論 文

Technical Papers

必至問題を解くプログラムとその評価

A Brinkmate Solver and Its Evaluation

橋本 剛 Tsuyoshi Hashimoto
静岡大学大学院理工学研究科
Graduate School of Science and Engineering, Shizuoka University
hasimoto@cs.inf.shizuoka.ac.jp

作田 誠 Makoto Sakuta
（同　上）
sakuta@cs.inf.shizuoka.ac.jp

飯田 弘之 Hiroyuki Iida
静岡大学情報学部情報科学科
Department of Computer Science, Shizuoka University
iida@cs.inf.shizuoka.ac.jp

keywords: computer shogi, brinkmate search, tsumeshogi, proof-number search

Summary

Brinkmate (*hisshi* in Japanese) is an important notion of accessing to the opponent's King in shogi. This is essentially the same as conventional chess mating problems, where all moves are considered. However, in shogi the problem is much more difficult, as the possibilities for delivering check or threatmate, and the number of defenses are much greater. The defending side may have 200 or 300 possible defensive moves to consider. Brinkmate search is resolved into an AND/OR-tree search based on the concept of *threat sequence*, proposed by Iida. The cost of brinkmate search is by far more expensive than mating (tsumeshogi) to find a solution.

Since not much is known about brinkmate itself or brinkmate search, we first explain it by giving the definition of brinkmate. In an AND/OR tree of brinkmate search, to determine effective branches (i.e., legal moves) at any internal node is often a time-consuming task. This paper proposes a new search algorithm, denoted by SPH, for an AND/OR-tree search. The algorithm is implemented in a shogi-hisshi (Japanese-chess brinkmate-problem) program, and evaluated by testing it on difficult hisshi problems. Moreover, it is enhanced by several methods including a new idea, denoted by TDSS. The experimental results are compared with those of other programs. The program with TDSS shows the best results for solving short-step problems, while the SPH program in general outperforms the other programs in solving long-step problems.

1. はじめに

必至は将棋の終盤において，詰みと同等かあるいはそれ以上に，重要な概念である．必至となった局面では，受け方はいかなる応手を用いても，自玉の詰みを防げない．それゆえ，詰みがない状況では必至に導く手順が存在すれば勝ちとなる．近年，詰将棋を解くプログラムの開発では顕著な成果 [伊藤 95, 脊尾 98] がみられるが，これとは対照的に，必至問題を解くプログラムの研究はあまり知られていない．プロ棋士であり詰将棋作家としても有名な内藤九段は「あれだけ詰めの能力にすぐれているのに必死が全然ダメというのは，コン君の頭の中はいったいどうなっているのかと不思議な気分になる.」とコメントした [内藤 99].

必至は攻め方が詰めろ (王手でもよい) の連続で受け方を強制的に受けなしに導くのであるから，必至探索は詰将棋探索と同様に AND/OR 木探索に帰着される [飯田 98]．しかし，攻め方の指し手が詰めろであるかどうか，または，受け方の指し手が防ぎ手となっているかどうかの判定は，非常に手間のかかる作業である．実際には，これらの判定はその度に詰みルーチンをコールして行うべきであるから，必至探索での AND/OR 木生成には膨大なコストを要する．飯田の報告 [飯田 98] によれば，最も簡単な 1 手必至の場合でさえ，平均して 100 〜 200 回詰め将棋をコールしなければならない．単純に計算すると，手数が増えればこれらは指数的に増加する．それゆえ，必至探索を効率的に行う必要がある．

本稿では，最良優先探索のようにふるまう深さ優先探索の新しいアルゴリズムを提案する．これは近年，詰将棋の分野で考案された PN*アルゴリズム [Seo 01](または C*アルゴリズム [脊尾 98]) と似たふるまいをするが，探索の方向付けのために証明数を用いる代わりに，その近似をより高速に与える評価関数を用いるのが大きな特徴である．必至の AND/OR 木探索では，詰将棋と比べる

と，証明数を求める手間がかなり大きいからである．また，必至探索での AND/OR 木生成を効率化する工夫を提案する．これらのアイデアを実装した必至探索プログラムを開発し，難問を集めたテスト問題を用いて評価実験を行う．

2. 必 至 と は

2·1 必至の規則と用語

詰めろと強制手順の概念を用いて必至を定義する ([飯田 98] より引用)．便宜上，必至をかける側を**攻め方**とし，受ける側を**受け方**とする．

【定義 1】 詰めろとは，次に受け方の玉を詰ますことができる局面に導く攻め方の手である．

【定義 2】 強制手順とは，攻め方による詰めろまたは王手，あるいは，それらを回避する受け方の応手を含む手順である．

【定義 3】 強必至とは，ある強制手順によって以下のような条件を満たす局面 (P とせよ) に導くことである．

1. 受け方は局面 P で勝ちがない．
2. 王手を除く受け方のいかなる手に対しても攻め方は受け方の玉を詰ますことができる．
3. 強制手順中に受け方は攻め方の玉に対して王手をすることができ，その王手とそれに対する受け方の手も強制手順に含める．

【定義 4】 弱必至とは，ある強制手順によって以下のような条件を満たす局面に導くことである．

1. 王手を除く受け方のいかなる手に対しても攻め方は受け方の玉を詰ますことができる．
2. 受け方は強制手順中には攻め方の玉に対して王手をしない．

定義 3 によれば，強必至探索において攻め方は常に以下のことを考慮しなければならない．

- 攻め方の玉に詰みはないか．
- 受け方が攻め方の玉に王手をして詰めろを避けられるか．

明らかに，強必至探索は弱必至探索よりはるかに多くの計算量を要する．この場合，計算量の違いは必至探索中の受け方による (攻め方の玉に対する) 王手の数に依存する．攻め方の玉がない必至問題 (双玉でない) を対象とする場合，強必至と弱必至は同等である．

以下，与えられた双玉でない必至問題を解くプログラムを議論するので，弱必至に焦点を当てて議論することになる．特に断らない限り，必至という用語は弱必至を意味する．

2·2 必至問題の正解手順

図 1 に必至問題の例を示す．攻め方は王手あるいは詰めろの手を選択する必要がある．まず☗３一角成と王手

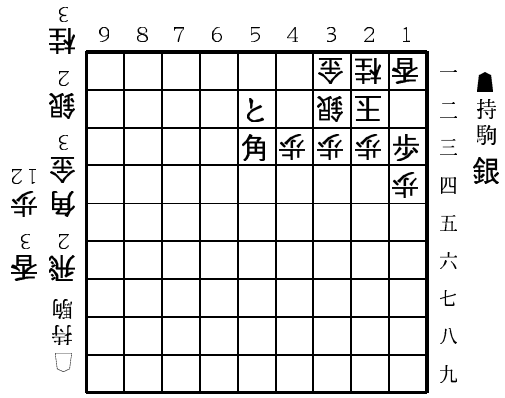

図 1 必至問題の例

をする．これに対する応手は☖３一同玉しかない．攻め方は次に☗１二歩成とする．これは☗２二金または☗４二金の１手詰めろである．受け方はこれを防ぐためには☖１二同香しかない．次に攻め方が☗１一銀とすると，受け方には☗２二金または☗４二金の１手詰めろを同時に防ぐ手が存在しないので，完全に必至がかかったことになる．正解手順は☗３一角成☖同玉☗１二歩成☖同香☗１一銀までの５手必至である．必至の解答手順には，必至がかかった後の詰み手順の手数はカウントしない．

2·3 必 至 と 探 索

必至問題の解決のための探索空間において，攻め方の手番での有効手は王手あるいは詰めろをかける手である．王手は局面から直接数え上げることができる．一方，詰めろをかける手は，王手以外のすべての合法可能手について，その手を指した後，受け方がパスしたと仮定して攻め方が玉を詰ませることができるかどうかを詰み探索によって調べ上げる必要がある．同様に，受け方の手番での有効手は王手を防ぐ手あるいは詰めろを防ぐ手である．王手を防ぐ手は局面から直接数え上げることができる．一方，詰めろを防ぐ手は，すべての合法可能手について，その手を指した後，攻め方から玉を詰まされないかどうかを詰み探索によって調べ上げる必要がある．

攻め方の手番局面で王手あるいは詰めろをかける手が存在しない局面は失敗末端局面となり，値 false を持つ．また，受け方の手番で王手を防ぐ手が存在しない局面，あるいは詰めろを防ぐ手が存在しない局面は成功末端局面となり，値 true を持つ．したがって，必至問題の探索空間は AND/OR 木探索に帰着される．(実際には，同一局面が生じる場合があるので AND/OR グラフになるが，多くの問題は AND/OR 木として探索され，別に同一局面チェックを行なうことによって対処される．) 一般に，必至問題では有効手の数はそれほど多くない．しかし，有効手かどうかの判断のために何度も詰み探索ルーチンをコールする必要があり，これが非常に重い処理となる．

3. 必至探索に関する従来の研究

ごく初期の 1987 年に吉川 [吉川 87] が強いヒューリスティックスを使った必至探索アルゴリズムを紹介しているが，その後しばらく必至探索に関する報告はなく，1998 年に飯田 [飯田 98] の必至探索に関する論文が出た．飯田はまず必至の明確な定義をし，一手必至と王手一手必至のアルゴリズムを示すとともに，N 手必至の概念を導入し強制手順による一般化された必至探索アルゴリズムを示している．これは深さ固定の単純なしらみつぶし型必至探索アルゴリズムで，攻め方受け方ともに各ノードでまず全候補手を生成し指し手の順序付けを行わないなど改良の余地を多く残している．また，このアルゴリズムで短手数 (1 〜 7 手) の問題を解くことに成功したことを報告している．

1998 年に山下 [山下 98] は必至探索を実戦に応用することの難しさ，つまり強必至と弱必至の相違が実戦での応用を困難にしていることを指摘している．最近では，2000 年に橋本 [橋本 00] が内藤 [内藤 99] の「今月はコンピュータで解けない上級必至問題です」との注釈付で出題された 17 手必至問題を約 2 時間 30 分で解いたことを報告している．さらに，有岡 [有岡 00] は必至探索に関する数々の手法や詳しい研究成果を報告している．有岡は必至探索の方法として，深さを閾値とする縦型の反復深化をはじめ，独自の方法で，証明数を閾値とした多重反復深化のアイデアも試み，金子 [金子 96] の必至問題集を非常に高速に解き優れた結果を報告している．有岡の探索は，攻め方，受け方とも玉の周りに利きを付ける手や駒を取る手など，有効そうな手だけを生成し，それらの手に対して詰みルーチンをコールして合法手かどうか調べる，という非常に実践的なヒューリスティックスを用いているのが特徴である．

将棋以外で必至探索に関係する研究としては，将棋の必至問題のように n 手先に fatal な手がある場合の探索の一般的な扱いに関して Lambda-search という名称を使い，その囲碁への応用について述べている Thomsen の研究 [Thomsen 00] があり，将棋の必至はこの研究の λ^2-search に相当する．

4. 必至探索プログラム TACO-H

我々の開発した必至探索プログラム TACO-H は，将棋プログラム TACOS を基に開発したもので，必至探索に関する研究としてよく知られている飯田 [飯田 98] のアルゴリズムに比べてかなり改良点は多い．飯田のアルゴリズムと TACO-H の相違点を表 1 に挙げる．

TACO-H の基本は縦型探索で深さを閾値として反復深化を行っている．以下，TACO-H の特徴について詳しく述べる．

表 1 飯田アルゴリズムと TACO-H の相違点

	飯田	TACO-H
探索深さ	固定	反復深化
合法手の生成	すべて	攻め方は枝刈
TDSS	なし	あり
指し手の順序付け	なし	あり
詰みルーチン	深さ固定	反復深化
その他		無駄手処理
		一手先読み

4·1 合法手の生成

必至探索での合法手の生成は詰みルーチンをコールする重い作業なので，反復深化の際には攻め方合法手のリストを局面表 (ハッシュ表を使った transposition table) に登録するなど，合法手生成のコストを削減するよう工夫している．合法手生成に関して，上述した有岡の方法は実践的には非常に有効な手法だと思われるが，この方法では受け方のあらゆる受け手を試みたことにならないため，必至であるという解が出ても，本当の意味での解を保証しない．もっとも，詰めろ判定用詰みルーチンの探索の最大深さを無限大にしないと厳密な意味で解は保証されないが，本稿では，与えられた詰みルーチンの探索最大深さにおける必至解の保証ということを重視する．そのため，TACO-H の基本方針として，攻め方は有岡と同様の手法で見込みのありそうな手だけを生成し，受け方はすべての指し手を合法手となり得るかどうかチェックする．しかし，すべての手に対して詰みの有無を確認していると膨大な計算量を要する．そこで，TACO-H では棚瀬 [棚瀬 00] と同様の方法（Threatmate-Defense Search using Similarity or TDSS と呼ぶ）を用いて，受け方が詰みルーチンをコールしないで合法手かどうか判定しコストを削減している．

4·2 TDSS (Threatmate-defense search using similarity)

詰めろのかかっている局面で，受け方がある指し手を選択し，その手が本質的に詰めろに対して影響を与えていなければ，本来の詰めろの手順と同じ手順で詰んでしまう．そこで，局面表に登録してある一つの詰み手順を完全に再現できるかどうかを調べれば，もし再現できた場合には詰めろを明らかに防いでないことが容易にわかり，詰みルーチンを実際にコールしなくてもよい．ただし，詰み手順が複数あるときは全ての手順を同時に防ぐ手を指さないといけない，すなわち，少なくとも一つの詰み手順を防いでいない手は明らかに詰めろを防ぐことを失敗している手である．詰み部分木 [棚瀬 00] を使うこの方法を TDSS と呼ぶ．

TDSS の基本方針は以下のようになる．

- 詰みがある局面を P，受け方が任意の可能手 S を指した局面を P' とする．

- P の詰み手順 T を記憶する．
- ここで，T は攻め方の手は最善手 1 手だけ，受け方の手はすべて記憶する．
- P' 上で P の詰み手順を攻め方の手は 1 手だけ，受け方の手はすべて再現する．
- 完全に再現できたら，S は明らかに詰みを防いでいない手である．

つまり，記憶する手順は木構造で，その木を完全に再現できるかどうかを調べることで，再現できた場合には少なくとも一つの詰み手順が存在することが保証される．完全に再現できない場合，すなわち指し手が詰みに何らかの影響を与えている場合には，詰みルーチンをコールして詰むかどうかを確認する．この場合，TDSS を実行したことが無駄になってしまうが，実際には詰めろに影響を与えない指し手の方が圧倒的に多いため，全体としてコストが削減される．なお，TACO-H では TDSS をより有効に実行するために，まず玉の安全度を計算し，詰みに影響を与えている可能性が高いと思われる手に対しては TDSS を適用しないで，直接詰みルーチンをコールする．また，TDSS を攻め方の節点にも適用して，受け方の手は最善手 1 手だけ，攻め方の手はすべて記憶し，詰めろにならないことを示す手順を木にし，明らかに詰めろにならない手を高速に判別している．

4·3 指し手の順序付け

探索の効率化には指し手の順序付けが重要であるが，TACO-H では詰みルーチンをコールする前に，局面の静的な情報（玉の安全度と玉の周りマスの利き制圧度，以下簡単にそれぞれ安全度，制圧度と呼ぶ）と指した手に関する情報 (無駄捨てかどうか，取り駒の価値，玉との距離) に基づいて，すべての手を順序付けしてから合法手かどうかチェックする．玉の安全度とは玉の移動できるマスの数を意味し，TACO-H ではこれに盤の端に接するマスの合計を 1 マスと数える補正 [有岡 00] を加える．玉の周りのマスの利き制圧度は，玉の 3x3 近傍で攻め方と玉方の利きの数をそれぞれ合計し差を取った値である．無駄捨てかどうかは，駒を取らない指し手のとき手を指した後の局面で駒を移動したマスの利きの数を調べ，指した側の利き数が相手方より少ないとき無駄捨てであると判断する．TACO-H では予備実験を元に各パラメータを設定し，これらの情報を以下のように計算して攻め方では低い順に，受け方では高い順に並び替える．

攻め方: 安全度 $\times 1000 +$ 制圧度 $\times 50 +$ 指し手情報
玉方: 安全度 $\times 1000 +$ 制圧度 $\times 50 -$ 指し手情報

指し手情報 ＝ 取る駒の基本価値/10（駒取りの場合）
－動かす駒の基本価値/20（無駄捨ての場合）
＋動かす駒と王との距離

ここで駒の基本価値は YSS の駒の価値表の値 [山下 98] を使っている．基本価値は最大の飛車で 1040 なので以下の様に順序付けしていることになる．

安全度 ≫ 制圧度 > 取り駒 > 無駄捨て駒 ≫ 王との距離

ここで ≫ は常に左が大きい，>はおおよそ左が大きいが右の方が大きい場合もあるという関係を示す．

4·4 詰み探索ルーチン

詰めろの判定をする詰み探索ルーチンは，深さ優先の縦型探索で深さを閾値とした反復深化を行っている．指し手の順序付けは一手指した後の相手方の合法手が少なくなる手を優先している．また，局面表を使用して同一局面と盤面が同じで持ち駒の優越関係が明らかになる局面の探索を省略している．

4·5 そ の 他

有岡の示した，詰め将棋における無駄合い処理に相当する「取られるだけの無駄な手数伸ばしへの対応」と,「1 手の先読み」の手法が TACO-H においても有効であったので，これを採用している．両者の定義を以下に示す．

- 「取られるだけの無駄な手数伸ばしへの対応」N 手必至の探索において，N 手目を指した局面で，攻め方の利きのあるマスに受け方が駒を移動もしくは打つ手を M として，M を指さないと（0 手）必至の状態であるとする．M を指し，攻め方がその指された駒を取って詰めろをかけた状態が（0 手）必至であるとき，M は無駄な手数延ばしであるとする．ここで,（0 手）必至を調べるときも，無駄な手数延ばしを考慮する．

- 「1 手の先読み」攻め方の手番のノードで 3 手以上手数が残っているとき，まず 1 手必至を調べる手法である．簡単な 1 手必至があるにもかかわらず，手を展開する順番が悪くてなかなか結果にたどり着けないという状況に対応するもので，必至では大きな効果がある．

4·6 アルゴリズム

TACO-H のアルゴリズムを図 2 と図 3 に C 風のコードで示す．互いに再帰的に呼び合う attack() と defence() を図 3 で定義し，探索のメインとなる図 2 の search() をルート局面で呼ぶ．結果が必至あるいは不必至（必至にならないことが証明できた状態）であることがわかれば探索を終了する．

5. 実験:深さを閾値とした縦型反復深化

TACO-H に，有岡 [有岡 00] と同じく金子の必至問題集 [金子 96] のうち難問の 7 手必至以上の問題 No. 69 〜 100 と上述した内藤 [内藤 99] の 17 手問題を解かせて性

```
int search(){
    for ( 末端深さ = 1;; 末端深さ += 2 ){
        int r = attack(0);
        if ( r = 必至 または 不必至 ) return r;
    }
}
```

図 2 TACO-H の基本アルゴリズム 1

能評価した．マシンは CPU が Pentium III 800MHz，メモリが 288M バイトである．局面表は内部詰み探索用（局面が詰む，詰まないの情報を保存）と必至探索用（局面が必至になる，ならない，不明の情報を保存）を別に用意した．詰み探索用は約 50 万局面分を確保し，詰み探索を呼ぶ前に局面表に全体の 75%以上の登録があれば局面表をクリアし探索を続ける．必至探索用は約 100 万局面分を確保し，全体の 75%登録した時点で探索を打ち切らせている．なお，実験用のパラメータとして重要なのは合法手の判別で呼ぶ詰みルーチンの最大探索深さであるが，ここではすべての問題を解いた中で最小の深さ 5 を採用している．なお，有岡のように攻め方と受け方でこのパラメータを違う値にはしていない．必至探索の場合，探索節点数の定義が難しいので，局面を更新する関数をコールした回数をカウントした．

5·1 結　　果

実験結果を表 2 に示す．参考のため，有岡の解を保証しない「深さをしきい値とした反復深化」アルゴリズムの詰手数パラメタを攻め方 5 手，受け方 11 手とした結果 [有岡 00] を一番右のカラムに示す．有岡の結果に比べて時間がかかっているが，解を保証する我々の方法でもすべて解くことができた．特筆すべきは，2000 年の報告 [橋本 00] で約 2 時間 30 分，局面更新数 2.76 億回かかっていた内藤の 17 手問題が，今回の実験では，12 分弱，局面更新数 95 万回で解けるようになったことである．時間短縮については CPU のスピードが上がったことも一つの理由であるが，攻め方の合法手候補をすべて読んでいたのを見込みのありそうな手だけに絞ったことや，指し手のソートに使う評価関数の改良などが大きく影響している．なお，実際に解いた手数が有岡の結果と違う問題がいくつかあるが，これは合法手判別用の詰みルーチンの最大深さを 5 にしているため，実際よりも手数を長く数えるためである．

6. 安全度優先必至探索 (Safety-Priority Hisshi Search or SPH)

6·1 背　　景

本節では，内藤九段の 17 手必至のような長手数の必至問題を解くために，PN*[Seo 01] のような探索アルゴリズムを必至探索に適用することを検討する．詰め将棋で

```
int attack( depth ) {
    局面表で同一局面を調べる
    if ( 登録値 = 必至 または 不必至) return 登録値
    if ( はじめて調べる局面 ){
        指し手候補を生成
        if ( 指し手数 = 0 ) return 不必至;
        指し手の順序付けをし並び替える
        if ( !末端 ) r = 必死探索深さ 1 の軽い探索
        if ( r = 必至 ) return 必至;
        for ( すべての指し手 ){
            if ( 詰み判定 = 王手でも詰めろでもない)
                continue;
            その手で局面を進める
            r = defence( depth + 1 );
            局面を戻す
            if ( r = 必至 ) return 必至;
            else if ( r = 不明 )
                指し手をリストに登録
        }
    }
    else{ /* すでに調べた局面 */
        for ( すでに登録してある指し手すべて ){
            その手で局面を進める
            r = defence( depth + 1 );
            局面を戻す
            if ( r = 必至 ) return 必至;
            else if ( r = 不必至)
                指し手をリストから削除
        }
    }
    if (r = 不明 となる手が一つでもある)
        return 不明
    else return 不必至;
}
int defence( depth ) {
    局面表で同一局面を調べる
    if ( 登録値 = 必至 または 不必至) return 登録値
    int r = 必至;
    if ( すでに調べた局面 ){
        前回調べた指し手で局面を進める
        r = attack( depth + 1 );
        局面を戻す
        if ( r = 不明 ) return 不明;
        else if ( r = 不必至) return 不必至
        /* 攻め方の勝ちが見つかった場合
        まだ調べていない局面から
        調べなおさないといけない */
    }
    else {
        指し手をすべて生成
        if ( 指し手数 = 0 ) return 必至;
    }
    指し手の順序付けをし並び替える
    for ( まだ調べていない指し手すべて ){
        if ( 詰み判定 = 詰み ) continue;
        局面を進める
        if ( depth = 末端 ){
            if ( 無駄捨て指し手 ) /* 2 手延長なので -1 */
                r = attack( depth - 1 );
            else r = 不明;
        }
        else r = attack( depth + 1 );
        局面を戻す
        if ( r != 必至 ) break;
    }
    return r;
}
```

図 3 TACO-H の基本アルゴリズム 2

表 2 深さを閾値とした縦型反復深化実行結果

No.	作意手数	実際手数	時間 (秒)	局面更新数	有岡 (秒)
69	7	7	5.2	8.0	0.3
70	7	7	3.5	5.5	0.7
71	7	7	20.9	36.3	1.7
72	7	7	2.7	4.1	0.5
73	7	7	1.6	2.2	0.5
74	7	7	635.9	1047.9	7.7
75	7	7	6.4	10.0	2.1
76	7	7	107.6	151.9	12.8
77	7	9	274.7	397.6	4.8
78	7	9	10.6	18.4	0.5
79	7	9	1620.5	2240.5	125.0
80	7	13	352.7	475.9	28.8
81	7	7	0.6	0.9	0.2
82	7	7	17.1	23.6	3.1
83	9	9	1.3	1.6	0.2
84	9	9	10.6	15.4	0.7
85	9	9	30.5	56.4	6.0
86	9	11	33.2	50.4	6.2
87	9	9	19.0	33.0	5.9
88	9	9	73.5	116.3	0.8
89	9	9	139.9	176.6	4.3
90	9	11	29.3	48.2	0.7
91	9	11	42.8	59.9	2.0
92	9	7	2.5	3.0	0.3
93	9	13	4205.2	6488.5	20.9
94	11	11	200.9	273.5	4.5
95	11	13	213.7	295.4	2.7
96	11	13	681.2	920.2	10.8
97	11	11	192.3	248.1	71.2
98	11	11	115.7	152.6	5.7
99	15	15	35.0	49.6	28.3
100	15	17	862.4	1197.4	300.0
内藤	17	17	685.9	954.6	

*局面更新数の単位は 10 万

は，受け方の応手の数を証明数として用い，1500 手を超える詰め将棋問題を解くなど，すばらしい成果が報告されている．必至探索も AND/OR 木探索であるから，証明数として受け方の応手の数を用いれば詰め将棋と同じ手法が可能である．しかし，必至探索では合法手の生成のために，たびたび詰みルーチンをコールするので，受け方の応手の数を証明数とする方法では膨大なコストを要する．そのため，詰め将棋で開発された手法をそのまま使うことは出来ず，新しい工夫が必要になる．本稿では，必至探索に玉の安全度という局面の静的な評価を閾値として用いる新たな探索手法である安全度優先必至探索 (Safety-Priority Hisshi search or SPH) を提案する．

6·2 有 岡 の 手 法

有岡 [有岡 00] は，受け方の応手の数を正確にカウントする代わりに擬似的な方法を使っている．受け方が必至にならないとき，その深さでの証明数への寄与を「まだ展開していない手の数」として評価値に反映させ，その証明数を閾値として PN*を適応するという方法を用いている．この手法は有岡が指摘しているように,「まだ展開していない手の数」に自玉が詰まされる手が含まれるので，正確な証明数が得られない可能性が生じる．

6·3 SPH の概念と工夫

玉の安全度とは，玉の移動できるマスの数を意味し，TACO-H ではこれに盤の端近くでの補正と，玉の周りのマスの利きの制圧度による補正を若干加味して計算される．一般に，玉の安全度の低い方が詰みやすいので，玉の安全度は受け方の応手の数と似た性質を持つ．しかし，玉の安全度は静的に評価できるため，受け方の応手の数に比べて計算コストが圧倒的に少なくてすむ利点がある．必至探索では，受け方の玉を詰めろと王手で攻めるが，玉の安全度が高い局面はそもそも詰めろにならないことが多く，必至をかけるためには受け方の指し手の後も玉の安全度の低い局面が続くのが一般的である．

そこで，SPH では受け方節点での玉の安全度を閾値とし，玉の安全度が低くなる手を優先して探索する．玉の安全度を閾値として反復深化するので，玉の安全度が高い局面は探索を後回しにされ，長手数の必至問題でも効率よく探索が行える．

ただし，玉の安全度が高くなる受け方の指し手が詰めろを防いでいるとは限らない，すなわち，合法手であるとは限らないため，ただ単純に玉の安全度が高くなる手だけを探索していては必至解を見つけることにはならない．そこで工夫が必要となるが，SPH では受け方節点で閾値にかかわらず合法手がみつかるまで応手を調べることでこの問題を解決している．つまり合法手の安全度を見て探索を打ち切るかさらに深く探索するかを決めるため，受け方節点で合法手がない場合にはすべての応手を調べるので，必至であると判断できる仕組みになる．

6·4 SPH の手順

SPH は次の手順で探索を行う．

- まず適当に安全度の閾値を決める．
- 攻め方の節点は反復深化の場合と同じ．
- 受け方の節点で以下のように探索を行う．
  - 受け方の応手をすべて生成し，玉の安全度が高くなる順にソートする．
  - ソートした上位の手から詰み探索をし，詰めろを防いでいる手すなわち合法手かどうか調べる．
  - 合法手が見つかったら，
    (1) 玉の安全度が閾値を超えている場合
    その手をさらに深く探索していき，値が返ってきたらその節点の他の手も調べていく．
    (2) 玉の安全度が閾値以下の場合
    その節点の探索を打ち切る．
  - 必至が見つからなかった場合，閾値をあげて再探索する．

SPH は玉の安全度を閾値として反復深化を行うが，受け方節点では閾値にかかわらず合法手がみつかるまで応手を調べるので，安全度が閾値を超えていながら合法手でないという手が存在しても，必至かどうかきちんと見極めることができる．例えば，閾値が 2 の場合，受け方が合法手を見つけ，その手を指した後の局面が玉の安全度 2 以下の場合，すなわち玉の動けるマスが 2 以下の場合はどんどん深く探索していき，安全度が 3 以上の場合は探索を後回しにする．

6·5 アルゴリズム

SPH のアルゴリズムを C 風のコードで示す．search() では末端深さの代わりに安全度閾値を反復深化している．attack() は図 3 から必死探索深さ 1 の軽い探索を外すだけなので省略する．defence() の変更箇所は末端深さで探索をやめる部分を玉の安全度が安全度閾値を超えているか，または深さが最大探索深さを超えているか，に替えた点と，無駄捨て延長を外した点だけである．

6·6 実　　験

SPH を実装した TACO-H で，深さを閾値とした反復深化と同じ方法で実験を行った．なお，第 4 節で述べた「取られるだけの無駄な手数伸ばしへの対応」と「1 手の先読み」の手法はここでは用いていない．玉の安全度の閾値は，初期値 2 から反復深化させている．また，探索深さに上限を設け，問題番号 69 から 98 までは 11，それ以外は 17 としたが，11 で解けなかった問題 80 と問題 95 は 17 にして解き直した．

6·7 結　　果

実験結果を表 3 に示す．「速度比較」は表 1 の結果と比べて速度が何倍速くなったかを示す．また参考のため有岡の「証明数をしきい値とした多重反復深化」で詰手数

```
int search(){
    for ( 安全度閾値 = 2;; 安全度閾値++ ){
        int r = attack(0);
        if ( r = 必至 または 不必至 ) return r;
    }
}
int attack( depth ) は省略
int defence( depth ) {
    最後の部分以外は図 3 と同じ
    for ( まだ調べていない指し手すべて ){
        if ( 詰み判定 = 詰み ) continue;
        指し手を一手進める
        if ( 玉の安全度 > 安全度閾値
        または depth = 最大探索深さ )
            r = 不明;
        else r = attack( depth + 1 );
        局面を戻す
        if ( r != 必至 ) break;
    }
    return r;
}
```

図 4 SPH のアルゴリズム

表 3 SPH 実行結果

No.	作意手数	安全閾値	時間（秒）	局面更新数	速度比較	有岡（秒）
69	7	2	0.3	48	19.2	3.1
70	7	4	24.3	3896	0.1	0.72
71	7	2	13.9	2101	1.5	34.1
72	7	4	27.0	3890	0.1	2.5
73	7	4	6.8	1017	0.2	0.32
74	7	3	518	81920	1.2	43.9
75	7	3	5.1	821	1.2	11.3
76	7	3	49.7	7121	2.2	1407
77	7	2	26.7	3927	10.3	6.7
78	7	3	6.6	1114	1.6	0.63
79	7	3	264	35472	6.1	X
80	7	3	394	53945	0.9	X
81	7	3	2.7	385	0.2	0.17
82	7	2	22.7	3134	0.8	304
83	9	2	1.6	226	0.8	1.3
84	9	2	0.1	14	81.5	0.82
85	9	3	46.6	7749	0.7	8.6
86	9	2	12.4	1711	2.7	29.6
87	9	4	194	30056	8.4	236
88	9	3	57.5	8686	1.3	0.7
89	9	4	336	43397	0.4	667
90	9	2	1.9	333	15.4	0.42
91	9	2	34.6	4880	1.2	8.4
92	9	3	14.9	1698	0.2	0.45
93	9	X	X	X	X	410
94	11	3	0.8	115	241	2
95	11	2	433	56622	0.5	9.9
96	11	X	X	X	X	3.1
97	11	3	54.8	7074	3.5	2.9
98	11	2	7.1	881	16.3	2
99	15	2	0.2	21	194	7
100	15	3	542	73304	1.6	4.4
内藤	17	2	225	29083	3.0	

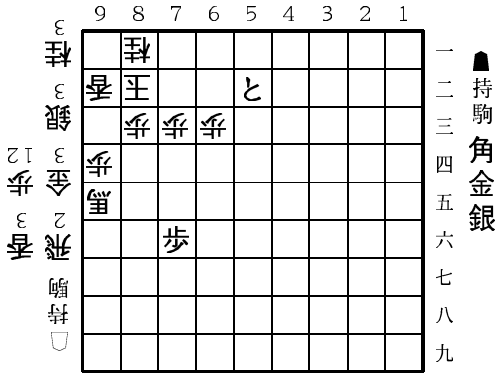

図 5 問題 99

☗７一銀△同玉☗６二金△８二玉☗７一角△９一玉
☗７二金△９三銀☗同角成△同香☗８二銀△９二玉
☗８一金△８四歩☗７五桂 まで 15 手必至

パラメタを攻め方 5 手，受け方 11 手，ルートで与える証明数のしきい値を 50，探索深さの上限を 17 手として解いた結果 [有岡 00] を一番右のカラムに示す．

解を保証しない有岡の方法より早く解いた問題も多数ある．問題 93，96 は局面表がいっぱいになり解けなかった．かえって遅くなったものもいくつかあるが，探索深さの上限を実際に解ける深さよりも深くしているためであると思われる．より速く解けた問題が多く，問題によっては非常に高速に解けたものも多数ある．99 番（図 5）は 15 手必至であるが，図に示したように比較的わかりやすい一直線の手順であるため，0.2 秒という驚くべき速さで解いた．また，内藤問題では 225 秒 (3.75 分) という好結果が得られた．

特に長手数の問題では速く解けており，長手数の必至問題を解くための有効な探索法と成り得ることが示せた．探索深さの上限をうまく設定すれば，短手数の問題でもかなり有効ではないかと思われる．

7. ま と め

本稿では，必至探索としての AND/OR 木探索とその効率化のための工夫を提案した．これらの工夫を必至探索プログラム TACO-H に実装し，難問を集めたテスト問題を用いて評価実験を行った結果，従来よりかなり短時間で必至問題が解けるようになった．解の保証を重視しながら行う我々の方法でも，これまで難しいとされてきた必至問題を十分に実用的な時間で解くことに成功した．

また，玉の安全度という静的な評価値を閾値として，長手数の問題にもうまく対応できる安全度優先必至探索 (SPH) を提案した．SPH は最良優先探索のようにふるまう深さ優先探索のアルゴリズムで，詰将棋の分野で考案された PN*と似たふるまいをするが，探索の方向付けのために証明数を用いる代わりに，その近似をより高速に与える玉の安全度の評価関数を用いている．テスト問題を解く評価実験によってその有効性を確認した．長手数の必至問題を効率良く解く有効な探索法と言える．

主要な結果として，難問の内藤 17 手必至を，深さを閾値とした反復深化で 11.4 分，SPH で 3.75 分で解いた．また，SPH では 15 手必至問題をわずか 0.2 秒で解くなど，SPH が長手数問題を解くための非常に有効な探索法であることを示した．

今後は，TACO-H を更に改良して短手数問題も長手数問題もより高速に解けるようにしたい．目標として，最大 37 手の必至問題を含む必至問題集 [来条 84] を高速に解けるようにし，そして，実戦形式の寄せ合い問題 (終盤の次の 1 手) を解くルーチンへと発展してゆきたい．さらに，実戦用の将棋プログラム TACOS にこれらを実装し，強い終盤ルーチンを開発する予定である．

◇ 参 考 文 献 ◇

[有岡 00] 有岡雅章：必至探索，on the Internet Web page http://plaza9.mbn.or.jp/~kfend/inside_kfend/hissi.html (2000).
[橋本 00] 橋本剛：必死探索について，コンピュータ将棋協会誌，Vol. 13, p. 44 (2000).
[飯田 98] 飯田弘之：必至問題を解くプログラム，松原仁（編），コンピュータ将棋の進歩 2，第 4 章，pp. 47–60，共立出版 (1998).
[伊藤 95] 伊藤，河野，脊尾，野下：詰将棋を解くプログラムの進歩，人工知能学会誌，Vol. 10, No. 6, pp. 853–859 (1995).
[金子 96] 金子タカシ：詰みより必死—終盤の超発想法，毎日コミュニケーションズ (1996).
[来条 84] 来条克由：来条克由必至名作集 (1984).
[内藤 99] 内藤國雄：新 妙手探し (26)，近代将棋，No. 8, pp. 26–27 (1999).
[脊尾 98] 脊尾昌宏：共謀数を用いた詰将棋の解法，松原仁（編），コンピュータ将棋の進歩 2，第 1 章，pp. 1–21，共立出版 (1998).
[Seo 01] Seo, M., Iida, H., and Uiterwijk, J. W. H. M.: The PN*-search algorithm: Application to tsume-shogi, *Artificial Intelligence*, Vol. 129, No. 1-2, pp. 253–277 (2001).
[棚瀬 00] 棚瀬寧：IS 将棋のアルゴリズム，松原仁（編），コンピュータ将棋の進歩 3，第 1 章，pp. 1–14，共立出版 (2000).
[Thomsen 00] Thomsen, T.: Lambda-Search in Game Trees - with Application to Go, *ICGA Journal*, Vol. 23, No. 4, pp. 203–217 (2000).
[山下 98] 山下宏：YSS - そのデータ構造，およびアルゴリズムについて，松原仁（編），コンピュータ将棋の進歩 2，第 6 章，pp. 112–142，共立出版 (1998).
[吉川 87] 吉川竹四郎：将棋プログラムの研究，*programmers'* Σ, No. 03, pp. 102–115 (1987).

〔担当委員：石塚 満〕

2001 年 4 月 16 日 受理

著 者 紹 介

橋本 剛

1994 年京都大学農学部卒業．1996 年東京大学大学院理学系研究科在学中に中国雲南民族学院へ留学．1997 年東京大学を中途退学し，台湾文化大学に留学．現在，静岡大学大学院後期課程在学中．

作田　　誠(学生会員)

2001 年静岡大学大学院理工学研究科後期課程修了．博士(工学)．現在，同大学院研究生．ゲーム・パズルを題材とした探索・推論・学習などに興味を持つ．

飯田　弘之(正会員)

1962 年生．将棋プロ棋士六段．1994 年東京農工大学大学院博士後期課程修了．博士 (工学)．科学技術振興事業団博士研究員，オランダマーストリヒト大学客員教授など．現在，静岡大学情報学部助教授．ゲーム情報学，エンターテイメントコンピューティング等に興味を持つ．