反復深化探索に基く協力詰将棋の解法

星　　由雄[†]　野下浩平[†]　柳井啓司[†]

協力詰将棋（協力詰）は詰将棋のルールを変形したパズルである．協力詰の問題は OR 木の探索問題として定式化できる．本論文では探索問題として協力詰をとりあげ，これを解くための新しいアルゴリズムを提案する．探索法として，反復深化法と局面表（トランスポジション表）の技法を組み合わせた深さ優先探索を用いる．局面表に新しい技法を導入し，探索の高速化と記憶領域の節約を図る．本論文のアルゴリズムを用いて作成したプログラムは，協力詰の最長手数問題である 49909 手詰の「寿限無 3」をはじめ，これまでコンピュータでは解けなかったいくつかの難しい問題を解くことができる．また実例によって，新しい技法は局面の数が多い問題で特に効果が大きいことを示す．

A New Algorithm for Solving the Cooperative Tsume-Shogi Based on Iterative-Deepening Search

Yoshio Hoshi,[†] Kohei Noshita[†] and Keiji Yanai[†]

The cooperative Tsume-shogi, or Shogi-helpmate, is one of the most popular variants of Tsume-shogi. Solving a Shogi-helpmate problem can be formulated as an OR-tree searching. We present a new algorithm for solving Shogi-helpmate. The basic framework of our algorithm consists of depth-first searching, iterative-deepening and a transposition table. Several new techiniques for handling information in the transposition table are introduced, by which we achieve considerable memory-savings as well as speed-ups. Our program has solved the longest-step problem named Jugemu 3 with 49909 steps, as well as several hard problems, all of which have been intractable by other previous programs. By experiments we show that our techniques are effective particularly for hard problems which need a large transposition table.

1. はじめに

近年，多くの探索問題がコンピュータによって解かれている．しかし，探索問題はその深さに伴って局面の数が爆発的に増加する特徴があり，コンピュータの性能向上だけでは解けない問題が数多くある．そこで探索問題を解くためには，アルゴリズムの改良が不可欠である．

探索アルゴリズムには数多くの研究がある [13),14)]．その中で基本的なものとして，深さ優先探索 (depth-first search, DFS)，幅優先探索 (width-first search, WFS)，最良優先探索 (best-first search, BFS) がある．このうち WFS と BFS は，探索中の節点の集合を記憶するために，一般に大きい作業用記憶領域が必要である．一方 DFS は，探索の深さに比例する作業用記憶領域ですむ線形記憶領域の探索法である．1985 年に IDA*が発表された [7),19)]．これは反復深化法 (iterative-deepening) の考え方を一般化した深さ優先探索で，A* を代表とするような BFS と同等の探索を線形記憶領域で実現する方法である．チェスで用いられてきた反復深化法は探索の深さを順次増加するのに対して，IDA*の反復深化法は評価関数で閾値を定め，それを順次増加するものである．この方法は強力であり [19)]，多くの研究が続いている．例えば，同様の性質を持つ再帰的最良優先探索 (recursive best-first search, RBFS)[8)]，深さ優先探索と局面表（トランスポジション表，transposition table）の技法を組み合わせた探索法 [15)] などがある．なお，WFS は，評価関数で定める閾値を探索の深さにとる IDA*の特殊なものとして実現できる．

詰将棋はよく知られているように玉を詰めるパズルである．与えられた問題図に対して，先手と後手は交互に指し，先手は王手の連続によって後手玉を最短手数で詰め，後手は詰手順が最長手数となるように王手を回避する．

† 電気通信大学情報工学科
Department of Computer Science, The University of Electro-Communications

協力詰将棋（協力詰）は，詰将棋のルールを変形したパズルである．与えられた問題図に対して先手と後手は協力して最短手数で後手玉を詰める．詰将棋と同様に数多くの問題が作られているが，協力詰には詰将棋よりはるかに詰手数の長い問題がある．

詰将棋や協力詰は，局面を節点，候補手を枝とするゲーム木で表現できる．詰将棋の問題が AND-OR 木の探索問題として定式化できるのに対し，協力詰の問題は OR 木の探索問題として定式化できる．

詰将棋は日本人になじみが深く，また人工知能研究のよい例題と考えられ，コンピュータで解くための研究が盛んに行われてきた．1990 年当時，詰将棋を解くプログラムは DFS を用いたものが主流であり，長手数問題はほとんど解くことができなかった [3),4)]．1994 年に，伊藤，河野，野下は BFS を用いたプログラムによって，超長手数問題の「寿」（611 手詰）を解いた [5)]．1997 年に，脊尾は共謀数を用いた探索法 [16)] によって，詰将棋の最長手数問題である「ミクロコスモス」（1525 手詰）を解いた．最近，証明数と反証数を用いた探索法 [10)] によって，ほぼすべての詰将棋問題をコンピュータで解くことができるとの報告がある [11)]．

協力詰を解くプログラムは，石黒による fm[6)] が知られている．また，野下による T3[5)] がある．T3 は最良優先探索で詰将棋を解くプログラムであるが，これを OR 木探索用に調整したものである．本稿ではこれを T3(OR) とよぶ．1994 年に，fm と T3(OR) の解答能力を評価する実験が行われた．性能評価には，橋本によって選ばれた 95 題の難しい問題が使用された．当時，fm と T3(OR) はそれぞれそのうち 85 題を解いている．

本論文は協力詰を解く新しいアルゴリズムを提示する．探索の枠組として反復深化法と局面表の技法を組み合わせた深さ優先探索を用いる．ここでの新しい技法は，主として局面表に関するものであり，多くの局面を枝刈りする技法，局面情報を整理する技法が中心である．本稿の主な結果は，従来解けなかった非常に解手数の長い問題を解けるプログラムを開発したことにある．

本論文のプログラムの基本となる方法 (アルゴリズム) の特徴を説明する．本方法では，深さ優先探索，反復深化，局面表を基本的な枠組に採用しているが，この 3 つの方法はそれぞれよく知られている．反復深化の繰り返しのパラメータに，(最近盛んに研究されている一般的な評価関数の値でなく) 探索の深さ (根からの距離) を採用している．反復深化の技法は，チェスや将棋のような実時間の対局プログラムで，読みの打ち切り時点における着手の選択，さらに，(次回の反復時の) 着手の順序付けに利用されている [9),12)]．詰将棋プログラムの AND/OR 探索で反復深化は，最適解，すなわち手数に関して先手後手の最強応手で定まる解を保証するために利用されている [3)]．なお，その後の最良優先探索のプログラムは，解答能力が著しく高いが，通常出力した解の最適性が保証できていない．本稿の扱う協力詰でも，反復深化により解の最適性が保証できる．さらに，反復深化によって，(はじめて走査する節点の走査順序の意味で) 幅優先探索と同等の探索を実現し，これを基礎にして，新しい局面表の技法を開発することができた．その主要なものとして第一に，同一局面の集合の代表に最も浅い (根に最も近い) 節点を用いる技法，第二に，節点の集まりを分類するための保護局面の技法である．この技法によって，記憶領域の大きい節約が実現でき，それによって時間的な能率向上も実現できる．その結果として，これまでのプログラムで解くことのできなかった難しい問題，特に，最長手数問題である「寿限無 3」（49909 手詰）[17)] を解くことに成功した．なお，本論文のアルゴリズムは，協力詰特有の知識を利用していないので，他の探索問題への応用も可能である．

2. 協力詰について

協力詰は，詰将棋のルールを変形したパズルである (例えば [6),17)])．協力詰の用語は大部分のものが詰将棋と共通である [3),4)]．協力詰の問題では，問題図 (駒の初期配置) と詰手数が与えられる．与えられる問題図に対して，先手と後手は協力して最短手数で後手玉を詰める．駒の動きのルールは普通の将棋と同じである．先手の着手は王手，後手の着手は王手を回避する手であることが義務づけられる．後手が王手を回避する着手がない時に詰みになる．また，詰将棋と異なり (普通の指し将棋と同じで) 無駄合の概念は無い．無駄合とは，詰手順に影響しない後手の合のことである．協力詰の問題は，解 (詰手順)，すなわち作意の解 (解答) が一意的に定まり，駒余でないものとして作られている．駒余は，後手玉を詰めた局面で先手に持駒が残るものである．協力詰の問題には，作成者の意図とは異なり完全な問題 (完全作) でないものがある．完全作でない問題には，不詰によるものと余詰によるものとがある．不詰は，問題図から詰局面が得られない問題である．余詰は，問題の作成者が意図とは異なる詰手順が存在する問題である．ここで，詰将棋とは異なり，問題の詰手数をこえる長い詰手数をもつ詰手順は余詰とみなさず，問題の正しさに影響しない．本稿

では一般に詰手順を解とよぶが，混乱のおそれがある場合には，作意の解，余詰の解，問題の詰手数をこえる解などとその都度区別して示すことにする．図 1に例題 (5 手詰) を示す．解 (作意) は 23 飛，12 玉，22 飛，11 玉，12 角成である．7 手以上の解 (詰手順) は多数ある．なお，月刊雑誌「詰将棋パラダイス」には毎号新しい問題が出題されるが，協力詰のルールの説明も時々載せられる．

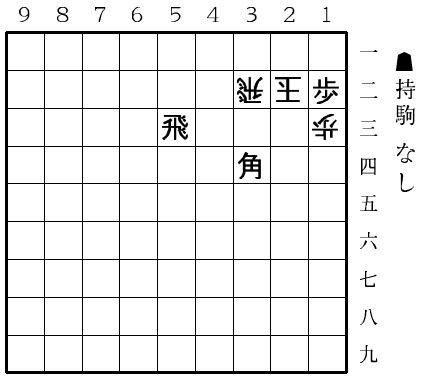

図 1　例題（5 手詰，柳田明）
Fig. 1　Sample problem (5 solution-steps) by Akira Yanagida

本論文のアルゴリズムは，問題図から出現しうるすべての局面を手数の短いものから順に網羅的に探索し，詰手順の最短のものを出力するものとして作られている (詰手順が見つかっても全探索が終了するまで探索を継続する)．

3. 探索アルゴリズム

3.1 探　索　木

協力詰の問題は，局面 (盤面と持駒の状態) を節点，候補手 (局面で指せる着手) を枝とする OR 木の探索問題として定式化できる．問題図で与えられる初期局面が根である．また，候補手が 0 の局面は葉である．葉には，先手の候補手が 0 となる不詰局面と，後手の候補手が 0 となる詰局面とがある．探索の最大深さは，問題で与えられる詰手数である．

本論文では，探索木の節点のうち，その性質によって次の 4 種類の節点を定義する．

- 打切り局面
- 既知局面
- 確定局面
- 保護局面と非保護局面

ここで打切り局面は，葉でないが，探索の限界として指定される深さにある節点であり，探索が打ち切られる局面である．既知局面，確定局面，保護局面と非保護局面については以下の必要なところで説明する．本論文では，誤解のおそれがない場合，節点とそれに付与された駒の配置 (盤と持駒の状態，すなわち単なる局面) を併せたものも局面と呼ぶことにする．

3.2 局　面　表

局面表 (トランスポジション表) の技法は，局面の情報を表に登録して，同一の局面が繰り返し出現した場合，表の参照によって同一の局面の再探索を省略 (枝刈り) するものである．現在ではゲームの探索に局面表を用いることは標準的な技法として確立している [2),4),9),12),20)]．局面表は，駒の配置をキーとするハッシュ法によって実現する．それぞれの局面は，確率アルゴリズム的手法 [9),20)] によって圧縮したビット列で表現する．ビット列の長さは 64 ビット以上が望ましいとされている．局面表に登録する局面情報は，駒の配置を表すビット列と，その局面の出現する深さである (その他局面の種類などアルゴリズムに応じた情報が付随する)．3.4 節で説明するように，本論文のアルゴリズムでは，同一局面同士はより浅い（根に近い）ものを優先して局面表に登録する．なお，同じ深さの同一局面については最初に走査した局面を優先する．

3.3 深さ優先反復深化

深さ優先探索 (DFS) は，根から葉に向かって深さを優先して走査し，葉に達すると一つ手前の節点に戻り，次の枝に対して再び深さを優先して走査を繰り返す．DFS は，探索の最大深さ (詰手数) に関して線形量の記憶領域で探索できる．反復深化法 (ID) は，まず浅い木の探索を行い，段階的に探索の最大深さを大きくしながら探索する局面を広げる．局面表を利用する本稿の探索法では，新たに走査した局面を局面表に登録する．DFS に反復深化法を組み合わせた深さ優先反復深化探索 (depth-first iterative-deepening, DFID) では，最初に探索する木の深さ $depth_1$ と，探索する深さの増分 $delta$ によって n 回目の反復深化での探索木の深さ $depth$ が定まる．DFID の枠組は次のように書ける：

for $depth := depth_1$ **step** $delta$ **until** $depth_{MAX}$
　　do $DFS(depth)$

ここで $DFS(depth)$ は深さ $depth$ までの DFS，$depth_{MAX}$ は探索の最大深さ (普通は詰手数) である．一般に，DFID は DFS よりも枝刈りする局面が増加する．なぜなら，浅い木の探索で得た結果を深い木の探索で利用できるからである．本稿では特にこの性質に着目する．

3.4 幅優先探索と同じ振舞をする探索法

幅優先探索 (WFS) は，浅い局面を優先的に探索する．WFS は探索途中での木の末端局面を全て記憶する必要があり，一般に探索が深くなるにつれて膨大な記憶領域が必要になる．

DFID において，$depth_1 = 1, delta = 1$ とすると，未走査の局面を走査する順序が幅優先探索と同じになる．本論文では，この探索法を IDWFS と呼ぶ．IDWFS は，局面表を利用すると，OR 木と WFS の特徴を生かした枝刈りを行うことができる．同一の局面同士において，この局面が詰局面に至るならば，局面表に登録した方の局面が最も短手数の解手順の上にある．つまり，別の場所の節点に現れる同一の局面の詰手数はかならず長くなる (または等しい)．なぜなら，WFS の順で節点を走査すれば，同一局面同士では最も浅い局面の方がより早い段階で局面表に登録されるからである．したがって，同一局面は最初に登録した局面以外を枝刈りできる．このように枝刈りされる局面を「既知局面」と定義する．なお，一般の IDA* などの反復深化型の最良優先探索では，同一局面のうち，最も浅い局面を最初に局面表に登録することは実際上不可能である．

協力詰では，詰将棋と同様に先手局面と後手局面との間で同一局面が生じることはない．そこで，先手局面（あるいは後手局面）のみに注目すると，$delta = 2$ としても先手局面（あるいは後手局面）は未走査の局面を走査する順序が WFS と同じになる．それで IDWFS は $delta = 2$ としてよい．$delta = 2$ の IDWFS は，反復深化の回数が $delta = 1$ の場合の半分になる．

3.5 確定局面

この節以下では，本稿のアルゴリズムは局面表に同一局面の最も浅いものを登録するということを前提にして説明する．探索結果として詰または不詰を判定してよい局面を確定局面と定義する．葉は，詰局面か不詰局面であるので，葉は探索結果が判明している．既知局面は，より浅い同一の局面がすでに局面表に登録されているので，この局面の探索を打ち切ってよい，つまり (IDWFS による OR 木探索であるので) 無視してよい．前項で説明したように，既知局面が詰手順の上にあれば，局面表に登録された同一局面が根に近いので，後者の方で先に短手数の詰手順として見つけられる．AND/OR 探索では必ずしもこのことが成り立たないことに注意されたい (AND 節点ではその子に同一局面があっても他の子の状況によって解に至るものと至らないものがある)．ある局面において，すべての子の探索結果が判明していれば，再帰的にその局面の探索結果も判明する (図 2)．

以上より「確定局面」は次のように定義できる．

- 葉
- すべての子が葉，または確定局面，または既知局面である局面

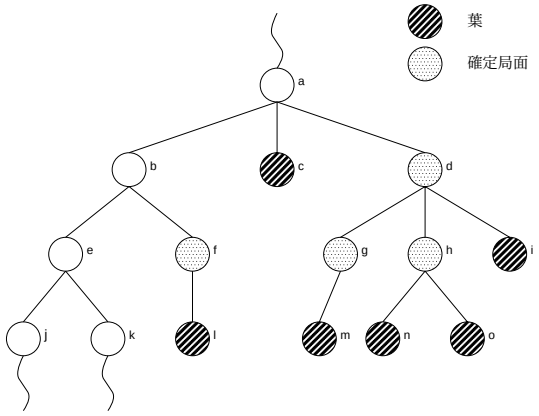

図 2 確定局面の例
Fig. 2 Example of decided positions

探索によってある局面が確定局面であることが判明すると，この局面が確定局面であることを局面表に登録する (詳しくはすでに局面表にある局面の局面情報を設定または更新する)．後の探索において，確定局面の同一局面が出現した場合，その局面を枝刈りできる．

本方法の確定局面については，IDWFS によって同一局面のうち最も浅い局面のみを局面表に登録するという点が本質的である．深い節点の局面を局面表に登録すると，相対的に浅い節点の同一局面に対して確定局面による枝刈りが常にはできない．いいかえると，同一局面の最も浅い局面を登録するという条件をはずして，確定局面の枝刈りをすると探索結果が誤ることがある．このように，最も浅い局面の登録を前提にすれば，OR 木のループに関するいわゆる GHI 問題 (graph history interaction problem)[1] が生じることはない．さらに，計算量の観点から見ると実験結果が示すように，時間と記憶領域に関して確定局面の枝刈りの効果は著しい．

3.6 保護局面と非保護局面

反復深化により DFS 探索を繰り返す場合，次回の反復以後の探索で必ず走査する局面をとりあげて，それを保護局面 (局面表の中で削除から保護する局面) とすることを考える．また，保護局面でない局面を非保護局面とする．

すべての局面は，最初保護局面とする．どの局面も局面表に登録された時点では後の探索で重要な局面で

あるかどうかを判定することはできないからである．

任意の局面において，確定局面によって枝刈りされるならば (すなわちその局面のすべての子の探索が省略できるならば)，それ以前の探索でその局面を根とする部分木が探索されている．それで，(局面表の登録情報を捨てないかぎり) この部分木に含まれる局面は局面表に記憶されている．一方，(確定局面の同一局面は枝刈りされるため) 確定局面を根とする部分木は，その確定局面自身を除いて，将来にわたって探索されることがない．それで，この部分木に含まれる局面のうち，他の探索で参照される同一局面を除くすべての局面は，(後の探索ではじめてそれに同一の局面が出現しないかぎり) 参照されることがない．それで，これらの局面を非保護局面に分類する (図 3)．

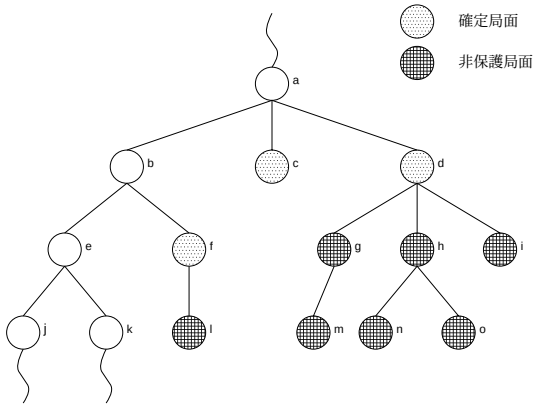

図 3　保護局面と非保護局面
Fig. 3　Protected and unprotected positions

非保護局面の登録されたアドレス（局面表の中の登録位置）へ新しい局面の登録要求があると，非保護局面の局面情報は，上書きされ削除される．また，局面表にある非保護局面が後の探索で同一の局面として参照されると，非保護局面は保護局面に戻される．

局面表には，後の探索で重要な局面とそうでない局面とが混在しているので，保護局面と非保護局面によって局面を差別化し，局面情報を整理する．参照される見込みの低い非保護局面を整理することで，限られた局面数しか登録できない局面表の記憶領域を有効に利用できる．

本論文では，ここで説明した操作を「保護ゴミ集め (ProtectGC)」とよぶ．ProtectGC は，ゴミ集め (garbage collection, GC) に似た振る舞いをする．一般に GC は，局面表が溢れた場合に局面情報を整理する．一方，ProtectGC は，反復深化の繰り返しの都度局面情報を整理する．整理された局面情報はただちには削除せず，新しい局面によって上書きされるまで保持する．それで，局面情報が削除されるまではその非保護局面の局面情報を探索に利用できる．なお，削除した非保護局面が後の探索で出現しても，(時間がかかるが) 再計算されるので結果の正しさ (一貫性) は保証される．

3.7　候補手生成の軽量化

局面を走査する時，最も計算時間を必要とするのは候補手の生成である．そこで，局面で生成した候補手を局面表に登録することが考えられる．しかし，記憶領域の大きさは限られているため，すべての局面ですべての候補手を登録することは困難である．そこで，ここでは候補手が高々1 つである局面についてのみ候補手を登録する．候補手を生成する前に局面表を参照し，候補手が登録されていればこれを探索に使用する．この工夫は特に新しいものではないが，本論文の計算実験では高速化の効果が著しい．それは，反復深化によって相対的に根に近い局面は，繰り返し走査され，その都度候補手を生成しなければならないからである．

4.　性能評価実験

協力詰を解くプログラムを作成し，性能評価実験を行う．実験には Linux の動作する Athlon 1.2GHz の CPU と 1.25GB のメモリを搭載した AT 互換機を用いる．

本実験ではアルゴリズムの異なるいくつかのプログラムを作成する．プログラムを表 1 に示す．表中の○はプログラムで使用したアルゴリズムである．△は次に説明する．表で局面表とは局面表に登録可能な局面数の最大値である．

表 1　実験に使用したプログラム
Table 1　Programs used in the experiments

	探索法	a	b	c	局面表
A	IDWFS, $delta = 1$	○	○		16777216
B	IDWFS, $delta = 1$	○			16777216
C	IDWFS, $delta = 1$	○	○		100663296
D	IDWFS, $delta = 2$	○	○	○	4194304
E	DFS	△			16777216
F	DFID, $delta = 1$				16777216
G	DFID, $delta = 1$	△			16777216
H	IDA*	△			16777216

a　：確定局面による枝刈り
b　：ProtectGC
c　：候補手の記憶による高速化

各プログラムについて簡単に説明する．A は本論文で提案する探索法で基本となるものである．B は A で ProtectGC を用いないものである．C は A で局面

表に登録できる局面情報の数を大きくとり，局面の数が多い問題を解くために使用する．D は局面表に登録できる局面情報の数を少なくとり，さらに局面表に候補手を登録することで候補手生成を高速化したものである．E，F，G，H の 4 つは性能比較のために作成した (従来の考え方に基く) プログラムである．これらは必ずしも (最初に見つけた) 詰手順の最短性を要求しない．E は通常の DFS である．F と G は DFID であり，一般の $delta(\geq 1)$ が使えるものである．表のものは単にパラメータ $delta$ を 1 に設定したものである．ここで，3.5 で説明したように，最も浅い同一局面の登録を前提にしていないので，既知局面に基く確定局面の枝刈りは使えない．それでも確定局面の一部を使って比較的自然なやり方で枝刈りが実現できる．すなわち，ある節点のすべての子の探索結果が定まると，その節点の探索結果を (再帰的に) 定めて，局面表にそのことを登録して同一局面の枝刈りに利用できる．この方法を△で示す．H は IDA*であり，評価関数として根から局面までの道の上の候補手 (分岐数) の総和とその局面から詰手数までの探索の深さの和を用いる．

4.1 解答能力の評価

協力詰を解くプログラムの解答能力を評価する．

4.1.1 橋本孝治による問題集

問題集として橋本孝治が選んだ難問集 95 題をとりあげる．これは 1994 年に，石黒俊太郎による fm と，野下浩平による T3(OR) の解答能力の評価に用いられている．当時，fm と T3(OR) はそれぞれこのうちの 85 題を解いている．なお，fm はその後も改良が続けられ，さらに 3 題を解いたとの報告がある [6]．1994 年当時，解けなかったり，解の出力に長時間を要した 11 題についての結果を表 2 に示す．本実験で使用したプログラムは A である．表中，○は解けた問題，×は解けなかった問題である．

プログラム A は，95 題のうち 92 題を解いた．解けた問題の大部分は，解を 1 つ求めるだけでなく全探索 (木全体の探索) を行うことができたが，それ以外の 4 題（問題 13，61，74，77）は全探索ができなかった．これらはいずれも多数の余詰を含む問題 (不完全作) であり，詰局面の駒の配置が微妙に異なるものが多数生成されている．これらの余詰に至るまでの局面の数は膨大であるため，局面表の溢れが生じている．この 4 題以外にも多数の余詰を含む問題が存在するが，いずれも局面の数が非常に多い．なお，3 題が解けないが，やはり局面表の溢れによるものである．

表 2 解答能力の評価
Table 2 Comparison of the solving power

問題	手数	fm	T3(OR)	A	計算時間 (s)
30	67	×	×	○	426
31	739	×	×	○	351
67	357	×*	×	○	662
72	135	○	×	○	135
79	77	×	×	○	86
88	35	×	○	○	69
91	1325	×	×	×	-
92	2157	×*	×	○	1168
93	5349	×	×	×	-
94	19447	×*	×	○	10152
95	5119	×	×	×	-

星印の問題は 1994 年以降に解けたとの報告がある (1999).

4.1.2 長手数問題

協力詰の長手数問題に対する解答能力を評価した [17]．本実験で使用したプログラムは C である．実験結果を表 3 に示す．これは，1000 手詰以上で完全作とされているもの 8 題に対する結果である．表の下の 3 題は完全性が十分検討されていないようである．そのほか 3451 手詰の問題 (鮎川哲郎) があるとされているが，問題図は不明である．

本プログラムは，プログラムによって初めて最長手数問題である「寿限無 3」（49909 手詰）を解くことに成功した．また，1000 手詰以上の問題のうち，「寿限無 3」を含む 5 題を解いた．

表 3 長手数問題
Table 3 Problems with long solution-steps

作者	作品名	手数	C
加藤徹	寿限無 3	49909	○
加藤徹	寿限無 2	38889	○
加藤徹	寿限無	19447	○
加藤徹		5349	×
西田尚史	ゴールドベルク	5119	×
西田尚史		3987	×
森茂		2157	○
鮎川哲郎		1325	×
加藤徹		7181	×
加藤徹		4197	×
加藤徹		4143	○

次に，協力詰の問題としては特に手数の長い「寿限無」,「寿限無 2」,「寿限無 3」(図 4) に対する実験結果を示す．

本実験で使用したプログラムは，A，D，および D の調整版である．最後のものは，D において問題ごとに局面表の大きさを調整したものである．

IDWFS は探索時に非常に多くの局面を枝刈りする．

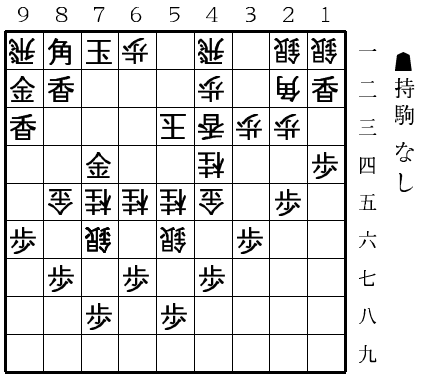

図 4　寿限無 3（49909 手詰, 加藤徹）
Fig. 4　Jugemu 3 (49909 solution-steps) by Toru Kato

表 4　「寿限無」問題の計算時間
Table 4　Computation time of Jumege problems

	寿限無	寿限無 2	寿限無 3
詰手数	19447	38889	49909
探索局面数	3452219	32829309	13418156
A (s)	10512	34426	38682
D (s)	1508	5254	5455
調整版 (s)	480	5254	3047

調整版で局面表に登録できる局面数の最大値は次の通りである：
寿限無： 262144，寿限無 2： 4194314，寿限無 3： 524288

それで，計算時間は詰手数と比べてそれほど長時間にはなっていない．本論文の探索法は局面表の大きさによって計算時間が大きく変化する.「寿限無」問題は，異なる局面の数が比較的小さいため，局面表の大きさは比較的小さくできる．また，局面表に候補手を登録することで，計算時間が短縮されている．これは，枝刈りによって局面の候補手が 1 つだけとなる局面が数多く存在することを示している．実験結果からわかったことであるが，探索が深くなると探索木は浅い局面で ‘痩せた’ 形状になり，(本稿の方法によれば) 相異なる局面の数が爆発することなく探索できる．

4.2　探索アルゴリズムの比較

次に，探索アルゴリズムの性能評価のため，その他の探索アルゴリズムによるプログラムとの比較実験を行った．実験に使用したプログラムは B，E，F，G，H である．橋本による問題集の中の問題 14，30 を例にして，実験結果を示す．問題 14，30 はどちらも 67 手詰で，比較的短手数の問題であるが，問題 14 は局面の数が少ない問題であるのに対して，問題 30 は局面の数が非常に多い問題である．

表 5 は，問題 14 に対する比較実験の結果である．ここで，走査局面数とは，走査した局面の延べ数であり，探索局面数とは，探索中に出現した相異なる局面の総数である．

問題 14 では，DFS が高速に全探索を終了している．その理由は，E(DFS) では反復深化を行わないためである．反復深化による探索では，反復深化の度に，DFS による重複した計算（節点の再走査と局面表の値の更新）が必要である．E 以外のプログラムの計算時間は，その大部分がこの重複した計算の時間である．すでに説明したように，プログラム F と G は，確定局面の考え方を利用した枝刈りを行うかどうかの点で異なり，またプログラム B と G は，既知局面を無視できるかどうかの点で異なる．これより，IDWFS は枝刈りによって走査局面数が大幅に減少することがわかる．

表 5　アルゴリズムの比較（問題 14）
Table 5　Comparion of the algorithms (Problem 14)

プログラム	走査局面数	計算時間 (s)
B	18508	26
E	7862	2
F	52704	26
G	31135	25
H	116724	34

詰手数： 67，探索局面数： 2862

表 6 は，問題 30 に対する比較実験の結果である．問題 30 では，IDWFS が高速に全探索を終了している．走査局面数の多い問題 30 では，反復深化で必要な重複した計算の時間が計算時間全体と比較して相対的に小さい．すなわち，計算時間は走査局面数に大きく依存する．問題 14 の結果と比較すると，問題 30 では既知局面を無視する効果が大きく，走査した局面の数は著しく減少している．

この実験に関連して刻み delta を選ぶことにふれておく．プログラム G において刻みを変えてその中で良い値を選ぶと，問題 30 ではかなり速くなる（この場合 delta=8 で 580 秒）．また，プログラム B において delta=2 に選ぶと，表の 432 秒が 250 秒まで速くなる（3.4 節参照）．比較の対象のプログラムで良い刻みを選んでも，(他の問題に対して) 解答能力をふやすほどの速度向上の効果はない．なお，多くの刻みの中から良い値を選ぶことは，一般に問題を何度も解いてはじめてできることであり，問題に依存するパラメータを利用したプログラムになるおそれがある．

4.3　局面表を整理する効果

次に ProtectGC によって局面表を整理する効果を検証する．実験に使用したプログラムは A と B である．橋本による問題集の問題 30，94 を例に用いる．実

表 6　アルゴリズムの比較（問題 30）
Table 6　Comparison of the algorithms (Problem 30)

プログラム	走査局面数	計算時間 (s)
B	27146076	432
E	113330341	1762
F	204106708	3108
G	189986648	2966
H	955175792	14629

詰手数：67，探索局面数：7909636

験結果を表 7に示す．なお，問題 94 は長手数問題の「寿限無」（19447 手詰）である．最大保護局面数とは，全探索を通じて整理しない局面の数の最大値である．

表 7　ProtectGC の効果
Table 7　Effect of ProtectGC

問題	ProtectGC	最大保護局面数	計算時間 (s)
30	使用 (A)	1818028	426
	不使用 (B)	7909636	432
94	使用 (A)	60992	10152
	不使用 (B)	3452219	9575

この表によると，ProtectGC の使用による計算時間の変化はわずかである．問題 30 では 77%，問題 94 では 98% 程度の局面が非保護局面として整理されている．整理される局面の数は問題によってばらつきがあるが，橋本による問題集では探索局面数全体の約 75% である．これは，局面表に登録される局面情報を 1/4 程度に整理したことを示す．ProtectGC により記憶領域の有効利用が可能になることがわかる．本プログラムで解けない問題はすべて局面表の溢れによるものであるので，ProtectGC のような技法の開発が重要であるといえる．なお，局面表があふれる都度適当な基準でゴミを回収して再利用する普通の GC を利用する方法も考えられる．すでに過去にも多くの実験されているが，本研究の過程でも実験を行ったが，解答能力でも計算時間でも本論文の結果より良い結果はえられなかった．本論文の方法に単純に普通の GC を組み込んでも解答能力がふえないと判断されるが，この方向での詳しい研究は今後の研究課題とする．

5. おわりに

本論文では，探索問題として協力詰を解くためのアルゴリズムを詳しく説明した．反復深化法と局面表の技法を組み合わせた深さ優先探索により幅優先探索と同じ振舞をする探索法を実現し，この枠組のもとで，局面表に関する新しい技法を提案した．この技法によって，探索中に走査する局面の数を減らすこと，局面情報の整理による記憶領域の有効利用ができることを示した．本論文の最大の成果は，協力詰の問題の中で詰手数の最も長い問題がプログラムで解けることをはじめて示したことである．

今後の課題としてまず考えられるものは，詰将棋を解くために最近提案されている強力な方法を協力詰将棋に応用することである．この場合，本稿のような解の最短性を保証するという条件をみたさなくてもよいとする．詰将棋とは異なり，そのような方法で本稿のプログラムより解答能力の高いものを開発することは容易でないと思われる．

本稿のアルゴリズムは協力詰に特化した知識を利用していないため，他の探索問題への応用も容易であるが，実際に他の問題に応用して性能評価することは今後の課題である．

最後に，協力詰を一般化した計算問題の複雑さについてふれる．詰将棋を N × N 盤の問題に拡張して，詰手順があるかどうかを判定する問題については指数時間完全であることが知られている [18]．一般化した協力詰問題について，詰手順の存在を判定することは NP 困難である (証明にはハミルトン路問題を用いる)．また入力に詰手数 (2 進数) が与えられるので，詰手順が与えられれば，(OR 木であるので) その正しさが直接的にチェックできる．それで一般化した協力詰問題は NP 完全であることがわかる．ただし，この問題は，解（詰手順）の存在を判定するものであり，一意的な解の存在を判定するものではない．

参 考 文 献

1) Breuker, D. M., van den Herik, H. J., Uiterwijk, J. W. H. M. and Allis, L. V.: A Solution to the GHI Problem for Best-First Search, *CG'98*, *LNCS* 1558, 25–49 (1999).
2) Greenblatt, R.D., Eastlake, D.E. and Crocker, S.D.: The Greenblatt Chess Program, Proc. FJCC, 801–810 (1967).
3) 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).
4) 伊藤琢巳, 河野泰人, 脊尾昌宏, 野下浩平: 詰将棋を解くプログラムの進歩, 人工知能学会誌, 10, 6, 853–859 (1995).
5) 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).
6) 神無七郎: フェアリー詰将棋のホームページ, http://www.coms.ne.jp/~k_7ro/.
7) Korf, R. E.: Depth-First Iterative-Deepening: An Optimal Admissible Tree Search, *Artificial*

Intelligence, 27, 1, 97–109 (1985).
8) Korf, R. E.: Linear-Space Best-First Search, *Artificial Intelligence*, 62, 1, 41–78 (1993).
9) Levy, D. N. L. and Newborn, M.: How Computers Play Chess, *W. H. Freeman and Company*, New York (1990).
10) Nagai, A.: A New AND/OR Tree Searching Algorithm Using Proof Number and Disproof Number, *Workshop of the First International Conference on Computers and Games (CG'98)*, 40–45 (1998).
11) 長井歩: コンピュータ詰将棋の最新成果: 超長手数問題への挑戦, コンピュータ将棋協会誌, 13, 34–42 (2000).
12) Newborn, M.: *Kasparov versus Deep Blue, Computer Chess Comes of Age*, Springer-Verlag, New York (1996).
13) Nilsson, N.J.: *Problem-Solving Method in Artificial Intelligence*, McGraw-Hill, New York (1971).
14) Pearl, J.: *Heuristics*, Addison-Wesley, Reading (1985).
15) Plaat, A., Schaeffer, J., Pijls, W. and de Bruin, A.: Best-First Fixed-Depth Minimax Algorithms, *Artificial Intelligence*, 87, 255-293 (1996).
16) 脊尾昌宏: 共謀数を応用した詰め将棋プログラムについて, ゲーム・プログラミングワークショップ'95 論文集, 128–137 (1995).
17) TETSU(加藤徹): おもちゃ箱 CD-ROM (1999), http://www.ne.jp/asahi/tetsu/toybox/.
18) 横田, 築地, 北川, 諸橋, 岩田, 一般化詰将棋問題の指数時間完全性, 電子情報通信学会論文誌, J84-D-I, 3, 239–246 (2001).
19) Zhang, W. and Korf, R. E.: Performance of Linear-Space Search Algorithms, *Artificial Intelligence*, 79, 2, 241–292 (1995).
20) Zobrist, A.L.: A Hashing Method with Applications for Game Playing, Technical Report 88, CS, University of Wisconsin (1970).