完全情報ゲームとAND/OR木探索

岸本　章宏

人工知能における問題の多くは，ある問題をいくつかの簡単な部分問題に分解し，その部分問題の一つを解くOR手続きと，すべての部分問題を解くAND手続きを行うAND/OR木探索へのモデル化が可能である．完全情報ゲームは，このようなAND/OR木探索が利用できる代表的な分野である．本論文では，ゲームを対象として発展してきたAND/OR探索アルゴリズムの概要と今後の展望について解説する．特に，現状で最も強力である証明数と反証数を用いた手法を中心にして取り扱う．

キーワード：完全情報ゲーム，AND/OR木探索，証明数，反証数，df-pn

1. はじめに

1.1 人工知能研究と完全情報ゲーム

人工知能研究として，ゲームは最も成功した研究テーマの一つである．チェス・プログラムのDeep Blue[4]が，人間の世界チャンピオンであるGarry Kasparovを1997年に破ったことは周知の事実である[13]．また，コンピュータ将棋において，人間のトッププロに伯仲するプログラムが，これから10年以内に出現しても全く不思議ではない[21]．

チェスや将棋，囲碁などのゲームは，2人のプレイヤーで行い，片方のプレイヤーが勝つときはもう片方が負ける（零和）ことが保証され，さらに自分と相手の状態がすべて公開されている（完全情報）ので，**二人零和完全情報ゲーム**と呼ばれる．二人零和完全情報ゲーム（以下，単にゲームと呼ぶ）の研究には次のような意義がある．

- 研究の動機が明確である．ゲーム研究における最大の目標は，ゲームという知的な分野で人間に勝つことである．この目標は，プログラムの開発への強いモチベーションとなりうる．
- ゲームのルールは単純であり，勝ち，負け，引き分けの3つの結果しかない．よって，他の実時間システムに比べて，開発した手法の有効性を調べやすい．
- ゲームは，計算機科学における面白い課題である．強い人間に勝つシステムを開発するためには，現状の手法の組み合わせだけでなく，大きなブレイクスルーが必要である．

このような理由から，ゲームの研究は50年以上にわたり，数多くの研究者によって行われ，プランニングや学習アルゴリズム，探索アルゴリズムなど様々な分野に応用できる研究成果を生み出してきた．

1.2 完全情報ゲームとAND/OR木

プレイヤーが勝つための重要なプロセスの一つとして，ある局面の勝ち負けを判定することがある．ある局面で，先手勝ちを証明したければ，先手番の局面では，すべての指し手の中の一つが先手勝ちであることを示せば十分である．一方，後手番の局面では，指し手のすべてが先手勝ちであることを示す必要がある．このようなゲームの必勝手順の証明は，先手を**OR手続き**，後手を**AND手続き**とした**AND/OR木探索**問題に変換して，それを解くことによって，計算可能である．このモデル化では，ゲームの局面が探索木の節点であり，指し手が探索木の枝に対応する．

AND/OR木探索アルゴリズムは，ゲームの部分問題やゲーム自体を解くことを題材にして，進歩してきた．対象となったゲームは，欧米のゲームであるチェス[3]やチェッカー[12]だけでなく，日本でも有名な五目並べ[1]や詰将棋[8]，[14]，詰碁[17]，[7]など幅広い．当然のことながら，一つのゲームで開発された手法は，他ゲームにも利用されている．さらに，コンピュータ将棋[5]における詰将棋エンジンなどゲームを指すプログラムにも搭載されている．

1.3 本論文の構成

本論文では，著しく進歩を遂げたAND/OR木探索手法について概説する．本論文の構成は次の通りである．2節では，AND/OR木の定義を行う．3節では，探索アルゴリズムの進歩について説明する．4節では，

きしもと　あきひろ
公立はこだて未来大学 システム情報科学部
〒041-8655 函館市亀田中野町116-2

まとめと今後の課題について議論する．

2. AND/OR 木の定義

具体的な探索手法について述べる前に，本論文で取り扱うAND/OR木を定義する．AND/OR木には**OR節点**と**AND節点**がある．OR節点が先手番の局面であり，AND節点が後手番の局面である．各節点は，**真**，**偽**，**不明**の3つの値を持つ．真が先手勝ち，偽が後手勝ちまたは引き分けに相当する．子節点を持つ節点を**内部節点**と呼ぶ．子節点を持たず，真か偽の値を持つ節点を**終端節点**と呼ぶ．終端節点とは，ゲームの決着のついた局面である．OR節点の子節点はAND節点であり，AND節点の子節点はOR節点である．未展開の節点を**先端節点**と呼ぶ．先端節点の値は不明である．**ルート節点**は，親節点を持たない節点である．本論文では，ルート節点は必ずOR節点であるとする．

AND/OR木の内部節点 n の値は，n の子節点の値によって，次のように定義される．

- n が内部OR節点のとき，n の値は：
  - 少なくとも1つの子節点が真ならば，真．
  - すべての子節点が偽ならば，偽．
  - それ以外のときは，不明．
- n が内部AND節点のとき，n の値は：
  - 少なくとも1つの子節点が偽ならば，偽．
  - すべての子節点が真ならば，真．
  - それ以外のときは，不明．

n を展開して，n の値が真であることを示すことを n を**証明**すると言う．逆に，n が偽であることを示すことを n を**反証**すると言う．n が証明されたときには，n が真であることを示す根拠である**証明木**を持つ．つまり，n の証明木が分かれば，n において必勝手順を指せる．n に複数の証明手段があるときには，証明木は複数存在する．同様に，n が偽であることの根拠である**反証木**を定義することができる．証明木は次のような性質を持つ．

1. n の証明木は n を含む．
2. n の証明木の各内部OR節点 m に対して，m の子節点の少なくとも一つが n の証明木にある．
3. n の証明木の各内部AND節点 m に対して，m の子節点のすべてが n の証明木にある．
4. n の証明木上の終端節点は必ず真である．

AND/OR木の例を図1に示す．この図では，D, G, H, I, J を終端節点とした．ルート節点 A の値は真であり，太線が A の証明木である．

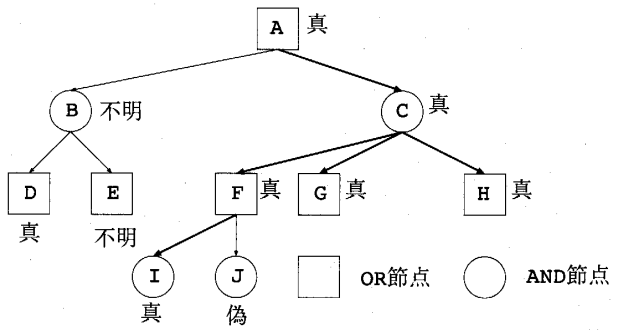

図1 AND/OR木の例

3. 探索アルゴリズムの研究動向

3.1 証明数・反証数に基づく最良優先探索

ルート節点の証明木か反証木を構築すれば，AND/OR木が解けたことになる．つまり，証明木や反証木の構築に関係ありそうな節点の探索を先に行えば，アルゴリズムの性能が向上する可能性が高い．Allisの**証明数**（proof number）と**反証数**（disproof number）[2]は最も成功している節点の選択戦略である．節点 n の証明数 $\mathbf{pn}(n)$ とは，n を証明するのに展開する必要のある先端節点の数の下限である．同様に，n の反証数 $\mathbf{dn}(n)$ とは，n を反証するのに展開する必要のある先端節点の数の下限である．$\mathbf{pn}(n)$ と $\mathbf{dn}(n)$ は，AND/OR木の性質より，次のように計算できる．ただし，$n_1, \cdots, n_k$ を内部節点 n の k 個の子節点とする．

- n が終端節点のとき
  $\mathbf{pn}(n)=0$, $\mathbf{dn}(n)=\infty$ （n が真）．
  $\mathbf{pn}(n)=\infty$, $\mathbf{dn}(n)=0$ （n が偽）．
- n が先端節点のとき，$\mathbf{pn}(n)=\mathbf{dn}(n)=1$．
- n が内部OR節点のとき
  $\mathbf{pn}(n)=\min(\mathbf{pn}(n_1), \cdots, \mathbf{pn}(n_k))$．
  $\mathbf{dn}(n)=\mathbf{dn}(n_1)+\cdots+\mathbf{dn}(n_k)$．
- n が内部AND節点のとき
  $\mathbf{pn}(n)=\mathbf{pn}(n_1)+\cdots+\mathbf{pn}(n_k)$．
  $\mathbf{dn}(n)=\min(\mathbf{dn}(n_1), \cdots, \mathbf{dn}(n_k))$．

図2に各節点の証明数と反証数の例を示す．各節点の左側の数値が証明数，右側が反証数である．

証明数・反証数を利用すると，最良優先探索である**証明数探索**（proof-number search）[2]を定義できる．この手法よって，キュービックや五目並べを解くことができた[1]．

証明数探索では，各プレイヤーが最善を尽くすと仮

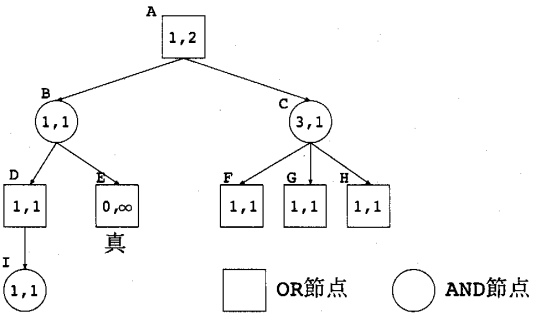

図2　証明数・反証数の例

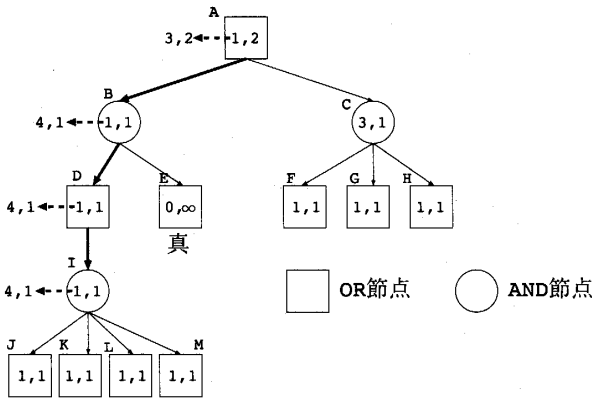

図3　証明数探索の様子

定する．ルート節点から先端節点に至るまで，OR節点では，証明数が最小の子節点を選択し，AND節点では，反証数が最小の子節点を選択を繰り返す．次に，この戦略で到達した先端節点を展開し，ルートからその先端節点の経路上にある節点の証明数と反証数を計算しなおす．このような先端節点の選択・展開作業を，ルート節点が解ける（証明数が0か∞である）まで繰り返す．

図3に証明数探索の動作を示す．この探索木では，ルート節点より，太線の経路を選び，Iを展開して，経路上にある証明数・反証数を再計算している．

3.2　証明数・反証数に基づく深さ優先探索

最近の研究では，最良優先探索の多くは，深さ優先探索に変換できることが知られている．深さ優先探索に変換することによって，最良優先探索の利点である有望な節点の優先的展開と，深さ優先探索の利点であるメモリ使用量の少なさを保持できる．

最良優先型のAND/OR木探索を深さ優先型にする試みは，脊尾によって最初に行われた[14]．この手法によって，1,525手という超長手順の問題が解ける強力な詰将棋ソルバーが作成された．

長井のdf-pnは，証明数探索と同じ振舞いをする深さ優先探索法である[8]．df-pnを利用して，脊尾の方法を凌駕する詰将棋ソルバーが開発された．df-pnは，詰碁[7]やチェッカー[12]にも利用されている．

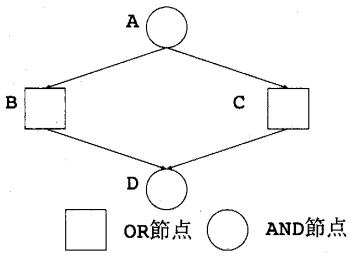

図4　証明数の二重カウント問題

df-pnは，深さ優先探索を行うために閾値を利用している．閾値には，証明数に対する閾値と反証数のための閾値の2種類がある．大まかには，証明数の閾値は，節点の証明がその閾値で可能かを調べるのに利用する．仮に，証明数の閾値が2で，節点nの探索を行うとする．この閾値では，nの証明数を2以上にすることなしにnの証明ができるかを調べる．閾値2で，nの証明や反証ができれば，nの探索は終了する．nの証明や反証ができなければ，証明数の閾値を増やして，nが証明できるかどうかを再び調べる．同様に，反証数の閾値を利用して，節点の反証がその閾値で可能かを調べる．

df-pnでは，様々な閾値で，同じ節点を何度も展開する．節点の再展開を減らすために，局面をハッシュ・キーとしたハッシュ表に証明数や反証数などの探索結果を保存している．ハッシュ表の利用によって，探索木の再展開の割合は全体の探索に比べて少ないことが実験的に知られている．例えば，詰碁ソルバーの再探索の割合は30%程度である[7]．

3.3　証明数・反証数の実用上の問題

前節までは，探索空間が木であることを仮定していた．多くのゲームでは，別手順で同一局面に至ることがある．また，繰り返し手順によって，元の局面に戻るサイクル手順が存在する．つまり，実際のゲームは，DAGやサイクルを含む空間である．

探索空間がDAGであるときには，証明数・反証数の**二重カウント問題**が生じる．証明数の二重カウントの例を図4に示す．この図では，Dを証明すればAが証明できるので，**pn**(A)＝**pn**(D)であるべきである．ところが，2節の証明数の計算では，**pn**(A)＝2×**pn**(D)になる．現状の対処法では，子節点の情報の一部のみを利用する経路分枝数[22]やハッシュ表に親局面へのポインタを一つ保持し，このポインタと違う経路で節点に到達した場合に二重カウントの可能性を調べる方法[8]などがある．

サイクルを含む空間では，df-pnは無限ループを引き起こすことがある．これは，節点nが自己に戻ることがあるときに$\mathbf{pn}(n)$を計算すると，$\mathbf{pn}(n)$自身を先天的に含んでしまうことに原因がある．また，ハッシュ表は探索経路を無視して結果を保存するために，**GHI問題**（Graph History Interaction Problem）というdf-pnが解答を時折間違える問題がある．岸本のdf-pn(r)アルゴリズムは，これらの2つの問題を解決している[6]．

3.4 その他の研究動向

3.4.1 先端節点の証明数・反証数の初期化

2節の定義では，先端節点の証明数と反証数は1であるが，実際には形勢が一方に傾いている局面が存在する．証明数・反証数を改良する自然な考え方として，先端節点の証明数と反証数に1以外の値を割り当てることが考えられる．つまり，ゲーム依存の知識によって証明数と反証数を推測する関数を$h_{\mathrm{pn}}(n)$と$h_{\mathrm{dn}}(n)$としたときに，先端節点nの証明数と反証数を次のようにすればよい．

$$\mathbf{pn}(n)=h_{\mathrm{pn}}(n),\ \mathbf{dn}(n)=h_{\mathrm{dn}}(n).$$

h_{pn}とh_{dn}は，証明や反証の予測を誤らない限りは，不正確な予測でも探索結果は正しい．h_{pn}やh_{dn}の決定には，開発者自身が記述する方法[3],[7]や学習による方法[9],[20],[19]，モンテカルロ法による方法[10]などの研究が盛んに行われている．

3.4.2 評価値を用いた反復法

節点nの真偽をほぼ正確に推測できる場合には，推測を利用した擬似的な証明を最初に作成し，徐々に正確な証明に改善する反復的な手法が利用可能である．Schaefferのチェッカーを解く手法[12]では，評価値の閾値thを用意する．この手法では，th以上の評価値を持つ局面を証明できたとみなし，$-th$以下の局面を反証できたとみなす．評価値には，チェッカーを指すプログラムChinookの探索結果を利用する．thには，最初は小さな値を設定する．このthで，ルートの証明や反証の終了後に，thの値を少しずつ大きくする．つまり，最初は後回しにしていた節点の証明や反証を徐々に正確に取り扱う．最初のルートの証明（反証）は，疑似証明（反証）であるが，$th=\infty$の探索終了後には，正しい証明（反証）が保証される．この手法によって，チェッカーの開始局面のいくつかを解くことができた．現在もチェッカーを解くプロジェクトは進行中である[11]．

3.4.3 脅威度を利用した手法

ゲームの局面で，ある手を指さなければ負けるなどという状況（**脅威度**（threat）と呼ばれる）では，その指し手のみを調べればよい．脅威度を利用して，指し手を限定することで，探索空間を大幅に減らすことができる．Thomsenの**λ探索**[16]は，自分が何回パスをしたときに相手が勝てるかという**λ順序**（λ order）を利用して，囲碁の石の捕獲問題の着手生成を制限している．λ探索とdf-pnを組み合わせる手法の研究が，将棋や囲碁で行われている[15]，[18]．

4. まとめと今後の課題

本論文では，ゲームを題材にしたAND/OR木探索手法の研究について概説した．証明数や反証数の概念やdf-pnに基づく手法により，AND/OR木探索アルゴリズムは非常に強力になり，様々なゲームで実用的に利用されている．その一方で，現状の手法では，ルート節点の証明木や反証木よりもはるかに大きな空間を探索してしまう．今後，ますます効率の良いアルゴリズムが開発されることを期待したい．また，最近の研究の多くは，証明数・反証数に基づいた手法の改良が中心である．もし，証明数・反証数を超える新しい概念の導入ができれば，探索技術が大きく進歩すると個人的には考えている．

謝辞 本論文の執筆にあたり，金子知適博士と副田俊介博士の有益な助言に感謝いたします．

参考文献

[1] L. V. Allis. *Searching for Solutions in Games and Artificial Intelligence*. PhD thesis, Department of Computer Science, University of Limburg, 1994.

[2] L. V. Allis, M. van der Meulen and H. J. van den Herik. Proof-number search. *Artificial Intelligence*, Vol. 66, No. 1, pp. 91-124, 1994.

[3] D. M. Breuker, L. V. Allis and H. J. van den Herik. How to mate: Applying proof-number search. In *Advances in Computer Chess* 7, pp. 251-272, 1994.

[4] M. Campbell, A. J. Hoane Jr. and F.-h. Hsu. Deep Blue. *Artificial Intelligence*, Vol. 134, No. 1-2, pp. 57-83, 2002.

[5] H. Iida, M. Sakuta and J. Rollason. Computer shogi. *Artificial Intelligence*, Vol. 1-2, pp. 121-144, 2002.

[6] A. Kishimoto. *Correct and Efficient Search Algorithms in the Presence of Repetitions*. PhD thesis, Department of Computing Science, University of Alberta,

2005.

[7] A. Kishimoto and M. Müller. Search versus knowledge for solving life and death problems in Go. In *Twentieth National Conference on Artificial Intelligence*, pp. 1374-1379, 2005.

[8] A. Nagai. *Df-pn Algorithm for Searching AND/OR Trees and Its Applications*. PhD thesis, Department of Information Science, University of Tokyo, 2002.

[9] A. Nagai and H. Imai. Application of df-pn$^+$ to Othello endgames. In *Game Programming Workshop in Japan '99*, pp. 16-23, Kanagawa, Japan, 1999.

[10] J.-T. Saito, G. Chaslot, J. Uiterwijk and H. J. van den Herik. Monte-Carlo proof-number search for computer Go. In *Computers and Games (CG '2006)*. To appear.

[11] J. Schaeffer. Chinook home page. http://www.cs.ualberta.ca/~chinook/.

[12] J. Schaeffer, Y. Björnsson, N. Burch, A. Kishimoto, M. Müller, R. Lake, P. Lu and S. Sutphen. Solving checkers. In *International Joint Conference on Artificial Intelligence*, pp. 292-297, 2005.

[13] J. Schaeffer and A. Plaat. Kasparov versus Deep Blue: The re-match. *International Computer Chess Association Journal*, Vol. 20, No. 2, pp. 95-101, 1997.

[14] M. Seo. The C* algorithm for AND/OR tree search and its application to a tsume-shogi program. Master's thesis, Department of Information Science, University of Tokyo, 1995.

[15] S. Soeda. *Game Tree Search Algorithms based on Threats*. PhD thesis, Graduate School of Arts and Sciences, University of Tokyo, 2006.

[16] T. Thomsen. Lambda-search in game trees—with application to Go. In *Computers and Games (CG 2000)*, Vol. 2063 of *Lecture Notes in Computer Science*, pp. 19-38. Springer, 2001.

[17] T. Wolf. Forward pruning and other heuristic search techniques in tsume Go. *Information Sciences*, Vol. 122, No. 1, pp. 59-76, 2000.

[18] K. Yoshizoe, A. Kishimoto and M. Müller. Lambda depth-first proof number search and its application to Go. In *Twentieth International Joint Conference on Artificial Intelligence*, 2007. To appear.

[19] 三輪誠. SVM を用いた将棋の詰みの予測とその応用. Master's thesis, 東京大学新領域創成科学研究科基盤情報学専攻, 2005.

[20] 金子知適, 田中哲朗, 山口和紀, 川合慧. 詰め将棋における df-pn$^+$ 探索のための, 展開後の証明数と反証数を予測する評価関数. 第 9 回ゲーム・プログラミングワークショップ, pp. 14-21, 2004.

[21] 瀧澤武信.「全幅探索」と学習による新感覚のコンピュータ将棋の成功とその高速アルゴリズムの及ぼす影響. 情報処理, Vol. 46, No. 8, pp. 875-881, 2006.

[22] 岡部文洋. 経路分枝数を用いた詰め将棋解図について. 第 10 回ゲーム・プログラミングワークショップ, pp. 9-16, 2005.