アルゴリズム 75－2
（2000．11．10）

df-pn アルゴリズムの詰将棋解答プログラムへの応用

長井 歩　　今井 浩
東京大学大学院理学系研究科情報科学専攻
〒113-0033 東京都文京区本郷 7-3-1

要旨

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，大雑把に言うと，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，ナイーブなアルゴリズムは，Allis の 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 と同等の振る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 アルゴリズムを詰将棋を解くプログラムに応用した．その結果，300手以上の詰将棋をすべて解くことに初めて成功するなど，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　　Hiroshi Imai
Department of Information Science, Faculty of Science, University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-0033, Japan

Abstract

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 a 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. As a result, 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，最終的な評価が2つしか存在しないような二人ゲームの探索は，人工知能研究の分野では，AND/OR木の探索として一般化される．AND/OR木には，ANDノードとORノードという二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が「詰め」ば「詰む」ようなノードのことである．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．

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．故に厳密には，チェスプロブレムを解くのは，純粋な AND/OR 木探索とは言えない．そう言う意味で，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．この時期の決定的なプログラムは野下による T2[10] である．T2 は25手詰以下の問題はほぼすべて解くことができる．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 [9]．

更に1995年，脊尾のプログラム「脊尾詰」が登場し，「寿」は勿論，873手の「新扇詰」など，それまでコンピューターで解かれていなかった数多くの詰将棋を解いている [11]．そして1997年，遂に1525手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成し遂げた．脊尾のアルゴリズムは，深さ優先探索法でありながら，振る舞いは最良優先と同じという，両者の良い面をあわせ持つ．脊尾のプログラムが劇的に解答能力の向上した背景には，証明数や反証数という新しい概念を導入したことに因るところが大きい．

脊尾のプログラムは素晴らしい成果を残してはいるが，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．この理由の一つには，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対等には扱っていない．反証数はおまけのような存在である．

これに対し1999年に我々の提案した df-pn アルゴリズムは，証明数と反証数を完全に対等に扱っている．そこで df-pn アルゴリズムを用いた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．

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 [4] 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 [7] を勧めたい．共謀数は minimax 木という，有名な $\alpha\beta$ 法で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 [3]．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**, **false** とすると，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．

証明数を pn，反証数を dn 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．

1. ノード n が先端ノード
   (a) 最終的な評価値が **true**
   $$n.pn = 0$$
   $$n.dn = \infty$$
   (b) 最終的な評価値が **false**
   $$n.pn = \infty$$
   $$n.dn = 0$$
   (c) 最終的な評価値が不明
   $$n.pn = 1$$
   $$n.dn = 1$$
2. ノード n が内部ノード
   (a) n が OR ノード
   $$n.pn = \text{子ノードの}pn\text{の最小}$$
   $$n.dn = \text{子ノードの}dn\text{の和}$$
   (b) n が AND ノード
   $$n.pn = \text{子ノードの}pn\text{の和}$$
   $$n.dn = \text{子ノードの}dn\text{の最小}$$

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．先端局面の選び方は一意ではない) となる．これは言い換えると，現在のところ考えられる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．証明数が大きいほど，詰まそうとするときの難易度が高くなる．攻め方はより詰ませ易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．また反証数は，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．言い換えると，現在のところ考えられる王手の掛けられ方の総数を表し，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．反証数が大きいほど，不詰めを示そうとするときの難易度が高くなる．王方はより不詰になり易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

証明数・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．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でありながら，探索の振る舞いは最良優先と同じになるという独自のアルゴリズムを提案した [8][11].

証明数をしきい値に利用

従来深さ優先探索法では深さをしきい値に用いていた．深さがしきい値以下の局面のみを探索するのである．そして解が見つかるまでしきい値を1ずつ増やし続ける．脊尾のアルゴリズムでは，証明数をしきい値に用い，証明数がしきい値に満たない限り，探索を続行する．そして解が見つかるまでしきい値を1ずつ増やし続ける．

多重反復深化

脊尾のアルゴリズムの特徴として，ルートと各ANDノードで反復深化するという特殊な反復深化をしていることが挙げられる．探索の様子を理解してもらうため，例を用いる．図1のように，ノード a の証明数が9になっているところへ，しきい値 th として10が割り当てられたとする．

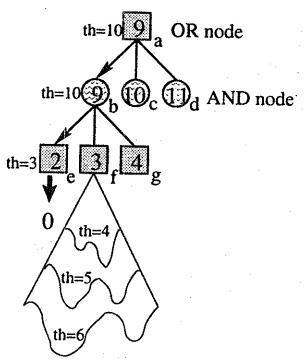

図 1: 多重反復深化(数字はそのノードの証明数)

この時，証明数最小のノード b にしきい値10を与えて，探索を進める．b ではノード e を選択し，しきい値3を与えて，探索を進めたとする．探索の結果，e が解かれて，証明数0になったとする．次にノード f を選択したとする．仮に多重反復深化しないと，f には，しきい値6を割り当てることが可能である (というのは，b のしきい値が10で，g の証明数が4だから)．しかし多重反復深化では，ここで f の証明数3を見て，まずはしきい値4を割り当てる．しきい値4で解けなかったら，しきい値5を割り当て，それでも解けなかったら，しきい値6を割り当てる．このようにして，ルート以外の内部ノードにおいてもしきい値を1ずつ増加させ，その範囲のノードを探索する．なぜいきなりしきい値6を割り当てないのかというと，もし f がしきい値4の探索で解ける場合，しきい値6で探索していると，解は確かに見つかるが，不必要に多くのノードを探索してしまう．そこで探索ノード数を極力抑えるためにしきい値を1ずつ増加させるのである．このような反復深化を多重反復深化 [8] と呼ぶ．

ところで，f にしきい値5を与えて解けなかった場合，f の証明数は5 (以上) になるので，証明数最小の立場からは次に探索すべきは g である．しかし脊尾のプログラムでは引き続き f を探索するという変則的なスタイルを取っている．

ハッシュの利用

脊尾のアルゴリズムは深さ優先ゆえ，今までに深さ優先探索法で培われてきた改良法 (enhancements) が利用できる．ハッシュの利用はその最たるものである．当然脊尾のアルゴリズムでもハッシュを利用している．多重反復深化を採っているので，単純な深さ優先探索よりも同じ局面を何度も探索する．従ってハッシュを利用するのが現実的である．

優越関係

ここからは将棋に特化した工夫である．詰将棋に適用したときには，複数の局面の間に優越関係の生じることがよくある．脊尾のアルゴリズムではこれらの優越関係をうまく利用している [8][11]．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全く同じで，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である．この時，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．A が B を優越している場合，それ以外に優越関係が使えるのは，B の証明数は A の証明数以上になるという事実である．従って

Bの証明数を知りたいとき，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取ればいい．

「脊尾詰」の限界

脊尾のアルゴリズムを脊尾自身が実装した詰将棋ソフト「脊尾詰」が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．以後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問題がある．300手以上の詰将棋で「脊尾詰」で解けていないものは，表1の通りである．

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
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
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

表 1: 300手以上の詰将棋で「脊尾詰」で解けていない10題(「詰パラ」は「詰将棋パラダイス」の，「近将」は「近代将棋」の略称)

4 df-pn アルゴリズム

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という深さ優先探索法を提案した[6].

証明数・反証数の両方をしきい値に利用

脊尾のアルゴリズムは証明数のみをしきい値に用いていた．また最近はAND局面(玉方手番の局面)で反証数を用いるという改良を施してはいるが，反証数はしきい値には用いていない．

本来証明数と反証数は対をなすもので，両者をナイーブに用いた最良優先のpn-search[3]でも全く対等に扱っている．しかし脊尾のアルゴリズムでは，証明数のみをしきい値に用いており，これでは片手落ちである．これに対し，df-pnアルゴリズムは，証明数・反証数の両方を完全に対等に扱い，両者ともしきい値に用いている．

全ノードで多重反復深化

脊尾のアルゴリズムでは，図1からも分かる通り，各ANDノードのみ(とルート)で反復深化している．これはしきい値に証明数のみを用いていたことに起因する．しかしdf-pnアルゴリズムでは，しきい値に証明数と反証数の両方を用いているので，すべてのノードで反復深化する．従って，脊尾のアルゴリズムに比べて，より多重度の高い多重反復深化を行う．

脊尾のアルゴリズムでは，ルートに対しては，まず最初に証明数のしきい値2を与え，解けないでルートに処理が戻ってくる限り，3，4…のようにしきい値を1ずつ増やして反復深化していた．しかしdf-pnアルゴリズムでは，ルートでは基本的には証明数・反証数とも最初にしきい値∞を与える．

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．この時何らかの方法で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，長井はいくつかの手法を提案している[5]．その中で今回の実装に用いたのは，SmallTreeReplacementとSmallTreeGC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．[5]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，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．

優越関係

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．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全く同様に，優越関係を効果的に利用できる．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．

GHI 問題対策

GHI(Graph History Interaction) 問題とは，最良優先探索において，ループのせいで，詰む局面を不詰としてしまう (あるいはその逆) 深刻な問題である [1][2]．我々はこの問題に対し，エレガントな対策を施している．GHI 問題を考えなくて良い場合，ルートでは証明数・反証数ともしきい値 ∞ を与える．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のプログラムリストのように，一旦しきい値に $\infty - 1$ を与えて探索し，探索後ルートの証明数・反証数のいずれかが ∞ なら何の問題もなく解けたことを意味し，いずれも ∞ に達していないなら，これは GHI 問題が絡んでいたことを意味する．そこでしきい値を ∞ に設定し直して再度探索する．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．尚，

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

でなくてはならない．

ヒューリスティックス

「脊尾詰」[11] と同様に，無駄合の手を後手玉に近い位置に打つ手から探索したり，同じ位置でも安い駒で合駒する手から探索する．また，玉方手番の局面の証明数を計算する際，本来はすべての子局面の証明数の和を計算するが，無駄合と予想できる手の証明数はカウントしない．有効合と予想する場合も，合駒のうち 1 種類だけカウントする，などというのも「脊尾詰」と同じである．

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」[12] の概念を導入している．我々のプログラムでも，この「証明駒」の概念と，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．

我々のプログラムの特徴としては，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

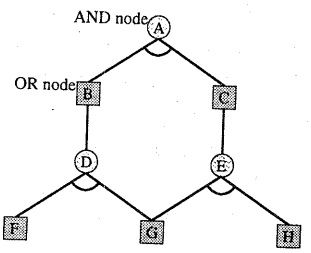

図 2: DAG に起因する証明数の二重カウント

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 [9][8] 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 に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．

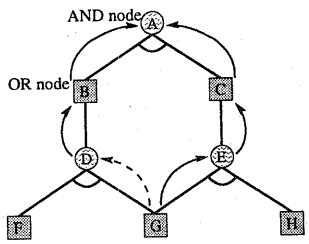

図 3: DAG の検出

我々は，各局面に親局面へのポインタを持たせ，場合によっては親局面を辿ることで DAG を検出し，証明数の二重カウントを回避した．図 3で詳しく説明する．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．この時 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

それぞれ辿り，Aで合流することを確認して DAG を検出する．この時闇雲にどこまでも辿る訳ではなく，OR局面では常に親を辿り，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辿る．Aで DAG を検出したら，その旨知らせる flag 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上で深刻で，その時の DAG の発生する状況に一定の規則性があったから，このような単純な方法で効果があった．一定の規則性とは，比較的 1 本道に近い手順同士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．ここで述べた検出の仕方はこのような特殊な状況の DAG に対してのみ，効果を発揮する．

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 1で取り上げた，「脊尾詰」では解けない 10 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極めて優れていることが分かる．

更に表 1にも載せてある，611 手の「FAIRWAY 修正図」が詰むことを初めて示した．この詰将棋は，不詰と判明した 615 手の原図に対する修正図で，詰むのか不詰なのか，これまではっきりとした結論が出ていなかった作品である．

また，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，余詰とは作意手順の中で，攻め方の手を作意とは別の手に変えても詰ますことができることをいう．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，不詰と言わ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，不詰と言われている問題を除いた 195 題すべてを解いた．更に「将棋図巧」「将棋無双」よりも数段難しいと言われている「将棋墨酔」についても，103 題中 95 題を解いた．

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．

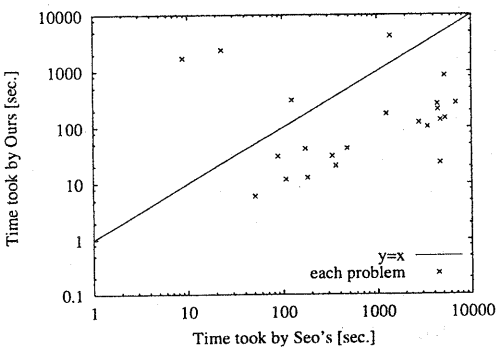

図 4: 脊尾弐 Ver.2.3 と我々のプログラムによる超長編の比較実験結果

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 400MHz，ハッシュに使用したメモリは 288MB(一部 352MB や 224MB 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 日と少々古い) 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 400MHz，ハッシュに使用したメモリは 280MB である．図 4を見ると，全体としてデータが右に偏っている．これは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 433MHz，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 160MB，「将棋墨酔」については 224MB である．バージョン

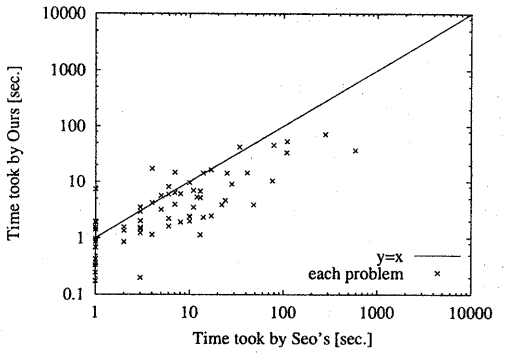

図 5: 脊尾参 Ver.2.3A と我々のプログラムによる「図巧」の比較実験結果

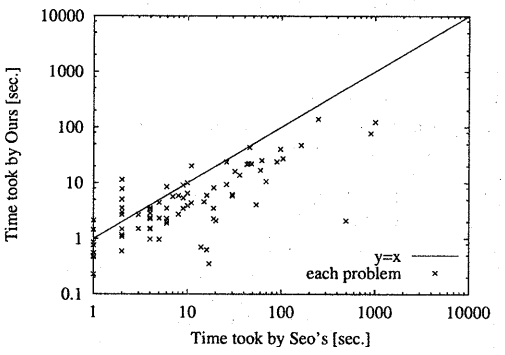

図 6: 脊尾参 Ver.2.3A と我々のプログラムによる「無双」の比較実験結果

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頂いたごく最近のものである．我々のプログラムの実験環境は，UltraSparcII 400MHz，ハッシュに使用したメモリは「将棋図巧」「将棋無双」が 28MB，「将棋墨酔」が 280MB である．図 5，図 6，図 7を見ると，やはり全体としてデータが右に偏っている．これは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 4，図 5，図 6，図 7から，我々のプログラムが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 アルゴリズムの妥当性をも示していると言えよう．

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

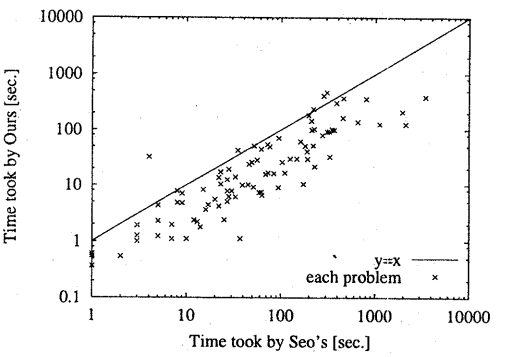

図 7: 脊尾参 Ver.2.3 と我々のプログラムによる「墨酔」の比較実験結果

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」を越える解答能力を持ち，更に解答時間の面でも優れたプログラムであることに示した．

謝辞

詰将棋の問題や実験データをはじめ，様々な情報を提供して頂いた岡崎正博氏，門脇芳雄氏，山田剛氏，加藤徹氏，更にプログラムの実装にあたって，ご自分の実装について色々ご説明下さった脊尾昌宏氏に深く感謝致します．また証明数の二重カウントの回避について，名取伸氏には貴重な助言を頂いたことに謝意を表したい．

参考文献

[1] Murray S. Campbell. The graph-histroy interaction: On ignoring position history. In *1985 Association for Computing Machinery Annual Conference*, pages 278–280, 1985.

[2] Jos W.H.M. Uiterwijk Dennis M. Breuker, H. Jaap van den Herik and L. Victor Allis. A solution to the ghi problem for best-first search. In H. Jaap van den Herik and Hiroyuki Iida, editors, *Computers and Games (CG'98)*, volume 1558 of *LNCS 1558*, pages 25–49, 1998.

[3] Maarten van der Meulen Louis V. Allis and H. Jaap van den Herik. Proof-Number Search. Technical Report CS 91-01, University of Limburg, Maastricht, Netherlands, 1991. Also available at *Artificial Intelligence*, Vol.66, pp. 91–124, 1994.

[4] David A. McAllester. Conspiracy Numbers for Min-Max Search. *Artificial Intelligence*, 35:287–310, 1988.

[5] Ayumu Nagai. A new Depth-First-Search Algorithm for AND/OR Trees. Master's thesis, Department of Information Science, University of Tokyo, Japan, 1999.

[6] Ayumu Nagai. Proof for the Equivalence Between Some Best-First Algorithms and Depth-First Algorithms for AND/OR Trees. In *KOREA-JAPAN Joint Workshop on Algorithms and Computation*, pages 163–170, 1999.

[7] Jonathan Schaeffer. Conspiracy Numbers. *Artificial Intelligence*, 43:67–84, 1990.

[8] Masahiro 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, Japan, 1995.

[9] 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, 36(12):2793–2799, 1995.

[10] 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, 35(8):1531–1539, 1994.

[11] 脊尾昌宏. 共謀数を用いた詰将棋の解法, volume 2, chapter 1 コンピューター将棋の進歩, pages 1–21. 共立出版, 1998.

[12] 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. In *Game Programming Workshop in Japan '99*, pages 129–136, 1999.

A df-pn のプログラムリスト

```
1   // ルートでの反復深化
2   procedure Df-pn(r) {
3     r.φ = ∞ − 1;   r.δ = ∞ − 1;
4     MID(r);
5     if (r.φ ≠ ∞ && r.δ ≠ ∞) {
6       r.φ = ∞;   r.δ = ∞;
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 (n.φ ≤ φ || n.δ ≤ δ) {
15      n.φ = φ;   n.δ = 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        n.φ = ∞;   n.δ = 0;
23      }else {n.φ = 0;   n.δ = ∞; }
24      PutInHash(n, n.φ, n.δ);
25      return;
26    }
27    GenerateLegalMoves();
28    // 3. ハッシュによるサイクル回避
29    PutInHash(n, n.φ, n.δ);
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 (n.φ ≤ ΔMin(n) || n.δ ≤ ΦSum(n)) {
34        n.φ = ΔMin(n);   n.δ = ΦSum(n);
35        PutInHash(n, n.φ, n.δ);
36        return;
37      }
38      n_c = SelectChild(n, φ_c, δ_c, δ_2);
39      if (φ_c = ∞ − 1) n_c.φ = ∞;
40      else if (n.δ ≥ ∞ − 1) n_c.φ = ∞ − 1;
41      else n_c.φ = n.δ + φ_c − ΦSum(n);
42      if (δ_c = ∞ − 1) n_c.δ = ∞;
43      else n_c.δ = min(n.φ, 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  }
```