不詰を正しく証明するアルゴリズム

A Correct Algorithm to Prove No-Mate Positions in Shogi

岸本　章宏
Akihiro Kishimoto
kishi@cs.ualberta.ca
Department of Computing Science, University of Alberta
Edmonton, AB, Canada, T6G 2E8

概 要

将棋では別手順によって同一局面に至ることがあるために、詰将棋解答プログラムはハッシュ表を用いるのが普通である。ハッシュ表の利用によって、探索に必要な空間を大幅に減らすことができるのだが、将棋には元の局面に戻る手順が存在するために詰将棋プログラムでは、詰む問題を不詰と間違えるGHI(Graph History Interaction)問題を引き起こすことがある。本論文では[8]の手法を利用してGHI問題に対する正当でかつ効率のよい解決策を提案する。さらに本手法を詰将棋プログラムに実装した結果、詰みの性能を保持しつつ、難解な不詰の問題を解くことに成功した。

Abstract

Since identical positions can be reached by more than one path in Shogi, tsume-shogi solvers are typically enhanced with transposition tables. This approach can reduce the search effort by a large margin. However, because Shogi contains repetitions, tsume-shogi solvers suffer from the Graph History Interaction (GHI) problem. GHI may cause a solver to falsely regard proven positions as disproven. This paper presents an efficient and correct method to cure the GHI problem, based on the algorithm in [8]. A tsume-shogi solver with our GHI solution is now able to solve harder fu-tsume positions without degradation in the ability to solve tsume positions.

1　はじめに

詰将棋解答プログラムは、ある局面が詰むのか詰まないのかを判定するプログラムであり、強いコンピュータ将棋プログラムには必ず高性能な詰将棋解答プログラムがある。詰将棋プログラムの性能はここ10年で著しい進歩を遂げてきた。最も効果的なアルゴリズムとしては、長井によって提案されたdf-pnアルゴリズム[18]が挙げられ、長井の詰将棋プログラムは300手以上の既存の詰将棋のすべてを解くことに成功している。

df-pnアルゴリズムは、証明数・反証数[1]を利用した反復深化法を用いており、探索効率向上のためにハッシュ表を利用している。ハッシュ表は、Zobristの方法[14]などによってコード化した局面をハッシュキーとして、詰み情報や証明数や反証数などの探索した結果を保存・参照する大きなテーブルである。ある局面のハッシュキーを計算するときは、その局面に至る経路は無視して、手番を含む局面の情報のみをコード化する。探索が終わった後に探索結果を

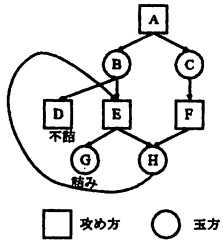

図 1: 詰む局面を不詰と間違える例

ハッシュ表に保存しておけば、df-pn などの反復深化法が行う無駄な同一節点の再探索を避けることができる。また、詰将棋の探索空間の性質上、別手順で同じ (または類似した) 局面に至ることがあるので、ハッシュはより効果的に働く。例えば、手順 p で局面 A を探索して、A が詰みであることが分かったとする。次に別手順 q で A に至ったときには、A のハッシュ・エントリーには詰み情報が入っているので、A 以下を再探索することなしにすますことができる。

詰将棋の探索空間では元の局面に戻るサイクル手順が存在する。詰将棋ではサイクル手順は無駄な手順なので、不詰とみなすのであるが、この不詰は局面に至る手順を考慮した結果である。ところが、ハッシュ表は局面に至る経路を無視した結果を返すので、上のようなハッシュ表による探索の効率化はサイクルに依存する不詰をハッシュに登録する際には必ずしも正しいとは言えない。もし、サイクル手順が起こった際にその局面を不詰として闇雲にハッシュ表に登録すると詰む問題を不詰だと解答してしまうことがある。例えば、図 1 のような探索空間で、次の順で探索する。

- $A \rightarrow B \rightarrow E \rightarrow H \rightarrow E$ と探索。サイクルによって不詰なので、H に不詰を登録する。
- $A \rightarrow B \rightarrow D$ を探索。玉方である B は D に逃れればよいので不詰。
- $A \rightarrow C \rightarrow F \rightarrow H$ を探索。H には不詰が入っているので C と F は不詰。
- B と C は不詰なので、A も不詰であるが、これは間違いである。実際は $A \rightarrow C \rightarrow F \rightarrow H \rightarrow E \rightarrow G$ で詰む。

このように探索アルゴリズムが解答を間違える問題は GHI(Graph History Interaction) 問題 [11] と呼ばれており、df-pn を用いた詰将棋プログラムでも起こる深刻な問題である。現状としては、常に正しい解答を返すためにハッシュ表の利用頻度を減らすか GHI 問題自体を無視して間違えた解答を返すことを甘受するなどの妥協策を取っている。

本論文では、[8] で開発した一般的でかつ効果的な GHI 問題の解決策を利用して、df-pn 系のアルゴリズムである df-pn$^+$[10] を用いた詰将棋プログラムを実装する。我々の方法は常に正しい解答を返すことを保証し、長井の方法よりも難解な不詰問題を解くことができた。

本論文の構成は以下の通りである。2 章では関連研究について述べ、3 章では我々の GHI 問題の解決策について説明する。4 章では詰将棋による実験結果を示した後、5 章でまとめを行う。

2 関連研究

2.1 df-pn アルゴリズム

df-pn は最良優先探索である証明数探策 [1] を深さ優先探索に変換したアルゴリズムである [18]。df-pn は最良優先探索の利点である有効そうな節点から展開することができ、さらに深さ優先探索として振る舞うことによって、内部節点の展開節点数が少なく、メモリの利用効率もよい。df-pn は証明数と反証数の閾値を持ち、これらの閾値は各節点で局所的に調整され、各節点で反復深化を行う。df-pn の探索は各節点での証明数・反証数がどちらかの閾値を超えるまで探索を行う。df-pn は深さ優先探索であるために、高性能なプログラムを書くためにはハッシュ表が必要不可欠である。また、先端節点で評価関数を呼ぶ df-pn$^+$ は、オセロによる実験では df-pn に比べて $\frac{1}{6}$ 程度の節点しか展開しないことが長井によって報告されている [10]。

2.2 GHI問題

GHI問題は1983年にPalayによって最初に指摘された[11]。Palayは2つの解決策を提案したのだが、実際のゲームによる実験は全く行われなかった。CampbellはGHI問題には2種類の起こり方があることを分析した[5]。具体的には、まずサイクル手順に関する結果がハッシュ表に保存され、その値がのちほどサイクル非依存の探索においてハッシュが参照される場合(draw-first case)とサイクル非依存の結果がハッシュに保存され、のちほどサイクル依存の探索においてハッシュ表が参照される場合(draw-last case)である。Campbellは、draw-first caseにおける$\alpha\beta$法用の解決策を提案したが、draw-last caseについては解決することができなかった。

Breukerらは、証明数探索[1]の場合におけるGHI問題を解決した[4]。ところが、Breukerの手法は明示的なグラフ構造を利用するので、df-pnのような深さ優先探索に利用できるかどうかは明らかではない。

長井はdf-pnに対してGHI問題を次のように解決している[18]。ルート局面から、証明数と反証数を大きな閾値($\infty - 1$とする)で探索を開始し、サイクルの時には不詰をハッシュに入れずに親局面に戻る。サイクルに関係なく詰み・不詰が証明されれば、その結果を使うのだが、どちらも証明されずにルートの閾値を超える場合は再探索を行う。再探索は最初の探索よりも大きな閾値(サイクル局面に至る手順を不詰にするために∞とする)で開始し、同様にdf-pnを呼び出す。さらに内部節点に対しても同様のことを行う。長井の方法はdf-pnの性質を利用したエレガントな方法であるが、不詰に対する解答能力についての実験がなされていない。また、サイクル依存の不詰を証明するためには最初に設定した閾値を超えるのを待つ必要があるので、無駄な探索が起こり、不詰を示すのに時間がかかる可能性がある。

その他にもGHI問題に対する様々な研究[2, 12, 13]があるが、実際に実装されなかったり、非常に効率が悪い、解答を間違える場合があるなど満足のいく解決策が提案されていない。

3 我々のGHI問題の解決策

3.1 アルゴリズム

これまでの探索アルゴリズムでは、不詰が行われたときに無条件にその結果をハッシュに保存していて、それがGHI問題を引き起こす原因になっていた。我々の方法は、サイクルに関する不詰が証明されたときにはその結果を無条件に信頼するのではなく、実際に探索を行って、それが正しいかどうかを確かめる。具体的には我々の解決策は次のことを行う。より一般的な場合は[8]に述べられている。

2種類のハッシュ・エントリー ハッシュ・エントリーにはコード化された局面情報[14]だけでなく、手順情報も追加する。ハッシュのエントリーを便宜上2種類に分ける。これらをベースをツインと呼ぶ。不詰以外の証明数・反証数はベースに入れる。局面nが手順pで不詰の場合は、ツインに「手順pのときはnは不詰」と保存し、手順pをたどったときのみ、手順p用のツイン・エントリーの情報を利用する。ツインはいくつも作られることがあり、例えば、さらに手順qでnが不詰のときはpとは別の新たなツイン・エントリーを作成し、「手順qでnは不詰」と保存する。ただし、この方法は「無条件に不詰」であるときに効率が悪いので、改良を行う(本章の残りを参照)。また、不詰をツインに保存するときにはnのベースの証明数と反証数を1に初期化する。

手順のコード化 手順は64ビットにコード化する。コード化はZobristの方法[14]と似た方法を用いる。ゲーム非依存のアイデアとしては、最大指し手数をMで最大探索深さDとしたときに$M * D$のサイズの64ビットの乱数表Rを用意する。Rはあらかじめ初期化しておく。そして手順pに至る指し手を$(m_1, \cdots, m_k)$とすると次のようにしてpをコード化する。

$$\mathrm{code}(p) = R[m_1][1]\mathrm{xor}\cdots\mathrm{xor}R[m_k][k]$$

このコード化の特徴としては、コードは指し手について非可換であり、さらにZobristの方法と同じよ

うに差分だけでコードの計算が可能である。

将棋の場合は、上記のアイデアを少しだけ変更している。まず、上のやり方でミクロコスモスを解くとすると数 MB の乱数表が必要なので、我々の実装では最大深さを 97 に制限して、それ以降の深さでは 97 で剰余を計算することで、乱数表を使いまわしている。手順のコードだけでなく、局面のコードもあるので、この使い回しによる手順コードの衝突は実際上はまず起こらない。さらに実装上の理由からどこの升目からどこに移動したか、成るかどうかなどで小さな乱数表を複数持ち、これらの XOR を取ることで手順のコードを生成している。

シミュレーションの利用　河野のシミュレーション [6] はある局面が詰むかどうかを小さなオーバーヘッドで調べる方法であり、これを不詰のチェックに利用する。ある節点 (局面)n_1 が不詰であるとする。似た節点 n_2 が不詰であることを示すのにシミュレーションでは n_1 が不詰であることを証明した探索木 (反証木) を利用する。つまり、n_2 の子孫節点が OR 節点 (攻め方) の場合は n_2 のすべての指し手をチェックし、AND 節点 (玉方) のときは n_1 を不詰にする指し手のみを n_2 で生成して調べる。このようにして、シミュレーションが不詰を返せば、n_2 は不詰であり、そうでなければ、詰みか不詰か分からないので、df-pn で通常探索を行う。我々の手法でシミュレーションの対象にした局面は次の通りである。局面 A が手順 p でサイクル依存の不詰であるとして、手順 q で局面 A に至ったとする。このときに q の場合が不詰かどうかをシミュレーションで調べる。

また、シミュレーション呼び出し自体がオーバーヘッドを必要とするので、呼び出し回数自体を減らすようにした。df-pn が探索中にサイクル手順に出会ったかどうかをチェックし、反証木にサイクルが含まれている場合のみにシミュレーションを呼ぶようにした。その他の場合は手順非依存で不詰であるので、ベースに不詰情報を入れ、常にこの結果を利用する。

3.2　我々の方法の正当性

我々の GHI 問題解決策が正しいことは次の定理によって証明されている。

定理 3.1 詰みと不詰と探索結果がハッシュから消えないと仮定する。我々の方法は GHI 問題を引き起こさない。

証明 [9] を参照。

■

上の定理は、df-pn が探索途中の証明数・反証数については間違えて計算することがあるのだが、いったん詰み・不詰まで探索してしまえば、常に正しい結果であることを保証している。しかし、定理 3.1 の前提条件から、ハッシュから詰み・不詰の結果を消すことができないので、以前に提案された GC[18] や Replacement[3] に比べてメモリの使用効率が悪い。次の定理は定理 3.1 よりは弱い定理であるが、我々の手法の正当性を保証している。

定理 3.2 詰みと不詰の探索結果がハッシュから消えることがあると仮定する。我々の方法が詰みか不詰の結果を返してきたとき、その値自体は常に正しい。

証明 我々の方法がハッシュに保存する詰み・不詰自体を間違えることがないことを示す。証明には探索結果のハッシュへの書き回数に関する帰納法を用いる。

- ハッシュの詰み・不詰の入ったエントリーが全く置き換えられないときは定理 3.1 より、ハッシュ内の詰み・不詰は全て正しい。
- ハッシュ表の詰み・不詰が正しいとして、GC や Replacement によってハッシュ表のいくつかの (詰み・不詰の入った) エントリーが消えたとする。ハッシュから詰み・不詰情報が消えたとしても、ハッシュ内の残りの詰み・不詰情報は正しい結果である。次に、新たに df-pn が詰み・不詰を探索して示し、その結果をハッシュに保存しよう

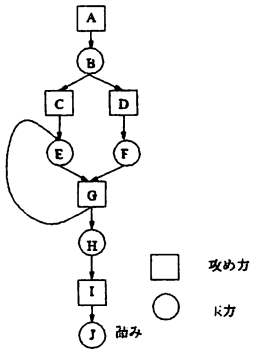

図 2: 間違えた証明木を作る例

としていると仮定する。このときにこの探索結果は節点の展開とハッシュの参照 (帰納法の仮定より正しい結果である) のみによって計算されたものであるので、df-pn の探索による詰み・不詰情報は正しい結果である。

■

ただし、上の定理はある局面が詰みであることを示す探索木 (証明木) という観点においては、正しい証明木 (または反証木) を返すとは限らない。例えば図 2 を次の順番で探索すると探索結果は詰みを返すのだが、サイクル手順を含んだ証明木を返す。

1. 最初に $A \to B \to C \to E \to G \to H \to I \to J$ を探索する。C、E、G、H、I および J は詰みとハッシュに保存する。特に G は H に至る指し手を指すことで詰みだと保存する。

2. Replacement によって、G、H、I と J をハッシュから消去する。

3. $A \to B \to D \to F \to G$ と探索する。ここで、df-pn が G の子節点 E の情報を調べると、E には詰み情報がまだハッシュに残っているので、G は $G \to E$ で詰みだとハッシュに保存する。ところが、G が詰みである理由は手順 $G \to H$ が詰みであるからであり、$G \to E$ はサイクル手順を取るので間違えた証明木である。

この問題自体は、Campbell の draw-last case[5] と類似の問題であり、我々の GHI 問題解決策を実装するかに関係なく起こる問題である。実際に長井は詰将棋プログラムに変別や余詰を求めさせるときに似たような問題が起こることを指摘している [17]。

我々の手法は、例えば詰みがハッシュに入っている場合はサイクルを含むような冗長な手順を経ることなく、詰みを見つけられるはずであることを定理 3.2 によって保証している。ただし、正しい証明木が必要な場合は再探索をすることで証明木を作り直さなければならない。我々の証明木構築アルゴリズムは、サイクル手順に伴う不詰はツインに入れられることを利用した単純な方法を取っている。アルゴリズムは以下の通りである。

- 探索とは別の小さなハッシュ表を用意して、OR 節点では詰みの AND 子節点、AND 節点ではすべての OR 子節点を展開する。

- OR 節点で、詰みの AND 子節点がないときは df-pn で、その OR 節点を探索しなおす。

- 手順 p で局面 n に至るとする。p がサイクル手順の場合は、n を除いた p 上のすべての局面の証明数・反証数を 1 にして、さらに p で n に至るときを不詰にする。次にルート節点から df-pn で探索しなおし、詰みを返してきたときには最初から証明木を構築しなおす。

反証木の構築も上記と似た方法によって行うことができる。

3.3 その他の実装上の問題

我々のハッシュの実装はチェイン・ハッシュを利用している。サイクル依存の不詰を保存するときに、別手順で同一局面にいたる場合が多いときには、多くのツイン・エントリーを作り出してしまうので、非常に長いチェインをハッシュが持つことがある。この問題を避けるために次のようなことを行う。手順 p の局面 n と手順 q の局面 n の両方がサイクル依存の不詰で、p の結果がハッシュに入っているとする。

将棋図巧	第 73 番
将棋無双	第 6 番 第 37 番 第 40 番 第 73 番 第 88 番 第 89 番
続詰むや詰まざるや	第 26 番 第 31 番 第 42 番 第 113 番 第 119 番 第 133 番 第 175 番

表 1: 不詰問題の一覧

もし、q が p と同じシミュレーションで詰むときは局面 n ではなく、手順 q 自体をハッシュ・キーとして結果を保存する。それ以外の場合は通常通り n をハッシュ・キーとして、ハッシュに保存する。

4 実験結果

我々の方法と長井の方法を df-pn$^+$ を用いた詰将棋プログラムに実装し、「将棋図巧」と「将棋無双」[15] および「続詰むや詰まざるや」[19] を用いて実験した。上記の問題集には詰みと不詰の問題がそれぞれ 388 題と 14 題ある。不詰の問題の一覧表を表 1 に示す。実験環境は Athlon 2800+マシン、ハッシュを 300MB で、各問題の制限時間は 15 分で行った。両方のプログラムには、df-pn のサイクルに関する証明数・反証数の計算対策 [7] や証明駒 [16]、SmallTreeGC[18] など性能向上のために最新の手法が用いられている。

我々の方法と長井の方法に関する解答能力の比較は表 2 の通りである。詰みの問題に関してはどちらの手法も全ての問題を解くことができたのだが、不詰問題は長井の方法は全く解けなかったのに対し、我々の方法が詰み能力を保持しつつ、不詰能力を大幅に向上させているのが分かる。上記の問題集に含まれる不詰問題は詰将棋作家が詰みを想定して作成された問題が多く、非常に難解な問題が含まれていると推測されるのだが、ほとんどの問題について我々の方法は不詰の証明に成功した。解けなかった不詰問題は無双 6 番および無双 73 番である。図 3 に不詰を証明した問題の例を示す。

図 4 に詰みの問題についての長井の方法と我々の方法の探索節点数の分布を示す。グラフでは X 軸に我々の方法による探索節点数、Y 軸に長井の方法に

	詰む問題	不詰の問題
長井の方法	388	0
我々の方法	388	12

表 2: 解答能力の比較

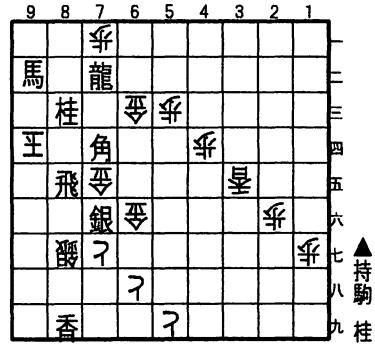

図 3: 「続詰むや詰まざるや」第 42 番

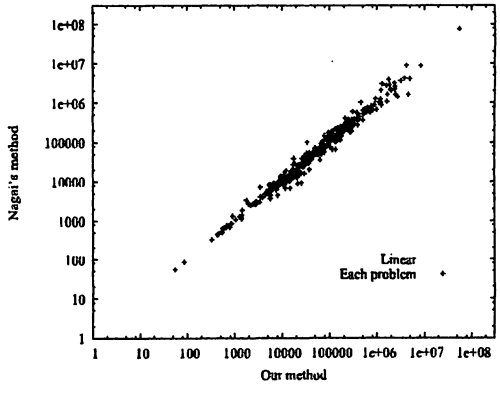

図 4: 詰みの問題による探索節点の比較

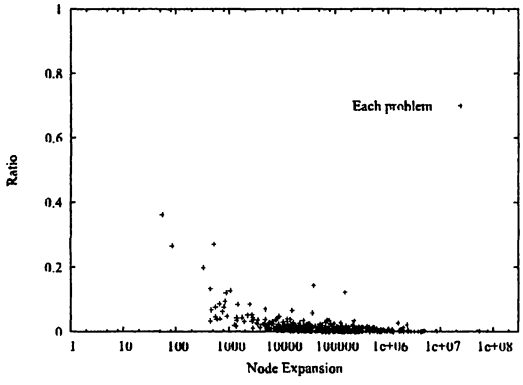

図 5: 証明木構築のオーバーヘッド

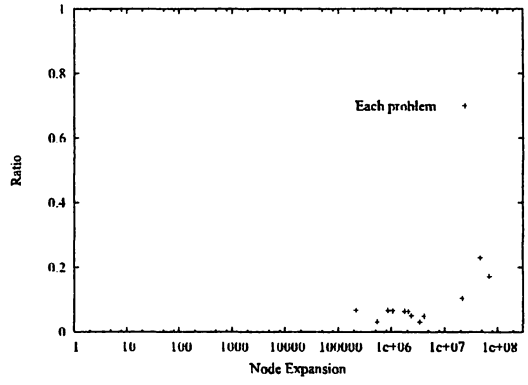

図 6: 反証木構築のオーバーヘッド

よる探索節点数を取り、各問題についてプロットした。このグラフのほとんどの点が $y = x$ のラインの側にあることから、我々の方法は詰みの問題に対して性能を落とすことはないと言える。この実験では、どちらの方法でも SmallTreeGC で消えてしまった証明木を構築することは行わなかった。なお、我々の方法と長井の方法の間には単位時間当たりの展開節点数に有為な差が見受けられなかった。また、我々の方法がシミュレーションに必要とした展開節点数は全体の探索節点のうち、0.7%程度であり、また成功回数は 66%であったので、シミュレーションを起動する頻度自体が全体の探索に比べて少ないと言える。シミュレーションが失敗する可能性としては 2 つ考えられる。1 つは GHI 問題が起こっていることであり、シミュレーションがハッシュに入っている反証木は利用できないことを見つけ出した場合である。もう 1 つは GC によって模倣する反証木が見つからなくなるためにシミュレーションに失敗する場合である。

次に df-pn+が詰みか不詰を返したときに証明木や反証木を構築し直すオーバーヘッドを測定した。図 5 と図 6 に再探索に用した df-pn+の節点数と証明木と反証木の再構築オーバーヘッドの分布を示す。X 軸は探索節点数、Y 軸に探索節点数に対する証明木または反証木構築に必要な節点数の割合として、各問題についてプロットした。証明木を作り直すオーバーヘッドは非常に少なく、平均的に 1.7%程度である。反証木の場合は、問題のサイズが大きくなるにつれて大きくなるようだが、不詰の問題のサンプル数が少ないので、これが一般的な特徴なのかは一概には言えない。反証木構築のオーバーヘッドは平均で 8.4%程度である。

5 まとめ

本論文では不詰を正しく効率よく証明するために GHI 問題の解決策を実装した。我々の方法は詰みの問題への性能を保持しつつ、難解な不詰の問題を解くことに成功した。

6 謝辞

本研究は Natural Sciences and Engineering Research Council of Canada(NSERC) および Alberta Informatics Circle of Research Excellence (iCORE) によって助成された。

参考文献

[1] L. V. Allis, M. van der Meulen, and H. J. van den Herik. Proof-number search. *Artificial Intelli-*

gence, Vol. 66, No. 1, pp. 91–124, 1994.

[2] E. B. Baum and W. D. Smith. Best play for imperfect players and game tree search: part I - theory. Technical report, NEC Research Institute, 1995. Available at `http://citeseer.nj.nec.com/baum95best.html`.

[3] D. M. Breuker, J. W. H. M. Uiterwijk, and H.J. van den Herik. Replacement schemes and two-level tables. *International Computer Chess Association Journal*, Vol. 19, No. 3, 1996.

[4] D. M. Breuker, H. J. van den Herik, J. W. H. M. Uiterwijk, and L. V. Allis. A solution to the GHI problem for best-first search. *Theoretical Computer Science*, Vol. 252, No. 1-2, pp. 121–149, 2001.

[5] M. Campbell. The graph-history interaction: On ignoring position history. In *1985 Association for Computing Machinery Annual Conference*, pp. 278–280, 1985.

[6] Y. Kawano. Using similar positions to search game trees. In *Games of No Chance*, Vol. 29 of *MSRI Publications*, pp. 193–202. Cambridge University Press, 1996.

[7] A. Kishimoto and M. Müller. Df-pn in Go: Application to the one-eye problem. In *Advances in Computer Games. Many Games, Many Challenges*, pp. 125–141. Kluwer Academic Publishers, 2003.

[8] A. Kishimoto and M. Müller. A general solution to the graph history interaction problem. In *Nineteenth National Conference on Artificial Intelligence (AAAI'04)*, pp. 644–649. AAAI Press, 2004.

[9] A. Kishimoto and M. Müller. A solution to the GHI problem for depth-first proof-number search. *Information Sciences*, To Appear.

[10] A. Nagai. Application of df-pn$^+$ to Othello endgames. In *5th Game Programming Workshop (GPW'99)*, pp. 16–23, 1999.

[11] A. J. Palay. *Searching with Probabilities*. PhD thesis, Carnegie Mellon University, 1983.

[12] M. Schijf, L. V. Allis, and J. W. H. M. Uiterwijk. Proof-number search and transpositions. *International Computer Chess Association Journal*, Vol. 17, No. 2, pp. 63–74, 1994.

[13] E. C. D. van der Werf, H. J. van den Herik, and J. W. H. M. Uiterwijk. Solving Go on small boards. *International Computer Games Association Journal*, Vol. 26, No. 2, pp. 92–107, 2003.

[14] A. L. Zobrist. A new hashing method with applications for game playing. Technical report, Department of Computer Science, University of Wisconsin, Madison, 1970.

[15] 伊藤宗看・看寿 門脇芳雄解説. 詰むや詰まざるや, 東洋文庫, 第 282 巻. 平凡社, 1975.

[16] 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. 第 5 回ゲームプログラミング・ワークショップ (GPW), pp. 129–136, 1999.

[17] 長井歩. 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. 第 6 回ゲームプログラミング・ワークショップ (GPW2001), pp. 9–16, 2001.

[18] 長井歩. df-pn アルゴリズムと詰将棋を解くプログラムへの応用. アマ 4 段を超える コンピュータ将棋の進歩 4, pp. 96–114. 共立出版, 2003.

[19] 門脇芳雄 (編). 続詰むや詰まざるや, 東洋文庫, 第 335 巻. 平凡社, 1978.