Vol.2014-GI-32 No.4
2014/7/5

一般ゲームからの知識獲得による盤面評価関数の自動生成

佐藤　祐一郎[1,a)]　飯田　弘之[1,b)]

概要：本稿では一般ゲームにおいて単純なゲームから知識を獲得し，より複雑なゲームに応用する手法について報告する．単純なゲームとして Tic-tac-toe を採用した．そのランダム着手によるゲームのシミュレーションから，勝利条件である「線」という知識を獲得した．獲得した知識を Connect4 や Breakthrough などの，より複雑なゲームの盤面評価関数を作るために応用し，結果を評価した．

キーワード：一般ゲームプレイング，知識獲得

Automatic Generation of Heuristics Function by Knowledge Acquisition from General Games

Sato Yuichiro[1,a)]　Iida Hiroyuki[1,b)]

Abstract: In this paper, we propose algorithms to extract explicit concepts from general games and these concepts. We extract explicit concepts of line from Tic-tac-toe simulation. These concepts are available to evaluate game positions of Connect4 and Breakthrough.

Keywords: General Game Playing, Knowledge acquisition

1. イントロダクション

近年のゲーム AI の発展はめざましく，様々なゲームで人間のトッププレイヤーに勝てるプレイヤーが作られている．だが多くのゲーム AI は特定のゲームのみをプレイできるだけで，人間のように様々なゲームから得た経験や知識を新しいゲームに応用し，臨機応変に物事に対応することはできない．特定のゲームに特化することなく，一般的にあらゆるゲームをプレイできるプレイヤーを作ることを目的とした研究分野が General Game Playing(GGP) である．

GGP は Pell の研究に端を発する [1]．この研究は，General Game を定義するための言語である Game Description Language(GDL) や，General Game をプレイできる Metagamer，General Game の盤面評価関数を作るための特徴量などについて付言しており先進的な研究だった．

その後の研究によって，全てのゲームに利用できる特徴量の発見や，GDL についてのさらなる研究，UCT モンテカルロ法の利用などが行われた．GGP には一人で行うゲームとしてパズルが含まれており，プラニング問題を含んでいる．よって人工知能研究の研究対象としてよいものであると期待できる．2005 年から GGP のコンペティションが AAAI にて行われるようになり，この分野の研究はさらなる盛り上がりを見せている [2].

私達は GGP のために小さなゲームから概念を学習し，それを大きなゲームに応用するようなゲーム AI を作ろうと試みた．そのために，ゲームの盤面を評価するための盤面評価関数を，そのゲームの対局中，ゲーム AI の実行時にコンパイルするシステムを作ることを目標とし，予め学習しておいた General Game に対する概念を利用した盤面評価関数を作り，性能を評価した．

[1] 北陸先端科学技術大学院大学
JAIST, Nomi, Ishikawa 923–1292, Japan
[a)] sato.yuichiro@jaist.ac.jp
[b)] iida@jaist.ac.jp

2. General Game Playing の概要

General Game Player は GDL で記述されたゲームを受け取り，ゲームをプレイする．ゲームの進行は Game Master と呼ばれるサーバによって制御され，各プレイヤーは Game Master と HTTP 通信を通じてやりとりすることでゲームをプレイしていく．Game Master はここから手に入る [3].

2.1 Game Description Language の概要

Game Description Language とは General Game を定義するための言語である．ゲームを定義するには，初期状態やゴール，合法手などを定義してやる必要がある．基本的な考え方は Pell の研究にあるが，その後より体系だった研究がなされた [4]．GDL は有限で離散的な完全情報ゲームを記述するための言語で，次のようなキーワードがある．

(1) `role(p)` ：p がプレイヤーであることを表す
(2) `init(f)` ：f はゲームの初期盤面において真であることを表す
(3) `true(f)` ：f が現在の盤面において真であることを表す
(4) `dose(p,m)` ：プレイヤー p が m という着手を行うことを表す
(5) `next(f)` ：次の盤面に f が真となるとこを表す
(6) `legal(p,m)` ：プレイヤー p にとって m が合法手であることを表す
(7) `goal(p,v)` ：現在の盤面が terminal である場合，プレイヤー p は v という得点を得ることを表す
(8) `terminal` ：現在の盤面がゲームの終わりであることを表す

近年には，不完全情報ゲームを扱えるよう拡張した GDL-II という言語も開発されている [6], [7].

2.2 GDL Reasoner の概要

GDL は Prolog のサブセットなので Prolog 上で動かすことができる．GDL Reasoner の役割は各プレイヤーの行う探索や学習のためにゲームをシミュレーションすることにある．特定のゲームだけに特化したプレイヤーなら盤面の遷移のコントロールや合法手の問い合わせは専用の関数を作ることで行えるが，GGP ではそうはいかない．GDL を読み込んで，スクリプト的に実行する必要がある．様々な GDL Reasoner があるが，代表的な物の性能評価はここに報告されている [5].

3. 代表的アプローチ

General Game に対するアプローチは大まかに，知識ベースアプローチとシミュレーションベースアプローチに分けられる．知識ベースアプローチとは，特徴量を用いて盤面を評価し，α-β アルゴリズムなどの探索アルゴリズムを利用してゲームをプレイするアプローチである．それに対しシミュレーションベースアプローチとは，UCT モンテカルロ法などにより，シミュレーションによって盤面を評価しゲームをプレイするアプローチである．

それとは他に，多人数のプレイヤーを持つゲームにも対応しなくてはならない．多人数のゲームをプレイするためには paranoid algorithm を使えばよいことが報告されている [8]．paranoid algorithm とは自分以外のプレイヤーが全員自分を倒すためにプレイすると仮定して手を選ぶアルゴリズムである．

3.1 知識ベースアプローチ

まずは各プレイヤーの合法手の数や初期状態からの距離などの，どのようなゲームであれ持っていると期待できる特徴量を使って盤面評価関数を作ることが試みられた [9]．ゲームの対称性に着目した研究もある [10]．これはシステム設計者が特徴量を見つける必要があるアプローチである．

他にはファジー論理を利用する試みがなされている．たとえば Fluxplayer は評価関数にファジー論理を取り入れている [11]．ある盤面がゴール状態にどれだけ近いかをファジー論理によって計算しようというものである．そのために succ 関係という，自然数の構造を持つ概念も利用している．

ニューラルネットワークによって知識を獲得しようという研究も行われている [12]．また，ニューラルネットワークの持つ学習能力を利用するため，ファジー論理で表現された知識をニューラルネットワークへと変換する研究も報告されている [13].

3.2 シミュレーションベースアプローチ

これらのアプローチとは別に，シミュレーションを利用して盤面を評価し，ゲームをプレイしようというアプローチもある [14]．これは UCT モンテカルロ法を使ったアプローチで，GGP コンペティションで優勝経験のある多くのプレイヤーが採用している手法である．モンテカルロ法によるシミュレーションの効率を上げるため，様々な手法が研究されている [15], [16], [17], [18].

4. General Game からの知識獲得

私達は知識ベースのアプローチを用いており，General Game のシミュレーションから様々な概念を学習する手法を研究し報告している [19]．本稿では提案したアルゴリズムは変わらないが，実行速度を向上させる手法を試みた．

4.1 提案手法

私達は General Game のランダムな着手によるシミュ

レーションから，勝ちで終わる盤面に多く含まれるパターンを取り出し，Tic-tac-toe のような小さなゲームから学んだパターンを Connect4 のようなより大きなゲームへ応用する手法を提案した．私達はこれらのパターンが General Game における概念であると考えている．だが，従来手法では学んだ概念を利用するためにファイル入出力を通して Prolog とやり取りする重い処理をしないといけなかった．その処理を高速化するために，学んだ概念を実行可能な Lisp プログラムに変換することを試みた．

Tic-tac-toe では，どちらかのプレイヤーが自分のシンボルで盤面上に線を作れば勝ちとなる．線という概念を Prolog で書くと次のようになる．

```
line(X,Y,Z) :- cell(X,Y,Z), Y2 is Y + 1,
  cell(X,Y2,Z), Y3 is Y + 2, cell(X,Y3,Z).
```

ただし X と Y は数値で座標を表し，Z はシンボルで，そのマス目があるプレイヤーの陣地であることを表す．このような盤面に現れる概念は Prolog のコードとして表現できるが，それは本質的な手法ではない．本質的なのは General Game に現れる概念はプログラムとして表現できるということである．よって，これを S 式で表現すれば次のようになる．

```
(concept line (variables p0 p1 p2)
  (and
    (piece x0 x1 x2 x3)
    (piece x0 x1 (+ x2 1) x3)
    (piece x0 x1 (+ x2 2) x3)))
```

ただし $x0$ は Prolog における述語の名前であり，盤面の中にその名前の駒があることを表す．また $x1$ と $x2$ は座標を表す数値，$x3$ はシンボルである．このような形式で概念を表現しておけば，必要に応じて Prolog のコードや Lisp のコードに自由に変換できる．今回は JVM 上で動作する Lisp である Clojure に変換した．

4.2 性能評価

私達が [19] で提案した手法を用いて Tic-tac-toe から二項関係から成る概念を学習し，それを Connect4 と Breakthrough の盤面評価関数へ応用した．学習された概念は，例えば次のようなものである．

```
(concept concept0 (variables p0 p1)
  (and
    (piece x0 x1 x2 x3)
    (piece x0 x1 (+ x2 1) x3)))
```

このような概念が 25 個学習された．

盤面評価関数は対象のゲームのランダムな着手によるシミュレーションから，勝ちで終わったゲームだけを抜き出し，終局の盤面にどのパターンが何回現れるかを数え上げることで作った．単純な数え上げによって作った評価関数と，学習した概念に何回マッチするかによって作った評価関数の 2 種類を用意した．単純な数え上げとは，ゲームの盤面の中に現れるマスのペアをそのまま数え上げたものである．一方，概念を利用した評価関数は，終局の盤面に含まれるマスのペアを全通り作り，それに学習した概念が何回マッチするか数え上げた．各概念の評価値はペアの合計数と，マッチした回数の比である．瑣末なペアの影響を避けるため，単純な数え上げの場合は上位 1%，概念を利用した場合は上位 10%だけのペアや概念を使って評価関数を作った．単純な数え上げによって作った評価関数は 100 個の盤面から作った．一方，概念を利用した評価関数は，より素早く評価関数を作れることを示すため，10 個の盤面から作った．

評価実験はランダムな着手をするプレイヤーに対し，評価関数を利用したプレイヤーとの対戦によって行った．実験回数は Tic-tac-toe，Connect4，Breakthrough の各ゲームについて 100 回行った．単純な数え上げによる評価関数と，概念を利用した評価関数のそれぞれに対し，探索の深さが 1 の場合と 3 の場合の 2 通りの実験を行った．ただし，Breakthrough の探索の深さが 3 の実験においては，実験の時間を短縮するため 40 回だけ行った．評価結果は次のようになった．

表 1 Tic-tac-toe の盤面評価関数の性能評価

type	search depth	win(%)	lose(%)	draw(%)
random	0	59	32	9
simple	1	88	7	5
simple	3	93	1	6
concept	1	84	9	7
concept	3	94	5	1

表 2 Connect4 の盤面評価関数の性能評価

type	search depth	win(%)	lose(%)	draw(%)
random	0	55	44	1
simple	1	89	11	0
simple	3	99	11	0
concept	1	91	9	0
concept	3	94	6	0

表 3 Breakthrough の盤面評価関数の性能評価

type	search depth	win(%)	lose(%)	draw(%)
random	0	49	51	0
simple	1	99	1	0
simple	3	100	0	0
concept	1	71	29	0
concept	3	70	30	0

学習に利用したシミュレーションの数が 1/10 であるにも関わらず，概念を使った評価関数は単純な数え上げによる評価関数とほぼ同等な結果を出していることが分かる．これは，予め概念を学習しておき，それを後から利用する

ことで，新しいゲームに素早く対応することができることを示している．

今回利用した GDL 中では Tic-tac-toe と Connect4 の盤面上のマス目は cell と表現されているのに対し，Breakthrough のマス目は cellholds と表現されている．これでは従来手法の，概念を Prolog のコードとして表現する方法では Tic-tac-toe から学んだ知識を Breakthrough に応用することはできないが，提案手法ではそれができる．また，従来手法に比べ，大幅な処理速度の向上がみられたため，Connect4 よりさらに大きなゲームである Breakthrough でも評価実験が行えた．

5. 今後の課題

二項関係が得られれば，それを多項関係に拡張することができる．また，論理和や論理積，再帰を用いてより複雑な概念の形成も可能である．今後はこのような，より複雑な概念を獲得することを目標に研究を進める．

また，今回は得られた概念を Clojure へと変換したが，これは実装が容易だったためであり，本質的な理由はない．今後は Common Lisp や Prolog，C 言語へ変換した場合に実行速度がどうなるか，なども検討していく．

6. 結論

本稿では私達の開発した General Game からの概念獲得手法と比べ，より汎用的で実行速度の早い手法を提案した．

参考文献

[1] Pell, B.: *A strategic metagame player for general chesslike games,* AAAI (1994).
[2] Gensereh, M.: *General Game Playing: Overview of the AAAI Competition*
[3] http://www.general-game-playing.de/
[4] Love, N., Hinrichs, T., Haley, D., Schkufza, E., Genesereth, M.: *General Game Playing: Game Description Language Specification,* http://games.stanford.edu/readings/gdlspec.pdf (2008).
[5] Schiffel, S., and Björnsson, Y.: *Efficiency of GDL Reasoners,* Computational Intelligence and AI in Games (2014).
[6] Thielscher, M.: *GDL-II,* KI - Künstliche Intelligenz (2011).
[7] Schiffel, S., and Thielscher, M.: *Reasoning About General Game Described in GDL-II,* AAAI (2011).
[8] Sturtevant, R. N., and Korf, E., R.: *On Pruning Techniques for Multi-Player Games,* AAAI/IAAI (2000).
[9] Kaiser, M., D.: *The Design and Implementation of a Successful General Game Playing Agent,* FLAIRS Conference (2007).
[10] Banerjee, B., Kuhlmann, G., and Stone, P.: *Value Function Transfer for General Game Playing,* Proceedings of the ICML-06 Workshop on Structural Knowledge Transfer for Machine (2006).
[11] Schiffel, S., and Thielscher, M.: *Fluxplayer: A successful General Game Player,* AAAI (2007).
[12] Michulke, D., and Thielscher, M.: *Neural Networks for State Evaluation in General Game Playing,* Machine Learning and Knowledge Discovery in Databases Lecture Notes in Computer Science (2009).
[13] Michulke, D.: *Neural Networks for High-Resolution State Evaluation in General Game Playing,* The IJCAI-11 Workshop on General Game Playing (2011).
[14] Walędzik, K., and Mańdziuk, J.: *Multigame playing by means of UCT enhanced with automatically generated evaluation functions,* Artificial General Intelligence (2011).
[15] Gudmundsson, F., S., and Björnsson, Y.: *Sufficiency-Based Selection Strategy for MCTS,* IJCAI'13 (2013).
[16] Walędzik, K., and Mańdziuk, J.: *An Automatically-Generated Evaluation Function in General Game Playing* Computational Intelligence and AI in Games (2013).
[17] Świechowski, M., and Mańdziuk, J.: *Self-Adaptation of Playing Strategies in General Game Playing,* Computational Intelligence and AI in Games (2013).
[18] Méhat, J., and Cazenave, T.: *A parallel general game player,* KI - Künstliche Intelligenz (2011).
[19] Sato, Y., and Cazenave, T.: *Automated Generation of New Concepts from General Game Playing,* Communications in Computer and Information Science (2014).