完全に解を保証する必死探索について

橋本　剛[1], 作田　誠[1], 飯田弘之[2]
[1] 静岡大学理工学研究科　[2] 静岡大学情報学部
E-mail: {hasimoto,iida}@cs.inf.shizuoka.ac.jp

概要

コンピュータ将棋にとって終盤の重大なテーマである必至探索は近年目覚しい進歩を遂げているが, 内部詰みルーチンに単純な深さを閾値とする反復深化法を用いているため, 長手数の詰めろが必要な必死問題が解けない, 必至解を完全に証明できない, という二つの大きな問題を抱えている. これらの問題を解決するため, 詰みルーチンにハッシュ表を利用してループ対策を施したPDSを使って必至探索を行った. その結果, 必死探索でPDSを使うには制限時間を設けざるを得ないため必死解を厳密に保証することはできないが, 適当な制限時間を設ける事で詰み探索に最大深さ5手の反復深化法を使う場合と同等以上の結果を得た.

Brinkmate search with guarantee of solution

Hashimoto Tsuyoshi[1], Makoto Sakuta[1], Hiroyuki Iida[2]
Departments of Computer Science
Shizuoka University
3-5-1 Johoku Hamamatsu, 432-8011 Japan
E-mail: {hasimoto,iida}@cs.inf.shizuoka.ac.jp

abstract

A quite important subject of Shogi endgame for computer Shogi, Brinkmate search, has been remarkably improved recently. However, because they use simply iterative deepning for inner mate search routine, they have two important problems. one is that they can never solve brinkmate problems which need thredmate with long steps, and the other is that they can not perfectly guarantee their brinkmate solutions. To solve these problems, we use PDS which is taked counter by hash tables as inner mate search for brinkmate search. The result is, it can not perfectly guarantee their solutions because PDS can not avoid a time limit in brinkmate search, but by a proper time limit we get results of the case with PDS which is equal to or more than the case with iterative deepning within 5 steps.

1 はじめに

将棋の必至問題をコンピュータに解かせることは詰めろ判定が高コストで難しいとされてきた. ところが近年実戦風の必至ルーチンで有岡 [2] が好結果を上げたほか, 筆者らの研究 [1] では受け側の全合法手を確認して解を保証しながら解く方法で, SPH という新しい手法を導入し 15 手必至問題を 0.2 秒で解くなど優れた結果を残した. どちらも 15 手までの必至問題を含み一般的な必至問題集としては最も難しいと思われる金子の問題を全て解いており, ほとんどの必至問題は解けると考えられる. 必至探索はそれ自体が AND/OR 木であるが, 内部で同じく AND/OR 木である詰みルーチンを何度も呼ぶいわば AND/OR 木の 2 重構造になっているのが特徴で, その外枠とでもいうべき必至探索自体には SPH 等が考案されている. だが, これまでの手法では内部で詰めろ判定に使う詰みルーチンに単純な深さを閾値とする反復深化法（以下 ID と呼ぶ）を用いており, 必至探索の中で長手数の詰めろが出てくる場合には対応できない. そのため二つの大きな問題が生じる. 一つは長手数の詰めろが必要な必至問題が解けないことであり, 寄せ合いの問題を解かせる場合や実戦で使う場合などにも大きな壁となる. 二つ目は詰めろでない事を完全に証明するのが難しいため必至解を厳密には保証できないことである. これらの問題を解決するため, 本稿では必至探索の内部詰みルーチンに証明数と反証命数を閾値として探索を行う PDS を使い, その結果と問題点を述べる.

2 PDS

長手数の詰め将棋を解くアルゴリズムには証明数を使う方法が一般的で, 脊尾の PN*（あるいは C*）[3] や長井の PDS[4] が有名である. 数十手の比較的長手数の詰み探索では PDS が最も効率が良いことが作田 [5] によって示されており, 最近では強豪将棋プログラムの終盤探索で用いられている. PDS は反証命数も閾値として探索するため不詰みの証明に優れており, 詰めろでない事を完全に証明して必至解を厳密に保証するという目的に合致する探索方法であると考えられる.

3 必至の内部詰みルーチンへの PDS の実装と問題点

必至プログラム TACO-H の内部詰みルーチンに PDS を実装した. TACO-H の詳細については [1] で述べている. （付録参照）. PDS は詰みあるいは不詰みの解を得るまで探索を行うので, その見極めが難しい局面やループ手順の生じる局面では解を得るまでにかなり時間がかかってしまう. 必至探索では詰めろの有無が容易に判定できる局面から見極めが難しい局面までさまざまな局面で何度も詰み探索を呼ばねばならず, ループ手順の生じる局面が何度も現れるためその対策が不可欠となる. また一般的に不詰の証明は詰みに比べて時間がかかるため, 時間に制限を設けないと詰み探索に最大探索深さの浅い ID を用いる場合に比べて PDS を用いる場合では時間がかかることは避けられない.

4 ループ対策

深さを閾値とした反復深化探索ではあまり問題にならないが, PDS のように一部のノードをどんどん深く読んでいく探索では同一局面が何度も現れループに陥ってしまうことがあり, 時間を浪費するだけでなく解を導けないなど深刻で厄介な問題になる. 特に必至問題では何千, 何万回も詰みルーチンを呼ぶため, ループが深刻な局面に当たってしまい, 解を得られないことも多い. そのため必至探索に PDS を用いるにあたってはループ対策が不可欠であるが, ここでは以下のような対策を施している.

4.1 ループ対策基本概念

将棋のルールでは連続王手の千日手は王手を掛ける側の負けであり, 詰め将棋では必ず連続王手なので同一局面が現れることは王手側の負けに等しい. そこで, 同一局面が現れた場合は不詰として扱えば解決しそうである. 王手側, すなわち OR ノードでは同一局面となる指し手を不詰と扱っても, 他の指し手も調べるので他に詰む指し手があれば詰みを返すし, 他の手がすべて不詰であれば同一局面になる手しか残らない事になりやはり王手をしている OR ノード側の負けになるので, 問題は無い. 具体的な例を挙げるため図 1 を見て頂きたい. ☗1一竜 ☖3二玉 ☗1二竜 ☖3一玉で同一局面になるので, ☗1一竜は不詰になる手として扱う. 次に他の指し手を探索していき, ☗3二金の詰みを発見するので, この局面が詰みであることはもし最初に ☗1一竜を探索しても理解できる. だが図 1 で持ち駒の金がない局面を考えると, 同様に ☗1一竜を不詰みとした後他に王手は竜を捨てる手しかなく, すべて不詰みになりこの局面は不詰の局面という事になるが, 実際にその局面は詰まないので問題は無い. ところが, 受け側, すなわち AND ノードでは一つ不詰となる指し手が見つかればそこで探索を打ち切ってしまうので, 同一局面となる指し手があっても不詰とすることはできない. そこで OR ノードで同一局面となる指し手が見つかった時だけ不詰として扱うようにする. ループが現れないようにするにはこれで十分である. また, 同一局面だけでなく, 盤面が同じで持ち駒の優越関係 [3] が明らかに攻め側にとって不利な場合も同様に OR ノードの指し手だけで不詰と扱っている.

4.2 局面表の利用

探索手順の中に同一局面が存在するか否かを調べる最も単純な方法は親局面と一つ一つ照合する方法だが, その方法は探索が深くなるほどコストがかかり時間も掛かってしまう. 我々は局面表をうまく利用する事でこの問題を解決している. まず同一局面や盤面が同じで持ち駒の優越関係が明らかな局面を調べるため, 局面を探索する前に探索中の局面であるという情報を局面表に登録する. その局面の探索が終われば必ず探索中でないという情報を局面表に再登録する. これで局面表にはルート局面から現在探索中の局面までが探索中であるという情報が入り, 局面表を参照するだけで探索中の同一局面などの有無がわかる.

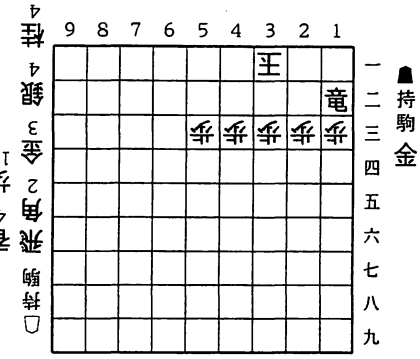

図 1: ループ例
☗1一竜 ☖3二玉 ☗1二竜 ☖3一玉で同一局面になる

5 制限時間

5.1 制限時間の必要性

必至探索で完全に解を保証するためには, 詰みあるいは不詰みであることを証明するまで内部詰みルーチンを走らせなければいけない. だが詰みルーチンを PDS にして制限時間なしで必至探索を行うと 3 手必至ぐらいまでならすぐに解けることが多いが, 7 手必至以上の問題を解かせると [1] では数秒で解を得た問題でも長時間なかなか解を返さない. 7 手以上かかる長手数の必至問題を解くためには詰みルーチンを数千回以上呼ばないといけないが, 中には容易に不詰を証明できない局面がいくつも現れ, 極端に時間を浪費してしまうためである. 不詰を証明するためには全ノードを王手が一つもかからない

局面まで展開しなければならない. そのため, 攻め側の駒が豊富な局面や, 王が広くいつまでも追い掛け回してしまう局面などでは相当時間が掛かってしまうのである. 以下実験によってこの事を検証する.

5.2 実験

ここで, 内部詰みルーチンに PDS を使い,PDS の制限時間を 1 ミリ秒と 1 秒にして金子タカシ「詰みより必至」[6] の第 69 問 (図 2) と第 75 問 (図 3) を TACO-H に解かせた結果を表 1 と表 2 に挙げる. なお, 実験に使用したマシンの CPU は ATHLON1.2GHz, メモリは 512Mbyte, OS は WINDOWS2000 である. 詰みルーチンを PDS にした以外はプログラムの実装の条件は [1] と同じである. これを見ると, 制限時間を変えても詰み探索ルーチンを呼び出した回数はそれほど変わらないが, 必至解を得るまでの時間, 局面更新数がともに PDS の制限時間 1 秒の方が制限時間 1 ミリ秒の場合の 20 倍近くかかっていることがわかる. 共に解は正確に出しているので, この違いはほとんど不詰の証明にかかるコストに起因していると考えられる. また, 問 69 では詰み探索の 90%近く, 問 75 では詰み探索の 70%が 1 ミリ秒以内に解を得ており, 必至解に必要な詰み探索の結果は多くとも 1 ミリ秒以内にすべて得られるにもかかわらず, 残りの重要性の低い不詰の証明のために高コストがかかっていると考えられ, 問 69 では全体の 0.6%, 問 75 では全体の 7.2%の局面に至っては制限時間千倍の 1 秒でも解が得られていない. 不詰みの証明に時間のかかる局面の例として, 第 69 問の変化手順に現れて不詰みの証明に 1 秒以上かかった局面 (図 4) を挙げる. この局面では☗ 1 三銀成 ☖ 同玉以下角を離して打つ変化や, 1 一の香車を取って打つ変化などがあり不詰みの証明に時間が掛かっている. このように, 必至探索において内部詰みルーチンに PDS を使う場合現実的には制限時間を設けざるを得ない.

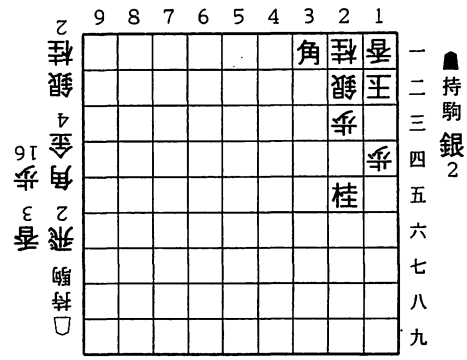

図 2: 金子タカシ「詰みより必至」第 69 問
☗ 1 三銀打 ☖ 同銀☗同桂成 ☖ 同桂
☗ 2 二銀打 ☖ 2 一桂打☗ 3 二銀打
まで 7 手必至

表 1: 詰み探索制限時間を変えた第 69 問の結果

詰み探索制限時間	1 ミリ秒	1 秒
時間 (ミリ秒)	677	22818
局面更新数	174257	7487680
詰みを読んだ回数	2260	2621
詰み探索時間切れ	305	18

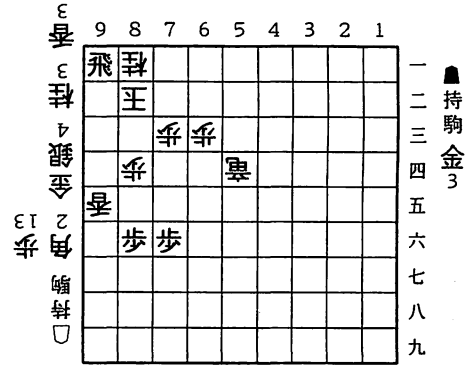

図 3: 金子タカシ「詰みより必至」第 75 問
☗ 9 三金打 ☖ 同桂☗ 8 一金打 ☖ 8 三玉
☗ 9 二飛成 ☖ 9 四玉☗ 8 三金打
まで 7 手必至

表 2: 詰み探索制限時間を変えた第 75 問の結果

詰み探索制限時間	1 ミリ秒	1 秒
時間 (ミリ秒)	12213	818709
局面更新数	174257	236970914
詰みを読んだ回数	8549	8918
詰み探索時間切れ	2473	649

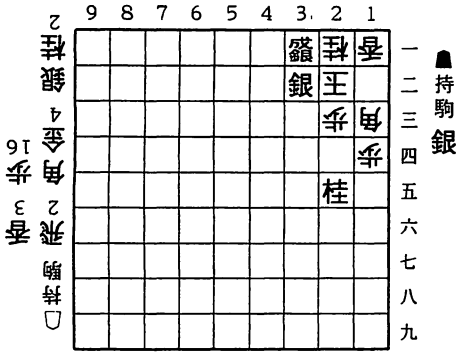

図 4: 不詰みの証明に 1 秒以上かかる局面の例

6 詰みルーチンに ID を使う場合との比較

TACO-H の内部詰みルーチンに制限時間 1 ミリ秒で PDS を使った場合と詰み探索深さの最大値を 5 にした ID を使った場合とで比較実験を行った. 結果を表 3 に示す. 前章同様, 実験に使用したマシンの CPU は ATHLON1.2GHz, メモリは 512Mbyte,OS は WINDOWS2000 である. [1] と同じく金子の必至問題集 [6] のうち 7 手必至以上の問題と内藤の 17 手必至問題 [7] を解かせてみた. 必至探索は深さ優先の反復深化である. 表中の数字は各詰みルーチンを使った場合の解答に要した時間で, 単位は秒である. 時間比とは PDS を使った場合の解答時間と ID を使った場合の解答時間との比を示している.

制限時間 1 ミリ秒の PDS の方が早く解ける事が多いが, 逆に遅くなっている場合もあり, 一概には言えないが, 制限 1 秒の PDS を使えば最大深さ 5 の ID を使った場合と同等か若干良い程度のパフォーマンスが得られていると言える. PDS の方が早く

表 3: 内部詰みルーチンの比較実験

No.	PDS	ID	時間比
69	1.0	3.5	0.3
70	5.8	2.4	2.5
71	17.4	13.9	1.3
72	14.7	1.8	8.2
73	8.8	1.0	8.5
74	52.8	423.9	0.1
75	12.2	4.3	2.9
76	79.9	71.7	1.1
77	83.6	183.1	0.5
78	2.8	7.1	0.4
79	675.3	1080.3	0.6
80	94.0	235.1	0.4
81	2.2	0.4	5.5
82	13.6	11.4	1.2
83	0.5	0.8	0.6
84	3.1	7.1	0.4
85	14.0	20.3	0.7
86	5.1	22.1	0.2
87	3.1	12.7	0.2
88	37.9	49.0	0.8
89	26.5	93.2	0.3
90	15.3	19.5	0.8
91	43.4	28.5	1.5
92	8.9	1.7	5.3
93	947.3	2803.5	0.3
94	25.0	133.9	0.2
95	33.2	142.4	0.2
96	498.5	454.1	1.1
97	67.3	128.2	0.5
98	83.1	77.1	1.1
99	43.2	23.3	1.9
100	474.6	574.9	0.8
naito	378.5	457.3	0.8

単位は秒

なる理由としては以下の理由が挙げられる. ID では変化手順の中に 5 手詰以上の詰みがある場合詰みを認識できないが, PDS ではより長手順の詰みを認識するため, 必至にならない変化で 5 手詰みの ID では不明として扱っていた枝に早く結論を出すことで, 早く解を得るのである. また, 正解手順の中にも 5 手詰み以上の詰みが必要な場合もあるが, 必至探索自身が王手を考慮することで結果として ID でも 5 手以上の詰みにも対応している. このような場合も,PDS ではより長手順の詰みを即座に認識するため, 上記の場合に比べて必至探索のノードを減らすことになり, やはり早く解を得られる事になる. PDS の方が遅くなる原因は, 前章で述べたように不詰の証明に時間がかかってしまうことが大きな原因として挙げられる.

7 まとめ

必至の内部詰みルーチンに ID を使うと二つ問題がある. 一つは長手数の詰めろが必要な必至問題が解けないことであり, 二つ目は必至解を完全に証明するのが難しいため必至解を厳密には保証できないことである. これらの問題を解決するため詰みルーチンに局面表を利用してループ対策を施した PDS を使って必至探索を行った. ID では最大深さ以上の探索は一切行わないのに対し,PDS では毎回詰みか不詰みを証明できるまで探索を行うため, 大量の局面で詰み探索を行わないといけない必至探索で PDS を使うには制限時間を設けなければならないという結果になり, 必至解を厳密に保証する必至ルーチンはできなかったが, 適当な制限時間を設ける事で最大 5 手の ID を使う場合と同等以上の結果を得た. [6] のような市販の必至問題を解くには ID で十分だが, 長手数の詰みが現れる必至にも対応できる本稿の手法は実戦風の問題を解く場合にもたいへん有効であると思われる.

8 今後の課題

PDS では不詰の証明が完全に王手がなくなるまで行われるため高コストになっているが, 人間は「この王は広い所に逃げてしまったので絶対捕まらない」「この変化は明らかに駒が足らない」などの判断を瞬時にすることでほとんどの場合完全に王手がなくなるまで考えると言うことをしない. 今後は不詰の証明にあくまで数学的厳密性を保ちながら人間のような高度な判断をさせることで, 不詰の証明の相当の高速化を図り, 厳密に解を保証する必至探索を完成させるとともに, 長手数の詰みに対応できるより高速で実戦的な必至ルーチンを完成させたい.

参考文献

[1] 橋本剛, 作田誠, 飯田弘之: 必至問題を解くプログラムとその評価, 人工知能学会論文誌, 16 巻,6 号 a,(2001)

[2] 有岡雅章: 必至探索, on the Internet Web page `http://plaza9.mbn.or.jp/~kfend/inside_kfend/hissi.html` (2000).

[3] Seo, M., Iida, H., and Uiterwijk, J. W. H. M.: The PN*-Search Algorithm, Application to Tsume-Shogi, *Artificial Intelligence*, Volume 129, Issue 1-2, pp. 253-277 (2001).

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

[5] 作田誠, 飯田弘之: 高性能な AND/OR 木探索アルゴリズムの詰め将棋問題による比較. 静岡大学 情報学研究, Vol. 5, pp. 15–22, 1999.

[6] 金子タカシ: 詰みより必死, 毎日コミュニケーションズ (1996).

[7] 内藤國雄:新 妙手探し (26), 近代将棋, No. 8, pp. 26–27 (1999).

付録: TACO-H の基本アルゴリズム

TACO-H の基本アルゴリズムを C 風のコードで示す.

```
/* 本体 */ 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;
    }
}

/* 攻め方 */
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 = 必至探索深さ 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;
}

/* 玉方 */
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;
}
```