Vol. 43 No. 6 情報処理学会論文誌 June 2002

df-pn アルゴリズムの詰将棋を解くプログラムへの応用

長井 歩[†] 今井 浩[†]

詰将棋を解くプログラムの研究はこの 10 年の間に大きく進歩した．その原動力となったのは，証明数や反証数という概念の導入である．詰将棋に適用すると，直感的にいうと，証明数は玉の逃げ方の総数を，反証数は攻め方の王手の総数を表す．前者は攻め方にとって，後者は玉方にとって非常に重要な値である．証明数・反証数を対等に扱った，最もナイーブなアルゴリズムは，Allis による pn-search という最良優先探索法である．我々は近年，df-pn アルゴリズムという，pn-search と同等の振舞いをする深さ優先探索法を提案している．この論文では，df-pn アルゴリズムを用いて詰将棋を解く強力なプログラムを作成し，その過程で導入した様々な技法を提案する．これらの技法を df-pn の上に実装することにより，我々のプログラムでは 300 手以上の詰将棋のすべてを解くことに初めて成功した．しかもそれは，シングルプロセッサのワークステーションで解くなど，解答能力と解答時間の両面で優れた結果を出すことができた．

Application of df-pn Algorithm to a Program to Solve Tsume-shogi Problems

Ayumu Nagai[†] and Hiroshi Imai[†]

During this decade, a study of programs to solve Tsume-Shogi problems has greatly advanced. This is due to the development of the concept of a proof number and a disproof number. Allis' pn-search is the most naive best-first algorithm that uses both proof numbers and disproof numbers on equal terms. We already developed a df-pn algorithm which is a depth-first algorithm that behaves the same as pn-search. In this paper, we applied df-pn algorithm to a program solving Tsume-Shogi problems. Moreover, we propose some techniques which we imported during implementing the program. As a result, by these techniques implemented on df-pn, our program solved all the Tsume-Shogi problems, for the first time, that require over 300 plies to reach to the checkmate.

1. はじめに

詰将棋の問題を解くのは，詰むか詰まないかのみを興味の対象とした探索である．このように，最終的な評価値が 2 種類しか存在しないような 2 人ゲームの探索は，人工知能研究の分野では，AND/OR 木の探索として一般化できる．AND/OR 木には，AND ノードと OR ノードという 2 種類のノードが存在する．OR ノードとは，詰将棋では攻め方手番の局面に相当し，その子ノードのどれか 1 つが「詰め」ば「詰む」ようなノードのことである（いい換えると，攻め方手番の局面については，複数ある王手のうち，どれか 1 つの王手で詰ませられれば十分ということ）．同様に AND ノードとは，詰将棋では玉方手番の局面に相当し，その子ノードすべてが「詰ん」で初めて「詰む」ようなノードのことである（いい換えると，玉方手番の局面については，玉方のすべての逃げ方について，玉が詰むことが分かって初めて詰むといえるということ）．逆に「不詰」を示すには，OR ノードでは，その子ノードすべてが「不詰」と分かって初めて「不詰」と判断でき，AND ノードでは，その子ノードのどれか 1 つが「不詰」ならば「不詰」といえる．

チェスにもチェスプロブレムという，詰将棋（正確には必至問題）に対応するものがある．しかし，最終的な評価値は「詰み」「不詰」「引き分け」の 3 つになる．ゆえに，1 回の AND/OR 木探索で厳密にチェスプロブレムを解くことはできない．そういう意味で，評価値が 2 つしかない詰将棋を解くのは，AND/OR 木探索の好例といえる．

詰将棋を解くプログラムの研究はこの 10 年の間に大きく進歩した．初期に作成されたプログラムは，深さによる反復深化を用いた，単純な深さ優先探索法をベースに様々な工夫を凝らしている．適当なヒューリ

† 東京大学大学院理学系研究科
Faculty of Science, University of Tokyo

スティックな評価関数も必要である．この時期の決定的なプログラムは野下による T2[14] である．T2 は 25 手詰以下の問題はほぼすべて解ける．また変化別詰を排除できたり，余詰検査できるような改良が施されている．

その後，最良優先探索が注目され，1994 年に 611 手の長編詰将棋「寿」を河野のプログラム，伊藤の Ito，野下の T3 が相次いで解いた．これら 3 つのプログラムはいずれも最良優先探索を用いている[13]．河野のプログラムと Ito は，AO*[10] の一種といえる．すなわちヒューリスティックな評価関数は用いず，証明数[1] という新しい画期的な概念を用いた最良優先探索法である．このように，解答能力の向上に，証明数という新しい概念の果たした役割は非常に大きい．

さらに 1995 年，脊尾のプログラム「脊尾詰」が登場し，「寿」はもちろん，873 手の「新扇詰」など，それまでコンピュータで解かれていなかった数多くの詰将棋を解いている[17]．そして 1997 年，ついに 1525 手の現在までのところの最長手数の詰将棋「ミクロコスモス」を解くという偉業をなしとげた．脊尾のアルゴリズムも証明数の概念を用いており，ヒューリスティックな評価関数は用いていない．脊尾のアルゴリズムの特徴は，深さ優先探索法でありながら，振舞いは最良優先の AO*と同じ[7),8)]（**表 1** 参照）という，両者の良い面をあわせ持つ．具体的には，最良優先のように有望な手だけを選択的に深く読む性質と，深さ優先特有の効率的なメモリ利用ができるという性質の両方を持つ．このように，脊尾のアルゴリズムによって劇的に解答能力の向上した背景には，最良優先的振舞いをする新しいタイプの深さ優先探索法を導入したことによるところが大きい．

脊尾のプログラムは素晴らしい成果を残してはいるが，後述のように 300 手以上の長編詰将棋で解けない問題があるなど，万能なわけではない．この理由の 1 つには，証明数と対をなす概念に反証数[1] という概念があるが，証明数と反証数を対等に扱っていないことが考えられる．本来証明数と反証数は対等な存在であるのに，脊尾のアルゴリズムでは基本的には証明数しか用いていない．最近の「脊尾詰」では反証数を用いている[16] が，対等には扱っていない．反証数は付加的な存在でしかない．このことが様々な歪みの原因となっている．

これに対し 1999 年に我々の提案した df-pn アルゴリズム[7] は，証明数と反証数を完全に対等に扱っている．df-pn は最良優先の pn-search[1] と同等の振舞いをする深さ優先探索法である（表 1 参照）が，反証数についても証明数と同様に積極的に利用している．df-pn はしきい値を 2 つ用いている点と，しきい値の計算法が独特である．そこで df-pn アルゴリズムを用いて詰将棋を解くプログラムを実装した．その結果，300 手以上の長編詰将棋を初めてすべて解くなど，素晴らしい成果をあげることができた．

表 1 4 つのアルゴリズムの関係
Table 1 Relation of the four algorithms.

	使っている概念	
	証明数のみ	証明数と反証数
最良優先	AO*[9]	pn-search[1]
深さ優先	脊尾のアルゴリズム[12]	df-pn[7),8)]

本論文では，2 章で証明数や反証数の説明をし，3 章で脊尾のアルゴリズムと「脊尾詰」についての説明をし，4 章で df-pn アルゴリズムに基づいた我々のプログラムで導入した，いくつかの技法について説明する．5 章で「脊尾詰」と我々のプログラムの比較実験の結果を示し，6 章でまとめる．

2. 証明数・反証数

証明数・反証数の元となったのは，McAllester によって提案された，共謀数[5] という概念である．本論文では共謀数の概念は直接関係ないので，興味のある方には文献 11) を勧めたい．共謀数は minimax 木という，有名な $\alpha\beta$ 法が探索の対象とする木に対して提案された概念で，これを AND/OR 木に適用すると，証明数・反証数の概念が出てくる[1]．AND/OR 木探索における最終的な 2 つの評価値（詰将棋の「詰み」「不詰」）を **true** と **false** で一般化すると，証明数・反証数の定義は，次のようになる．

定義 1 ［証明数（反証数）］ 各ノードに対して定義される値で，そのノードの最終的な評価値を **true**（**false**）にするために，**true**（**false**）となることを示さないといけない先端ノードの個数の最小値．

ノード n の証明数を $\mathbf{pn}(n)$，反証数を $\mathbf{dn}(n)$ で表すと，具体的な証明数・反証数の計算法は以下のように再帰的に行われる（グラフではなく，木の探索を仮定）．

(1) ノード n が先端ノード
  (a) 最終的な評価値が **true**
    $\mathbf{pn}(n) = 0$
    $\mathbf{dn}(n) = \infty$
  (b) 最終的な評価値が **false**
    $\mathbf{pn}(n) = \infty$
    $\mathbf{dn}(n) = 0$
  (c) 最終的な評価値が不明

$$\mathbf{pn}(n) = 1$$
$$\mathbf{dn}(n) = 1$$

(2) ノード n が内部ノード

(a) n が OR ノード

$$\mathbf{pn}(n) = \text{子ノードの } \mathbf{pn} \text{ の最小}$$
$$\mathbf{dn}(n) = \text{子ノードの } \mathbf{dn} \text{ の和}$$

(b) n が AND ノード

$$\mathbf{pn}(n) = \text{子ノードの } \mathbf{pn} \text{ の和}$$
$$\mathbf{dn}(n) = \text{子ノードの } \mathbf{dn} \text{ の最小}$$

証明数・反証数の定義（定義 1）を詰将棋に適用すると，証明数は，その局面を詰ますために詰まさないといけない先端局面の個数（正確にはその最小値．また，先端局面の選び方は一意ではない）となる．そのような先端局面は，現在のところ考えられる，玉のある逃げ方を表し，その個数である証明数は，玉の逃げ方の総数を表し，その局面を詰まそうとするときの難易度の非常に良い尺度となる．証明数が小さいほど，詰ますときに少ない先端局面を詰ますだけで済むので，少ない労力で詰ませられる可能性が高くなる．攻め方はより詰ましやすそうな手を選ぶので，攻め方手番の局面（一般的には OR ノード）では証明数最小の手を選ぶことが重要になってくる．逆に反証数は，その局面が不詰であることを示すために不詰を示さないといけない先端局面の個数（の最小値）となる．そのような先端局面は，現在のところ考えられる，ある王手の掛け方を表し，その個数である反証数は，王手の掛け方の総数を表し，その局面が不詰であることを示そうとするときの難易度の非常に良い尺度となる．反証数が小さいほど，不詰を示すときに少ない先端局面の不詰を示すだけで済むので，少ない労力で不詰を示せる可能性が高くなる．玉方はより不詰になりやすそうな手を選ぶので，玉方手番の局面（一般的には AND ノード）では反証数最小の手を選ぶことが重要になってくる．

3. 脊尾のアルゴリズムと「脊尾詰」

3.1 「脊尾詰」の技法

証明数や反証数をナイーブに探索に用いれば，それは自然に最良優先探索になる．証明数や反証数を用いた最良優先探索法には 2 つある．証明数のみを用いた AO*と，証明数と反証数の両方を用いた pn-search である．しかし最良優先探索法にはメモリを大量に消費し，メモリを使い尽くした後の一般的な解決法は知られていないという大問題がある．そこで脊尾は，メモリ消費に問題の少ない深さ優先探索法でありながら，探索の振舞いは最良優先の AO*と同等になる[7),8)]という独自のアルゴリズムを提案した[12),17)]．脊尾のアルゴリズムの概略は，証明数をしきい値に用い，しきい値を 1 つずつ増やしては探索範囲を広げる．その際，内部ノードにおいても反復深化するという，多重反復深化という手法を採用している．詳しくは文献 12), 17) を参照されたい．

脊尾自身が実装した詰将棋ソフトである「脊尾詰」では様々な技法を採用しているが，特筆すべきは局面間の優越関係の利用と，証明駒の導入である．詳しくは次章で説明するので，ここでは簡単に述べておく．優越関係を利用すると，以前に調べて詰むことが分かった局面の情報から，今調べている局面が探索する前から即座に詰むと判断できたりする．また，証明駒を導入するとは，ある局面が詰んだときに，その局面を詰ますために最小限必要な持駒のことで，優越関係の利用と併用することにより非常な効果を発揮する[18)]．

表 2 300 手以上の詰将棋で「脊尾詰」で解けていない 12 題（「詰パラ」は「詰将棋パラダイス」の,「近将」は「近代将棋」の,「めいと」は「詰棋めいと」の略称）

Table 2 The list of 12 Tsume-shogi problems which take over 300 plies to checkmate and which "Seo-Tsume" could not solve.

名前	作者	手数	発表誌	発表年月
メタ新世界	山本昭一	941 手	詰パラ	1982 年 7 月
アルカナ	橋本孝治	639 手	詰パラ	1999 年 8 月
FAIRWAY 修正図	馬詰恒司 摩利支天	611 手	看寿賞 作品集	1999 年
風神	相馬康幸	589 手	詰パラ	1985 年 4 月
まだら	田島秀男	569 手	めいと	2001 年 5 月
赤兎馬	山崎隆	525 手	詰パラ	1979 年 7 月
乱	田島秀男	451 手	詰パラ	1999 年 10 月
呪われた夜	添川公司	393 手	詰パラ	1981 年 2 月
————	田島秀男	345 手	めいと	2000 年 2 月
夢の旅人	藤本和	307 手	近将	1984 年 12 月
金馬車	添川公司	303 手	近将	1995 年 7 月
————	藤本和	301 手	詰パラ	1985 年 4 月

3.2 「脊尾詰」の限界

さて「脊尾詰」は現在最も解答能力の高いソフトとされている.「脊尾詰」は一連のシリーズで，脊尾詰，脊尾弐，脊尾参，と進化してきている．本論文ではこれらをひっくるめて「脊尾詰」と呼ぶ．しかし「脊尾詰」でも解けていない詰将棋がある．300 手以上の詰将棋で「脊尾詰」で解けていないものは，**表 2** のとおりである．これらの問題のうち,「金馬車」と藤本和氏の 301 手の作品については，市販ソフト「柿木将棋」によって解けたという報告があるが，残りの問題についてはこれまで計算機で解けたという報告のない作品である．

4. 我々の詰将棋プログラムで導入した技法

脊尾のアルゴリズムはそれなりに成功を収めているが，証明数のみをしきい値に用いている点に最大の改善の余地がある．そこで 1999 年，我々は証明数・反証数の両方をしきい値に用いた df-pn アルゴリズムという深さ優先探索法を提案した[7]．df-pn の概略は，本来対等な存在である，証明数と反証数を完全に対等に扱い，両者ともしきい値に用いることで，最良優先の pn-search と同等の振舞いを実現している．詳しくは文献 7)，8) を参照されたい．

我々はさらに，df-pn アルゴリズムを用いて詰将棋を解くプログラムを作成した．この章では，その際に我々のプログラムに導入した，いくつかの技法について提案し説明する．

4.1 ハッシュの効率的な利用

df-pn でも脊尾のアルゴリズム同様にハッシュを使える．そしてハッシュを使うのが現実的である．ハッシュが十分にあれば何も問題はないのであるが，現実には解くのに非常に時間のかかる問題もある．そのような問題を解く際には，ハッシュを使い尽くしてしまうことがある．このとき何らかの方法でハッシュ上の古いエントリを捨てて新しいデータを書き込めるようにしないと，探索を続行できない．そのためのアルゴリズムとして，Nagai はいくつかの手法を提案している[6]．その中で今回の実装に用いたのは，SmallTreeReplacement[6] と SmallTreeGC[6] である．最適なエントリの捨て方のアルゴリズムは，ハッシュの実装に依存する．文献 6) ではハッシュにはチェーンハッシュを用いていた．今回はゲームの実装でよく用いられる，開番地法（単純な配列で実装したハッシュ）を用いた．そのため，エントリの捨て方のアルゴリズムも次のようにした．

(1) ハッシュの占有率が 80%に達したら，全ハッシュの少なくとも 30%を SmallTreeGC で捨てる．SmallTreeGC というのは，全エントリを見て，ノードの部分木の小さい順にエントリを消していくというものである．ノードの部分木とは，そのノードの下に存在するノードの個数のことである．

(2) 単純に置き換えられるなら，SmallTreeReplacement で置き換える．SmallTreeReplacement とは，置き換え候補となっているいくつかのエントリを見て，部分木の最も小さいエントリを消して新しいエントリに置き換えるというものである．

このような実装にした理由は次のとおりである．ハッシュの占有率が低い間は，単純な SmallTreeReplacement で置き換える．占有率が上がってくると，SmallTreeReplacement ではオーバヘッドが次第に大きくなってくるので，ある程度の占有率になったら，SmallTreeGC で大量に捨てる．こうして再び SmallTreeReplacement を使うということを繰り返すのが妥当と考えたからである．

4.2 優越関係のさらなる利用

まず始めに，2 つの局面 A と B の間に優越関係が生じるのは，A と B の盤上の駒の配置はまったく同じで，A の攻め方の持駒が，B の攻め方の持駒を真に含むような場合に限る[15]．このとき，A は B を優越しているといい，B が詰むなら，A も自動的に詰む．逆に A が不詰なら，B も不詰である．それ以外の優越関係の使い方としては，B の証明数は A の証明数以上になるという事実を利用して，B の証明数を知りたいときに，ハッシュを検索して，B と B を優越するすべての局面の証明数の最大を B の証明数とすることができる．脊尾のアルゴリズムで利用できるのはここまでである．このように脊尾のアルゴリズムでは，証明数をめぐる優越関係の効果的な利用をしていた．

これに加えて df-pn アルゴリズムでは，反証数についても証明数の場合とまったく同様に，優越関係を効果的に利用できる．すなわち，A の反証数は B の反証数以上になるという事実を利用して，A の反証数を知りたいときに，ハッシュを検索して，A と A によって優越されるすべての局面の反証数の最大を A の反証数とすることができる．

脊尾のアルゴリズムを採用している場合にも，「脊尾詰」のように反証数を付加的に利用することはできる．しかしそのような状況下で，反証数について優越関係を効果的に利用することはできない．これは反証数をしきい値に用いていないことによる影響である．

4.3 GHI 問題対策

GHI（Graph History Interaction）問題とは，最良優先探索において，詰む局面を不詰と誤断してしまう（あるいはその逆）深刻な問題である[2),3)]．これはループと DAG のある木の探索をするときに起こりうる．それなりに普及している，面白味のある思考ゲームで，DAG の出現しないものはないと思われるので，ループの出現するゲームには GHI 問題の危険性がある．したがって詰将棋にもその危険性はある．将棋には，連続王手の千日手は王手をかけている側の負けというルールがある．したがってループを含むような詰手順はあ

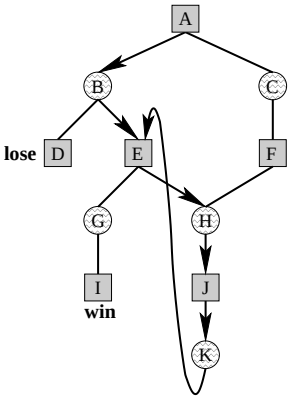

図 1 GHI 問題
Fig. 1 GHI problem.

りえない．だからといって，探索中にループに遭遇したら，ただちに「不詰」と判断してはならない．これが GHI 問題の原因となる．ループを避けつつ，別の手順で同一局面に至る可能性があるからである．**図 1** で，*A-B-E-H-J-K-E* と探索して，*E-H-J-K* のループを発見したからといって，*K* をただちに「不詰」扱いすると，正解手順である *A-C-F-H-J-K-E-G-I* を見つけることはできず,「不詰」と誤断してしまう．

我々はこの問題に対し，次のような対策を施している．GHI 問題を考えなくてよい場合は，ルートでは証明数・反証数ともしきい値 ∞ を与えていた．しかし GHI 問題が絡む場合は，このままでは過ちを冒す恐れがある．そこで付録 A.1 のプログラムリストのように，

(1) ルートにて，しきい値に $\infty - 1$ を与えて探索する．このフェーズは，ループを避けつつ詰・不詰を見つけられるかどうか，という探索を意味する．
(2) (1) のあと，
- ルートの証明数と反証数のペアが 0 と ∞，もしくは ∞ と 0 なら，何の問題もなく解けたことを意味する．前者は詰んだことを意味し，後者は不詰であることを意味する．
- ルートの証明数と反証数が 0 や ∞ でないなら，これはループを避けることができなかったことを意味する．そこで，ルートにてしきい値を ∞ に設定し直して再度探索する．このフェーズでは，ループをただちに「不詰」扱いする探索になる．

ルート以外のすべてのノードについても，同様の処理を行う．なお，

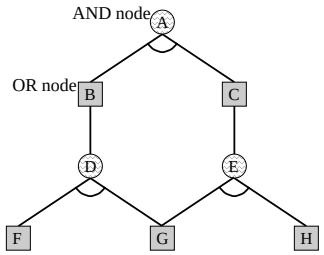

図 2 DAG に起因する証明数の二重カウント
Fig. 2 Double counting proof number caused by a DAG.

$$\sum \text{有限の証明数} < \infty - 1 < \infty$$

でなくてはならない．

4.4 反 証 駒

最近の「脊尾詰」ではハッシュをより有効利用するために,「証明駒」[18] の概念を導入している．ある局面が詰んだとして，その局面を詰ますためにすべての持駒が必要だったとは限らない．そこで，そのときの詰手順を眺めながら，本当に必要な最小の先手の持駒集合を証明駒と呼び，証明駒を持駒とした局面をハッシュに記録するのである．我々のプログラムでも，この証明駒の概念と，それに加えて，同様の概念を反証する場合に適用した「反証駒」の概念を導入している．反証駒とは，証明駒とは逆に，ある局面が不詰だと示せたとして，その局面から不詰に逃れるための最小の後手の持駒集合のことである．

4.5 証明数の二重カウントの回避

我々のプログラムのもう 1 つの特徴としては，証明数の二重カウントの回避を行っている点である．従来 DAG に起因する証明数の二重カウントは，解決法の未発見な問題[12),13)] であった．証明数の二重カウントがどういうものかというと，**図 2** で，局面 *G* の証明数は局面 *B-D* 経由で局面 *A* にカウントされる．一方それとは別に，*G* の証明数は局面 *C-E* 経由でも局面 *A* にカウントされる．こうして，*G* の証明数は *A* で二重にカウントされる．本来の証明数・反証数の定義（定義 1）に従えば *G* の証明数を二重カウントするのは間違いである．しかし現実には，証明数・反証数の計算は 2 章で述べたようにして行われ，DAG が存在すると，二重カウントが起こってしまう．本来証明数はその局面を詰まそうとするときの難易度を表すが，証明数の二重カウントが起こると，難易度が不当に高く見積もられてしまい，それほど難易度の高い問題でもないのに解くのに異常に時間がかかったり，現実的な時間内では解けなくなってしまったりしてしまう．

我々は，各局面に親局面へのポインタを持たせ，場

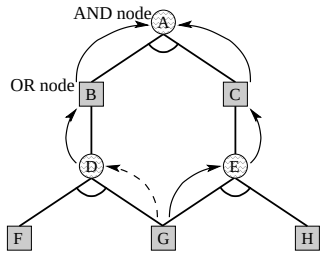

図 3 DAG の検出
Fig. 3 Detecting a DAG.

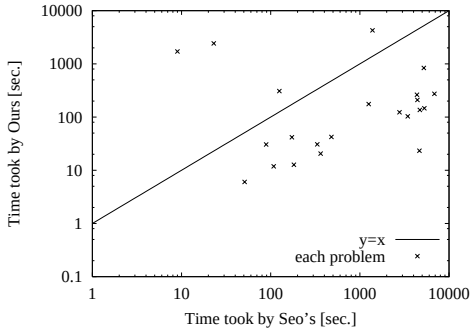

図 4 脊尾弐 Ver.2.3 と我々のプログラムによる超長編の比較実験結果
Fig. 4 The comparison between Seo-Ni Ver.2.3 and our program of the result of very long tsume-shogi problems.

合によっては親局面をたどることで DAG を検出し，証明数の二重カウントを回避した．**図 3** を用いて詳しく説明する．以前に *A*-*B*-*D* 経由で局面 *G* を探索していたとする．すると *G* の親局面には *D* が記録されている．その後 *A*-*C*-*E* 経由で局面 *G* を探索したとする．このとき *G* の親は *E* なのに，*D* が記録されている．このような状況の場合，DAG が存在している可能性がある．そこで *D* の親と，*E* の親をそれぞれたどり，*A* で合流することを確認して DAG を検出する．このとき闇雲にどこまでもたどるわけではなく，OR 局面ではつねに親をたどり，AND 局面ではその AND 局面の証明数と，子局面の証明数が一致する場合に限り親をたどる．*A* で DAG を検出したら，その旨知らせるフラグを立てて，*A* において証明数の計算をするとき，子局面の証明数の**和**を計算する代わりに，子局面の証明数の**最大**を計算する．こうすることで，本来の証明数・反証数の定義（定義 1）に近づける．

この方法はどのようなゲームの探索においても二重カウントを回避できるわけではない．**詰将棋**における**証明数**の二重カウントが，一部の詰将棋を詰ますうえで深刻で，そのときの DAG の発生する状況に一定の性質があったから，このような単純な方法で効果があった．一定の性質とは，比較的 1 本道に近い手順どうしが先の方で同一の局面に至り DAG を形成しているということである．人工的に作られた長編詰将棋によく見受けられる性質である．ここで述べた検出の仕方は，このような特殊な（単純な）状況の DAG に対してのみ，効果を発揮する ad-hoc な方法であり，一般的な解決法ではない．

5. 実 験 結 果

我々のプログラムは，300 手以上の長編詰将棋に関しては，表 2 で取り上げた,「脊尾詰」では解けない 12 題を含む，すべての問題を解くことができた．これは初めてのことである．このことから，我々のプログラムの解答能力がきわめて優れていることが分かる．これらのどの問題についても，どの詰将棋プログラムからも，解けたという報告のない難問である．そのような一連の問題を，我々のプログラムは通常の 1 台のワークステーション環境でたかだか 92,000 秒（最も時間のかかったのは 451 手詰の「乱」である）で解いたことは，df-pn アルゴリズムと，本論文中で提案したいくつかの技法の有効性を十分に示すものである．

さらに表 2 にも載せてある，393 手の「呪われた夜」に 183 手目以降少なくとも 265 手の早詰があることを初めて示した．早詰は余詰の一種で，余詰とは，攻め方の手を作意手順とは別の王手に変えても詰ませられるような手のことをいう．余詰のうち作意手順より短いものを早詰という．詰将棋の作品は，余詰がなくて初めて完全作と認められる．この作品はこれまで完全作と見られていたが，余詰を発見した．

「将棋図巧」「将棋無双」は難解作の多いことで有名な，江戸時代の詰将棋作品集である．我々のプログラムはこれらの作品集の問題のうち，不詰とされている問題を除いた「将棋図巧」99 題,「将棋無双」94 題すべてを解いた．また，江戸時代から昭和までの名作詰将棋を集めた「続詰むや詰まざるや」についても，不詰とされている問題を除いた 195 題すべてを解いた．さらに「将棋図巧」「将棋無双」よりも数段難しい「将棋墨酔」についても，103 題中 95 題を解いた．残り 8 題中 5 題は不詰であるが，3 題は詰むのか不明である．

ここで，詰将棋を解くソフトとして評価の高い「脊尾詰」との比較実験の結果を載せる．

図 4 は「脊尾詰」と我々のプログラムで，作意が 300 手以上（不完全作を含む）の長編詰将棋問題を解

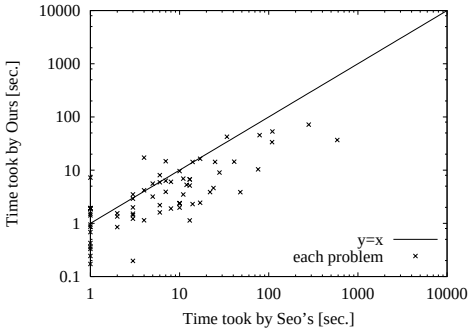

図 5 脊尾参 Ver.2.3A と我々のプログラムによる「図巧」の比較実験結果

Fig. 5 The comparison between Seo-San Ver.2.3A and our program of the result of "Zuko".

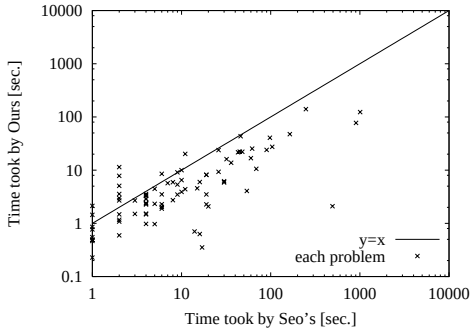

図 6 脊尾参 Ver.2.3A と我々のプログラムによる「無双」の比較実験結果

Fig. 6 The comparison between Seo-San Ver.2.3A and our program of the result of "Muso".

かせた結果を図示したものである．当然のことながら，「脊尾詰」で解けなかった問題については図示していない．実験環境は,「脊尾詰」は PentiumII 400 MHz, ハッシュに使用したメモリは 288 MB（一部 352 MB や 224 MB のデータも混じっている）「脊尾詰」のバージョンは脊尾弐 Ver.2.3 であるが，このバージョンのデータのないものについては，脊尾詰 Ver.2.1 のデータで代用している．このデータは加藤徹氏作成（1998 年 9 月 23 日．古い）の資料に基づいたものである．我々のプログラムの実験環境は，UltraSparcII 400 MHz, ハッシュに使用したメモリは 280 MB である．脊尾詰と我々のプログラムでマシン環境が異なるが，これは次のような理由による．脊尾詰は Windows 上のソフトとして販売されており，Windows 上でしか動作しないのに対し，我々のプログラムはワークステーション環境で開発を行っており，現在のところ Unix 上でしか動作しないので，同じマシン環境での比較はできなかったのである．図 4 を見ると，全体としてデータが右下に偏っている．これは，マシン環境はほぼ同じなのに，我々のプログラムの方が解答時間が短いことを示している．

図 5，**図 6**，**図 7** はそれぞれ「将棋図巧」「将棋無双」「将棋墨酔」を解かせた結果を図示したものである．実験環境は,「脊尾詰」は Celeron 433 MHz，ハッシュに使用したメモリは「将棋図巧」「将棋無双」については 160 MB,「将棋墨酔」については 224MB である．バージョンは,「将棋図巧」「将棋無双」については脊尾参 Ver.2.3A,「将棋墨酔」については脊尾参 Ver.2.3 である．このデータは岡崎正博氏よりいただいた 2000 年夏のものである．我々のプログラムの実験環境は，UltraSparcII 400 MHz，ハッシュに使用したメモリは「将棋図巧」「将棋無双」が 28 MB,「将棋墨酔」が 280 MB である．マシン環境が違うのは前述のように，開発環境の違いに起因する．図 5，図 6，図 7 を見ると，やはり全体としてデータが右下に偏っている．これは，マシン環境は大体同じなのに，我々のプログラムの方が解答時間が短いことを示している．

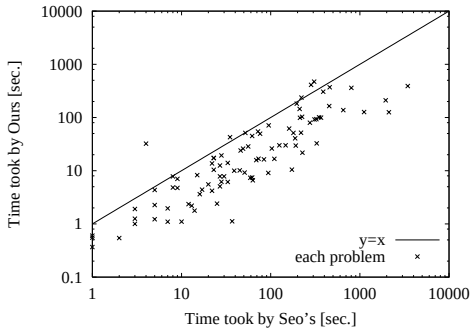

図 7 脊尾参 Ver.2.3 と我々のプログラムによる「墨酔」の比較実験結果

Fig. 7 The comparison between Seo-San Ver.2.3 and our program of the result of "Bokusui".

図 4，図 5，図 6，図 7 から，我々のプログラムが解答能力のみならず，解答時間の面でも優れていることが分かる．これらの結果はまた，df-pn アルゴリズムと，本論文中で提案した，ハッシュの利用法や証明数の二重カウント回避などの技法の有効性を十分に示す．

6. ま と め

df-pn アルゴリズムを詰将棋に適用し，詰将棋を解くプログラムを実装した．そして初めて 300 手以上の詰将棋をすべて解いた．また「続詰むや詰まざるや」195 題,「将棋図巧」99 題,「将棋無双」94 題,「将棋墨酔」95 題を解いた．我々のプログラムがこの分野で最高に高く評価されているソフト「脊尾詰」を超える

解答能力を持ち，さらに解答時間の面でも優れたプログラムであることを示すと同時に，df-pn アルゴリズムと，本論文中で提案したいくつかの技法の有効性を示した．

参 考 文 献

1) Allis, L.V., van der Meulen, M. and van den Herik, H.J.: Proof-Number Search, Technical Report CS 91-01, University of Limburg, Maastricht, Netherlands (1991). Also available as *Artificial Intelligence*, Vol.66, pp.91–124, (1994).
2) Breuker, D.M., Uiterwijk, J.W.H.M., van den Herik, H.J. and Allis, L.V.: A Solution to the GHI Problem for Best-First Search, *Computers and Games* (*CG'98*), van den Herik, H.J. and Iida, H. (Eds.), Vol.1558 of LNCS 1558, pp.25–49 (1998).
3) Campbell, M.S.: The graph-histroy interaction: On ignoring position history, *1985 Association for Computing Machinery Annual Conference*, pp.278–280 (1985).
4) Knuth, D.E. and Moore, R.W.: An Analysis of Alpha-Beta Pruning, *Artificial Intelligence*, Vol.6, pp.293–326 (1975).
5) McAllester, D.A.: Conspiracy Numbers for Min-Max Search, *Artificial Intelligence*, Vol.35, pp.287–310 (1988).
6) Nagai, A.: A new Depth-First-Search Algorithm for AND/OR Trees, Master's Thesis, Department of Information Science, University of Tokyo, Japan (1999).
7) Nagai, A. and Imai, H.: Proof for the Equivalence Between Some Best-First Algorithms and Depth-First Algorithms for AND/OR Trees, *KOREA-JAPAN Joint Workshop on Algorithms and Computation*, pp.163–170 (1999).
8) Nagai, A. and Imai, H.: Proof for the Equivalence Between Some Best-First Algorithms and Depth-First Algorithms for AND/OR Trees, *IEICE Trans. Information and Systems* (to appear).
Preliminary version appeared in *KOREA-JAPAN Joint Workshop on Algorithms and Computation*, pp.163–170 (1999).
9) Nilson, N.J.: *Principles of Artificial Intelligence*, Tioga Publishing Company, Palo Alto, CA (1980).
10) Pearl, J.: Asymptotical properties of minimax trees and game searching procedures, *Artificial Intelligence*, Vol.14, pp.113–138 (1980).
11) Schaeffer, J.: Conspiracy Numbers, *Artificial Intelligence*, Vol.43, pp.67–84 (1990).
12) Seo, M.: 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, Japan (1995).
13) 伊藤琢巳，河野泰人，野下浩平：非常に手数の長い詰将棋問題を解くアルゴリズムについて，情報処理学会論文誌，Vol.36, No.12, pp.2793–2799 (1995).
14) 伊藤琢巳，野下浩平：詰将棋を速く解く 2 つのプログラムとその評価，情報処理学会論文誌，Vol.35, No.8, pp.1531–1539 (1994).
15) 脊尾昌宏：C*アルゴリズムによる AND/OR 木の探索および詰将棋プログラムへの応用，情報処理学会人工知能研究会，No.99-14, pp.103–110 (1995).
16) 脊尾昌宏：メールによる対話 (2000).
17) 脊尾昌宏：共謀数を用いた詰将棋の解法，第 2 巻，1 コンピュータ将棋の進歩，pp.1–21, 共立出版 (1998).
18) 脊尾昌宏：詰将棋を解くアルゴリズムにおける優越関係の効率的な利用について，*Game Programming Workshop in Japan '99*, pp.129–136 (1999).

付　　録

A.1 df-pn のプログラムリスト

df-pn のプログラムリストを載せる．OR ノードにおける証明数と AND ノードにおける反証数は本質的には等価である．同様に，OR ノードにおける反証数と AND ノードにおける証明数も本質的には等価である．両者は互いに双対をなすので，minimax 木探索探索における negamax アルゴリズム[4] に相当する，よりコンパクトなアルゴリズムを構成できる．前者を ϕ，後者を δ と名付ける．すなわち，

$$\phi(n) = \begin{cases} \mathbf{pn}(n) & (n \text{ は OR ノード}) \\ \mathbf{dn}(n) & (n \text{ は AND ノード}) \end{cases}$$

$$\delta(n) = \begin{cases} \mathbf{dn}(n) & (n \text{ は OR ノード}) \\ \mathbf{pn}(n) & (n \text{ は AND ノード}) \end{cases}$$

と定義する．

```
1   // ルートでの反復深化
2   procedure Df-pn(r) {
3       φ(r) = ∞ − 1;    δ(r) = ∞ − 1;
4       MID(r);
5       if (φ(r) ≠ ∞ && δ(r) ≠ ∞) {
6           φ(r) = ∞;    δ(r) = ∞;
7           MID(r);
8       }
9   }
10  // ノード n の展開
11  procedure MID(n) {
```

```
12      // 1. ハッシュを引く
13      LookUpHash(n, φ, δ);
14      if (φ(n) ≤ φ ‖ δ(n) ≤ δ) {
15          φ(n) = φ;    δ(n) = δ;
16          return;
17      }
18      // 2. 合法手の生成
19      if (n が末端ノード) {
20          if ((nが ANDノード && Eval(n)=true) ‖
21              (nが ORノード && Eval(n)=false)) {
22              φ(n) = ∞;    δ(n) = 0;
23          }else {φ(n) = 0;    δ(n) = ∞; }
24          PutInHash(n, φ(n), δ(n));
25          return;
26      }
27      GenerateLegalMoves();
28      // 3. ハッシュによるサイクル回避
29      PutInHash(n, φ(n), δ(n));
30      // 4. 多重反復深化
31      while (1) {
32          // φ か δ がそのしきい値以上なら探索終了
33          if (φ(n)≤ΔMin(n) ‖ δ(n)≤ΦSum(n)) {
34              φ(n)= ΔMin(n);    δ(n)= ΦSum(n);
35              PutInHash(n, φ(n), δ(n));
36              return;
37          }
38          n_c = SelectChild(n, φ_c, δ_c, δ_2);
39          if (φ_c = ∞ − 1) φ(n_c) = ∞;
40          else if (δ(n) ≥ ∞ − 1) φ(n_c) = ∞ − 1;
41          else φ(n_c) = δ(n) + φ_c − ΦSum(n);
42          if (δ_c = ∞ − 1) δ(n_c) = ∞;
43          else δ(n_c) = min(φ(n), δ_2+1);
44          MID(n_c);
45      }
46  }
47  // 子ノードの選択
48  procedure SelectChild(n, &φ_c, &δ_c, &δ_2) {
49      δ_c = ∞;    δ_2 = ∞;
50      for (各子ノード n_child について) {
51          LookUpHash(n_child, φ, δ);
52          if (δ < δ_c) {
53              n_best = n_child;
54              δ_2 = δ_c;    φ_c = φ;    δ_c = δ;
55          }else if (δ < δ_2) δ_2 = δ;
56          if (φ = ∞) return n_best;
57      }
58      return n_best;
59  }
60  // ハッシュを引く (本当は優越関係が使える)
61  procedure LookUpHash(n, &φ, &δ) {
62      if (n が登録済み) {
63          φ = Table[n].φ;    δ = Table[n].δ;
64      }else {φ = 1;    δ = 1; }
65  }
66  // ハッシュに記録
67  procedure PutInHash(n, φ, δ) {
68      Table[n].φ = φ;    Table[n].δ = δ;
69  }
70  // n の子ノードの δ の最小を計算
71  procedure ΔMin(n) {
72      min = ∞;
73      for (各子ノード n_child について) {
74          LookUpHash(n_child, φ, δ);
75          min = min(min, δ);
76      }
77      return min;
78  }
79  // n の子ノードの φ の和を計算
80  procedure ΦSum(n) {
81      sum = 0;
82      for (各子ノード n_child について) {
83          LookUpHash(n_child, φ, δ);
84          sum = sum + φ;
85      }
86      return sum;
87  }
```

(平成 13 年 5 月 7 日受付)
(平成 14 年 3 月 14 日採録)

長井 歩

1974 年生．1997 年東京大学理学部情報科学科卒業．1999 年東京大学大学院理学系研究科情報科学専攻修士課程修了．現在，同大学院博士課程在学中．探索，人工知能に関する研究に従事．

今井 浩（正会員）

1958 年生．1981 年東京大学工学部計数工学科卒業．1983，1986 年東京大学大学院工学系研究科情報工学専門課程修士，博士課程修了．工学博士取得．1986〜1990 年九州大学工学部情報工学科助教授，1990 年東京大学理学部情報科学科助教授，現在に至る．1987 年冬カナダ McGill 大学訪問副教授．1988 年秋 IBM T.J. Watson 研究所訪問研究員．アルゴリズム論，計算幾何学，最適化，学習理論の研究に従事．電子情報通信学会，日本 OR 学会，ソフトウェア科学会，人工知能学会，数学会，ACM IEEE 各会員．