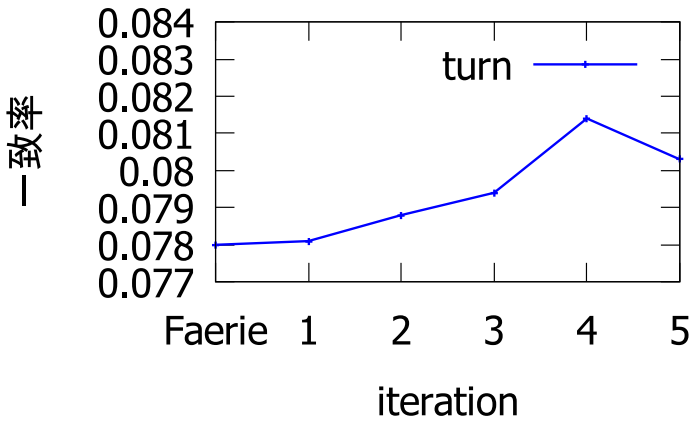

図 4.7. テストデータに対する一致率（ターン）：静止探索なし

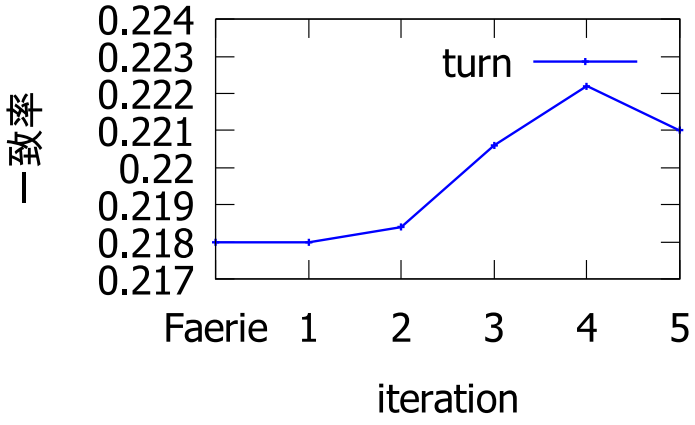

図 4.8. テストデータに対する一致率（ステップ）：静止探索なし

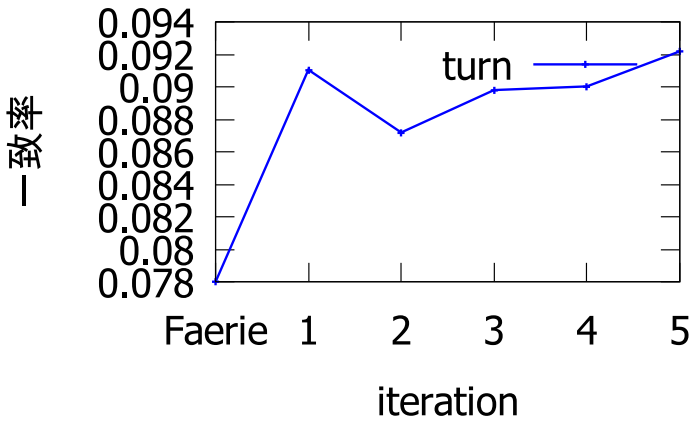

図 4.9. テストデータに対する一致率（ターン）：静止探索あり

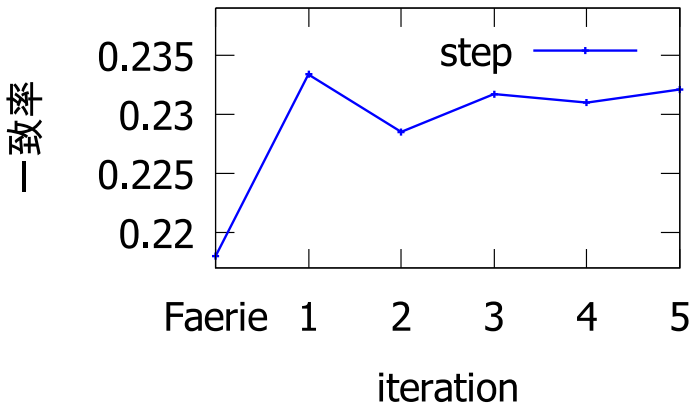

図 4.10. テストデータに対する一致率（ステップ）：静止探索あり

第5章　おわりに

5.1　本研究のまとめ

本稿では、 Arimaa の評価関数の精度を高めるために、Arimaa の評価関数を比較学習を用いて学習することを目指した。

まず、予備実験として Faerie で用いられている特徴量と同じ特徴を用いて実験を行った。その結果から、学習時に探索を行おうとすると大きな時間がかかり学習に利用できる局面が限られてしまうこと、Faerie で用いられている特徴量のみでは特徴量が不足しておりうまく学習を行なうことが難しいことが分かった。

探索の時間の問題を解決するために、Arimaa における静止探索を実装した。その評価を行ったところ、静止探索を行なうことで先の局面との評価値の差が小さくなるようなことは起きなかった。

Faerie で用いられているものに他の特徴量を加えて、再度評価関数の比較学習の実験を行った。静止探索を行わずに比較学習を行った場合と、静止探索を用いて比較学習を行う場合の二つの手法について評価を行った。実験の結果、探索を行わない場合と静止探索を行った場合の共に、学習前の評価関数よりも良い評価関数を学習することができた。特に、静止探索を行った場合では勝率、一致率ともに大きく向上しており、静止探索を行なうことにより、評価関数の学習がうまくいくということが分かった。静止探索を利用することで、棋譜の指し手の意味をうまく理解して学習できたと考えられる。

5.2　今後の展望

今回の実験で行った静止探索は Arimaa の特徴を利用したヒューリスティックな実装である。機械学習を利用することで、静止探索中に探索するべき指し手と探索する必要のない指し手とをうまく分類することが可能になれば、ヒューリスティックを用いずに静止探索を行なうことが可能になる。また、ヒューリスティックを用いない静止探索が実現できれば、Arimaa 以外の他のゲームで比較学習を利用する際にも、そのゲームに応じた静止探索を行なうことができるようになる。

また、今回の実験では評価関数の改善を目指し、探索部分に関する改善は行わなかった。強い Arimaa プログラムを作ることを目指すためには、探索部分の改善は不可欠となってくる。探索部分の改善を行っていくことにより、より強い Arimaa プログラムが作成可能になると思われる。探索部分の具体的な改善案としては、まず静止探索を対戦時に用いることが考えられる。今回の実験では作成した静止探索は評価関数を学習するときのみ用いており、対戦実験時に静止探索を利用した対戦は行っ

ていない。静止探索を学習時だけでなく対戦時においても利用することで、対戦時での性能が上がることが期待される。

探索の改善手法として、探索時にゲーム木の枝刈りを行なうことが考えられる。今回の実験では、Minimax 探索と最終的な結果が変わらない後ろ向き枝刈りと呼ばれる種類の枝刈りしか行っていない。ゲーム木の枝刈りは大きく二つに分類でき、一つが Minimax 探索の結果に影響を与えない枝を刈る後ろ向き枝刈り、もう一つが有望ではないと考えられる手は例え Minimax 探索の探索結果に影響を与える可能性が存在しても枝刈りしてしまう前向き枝刈りである。Arimaa では各ターンの合法手が非常に多いため、それに伴い有望でない合法手の数も非常にたくさんあると考えられる。それらをうまく枝刈りすることができれば探索の高速化に繋がるが、適当に枝刈りを行ってしまうと有望な指し手までも枝刈りが発生してしまい、逆に悪影響をもたらす結果になってしまう。うまく高速に悪手とそうでない手を見極めることができれば、良い前向き枝刈りが可能になる。

参考文献

[1] Omar Syed. Arimaa - Intuitively simple ... intellectually challenging. http://arimaa.com.

[2] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM J. Res. Dev.*, Vol. 3, No. 3, pp. 210–229, July 1959.

[3] Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, Vol. 3, No. 3, pp. 210–229, 1959.

[4] 善行小谷. 第 3 回将棋電王戦を振り返って：3．コンピュータ将棋の棋力の客観的分析-人間のトップに到達したか？-. 情報処理, Vol. 55, No. 8, pp. 851–852, jul 2014.

[5] Haskin Brian. A look at the arimaa branching factor. http://arimaa.janzert.com/bf_study/, 2006.

[6] 松原仁, 半田剣一. ゲームとしての将棋のいくつかの性質について. 情報処理学会研究報告. 人工知能研究会報告, Vol. 94, No. 83, pp. 21–30, oct 1994.

[7] Gerald Tesauro. Connectionist learning of expert preferences by comparison training. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems 1*, pp. 99–106. Morgan-Kaufmann, 1989.

[8] Gerald Tesauro. Comparison training of chess evaluation functions. In Johannes Fürnkranz and Miroslav Kubat, editors, *Machines that learn to play games*, pp. 117–130. Nova Science Publishers, Inc., Commack, NY, USA, 2001.

[9] 保木邦仁. 局面評価の学習を目指した探索結果の最適制御. 第 11 回ゲームプログラミングワークショップ 2006, pp. 78–83, 2006.

[10] 金子知適. コンピュータ将棋の評価関数と棋譜を教師とした機械学習 (<レクチャーシリーズ>コンピュータ将棋の技術〔第 5 回〕). 人工知能学会誌, Vol. 27, No. 1, pp. 75–82, Jan 2012.

[11] Jonathan Schaeffer, Neil Burch, Yngvi Björnsson, Akihiro Kishimoo, Martin Müller, Robert Lake, Paul Lu, and Steve Sutphen. Checkers is solved. *science*, Vol. 317, No. 5844, pp. 1518–1522, 2007.

[12] David Jian Wu. *Move Ranking and Evaluation in the Game of Arimaa*. PhD thesis, Harvard University, 2011.

[13] 矢野友貴, 三輪誠, 横山大作, 近山隆. ゲーム構成要素を組み合わせた特徴の最適化. 第 15 回ゲームプログラミングワークショップ 2010, pp. 15–22, Nov 2010.

[14] Vivek Choksi, Neema Ebrahim-Zadeh, and Vasanth Mohan. Leveraging game phase in arimaa. 2013.

[15] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, Vol. 65, No. 6, pp. 386–408, 1958.

[16] Yoav Freund and Robert Elias Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, Vol. 37, No. 3, pp. 277–296, Dec 1999.

[17] 原一之, 岡田真人. マージンを用いた単純パーセプトロン学習法のオンラインラーニングの理論. 電子情報通信学会技術研究報告. NC, ニューロコンピューティング, Vol. 101, No. 534, pp. 51–57, Dec 2001.

[18] Larry R. Harris. The heuristic search and the game of chess - A study of quiescene, sacrifices, and plan oriented play. In *Advance Papers of the Fourth International Joint Conference on Artificial Intelligence, Tbilisi, Georgia, USSR, 3-8 September 1975*, pp. 334–339, 1975.

[19] Haskin Brian. Opfor - the ”opposing force” bot, information page. http://arimaa.janzert.com/opfor/.

[20] 鶴岡慶雅. 将棋プログラム「激指」のページ. http://www.logos.t.u-tokyo.ac.jp/˜gekisashi/.

[21] 鶴岡慶雅. 3 選手権優勝記 : 激指の技術的改良の解説 (<ミニ特集>コンピュータ将棋の不遜な挑戦). 情報処理, Vol. 51, No. 8, pp. 1001–1007, Aug 2010.

[22] Arzav Jain, Neema Ebrahim-Zadeh, Vasanth Mohan, and Vivek Choksi. Move ranking in Arimaa, 2013.

[23] David Wu. Designing a winning arimaa program. *International Computer Games Association Journal*, Vol. 38, No. 1, pp. 19–40, 2015.

[24] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, Vol. 529, No. 7587, pp. 484–489, 2016.

[25] Tomas Kozelek. Methods of MCTS and the game arimaa. *Charles University, Prague, Faculty of Mathematics and Physics*, 2009.

[26] Tomáš Jakl and Tomáš Jakl. Arimaa challenge–comparission study of MCTS versus alpha-beta methods. *Memory*, Vol. 37, pp. 37–5, 2011.

[27] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, Vol. 3, No. 1, pp. 9–44, 1988.

[28] Jonathan Baxter, Andrew Tridgell, and Lex Weaver. Learning to play chess using temporal differences. *Machine Learning*, Vol. 40, No. 3, pp. 243–263, 2000.

[29] Joel Veness, David Silver, Alan Blair, and William W Cohen. Bootstrapping from game tree search. In *Advances in neural information processing systems*, pp. 1937–1945, 2009.

[30] Thitipong Kanjanapa, Komiya Kanako, and Kotani Yoshiyuki. Design and implementation of bonanza method for the evaluation in the game of arimaa. Technical Report 4, Department of Computer and Information Sciences, Tokyo University of Agriculture and Technology, feb 2012.

[31] Tomáš Hřebejk. Arimaa challenge-static evaluation function. Master's thesis, Charles University in Prague, 2013.

本研究に関する発表文献

(1) 川上裕生, 鶴岡 慶雅. 多様な特徴量を用いた Arimaa 評価関数の比較学習. ゲームプログラミングワークショップ 2014 論文集, 2014, 151-156.

(2) 川上裕生, 鶴岡 慶雅. 静止探索を用いた Arimaa 評価関数の比較学習. ゲームプログラミングワークショップ 2015 論文集, 2015, 69-76.

謝辞

本研究を進めるにあたり、多くの方々のお世話になりました。

指導教員の鶴岡慶雅准教授には、卒論生の頃から 3 年間に渡ってご指導いただきました。研究室ミーティングでは、研究の方針などに関して多くのアドバイスを頂きました。毎回締め切り直前まで論文の添削に付きあっていただき、ありがとうございました。私の力不足のため、多くのご迷惑をおかけしてきたように思いますが、心から感謝しております。

先輩方をはじめとする研究室の方々には、ミーティングでのアドバイスのみならず、研究室生活でも大変お世話になりました。

また、私を金銭的・精神的に応援し支えてくれた家族には大変感謝しています。

この場をお借りして皆様に厚くお礼申し上げます。