詰将棋における変別チェックと余詰探索を同時に行える新しいアルゴリズムの提案

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

{nagai, imai}@is.s.u-tokyo.ac.jp

要旨

詰将棋の用語に，余詰と変化別詰 (変別) がある．詰将棋において，作者の作意手順中では，詰ますための攻め方の王手は原則として唯一でなくてはならない．作意以外の王手でも詰ますことが出来るなら，その手順を余詰と呼ぶ．詰将棋において，余詰の発見は重要である．余詰を発見するための探索を余詰探索という．また，余詰は作意手順中にあってはならないが，作意手順以外の枝葉の部分にはあっても良い．そのせいで，詰将棋の手順として，作意手順の途中から玉方の受けの違う手順を答えてしまうことがある．このような手順は変別と呼ばれ，間違いではないが不満が残る．そのため，普通に詰むことを調べた後，変別を除去するような処理をしなくてはならない．これを変別チェックと言う．既存のアルゴリズムとして，脊尾によるアルゴリズムがあるが，余詰探索と変別チェックを別々に行う上に，原理的に発見不可能な余詰の存在を否定できないなどの欠点がある．この論文では，余詰の定義に立ち返って，余詰探索と変別チェックを同時に行える，ナイーブなアルゴリズムを提案する．その結果，証明数探索に対する既存の余詰探索アルゴリズムでは原理的に発見できなかった類いの余詰も見つけるなどの成果を得た．

A New Algorithm to Remove Henbetu and to Find Yodume of Tsume-Shogi

Ayumu Nagai　　Hiroshi Imai

Department of Information Science, Faculty of Science, University of Tokyo

Abstract

There are Yodume and Henbetu in the terms of Tsume-Shogi. As with Tsume-Shogi, attacker's check must be the only move to mate the defender's king. If there is any other move, it is called Yodume, which makes the problem incomplete. It is significant to find Yodume on Tsume-Shogi problems. While there must be no Yodume in the intentional sequence, it is allowed to have a Yodume in the sequence except the intentional sequence. Due to this rule, an answerer may reply a winning sequence whose defensive move is different from the intentional sequence. This sequence is called Henbetu, which is not wrong, but not perfect. It is better to remove Henbetu. For such purpose, Seo's algorithm is known, but the algorithm to find Yodume and the algorithm to remove Henbetu must be carried out separately. We propose a naive algorithm to find Yodume and remove Henbetu at the same time. As a result, our program can find a Yodume that are impossible to find by already known algorithm to find Yodume used with Proof-Number Search.

1 はじめに

詰将棋を解く研究に関しては古くから行われており，最近10年ほどの間に驚くほど進歩した [1][2][3][4]．特に証明数や反証数 [5] という概念の登場以来の進歩には目を見張るものがある．証明数とは，直感的には，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非常に良い尺度となる．証明数が小さいほど，少ない労力で詰ませられる可能性が高くなる．また反証数とは，直感的には，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非常に良い尺度となる．反証数が小さいほど，少ない労力で不詰を示せる可能性が高くなる．より詳しい解説としては文献 [6] を参照されたい．このように，詰将棋を解くこと (あるいはより一般的にAND/OR木探索) に焦点を絞ったアルゴリズムの研究には多くの例がある．最近では解答能力の優れたものとしては，市販ソフトである「脊尾詰」をはじめ，証明数や反証数の概念を取り入れた探索法を用いている．AND/OR木探索の決定的な探索法として，長井はdf-pnアルゴリズム [7] を提案し，詰将棋への適用を行っている [8]．

詰将棋はただ単に詰むか不詰かを判定すればいいわけではなく，手順を答えなければならない．しかし作意手順通りに正しく答えるのは現実にはそう簡単ではない．詰将棋のルールとして，作者の作意手順中では，詰ますための攻め方の王手は原則として唯一に限定されていなくてはならない．作意以外の王手でも詰ますことが出来るなら，その手順を余詰と呼び，その作品は不完全作として扱われてしまう．従って余詰の発見は詰将棋においては重要である．余詰を発見するための探索を余詰探索という．余詰のうち，作意より短いものを特に早詰という．

余詰は作意手順中にあってはならないが，作意手順以外の枝葉の部分にはあっても良い [9]．詰将棋の手順を答えるとき，そのような，より手数の短い詰みを発見出来ないばかりに，作意手順の途中から，玉方の受け手の違う手順を答えてしまうことがある．このような手順は，変化別詰 (変別と略す) と呼ばれ，間違いではないが，不満の残るところである．(図1参照) 人間の場合は，詰将棋の作者の意図を汲み取って，自分の暫定手順を正すことができるが，コンピューターの場合はそうはいかない．詰めルーチンで普通に詰むことを調べた後，変別を除去するような処理をしなくてはならない．これを変別チェックと言う．

古典的な，深さをしきい値に用いた反復深化法を採用する限り，より手数の短い詰みを発見出来ない，というようなことは起こり得ないが，反復深化法には，長手数の詰将棋を解くことが困難という，致命的な限界がある．そこで最近は，証明数・反証数を用いた，pn-search [5] や df-pn のような証明数探索が注目されている．それらの探索法は，長手数の詰将棋でも非常によく解ける代わりに，より手数の短い詰みを発見出来ない，ということが起こりうる．従って，変別を答えてしまうことがあり，変別チェックのアルゴリズムが必要になってくる．しかし詰将棋を解くアルゴリズムに比べると，まだあまり研究されていない．脊尾による研究以外に我々はその例を知らない．

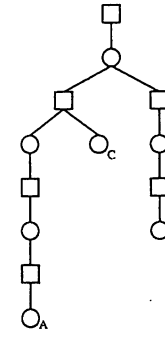

図 1: 作意手順は B であっても，C が詰むことに気付かないと，A という変別を答えてしまう

2 既存のアルゴリズム

2.1 脊尾の変別チェックアルゴリズム

変別チェックのアルゴリズムとして現在知られているものとしては，脊尾による変別チェックアルゴリズム [10] が唯一の例である．脊尾の変別チェックの手法をまとめると次のようになる．

1. まず通常の詰・不詰探索ルーチンによって，詰・不詰判定を行う．
2. 暫定手順中の局面の証明数を1にする．
3. 最大探索深さを (暫定手順の手数 −2) に設定する．ただし，探索手順中に合駒が入れば2手延長しないといけない．
4. その上で通常の詰・不詰探索ルーチンを呼ぶ．
5. その結果がもし不詰なら，暫定手順は変別ではなかったということが分かる．(厳密には100%そうとは言いきれない; 後述) もし詰むなら，「最長の受け手」「最短の王手」の原則に従って手順を構成し直せば，変別を排除した手順を得られる．こうして得られた新たな手順を再度暫定手順として 2. に戻る．もし十分に探索が済んでいれば，適当に設定したしきい値によって打ち切る．(具体的には「脊尾詰」で

は，ルートの証明数が 1. の時の値に達するか，展開局面数が一定の値に達したとき)

脊尾の変別チェックは，非常に単純でエレガントな方法である反面，現実にはいろいろな問題点が出てくる．最大探索深さを設定するので，探索が途中で打ち切られ，証明数が不当に大きくなる傾向が出てくる．現実にどの程度の影響があるかはともかく，可能性としては，いろいろな悪さをする原因になりうる．例えば図2のような木になっているとする．A を暫定手順とする．A は深さ7なので，変別チェックは最大探索深さを5に制限した探索となる．そのときに，F-D-C の手順で C にしきい値 n を割り当てた探索を行い，その後に F-E-C の手順で C にしきい値 n を割り当てて探索…ということを運悪く繰り返すと，F-E-C-B の手順は5手以下で，かつ非常に簡単な詰手順であるにも関わらず，発見することはできない．F-D-C で制限深さに達してしまい，実際よりも大きいのに，しきい値 n を C の証明数としてハッシュに書き込むことになるからである．また，もし変別チェックのあとに余詰探索などをするなら，ここでも問題が発生しうる．証明数が不当に大きくなっているために，簡単な余詰であったとしても，なかなか発見できないというような悪影響が出る場合がある．変別チェックに時間を掛ければ掛けるほど，証明数が不当に大きくなる傾向が強くなり，悪影響も大きくなる．

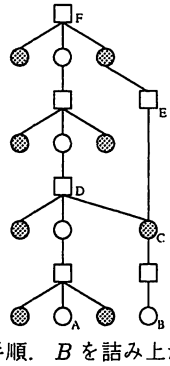

図 2: A が暫定手順．B を詰み上がりとする．常に F-D-C の手順を，F-E-C の手順より前に探索すると，F-E-C-B の詰手順を発見できない．

2.2 脊尾の余詰探索アルゴリズム

余詰探索についても，証明数探索に対するものに限定すると，脊尾によるアルゴリズム [10] が唯一である．脊尾の余詰探索をまとめると次のようになる．

1. まず通常の詰・不詰探索ルーチンによって詰手順を得る．これを本手順と呼ぶ．
2. 本手順の末端局面 (のハッシュ) は不詰に書き換え，本手順中の局面の証明数を 1 にする．
3. 通常の詰・不詰探索ルーチンを呼ぶ．
4. その結果がもし不詰なら，本手順以外の余詰はなかったことが分かる．(厳密には 100% そうとは言いきれない; 後述) もし詰むなら，その手順は本手順に対する余詰である．

これも非常にエレガントな方法なのだが，やはり現実にはいろいろな問題点を含む．諸悪の根元は，本手順の末端局面に「不詰」という，実際とは違う虚偽のデータを書き込むことに端を発する．本手順に余詰が存在するとして，その余詰手順 (あるいはその枝葉) に本手順の末端局面を含む場合，この局面は「不詰」に書き変わっているので，この余詰を発見することはできない．(図3参照) 更に「不詰」の局面に至る局面にも「不詰」が書き込まれるので被害は拡大する．本手順の末端局面を共有するような余詰，言い換えると，あとで結局本手順に合流するような余詰 (具体的には，手順前後や迂回手順や成・不成のどちらでもいい余詰) は，実際の詰将棋に多く見られるのでこの問題点は重大である．

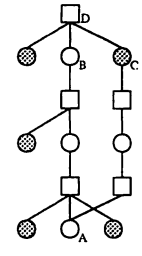

図 3: A が暫定手順．A に「不詰」を書き込むと，D-C-A の余詰を原理的に発見できない．

3 提案手法

我々は，余詰や変別の定義に立ち返って，変別チェックと余詰探索を同時に行えるアルゴリズムを提案する．基本的には，余詰や変別の定義に則った比較的ナイーブなアルゴリズムであるが，脊尾の変別チェックや余詰探索のアルゴリズムに伴う問題点を解消している．基本方針としては，

- 本手順に着目して，本手順から 1 手だけ外れた王手をかけた局面 (複数存在; 候補局面と呼ぶことにする) の中から 1 つの局面を選びだし，設定したしきい値内で詰むかどうか調べる．(これは通常の詰・不詰探索ルーチンを呼び出す)
- 詰まないなら，別の候補局面で調べる，というのを繰り返す．

- 発見した余詰が暫定手順より短いなら，「最長の受け手」「最短の王手」の原則に従うことで変別チェックできる．そうでなかったら，余詰を見つけたことになる．(図4参照)

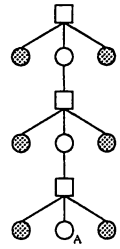

図 4: A が暫定手順．網掛けの付いた局面が候補局面．候補局面が詰むことが分かれば，変別を修正できるか，あるいは余詰を発見できる

さて，候補局面の選び方であるが，次の2通り提案する．

3.1 証明数ベース

まず1つ目は，候補局面のうち，証明数最小の局面を選択するというものである．これを「証明数ベース」と呼ぶことにする．具体的には，証明数のしきい値を2からスタートさせて，しきい値以下ならその候補局面を選択して探索する．(図5参照) 全候補局面の探索が終わったら，しきい値を3，4…と上げていく．これは大まかには脊尾の余詰チェックの方法と等価で，しかもハッシュに事実と異なるデータを書き込まない．「大まかには」と言ったのは，脊尾の方法だと深い方の局面から調べようとするが，この方法だとどちらから調べても良い．今回は浅い方から調べる．

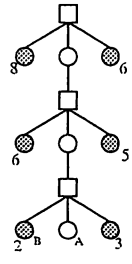

図 5: 候補局面のうち，証明数の最小な B を選ぶ

3.2 マージンベース

候補局面を選ぶ2つ目の方法について説明する．候補局面の親の局面は既に詰んでいるので，証明数は0のはずだが，0になる直前の証明数を記録しておく．そして，親の証明数 (もどき) と候補局面の証明数の差 (マージン) の最小の局面を選択する．(図6参照) これを「マージンベース」と呼ぶことにする．証明数の差を考える意味は，仮に今の暫定手順で詰まなかった場合，証明数の差の小さい候補局面は，詰・不詰探索ルーチンにおいて有力な探索対象となっていたはずだからである．具体的には，マージンのしきい値1からスタートさせて，浅い方の局面から眺めて，しきい値以下ならその局面を選択して探索する．全候補局面の探索が終わったら，しきい値を2，3…と上げていく．「証明数ベース」の方法だと，どうしても深い方の局面を一生懸命探索してしまうので，この方法を考えた．ただし，証明数のしきい値的には大きくなることがあるので，コストの大きな探索になる場合がある．この辺りにトレードオフがある．この方法も浅い方から調べても良いし，深い方から調べても良い．今回は浅い方から調べている．

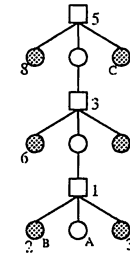

図 6: 候補局面のうち，証明数の差の最小な B や C を選ぶ

3.3 提案手法の利点と欠点

- 最大探索深さの制限は設けないので，証明数・反証数の値が不当に大きくなるようなことはない．その反面，変別チェックに関してはいくらか無駄な探索をしている．
- ハッシュに虚偽のデータを書き込むようなことはしないので，原理的に発見不可能な余詰が存在したりはしない．明確な余詰から非常に軽微なキズまで発見できる可能性がある．その反面，軽微なキズに埋もれて，重要な余詰に気づきにくくなるかも知れない．
- 候補局面の探索においては，あくまでも証明数探索を用い，非常に高い解答能力で詰・不詰の判断ができる．

4 詰・不詰判定と比較して，変別チェック・余詰探索の厄介な点

- 詰・不詰判定では詰むのか詰まないのかが重要であった．ハッシュに書き込む詰手数の情報はわりといい加減で良かった．しかし，変別・余詰チェックでは，詰む・詰まないは勿論，詰手

数の情報が非常に重要になってくる．(変別・余詰を詰手数なしには語れない) 従って，詰・不詰判定の時より厳密なルーチンを用意しないといけない．

- 優越関係の出番が減ってしまう．詰・不詰判定では，ある局面の先手の持駒が銀で詰むことが分かっているなら，同じ盤面で持駒が銀香でも当然詰む．変別チェック・余詰探索では，手数が重要なので，例えば持駒が銀の局面が11手で詰むことが分かっているとして，同じ盤面で持駒が銀香だったら，詰むことは詰むが，何手で詰むのかまでは分からない．少なくとも11手で詰むということしか分からない．それで十分な場合もあるし，9手以内で詰むのかが焦点になってくる場合もある．後者の場合は結局9手以内で詰むのか再度調べ直さないといけない．
- ループの絡む場合の余詰判定が難しくなる．図7のような木があるとする．詰・不詰探索ルーチンにより，*F-D-B-G-A* を本手順とする．*F-E-C-B* の探索のあと *F-D-B-G-C* のように探索すると，前者の手順の探索で *C* が詰むという情報がハッシュに書き込まれる．後者の手順の探索ではそれを参照して，後者の手順も詰む，即ち *G-C* は余詰と判断してしまう．我々の実装では余詰を表示する直前に，余詰がループになっていないかチェックして，ループなら余詰としては扱わない．
- ややこしいことに，戦前の詰将棋では，「妙手説」といって，玉方の手は最長手数の受けである必要はなく，妙手を含んでいる変化を作意として良かった．詰将棋のプログラムは現代のルールに基づいて作っているので，作意手順を返すのが不可能な場合がある．
- 探索打ち切り条件に達して，変別チェック・余詰探索を終了するとき，完全に強制的に打ちきることは出来ない．明らかにおかしい手順を返してしまう．このように，急には探索を止められないという問題点がある．我々の実装では，打ち切り条件に達すると，一部はしょり気味の探索にして，なるべくすぐに変別チェック・余詰探索を終われるようにしている．

5 実験結果

ここでは「将棋無双」のうち，不詰の問題を除いた94題に関して実験を行った．

比較対象は次の4つとする．

- 変別チェック全くなしの場合

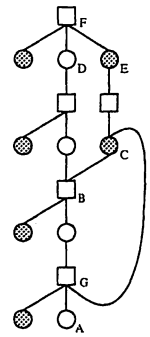

図 7: *F-D-B-G-A* を本手順とする．*F-E-C-B* の探索のあと *F-D-B-G-C* のように探索すると，後者の手順も詰む，即ち *G-C* は余詰と判断してしまう．

- 「証明数ベース」のみ
- 「マージンベース」のみ
- 「証明数ベース」と「マージンベース」を単純に組み合わせた手法．「証明数」をしきい値2で実行した後，「マージン」をしきい値1で実行し，そのあとは「証明数」を4回行うごとに「マージン」を1回行う．

変別チェック・余詰探索の打ち切り条件は，詰み探索で要した探索局面数の1/10もしくは，この数字が10000局面を下回る場合は，10000局面を上限としている．このように設定した理由は，現実には変別チェックや余詰探索にあまり時間をかけたくないという欲求があるためである．マシン環境は UltraSPARC II 400MHz，ハッシュには5.6MB使用．

5.1 変別チェックの結果

直接的な評価，すなわち作意手順と完全に一致するかどうかをもって評価したいところであるが，それはあまり意味を持たない．そのことを実例を挙げて示したい．図8は「将棋図巧」第18番(27手詰)である．我々のプログラムでこの問題を解かせ，変別チェックに掛けると，最終的に27手の手順を返す．しかし収束の部分で作意手順とは微妙に異なるのである．24手目まで進めたのが図9であるが，作意ではここから▲6三桂成▽6一玉▲5二角成 にて詰みだが，我々のプログラムは▲6三桂成▽5一玉▲5二角成 を答えてしまう．本質的には両手順の間に差はないと見るべきであるが，そのような判断を行うには人間の判断を必要とする．そこで我々は評価法として，手順そのものの比較は行わないことにした．

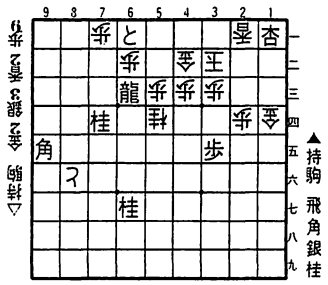

図 8: 「将棋無双」第 18 番 27 手詰

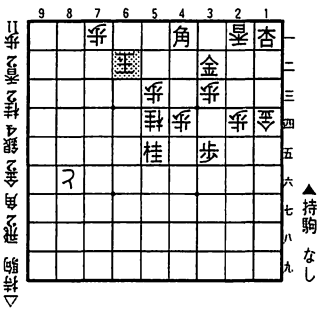

図 9: 「将棋無双」第 18 番 24 手目▽6 二同玉の局面. 作意はこのあと▲6 三桂成▽6 一玉▲5 二角成にて詰みだが, 我々のプログラムは▲6 三桂成▽5 一玉▲5 二角成 を答えてしまう.

そこで, 変別チェックの評価として, 間接的なものではあるが, 次の 4 つを考える.

- 作意手順の手数と同じ手数の手順を返した問題数がどのくらい多いか : 上の例のように, 作意とは微妙に異なってはいても, 作意を捕らえている可能性が高い.
- 手数がどのくらい短くなっているか : 変別チェックが働けば働くほど, 手数は一般に短くなる.
- 駒余りがどのくらい少ないか : 変別チェックにより作意手順に至れば, 駒余りはないはずである.
- 探索を打ち切ったつもりでも, 現在の実装では余分に探索してしまうが, この余分な探索がどのくらい少ない方か.

実際には, 手数は短くなっているのに, 駒余りになったりすることもあり, 単純ではない.

作意手順の手数と同じ手数の問題数については表 1のようになる. 3 種類の方法はすべて 50 題となった. 変別なしの場合, 27 題にしかならない. 提案法の導入により 2 倍近くまで増やすことができた.

変別なし	証明数	マージン	証明数 + マージン
27 題	50 題	50 題	50 題

表 1: 作意手順の手数と同じ手数の手順を返した問題数. 作意を捕らえている可能性が高い

短くなった手数については, 変別チェックを全くしなかった場合に比べて, どのくらい手数を短くできているか調べた. 最終結果だけを表にすると表 2のようになる. 3 種類の方法の間で殆んど差は見られない.

変別なし	証明数	マージン	証明数 + マージン
0	−2.43 手	−2.43 手	−2.45 手

表 2: 変別チェックにより短くなった手数. 一般に短くなっているほど変別チェックが効いている

駒余りについても, 最終結果のみ表にまとめると表 3のようになる. 3 種類の方法の間で差は全くない. (付録 A参照)

変別なし	証明数	マージン	証明数 + マージン
?	28 題	28 題	28 題

表 3: 駒余りとなった問題数. 一般に駒余りが少ないほど変別チェックがうまく機能している

実際の探索局面数について調べた. 詰・不詰探索に 100000 局面以上要した問題は 48 題あり, 変別チェックに費やした局面数と, 詰・不詰探索に費やした局面数との比の平均をとり, 表にすると表 4のようになる. 3 種類の方法の間に殆んど差はない. 詰・不詰探索ルーチンで 100000 局面未満しか探索していなくても, 今回の実験の探索打ち切り条件では少なくとも 10000 局面は探索させるので, そのような問題は平均から省いた. 当然ではあるが, 探索局面数の多い問題ほど, 探索局面数の比は 0.1 に近づいている. (付録 A参照)

以上の 3 つの評価から「証明数ベース」の方法も「マージンベース」の方法も性能的に殆んど差がないと言える. また, 提案手法を変別チェックとして見た場合, 表 1のように, 提案法の導入により, 作

変別なし	証明数	マージン	証明数＋マージン
0	0.1944	0.1941	0.1940

表 4: 本来，変別チェック・余詰め探索の打ち切り条件は，詰・不詰探索の 0.1 倍であるが，多くの場合これを上回って探索してしまう．上回り方の少ない方が制御しやすい

意手順の手数と同じ手数の問題数 (ほぼ作意を捕らえている問題数) を 2 倍近くまで増やすことができた．

5.2 余詰探索の結果

表 5は「将棋図巧」の余詰・キズの一覧である [11]．一番右の蘭は，変別チェック・余詰探索でこれらの余詰・キズを発見できたか否かを表す．(尚，第 40 番は「—」となっているが，これは不詰問題である．) 変別・余詰探索の打ち切り条件は，詰・不詰探索で要した探索局面数の 1/10 もしくは，この数字が 10000 局面を下回る場合は，10000 局面を上限とした場合である．条件内では発見できなかった余詰もあるが，7 割ほどは発見できている．注目すべきは，第 5 番 29 手目のような合流余詰である．合流余詰は，結局は作意手順に戻ってくるような余詰である．脊尾の余詰探索のようなアプローチを採る限り，このような余詰は原理的に発見できない．しかし我々のアルゴリズムでは発見できる．

6 まとめ

詰将棋を，非常に高い解答能力を持った証明数探索で解くと，変別手順を解答する可能性がある．脊尾の変別チェックや余詰探索アルゴリズムは個別に実行する上に，原理的に発見不可能な余詰が存在するなど，様々な問題がある．我々は変別チェックと余詰探索を同時に行うことのできる，新しい手法を提案した．余詰や変別の定義に則ったナイーブなアルゴリズムであるが，脊尾のアルゴリズムの持つこれらの問題を解消できる．提案手法は「証明数ベース」の方法と「マージンベース」の方法の 2 つあるが，実験結果から，どちらも性能の面では殆んど同じであることが分かった．

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．

番号	手数	場所	作意	余詰	発見
5 番	43 手詰	29 手目	3 三香打	3 二香打 (早詰)	○
8 番	31 手詰	5 手目	5 三馬	5 三角	×
14 番	83 手詰	79 手目	8 四金打	7 四龍 (キズ)	○
16 番	53 手詰	35 手目	3 三桂成	4 四桂	○
27 番	25 手詰	17 手目	7 四馬	6 四馬	○
31 番	31 手詰	3 手目	9 六香打	8 四角打 (早詰)	○
40 番	33 手詰	5 手目	5 四同銀成	5 六桂	—
41 番	47 手詰	37 手目	3 二角打	1 二角 (キズ)	○
57 番	29 手詰	3 手目	7 五銀打	9 四馬 (早詰)	○
62 番	23 手詰	1 手目	4 七金打	4 七銀打 (キズ)	×
		7 手目	5 六銀行	5 五馬	○
63 番	25 手詰	9 手目	6 四角	7 四金	○
64 番	19 手詰	11 手目	2 四角	2 五桂	×
71 番	25 手詰	7 手目	6 八馬	9 六飛 (キズ)	○
		21 手目	9 四金打	9 五金 (キズ)	○
76 番	31 手詰	19 手目	5 三香成	5 二香成	×
92 番	51 手詰	5 手目	6 八銀打	6 八角打 (キズ)	×
		39 手目	3 四龍	3 四銀	○

表 5: 「将棋無双」の余詰・キズ一覧

参考文献

[1] 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.

[2] 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.

[3] 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.

[4] M. 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.

[5] L.V. Allis, M. van der Meulen, and H.J. van den Herik. 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.

[6] L.V. Allis. *Searching for Solutions in Games and Artificial Intelligence*. PhD thesis, Department of Computer Science, University of Limburg, Netherlands, 1994.

[7] A. Nagai and H. Imai. 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*, pp. 163–170, 1999.

[8] 長井歩，今井浩．df-pn アルゴリズムの詰将棋解答プログラムへの応用．情報処理学会 アルゴリズム研究会 AL-75-2, 2000.

[9] 野下浩平．詰将棋を解くプログラム T 2，第 1 巻，3 コンピューター将棋の進歩，pp. 50–70. 共立出版，1996.

[10] 脊尾昌宏．メールによる対話，2000.

[11] 岡崎正博．提供資料，2001.

A 「将棋無双」の実験結果

「手数」の＊印は駒余りであることを表す.

番号	作意手数	詰・不詰ルーチン 手数	時間	局面数	変別チェック 手数	差	時間	局面数	比
1	11	57	324.65	9580436	41	16	32.58	1000060	0.104
2	47	51	5.90	230088	47	4	1.46	62198	0.270
3	39	39	5.85	218699	39	0	0.59	21921	0.100
4	25	43	2.72	108058	27	16	0.43	19037	0.176
5	43	41	18.83	703668	41	0	1.96	75180	0.107
6	43	-	—	—	-	-	—	—	—
7	15	29	5.56	197230	25*	4	1.01	39025	0.198
8	31	47	9.32	321963	37*	10	0.93	32307	0.100
9	29	31	0.46	18116	27	4	0.25	10536	0.582
10	19	27	8.07	291546	25*	2	0.58	29517	0.101
11	31	31	3.68	150546	31	0	0.39	16571	0.110
12	61	65	59.02	1907236	61	4	12.25	389651	0.204
13	41	55	2.92	113131	49*	6	0.26	13536	0.120
14	83	83	2.58	88319	83	0	0.52	20687	0.234
15	33	35	3.40	134116	33	2	0.78	30543	0.228
16	53	55	8.05	328549	53	2	0.74	32972	0.100
17	23	63	25.74	864375	59*	4	1.10	98291	0.114
18	27	31	1.05	41774	27	4	0.33	13228	0.317
19	29	35	8.80	328161	31*	4	0.86	33124	0.101
20	45	45	5.73	206931	45	0	0.62	23329	0.113
21	21	25	0.62	24116	25*	0	0.29	12029	0.499
22	33	33	0.91	32575	33	0	0.31	11701	0.359
23	25	25	1.42	53735	25	0	0.31	11747	0.219
24	31	31	0.46	17633	31	0	0.25	10117	0.574
25	17	19	3.20	111671	19*	0	8.35	288413	2.583
26	25	33	5.32	175411	33*	0	0.92	32980	0.188
27	25	37	5.69	202961	37*	0	0.70	31668	0.156
28	47	49	20.40	776434	49	0	2.85	99414	0.128
29	21	23	1.45	59072	23*	0	0.30	11442	0.194
30	119	117	22.26	774180	117	0	5.17	175708	0.227
31	107	51	25.45	901078	39*	12	2.52	98253	0.109
32	35	37	1.41	53387	35	2	0.24	10254	0.192
33	25	25	0.47	17207	25	0	0.40	14950	0.869
34	31	33	16.56	621521	31	2	2.30	90300	0.145
35	17	21	2.65	96883	21*	0	0.92	32676	0.337
36	37	39	0.83	34046	37	2	0.27	11107	0.326
37	47	-	—	—	-	-	—	—	—
38	13	15	0.26	11190	15	0	0.45	19019	1.700
39	25	33	6.12	214051	29	4	0.59	21994	0.103
40	33	-	—	—	-	-	—	—	—
41	47	47	2.26	89779	47	0	0.30	12912	0.144
42	43	51	2.19	82346	47	4	0.29	10877	0.132
43	47	55	10.23	373692	55*	0	1.57	58668	0.157
44	25	35	3.34	123822	29*	6	0.24	14292	0.115
45	47	51	26.81	915887	47	4	4.28	154944	0.169
46	27	41	35.23	1168424	29*	12	5.71	210191	0.180
47	13	19	2.67	95730	19*	0	0.68	25917	0.271
48	37	45	199.22	6458525	43*	2	33.70	1106088	0.171
49	27	27	0.12	4862	27	0	0.32	12671	2.606
50	17	35	2.08	74959	31	4	0.44	15540	0.207
51	33	39	96.28	3392270	33	6	12.58	429631	0.127
52	55	59	30.35	1070422	55	4	3.08	109165	0.102
53	37	51	4.41	164217	51*	0	0.49	19064	0.116
54	19	33	3.49	128071	19	14	0.38	15175	0.118
55	11	17	0.45	18304	13	4	0.26	11348	0.620
56	45	45	1.57	61471	45	0	0.47	18895	0.307
57	29	33	4.78	171622	31*	2	0.45	17507	0.102
58	31	35	2.55	96041	31	4	0.49	19173	0.200
59	45	49	3.87	152802	45	4	0.39	17793	0.116
60	45	45	0.59	22490	45	0	0.41	15118	0.672
61	39	39	3.16	118492	39	0	0.53	21540	0.182
62	23	23	2.04	76205	23*	0	0.37	13931	0.183
63	25	29	2.14	81222	27	2	0.41	17373	0.214
64	19	27	5.03	173664	25	2	0.41	17589	0.101
65	9	29	3.30	120559	29*	0	0.54	20419	0.169
66	29	29	2.18	83391	29	0	0.61	23615	0.283
67	33	73	7.60	251764	71	2	0.32	91451	0.363
68	39	41	1.11	41535	39	2	0.33	13456	0.324
69	31	31	1.14	42528	31	0	0.35	13524	0.318
70	79	79	2.16	85793	79	0	0.69	28115	0.328
71	25	31	0.95	34391	27	4	0.28	11371	0.331
72	41	43	0.66	25282	41	2	0.25	10243	0.405
73	75	-	—	—	-	-	—	—	—
74	15	31	3.15	120940	29*	2	0.32	12740	0.105
75	225	225	15.89	571850	225	0	2.26	83303	0.146
76	31	35	1.96	74397	35*	0	0.30	12755	0.171
77	11	11	0.03	1434	11	0	0.27	11183	7.798
78	13	23	0.71	22207	19	4	0.31	11139	0.502
79	11	11	0.14	5516	11	0	0.48	17552	3.182
80	35	39	7.32	253006	35	4	1.52	53159	0.210
81	23	23	0.32	10917	23	0	0.31	10949	1.003
82	21	25	1.34	50079	25*	0	0.31	11973	0.239
83	29	33	0.54	19723	29	4	0.43	16486	0.836
84	29	29	0.60	23004	29	0	0.38	14853	0.646
85	55	57	4.22	155670	57*	0	0.69	25833	0.166
86	15	21	1.39	54585	21	0	0.33	13097	0.240
87	57	59	9.06	352827	57*	2	0.27	35386	0.100
88	31	-	—	—	-	-	—	—	—
89	69	-	—	—	-	-	—	—	—
90	37	37	0.05	1957	37	0	0.32	13187	6.738
91	31	31	0.52	20546	31	0	0.30	12209	0.594
92	51	53	1.80	69130	51	2	0.13	10146	0.147
93	35	37	1.91	71581	35	2	0.44	17390	0.243
94	57	65	27.87	952551	59	6	3.07	103271	0.108
95	35	39	4.01	143336	35	4	0.23	15043	0.105
96	51	63	1.94	74977	57*	6	0.28	12359	0.165
97	51	51	2.14	78633	51	0	0.32	13084	0.166
98	61	65	12.85	463195	61	4	1.34	50838	0.110
99	41	41	0.22	8229	41	0	0.32	13140	1.597
100	163	163	182.26	6021868	163	0	19.60	628385	0.104
平均						2.4			0.194