情報処理学会論文誌 Vol. 51 No. 11 2040–2047 (Nov. 2010)

新規節点で固定深さの探索を行う df-pn の拡張

金子知適[†1] 田中哲朗[†2]
山口和紀[†1] 川合慧[†3]

本稿では df-pn 探索の拡張として，末端節点での浅い探索木の活用を提案し，実際に効率的に詰を発見できることを示す．AND/OR 木の探索において，df-pn は探索節点数の観点で効率が良く，固定深さの探索は節点展開速度に優れている．そこで末端で固定深さの探索を行い，詰む場合はその情報を，詰まない場合は固定深さの探索が展開した探索木の証明数と反証数を計算して，df-pn で活用する．実験の結果，棋譜に表れる実戦の詰み局面に対して，平均探索時間が標準的な df-pn の約 $\frac{1}{3}$，使用した局面表の節点の数は平均約 $\frac{1}{5}$ という大きな効率の向上を実現した．さらに，大規模な詰将棋問題において各種の拡張を組み込んだ df-pn$^+$ との比較においても，所要時間で約 $\frac{2}{3}$，使用した局面表の節点数で半分強という効率の向上を実現した．

Enhancement of Df-pn with Fixed-depth Search at Frontier Nodes

Tomoyuki Kaneko,[†1] Tetsuro Tanaka,[†2]
Kazunori Yamaguchi[†1] and Satoru Kawai[†3]

This paper presents an enhancement on df-pn for AND/OR tree search, by utilization of fixed-depth look ahead. In this enhancement, a shallow fixed-depth search is performed to find out a short proof at each new frontier node visited in the df-pn search. This combination utilizes both the efficiency of df-pn regarding to the number of nodes needed to find a proof and the efficiency of fixed-depth searches in execution time for node expansion. Moreover, even when checkmate is not found, the result of fixed-depth search yields a good estimation of future proof and disproof numbers at the node. Our experiments on checkmate search in shogi showed that the presented combination solved problems with about $\frac{1}{3}$ in execution time and about $\frac{1}{5}$ in memory usage, compared to the df-pn. Also for large problems, it achieved efficiency of about $\frac{2}{3}$ in execution time and about $\frac{1}{2}$ in memory usage, compared to df-pn$^+$ with state-of-the-art techniques.

1. はじめに

詰将棋解答プログラムは，df-pn[18] に代表される技術により大きく進歩してきたが，対局での利用においてはよりいっそうの効率化が望まれている．詰探索においては詰みや不詰みの証明に必要な局面の 10 倍から 100 倍の局面を探索している[17] ため，もし必要な局面だけを読むことができれば大幅に探索局面数や探索時間を削減可能である．df-pn$^+$[6] は不要な探索を減らす技術であり，詰みやすさを評価する評価関数を用いて証明数と反証数の初期化を行う．しかし，将棋においてはこの評価関数の作成自体にも難しさがある．

本研究では詰探索を効率化するために，df-pn の末端節点で固定深さの探索を行い，評価関数を使わずに証明数と反証数の初期値を設定することを提案する．節点あたりの実行時間では固定深さの探索は df-pn よりも効率が良いため，この組合せにより，効率の向上が期待できる．実際の固定深さの探索の実装では，対象とするゲームの性質を利用して探索を効率化できることが多い．本稿では大規模な例題が豊富に存在することから将棋を対象に選んだが，将棋では，一部の一手詰を王手を生成することなしに高速に判定することや，すべての王手後の状態を考慮した証明数と反証数を予想する評価関数の作成が可能である．実験では，棋譜に表れた詰み局面に対して，提案手法は平均探索時間でオリジナルの df-pn の平均で $\frac{1}{3}$ 未満，局面表に登録する節点の数は平均 $\frac{1}{5}$ という望ましい結果となった．

次章で関連研究について紹介した後，3 章で証明数と反証数および，それらを用いた探索手法である df-pn と df-pn$^+$ を説明する．続いて，4 章で df-pn と固定深さの探索の併用方法について提案し，5 章で実験結果を報告した後に，6 章で結論を述べる．

2. 関連研究

df-pn は，pn 探索[1] と同じ振舞いを保ちつつ深さ優先にすることで効率を改善した探索手法である．発表時点で 300 手以上の詰将棋をすべて解くという成果をあげている[18] ほかに，詰碁でも効果が確認されている[3]．将棋を含めてサイクルがあるゲームでは GHI など

†1 東京大学大学院総合文化研究科
Graduate School of Arts and Sciences, The University of Tokyo
†2 東京大学情報基盤センター
Information Technology Center, The University of Tokyo
†3 放送大学
The University of the Air

の問題が生じうるが，df-pn における対策[3] も提案されている．他のアルゴリズムとしては，PN*[7] や PDS[5] が先に考案されているが，現在では df-pn の方が効率が良いとされている[18]．

df-pn の効率をさらに向上させるために，評価関数を組み合わせる改良（df-pn$^+$）が提案されており，オセロの終盤の問題で探索節点数を約 $\frac{1}{6}$ に抑えたという報告がある[6]．関連して，評価関数を用いる場合に起こりやすい再探索の回数を抑えるために，子節点の証明数や反証数の平均値を閾値に反映させる方法も提案されている[3]．

一方で，詰探索における評価関数の作成は比較的難しいとされている．詰将棋のための評価関数作成を目指した過去の研究[17] では探索節点数は減らしたものの，探索時間を減らすにはいたっていない．他の手法では，三輪[15] が SVM を用いて詰みの予測関数を作成し，PDS[5] の探索制御に応用して成果をあげている．他にも様々なアイデアが提案されているが[11),14]，df-pn$^+$ について実戦の局面を対象に十分な量の実験を行った研究は著者らの知る限りではない．本研究は，単純な評価関数と固定深さの探索とを組み合わせることで，複雑な評価関数を用いなくても，優れた効率を得られることを示すものである．

将棋独自の技術として本研究でも用いた一手詰を判定する関数は，山下[10]，佐野ら[13] が文献中で提案しているほか，多くの将棋プログラムが利用している．本研究で利用したものは，王手の生成の必要がなく，また局面を動かさずに判定できる点で効率が良い．

3. 証明数と反証数と df-pn 探索

3.1 証明数と反証数

まず，df-pn において重要な証明数と反証数という変数を説明する．証明数と反証数は節点ごとに定義される値で，探索木の状態により決まる次のような意味を持つ[18]：

証明数　ある節点の詰みの証明のために，詰みを証明する必要がある先端節点の数の最小値．

反証数　ある節点の不詰みの証明のために，不詰みを証明する必要がある先端節点の数の最小値．

df-pn では，この証明数と反証数を用いて次に展開する先端節点を決める．具体的には，ルート節点から攻方では証明数が最小の節点を，受方では反証数が最小の節点をたどった末端の局面を展開する．これを定義どおりに実装した手法が最良優先探索の一種である pn 探索であり，深さ優先探索の枠組みで同じ節点を展開する手法が df-pn である．

証明数と反証数は，ゲーム木に合流がない場合は**表 1** のように再帰的に計算できる．将棋を含めて合流のあるゲームにおいてはこの計算方法は誤りであるが，正確な計算は困難であるために基本的には表 1 の方法が使われる．この場合，合流が起こると多重に計算される節点が生じるため探索の効率は悪化し，サイクルが起こると無限ループなどの問題が生じうる．そこで，閾値の計算の工夫や特別なケースごとの対応が使い分けられている[3),4),8)]．

表 1　証明数・反証数の計算方法
Table 1　Proof numbers and disproof numbers.

節点の種類		証明数	反証数
先端	詰み	0	∞
	不詰み	∞	0
	不明	1	1
内部	攻方	min（子節点の証明数）	$\sum$（子節点の反証数）
	受方	$\sum$（子節点の証明数）	min（子節点の反証数）

3.2 評価関数

オリジナルの df-pn では先端節点の (証明数, 反証数) を $(1, 1)$ とするが，評価関数を使う df-pn$^+$ では，表 1 の計算方法を一部変更し，先端節点では（証明数の予測値，反証数の予測値）を用いる．この予測を行う関数を本稿では評価関数（H）と呼ぶ．この初期化で，予測が正しければ（たとえば詰みがある節点に関して正しい王手の証明数が最も小さくなっていれば），重要な節点が先に展開されて詰みや不詰みが早く見つかると期待される．

df-pn$^+$ には，H のほかに cost という評価も導入されている．こちらは指手ごとに保持する値で，その指手以降の節点の探索で用いる証明数や反証数の閾値を制御する．cost の値を大きくすれば探索は早目に打ち切られ，小さくすると深くまで探索が行われる．0 の場合はオリジナルの df-pn と同じ振舞いとなる．

4. 固定深さの探索と df-pn の組合せ

提案する手法は，df-pn の探索で訪れた新規節点に対して固定深さの探索を行い，その結果を df-pn の探索の制御に活用することである．具体的には，詰みや不詰みの場合はその結果を利用し，そうでない場合は，展開した木について表 1 の計算方法に従って証明数と反証数を計算して利用する．また，詰みの場合には証明駒[12] も固定深さの探索で計算する．

総合的な効率を得るためには，固定深さの探索を効率的に実装する必要がある．予備的な実験の結果，固定深さの探索の末端は攻方の手番で打ち切ることとし，末端節点で一手詰判定関数を用いることが効率的であることが分かった．そのため，固定深さの探索の深さは，

df-pn の新規節点が攻方の手番のときに呼ぶ場合には奇数深さ，受方の手番の場合では偶数深さとした．さらに長手数の固定深さ探索は効率的でないため，実際には 1–3 手読みを用いた．

4.1　攻方の手番での固定深さの探索

攻方の手番の節点で固定深さの探索を行う場合は，df-pn で新規節点を訪問してすぐに，1 手または 3 手の探索を行う．先に述べたように固定深さ探索の葉では一手詰判定関数を用いる．詰みや不詰みの場合はそのまま扱い，そうでない場合は，展開した木について証明数と反証数を計算する．計算した証明数や反証数が df-pn の閾値を超えていれば，その節点の探索を終了させ，そうでなければ通常どおりに df-pn の探索を続ける．節点を訪問した直後に証明数と反証数を計算して閾値と比較する点で，この探索は 3.2 節で説明した評価関数（H）とは使われるタイミングが異なる．

他に，王手それぞれについて一手進めた後で固定深さの探索を行う方法も考えられるが，予備的な実験では性能がふるわなかった．このことは，固定深さ探索により詰みや不詰みが見つかったり証明数や反証数が df-pn の閾値を超えたりする可能性が高いことを示唆している．

4.2　受方の手番での固定深さの探索

受方の手番の節点で固定深さの探索を組み合わせる方法を説明する．初めに王手回避の指手をすべて生成する．続いて，それぞれの応手に対する詰みの有無を，王手を回避した後の局面から固定深さ探索を用いて調べる．実験では深さ 1 の探索を用いた．本稿では区別のために，王手回避の手を含めて深さ 2 と表記する．もし王手回避後の局面が不詰みであれば元の節点も不詰みでありそれ以上の探索は不要となる．詰みの場合は，その王手回避の手は無視して探索が続けられる．詰みとも不詰みとも分からなかった場合は，展開した木について証明数と反証数を計算した値を，3.2 節で説明した評価関数（H）として用いる．

この組合せ方は，攻方の手番で固定深さを組み合わせる場合とは異なる．予備的な実験では攻方の手番同様に指手生成前に固定深さの探索を行う方法も調べたが，それよりも採用した方法の方が効率的であった．原因としては，固定深さの探索の葉で用いる高速な一手詰判定関数において，不詰みの判定ができないことが考えられる．王手をすべて生成すれば不詰みの判定もできるようになるが，そのコストは大きかったため見送った[⋆1]．

⋆1 将来的には，一手で王手がかからなくなるような受けの存在を判定する関数を効率的に実装できれば状況が変わる可能性がある．

1	2	3
4	王	5
6	7	8

(1)	(2)	持駒
7	4, 5, 6, 8	金
5, 6	4	飛金銀
7	1, 2, 3	-

図 1　一手詰判定関数で駒打で詰む可能性を表す例
Fig. 1　Example of pieces needed for checkmate by a drop move.

4.3　一手詰判定関数の設計

続いて，固定深さの探索の末端で用いられる一手詰判定関数について説明する．効率のために，実際の王手を生成せず，また局面も動かさないで判定する方法を実装した．判定を容易にするため，直接の王手のみを対象とし，空き王手や合駒が可能な王手の考慮は見送った．まず玉の 8 近傍（周囲の 8 マス）について次の情報を計算する（各 8 bit）：

(1)　駒を打つ王手の候補のマス（玉以外の利きがなく，攻方の利きがあり，駒がない）
(2)　玉が移動可能なマス（攻方の利きがなく，受方の駒がない）
(3)　利き次第では玉が移動可能なマス（空白か攻方の駒がある）
(4)　駒を動かす王手の候補のマス（玉以外の利きがなく攻方の利きが 2 つ以上あり，空白または受方の駒がある）

続いて，駒打と移動王手の判定を順に説明する．なお，実際には移動王手を先に判定する．これは証明駒が小さい詰みを優先して見つけるためである．

4.3.1　駒打による一手詰の判定

上記の (1) と (2) を組み合わせた 16 bit に関して，(1) のどこかに駒打を行って (2) を網羅可能となるような駒の種類を，事前に計算して表に保存しておく．判定時には，まずその表と持駒を比較することで，一手詰がない場合を効率良く判定する．**図 1** に表に保持する情報の例を示す．玉の周囲のマスに左図のように番号を振るとして，玉が 4，5，6，8 のみに移動可能で，駒打が 7 に可能な場合は，右の表の 1 行目のように持駒に金が必要である．同様に玉が 4 のみに移動可能で 5，6 に駒打が可能な場合には飛金銀のどれかを持っていれば詰む可能性がある（表の 2 行目）．表の 3 行目は何を持っていても詰まない場合に相当する．

必要な持駒のどれかを所持していた場合は，駒打により利きが止まって詰まない例外[10]かどうかを盤面を調査して判定する．その際も，駒の種類と打つ場所から自分の利きを止める可能性のある場所を事前に求めておくことで，効率的に判定を行う．

4.3.2　移動による一手詰の判定

駒打の場合と同様に，上記の (4) の各マスについて，そのマスに移動可能な駒それぞれについて，移動により (2) を網羅できるかをビット演算により求める．可能な場合には，駒の移動により利きを止めて玉の逃げられる場所が増える例外かどうかを詳細に調査する．その際に，移動した駒の直接の利きのあるマスを求めておき，それと (3) の情報をもとにビット演算を行い判定を効率化する．また，移動による王手では，駒の移動により攻方の玉が王手状態になる反則にも注意を払う必要がある．

4.4　一手後の証明数と反証数を予測する評価関数（H'）の設計

一手詰判定関数により一手詰が見つからなかった場合には，その節点の証明数と反証数を予測する．単純な方法としては (1, 1)，あるいは王手を生成して (1, 王手の数) とすることが考えられる．しかし，詰みにくい局面には 1 より大きな証明数を与えることが効率の観点からは重要で，これらの単純な方法には大きな改善の余地がある．また後者の場合には，王手を生成するコストがかかるという問題点もある．

そこで，攻方の手番ですべての王手をかけた後の局面についての証明数の最小値の下限を求める関数 H' を作成した．関数 H' の実装では，4.3 節で説明した一手詰判定関数のために求めた情報を利用することで効率化を図った．

4.4.1　証明数の予測

攻方の手番で深さ 2 の探索木を考えると，表 1 の計算方法を用いる場合の証明数は，各王手の後の局面に関して王手を回避する指手の数の最小値に相当する．効率を落とさないために，ここでは王手を生成せずにそのような予測を行いたい．そこで，一手詰判定関数の王手の候補である (1) と (4) のビットが立っている各マスに関して，駒打または移動で最も効果的な王手がかかった場合の逃げる場所の数を王手回避の手の下限値として用いる．

証明数を予測する際には admissible である（本来の証明数と同じかより小さい）ことが望ましい．この性質は，df-pn$^+$ で df-pn と同じ解を得るために必要であるが，実際の探索効率（節点数）にも大きく影響する．そこで，以下の 3 つの場合の最小値をとる楽観的な予測を行うこととした．ここで自由度とは玉が合法的に移動できるマスの数を意味する．

- 現在の自由度 −1：遠くからの王手で合駒ができない場合（そのような王手はいつでも可能と仮定する）
- 駒打の王手による自由度：一手詰判定関数用の情報の (1) と (2) の組合せ（16 bit）と持駒から，最小自由度を求められる表を準備しておく．
- 移動の王手による自由度：(4) と (2) の組合せと盤の状況から，次のいずれかの最小自由度を求める表を用いる．
  - 竜もしくは馬が玉の 8 近傍に利いている場合：すべての (4) のマスに竜もしくは馬が移動できると仮定した予測
  - 玉の真上に金相当の駒の利きがある場合：すべての (4) のマスに竜，馬，玉以外の任意の駒が移動できると仮定した予測
  - それ以外の場合：すべての (4) のマスに竜，馬，玉以外の駒が移動できるが，例外として真上に金相当の駒は来ることができないと仮定した予測

なお駒打や移動により利きが止まることで最小自由度は増える可能性がある．一手詰判定関数の実装においてはこれを検査する必要があったが，H' においては最小自由度が増えても admissible な予測である性質を満たしているので問題としていない．一方，この予測が admissible でない場合としては，捨駒により最小自由度が低い王手が存在し，かつそれを取り返した場合に「現在の自由度 −1」より自由度が小さくなる場合があげられる．しかし，そのような局面の頻度は少ないと考えられる．

4.4.2　反証数の予測

反証数の予測は，(1) と (4) のビットの数の和（ただし 0 の場合は 1）とした．これは取り返されない，有効な王手の数の最小値に相当し，王手の数の有力な近似となっている．またこの予測は admissible である．各ビットに対して，利きの数や，王手になる持駒の数を考慮すればより正確な予測となると考えられる．

5. 実 験 結 果

提案手法の有効性を示すために，2 種類の詰探索の実験を行った．1 つは，コンピュータ将棋プログラムが実戦で解く必要に迫られるような，比較的小規模な実戦の局面を題材に用いた．もう 1 つは，大規模な問題での効果を示すために，将棋無双と将棋図巧の問題を題材に比較を行った．両方の実験で良い結果を得たことを順に紹介する．

5.1　実戦の局面を用いた性能評価

この実験には，オープンソースプログラムである GPS 将棋[16] を用いた．局面の優越関係，証明駒[12]，GHI 対策，ループ対策[3] など詰将棋探索における標準的な効率化は実装されている．また測定は AMD Opteron 252（Linux 64 bit）の機器で行った．

まず，人間同士の対局（将棋倶楽部 24 の棋譜集[9]）から 10,000 節点以内の探索で詰みがある局面を集めた．似たような局面が集まることを避けるため，1 つの棋譜からは最初に見つかった 1 つのみを採用した．その結果，1,000 棋譜から 777 局面を得た．同様に，不詰

表 2　固定深さ探索の節点あたりの平均所要時間
Table 2　Average time needed for each node in fixed-depth search.

深さ	H’	詰みのある局面 平均	詰みのある局面 標準偏差	詰みのない局面 平均	詰みのない局面 標準偏差
1	無	1,528.4	41.0	1,212.8	22.6
1	有	1,808.3	44.4	1,656.8	16.8
3	無	1,575.4	12.6	1,627.8	11.0
3	有	1,607.2	9.80	1,676.8	13.2

みの局面の例として，965 局面を収集した．不詰みの場合はあまりに簡単な局面を避けるために，1,000 節点の探索では不詰みが証明できず，10,000 節点以内の探索で不詰みが証明できるという条件で収集した．局面の収集の際には，評価関数の影響を排除するために H や cost は使用していない．

5.1.1　固定深さ探索の効率と評価関数 H’ の性質

まず，固定深さ探索の速度を測定するために，詰みのある局面と詰みのない局面の両方に対して探索を行い，節点あたりの所要時間の平均値を求めた．探索深さ 1 および 3 の結果を**表 2** に掲載する．なお単位はマシンサイクルである．詰みの問題セットと不詰みの問題セットのそれぞれについて，実行時間の揺らぎを吸収するために 10 回ずつ測定を行い，平均と標準偏差を測定した．その際に同じ局面で連続して測定を行うと，CPU のキャッシュや分岐予測の影響で 2 回目以降は何割も高速に実行される．その影響を防ぐため，各局面を 1 回ずつ測定することを 10 回繰り返している．また，詰み探索のコードをロードする時間の影響を抑えるためにテストとは関係ない局面で詰み探索を行った後に，各測定を行った．

一手詰があるかどうかを判定するだけならば，測定に用いた 2.6 GHz の CPU ならば毎秒 100 万局面以上行うことができて十分に高速である（通常の df-pn は節点あたりに平均 7,000 から 8,000 サイクルを要するため，1,500 サイクル程度の固定深さの探索とは 5 倍程度の差がある）．なお，王手のかからない局面での一手詰判定はさらに高速である．また一手詰判定に加えて評価関数 H’ の評価も行うと，深さ 1 の場合は 2–3 割程度のコストが余分にかかっている．一方，深さ 3 の場合は，評価関数 H’ の有無でほとんど差はない．

続いて，評価関数 H’ の性質を調べるために，詰みのある局面と詰みのない局面の両方について，固定深さ探索が返す証明数と反証数の平均値を求めた．**表 3** に結果を掲載する．深さ 1 で H’ を使わない場合は，証明数と反証数の予測値は必ず $(1, 1)$ になるため掲載していない．実験結果から，探索深さが深い方が，また H’ を使う方が証明数は大きくなり，深く読むことや H’ の利用が有効であると期待される．表の 2 行目は，深さ 3 で展開した木の

表 3　固定深さ探索が返す証明数と反証数の平均値
Table 3　Average proof and disproof numbers returned by fixed-depth search.

深さ	H’	詰みのある局面 証明数	詰みのある局面 反証数	詰みのない局面 証明数	詰みのない局面 反証数
1	有	1.62	1.25	2.29	1.01
3	無	1.53	4.20	2.63	5.04
3	有	1.88	4.18	3.12	5.05

表 4　実戦に現れる局面を用いた性能評価
Table 4　Performance with checkmate problems in real game positions.

手法	詰問題 所要時間（$\times 10^4$）	詰問題 節点数（展開）	詰問題 節点数（局面表）	詰問題 失敗	不詰問題 所要時間（$\times 10^4$）	不詰問題 節点数（展開）	不詰問題 節点数（局面表）
df-pn	3,869	7,029	5,429	1	1,607	3,383	2,171
df-pn/a1	2,312	4,370	3,573		1,493	3,216	2,067
df-pn/a3	1,205	1,355	906		2,569	4,060	1,497
df-pn/d2	1,382	2,008	1,680		2,108	3,554	2,386

（H を使わない場合の）証明数と反証数を示している．これと表の 1 行目との比較から，H’ は深さ 3 の証明数をある程度近似していると考えられる．一方，反証数については 1 に近い値である．これは H’ のような表を持っていないためである．

5.1.2　詰探索の性能の改善

詰探索の性能の改善を示すために，提案手法の有無を変えた詰み探索プログラムで探索を行い，その効率を調べた．評価は，所要時間（マシンサイクル）と局面表に登録した局面の数などを測定し，平均をとった．探索は 40 万節点で打ち切ったため，解けなかった問題が一部に存在した．また，探索節点数では，df-pn の節点のみを数え，固定深さの探索の節点は数えていない．

まず，詰みのある局面に対しては，**表 4** の左側のように有望な結果が得られた．全部で 777 局面あるが，解けなかった問題の影響を排除するために，所要時間などの平均値はすべての手法で解けた問題を対象とした．解けなかった問題の数は各手法ごとに表の「失敗」の項に記載した．また，平均所要時間と局面表に登録された節点数の平均に関しては，最も良かった手法の数値を 下線 で表現した．なお，平均所要時間の単位はマシンサイクルで，桁が大きいため一万で割った値を記載している．

各行が各設定のアルゴリズムに対応し，初めの “df-pn” は改良前のもの，それ以降が提

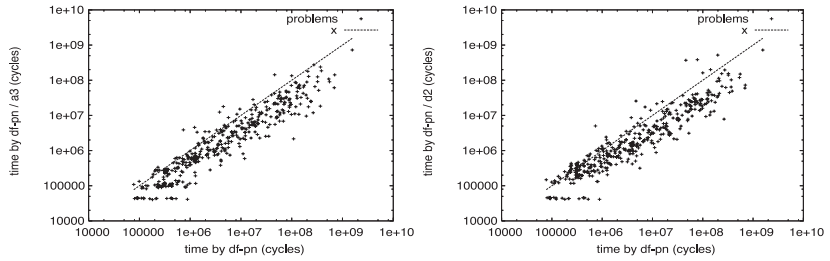


図 2　df-pn との所要時間の比較（左：df-pn/a3，右：df-pn/d2）
Fig. 2　Efficiency compared to df-pn (Left: df-pn/a3, Right: df-pn/d2).

案手法である．“df-pn/a1” は攻方の末端節点で一手読みを，“df-pn/a3” は攻方の局面で三手読みを，“df-pn/d2” は受方の局面で二手読みをそれぞれ行う拡張である．

提案手法は，どれもオリジナルの df-pn より所要時間の観点からも局面表の使用量の観点からも優れている．特に，df-pn/a3 では速度にして 3 倍以上，メモリ使用量削減に関しては約 5 倍の効率化が達成されていることが読みとれる．**図 2** に，df-pn/a3 と df-pn/d2 に関して，df-pn と所要時間を比較する散布図を掲載した．ほとんどの問題が $y = x$ の線よりも右下に位置し，解を得るために必要な時間が偏りなく短くなっていることが分かる．

不詰みの局面での結果は表 4 の右側にまとめられている．この実験では，どの手法も 965 局面すべてを不詰みと判定した．所要時間は，攻方で一手読みを行う手法（a1）が効果的であった．このことは，一手詰判定関数の利用は不詰みの証明にも役に立つことを示している．

5.2　大規模な詰将棋問題を用いた性能評価

将棋無双と将棋図巧の問題を題材に大規模な詰将棋問題での性能を評価する．両者は詰将棋の古典的な名作を集めた問題集で，詰みまでに 600 手を超す問題も収録されている．この実験では Xeon 5470 3.33 GHz の機器を使用した．大規模な問題では，オリジナルの df-pn に対して評価関数を用いる df-pn$^+$ の効果が強く現れるため，この実験では手調整の評価関数を用いる df-pn$^+$ を基準に提案する拡張の効果を示す．df-pn$^+$ の評価関数の H としては，攻方の手番で王手後の逃げ方をビット演算で数える関数を証明数に用いた．また，もう 1 つの評価関数である cost には，**表 5** に掲載するような攻方がただで駒を捨てる場合のペナルティを用いた．これらの値は，本稿の実験で使った問題とは別の実戦の詰み問題約 20 題での df-pn$^+$ の性能を参考に簡単に調整した．同様の理由で，速度を追求した設定での効果を示すために，df-pn$^+$ の実装においても前節の実験からさらに改良が加えられた OSL revision

表 5　攻方がただで駒を捨てる cost
Table 5　Cost for sacrifice moves in attack.

と	成香	成桂	成銀	馬	龍	金	歩	香	桂	銀	角	飛
1	2	2	2	4	4	4	1	2	2	2	3	3

表 6　将棋図巧と将棋無双を用いた性能評価
Table 6　Performance with problems in Shogi-Zuko and Shogi-Muso.

	将棋図巧				将棋無双			
手法	所要時間（秒，比）		局面表（$\times 10^4$，比）		所要時間（秒，比）		局面表（$\times 10^4$，比）	
df-pn$^+$	377.9	1.00	2,738.9	1.00	251.7	1.00	2,041.7	1.00
df-pn$^+$/a1	271.6	0.72	2,010.1	0.73	208.1	0.83	1,680.3	0.82
df-pn$^+$/a3	331.7	0.88	1,386.1	0.51	298.7	1.18	1,341.3	0.66
df-pn$^+$/d2	264.7	0.70	1,683.6	0.61	193.7	0.77	1,342.6	0.66
df-pn$^+$/d2+	233.8	0.62	1,431.4	0.52	184.7	0.73	1,224.2	0.60

4,050 のものを用いた[⋆1]．子節点の平均値に基づく閾値の拡張に加えて，ループなどの扱いを深さに基づく対策[3] よりも改善させるために，次の拡張を導入した．(1) 100 を超えるかつ，親節点の閾値より大きな証明数（反証数）を持つ子節点は他に候補がなくなるまで無視することと，(2) ある節点の最初の訪問した際に証明数や反証数の合計が閾値を超えた場合は閾値の方を増やすことで探索を続ける[⋆2]拡張の 2 つである．さらに，以下の将棋の性質を利用した工夫も加えた．合駒で王手を防いだ後に攻方がその合駒をとった場合，受方は取り返す手以外は後回しにする．また，他の手を読む場合にも合駒前の局面で可能な着手であればその局面での詰み筋をシミュレーションで試す．また，無駄合いと予想される受けは他の指手がすべて詰みとなるまで生成せず，生成した後もその節点の証明数の計算方法を子の合計から最大値に変更する．これらの手法の効果は，予備的な実験により確認されている．たとえばミクロコスモスという 1,525 手詰めの詰将棋を PN*[7] は 5 億局面かけて解いているが，実験で用いた df-pn$^+$ は 5,000 万局面かからずに解く．

表 6 に，将棋図巧と将棋無双から不詰みの問題を除いたそれぞれ 99 題と 94 題を用いた性能評価を示す．ただし df-pn$^+$ で 1 題だけ制限節点数で解けない問題があったため，それを除いた結果を示している．ここでは，詰みを見つけるのに要した所要時間の合計を秒で，局面表に記録した局面の数を 1 万局面単位で記した．この実験の目的は現実的な性能を測る

⋆1 http://gps.tanaka.ecc.u-tokyo.ac.jp/cgi-bin/viewvc.cgi?view=rev&revision=4050
⋆2 (2) の部分は岸本の手法[4] である．ただしこの実験では証明数は 100，反証数は 200 を超える場合だけ有効にした

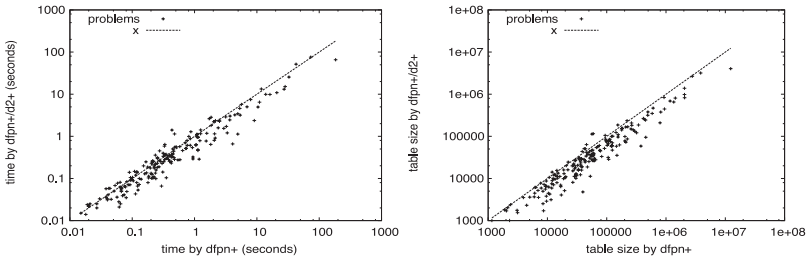


図 3　df-pn$^+$ と df-pn$^+$/d2+の性能比較（左：秒数，右：局面表の大きさ）
Fig. 3　Comparison of effeciency of df-pn$^+$and df-pn$^+$/d2+ (Left: seconds, Right: table size).

ことであるため，探索節点数については今回は計測しなかった．理由は，固定深さの探索，シミュレーション[2)]，df-pn のそれぞれで節点あたりの所要時間が大きく異なるためである．

df-pn$^+$ と df-pn$^+$/a1，df-pn$^+$/a3，df-pn$^+$/d2 を比べると，局面表の使用率では df-pn$^+$/a3 が df-pn$^+$ の 5，6 割に抑えることができており，所要時間では df-pn$^+$/d2 が $\frac{2}{3}$ 程度に抑えることができていて，それぞれ改善された．特に，df-pn$^+$ と df-pn$^+$/a1，df-pn$^+$/d2 の 3 者の間にはどの基準で評価しても df-pn$^+$ より df-pn$^+$/a1 が，df-pn$^+$/a1 より df-pn$^+$/d2 が良いという結果になっており，新規節点で深く読むことの効果を示している．一方，df-pn$^+$/a3 では遅くなっている問題もあり，3 手の展開は無駄な場合もあることが読み取れる．

最後に，df-pn$^+$/d2 と df-pn$^+$/a3 のそれぞれの良さを活かして適応的に組み合わせる実験を行った．3 手の固定深さの探索における詰みの発見率を調べたところ，攻方が持駒に金を持っていて，受方の玉が端（後手玉だとすると $x \geq 7$ または $x \leq 3$ または $y \leq 3$）の場合と，シミュレーション中の場合に成功率が高いことが分かった．前者は将棋の性質から，後者はシミュレーション中は詰み筋に近い局面を訪問するという理由から，両方とも自然な性質と思われる．そこで，df-pn$^+$/d2 を基本に，上記の条件を満たした場合だけ 3 手の詰みを探すことを行った．その結果を，df-pn$^+$/d2+ の行に示す．df-pn$^+$/d2+ は df-pn$^+$/d2 よりもすべての基準で良い結果となり，局面表使用量も df-pn$^+$/a3 に並ぶほど少なく抑えることができている．**図 3** に，df-pn$^+$/d2+と df-pn$^+$ を比較する散布図を示す．左が所要時間の比較で，右が使用した局面表の比較である．両者とも，ほとんどの問題が $y = x$ の線よりも右下にプロットされ，安定して性能が向上していることが読み取れる．

6. おわりに

詰みの発見をはじめとする AND/OR 木の探索を効率化するために，現在最良の探索アルゴリズムと考えられている df-pn に 1 手から 3 手の固定深さの探索を組み合わせることを提案した．df-pn は探索節点数の観点ではきわめて効率の良いアルゴリズムであるが，節点あたりに必要な時間に関しては固定深さの探索の方が効率の良い実装が可能である．そこで，新規節点で固定深さの探索を行い，詰む場合はその情報を用い，詰まない場合は探索木の証明数と反証数を計算し df-pn$^+$ の評価関数として活用することで，探索節点数と解を得るまでの時間の両方の観点に関して効率の良いアルゴリズムを得ることができた．固定深さの探索の末端では一手詰判定関数と，王手後の証明数と反証数を予想する評価関数（H'）を利用した．最良優先探索に深さ固定の先読みの併用することは 2 人ゲーム以外の分野でも行われているため，本手法も将棋以外のゲームでも効果があると期待される．

将棋を題材とした実験により，df-pn で求めた棋譜に表れる詰み局面に対して，平均の探索時間でオリジナルの df-pn の約 $\frac{1}{3}$，局面表に登録する節点の数で約 $\frac{1}{5}$ という大きな効率の向上を示した．また，各種の拡張を組み込んだ df-pn+との比較を将棋図巧や将棋無双といった大規模な詰将棋問題で行ったところ，提案手法は安定して性能向上を示した．特に，受方の手番の末端での 2 手の探索と限定的な 3 手の探索を併用することで，所要時間を約 $\frac{2}{3}$ に，使用した局面表の節点数を半分強に抑えられ，大きな効率の改善を実現した．

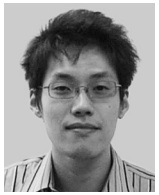
謝辞　有益なコメントをいただいた査読者の方々に感謝する．

参考文献

1) Allis, L.V., van der Meulen, M. and van den Herik, H.J.: Proof-number search, *Artificial Intelligence*, Vol.66, pp.91–124 (1994).
2) Kawano, Y.: Using similar positions to search game trees, *Games of No Chance*, Nowakowski, R.J. (Ed.), MSRI Publications, Vol.29, pp.193–202, Cambridge University Press (1996).
3) Kishimoto, A.: Correct and Efficient Search Algorithms in the Presence of Repetitions, Ph.D. Thesis, University of Alberta (2005).
4) Kishimoto, A.: Dealing with Infinite Loops, Underestimation, and Overestimation of Depth-First Proof-Number Search, *AAAI*, pp.108–113 (2010).
5) Nagai, A.: A new AND/OR Tree Search Algorithm Using Proof Number and Dispr oof Number, *Complex Games Lab Workshop* (1998).
6) Nagai, A. and Imai, H.: Application of df-pn$^+$ to Othello Endgames, *Game Pro-*

gramming Workshop in Japan '99, pp.16–23 (1999).
7) Seo, M., Iida, H. and Uiterwijk, J.W.: The PN*-search algorithm: Application to tsume-shogi, *Artificial Intelligence*, Vol.129, No.1-2, pp.253–277 (2001).
8) 柿木義一：詰将棋プログラムにおける証明数の 2 重カウント対策の一手法，コンピュータ将棋協会誌，Vol.17, pp.36–37 (2005).
9) 久米　宏：将棋倶楽部 24 万局集，ナイタイ出版 (2002).
10) 山下　宏：YSS–そのデータ構造，およびアルゴリズムについて，コンピュータ将棋の進歩 2，松原　仁（編），chapter 6, pp.112–142, 共立出版 (1998).
11) 野下浩平：詰将棋を解くプログラム T2，コンピュータ将棋の進歩，松原　仁（編），共立出版，chapter 3, pp.50–70 (1996).
12) 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，ゲームプログラミングワークショップ '99，pp.129–136 (1999).
13) 佐野晶彦，橋本　剛，長嶋　淳，飯田弘之：一手詰み判定関数の実装および終盤探索への応用，第 9 回ゲームプログラミングワークショップ，pp.9–13 (2004).
14) 田中盛一，飯田弘之，小谷善行：詰み判定評価関数と pn 探索の融合，ゲームプログラミングワークショップ '95，pp.138–147 (1995).
15) 三輪　誠：SVM を用いた将棋の詰みの予測とその応用，修士論文，東京大学新領域創成科学研究科基盤情報学専攻 (2005).
16) 金子知適：コンピュータ将棋の新しい波：3. 最近のコンピュータ将棋の技術背景と GPS 将棋，情報処理，Vol.50, No.9, pp.878–886 (2009).
17) 金子知適，田中哲朗，山口和紀，川合　慧：詰将棋における dfpn+探索のための，展開後の証明数と反証数を予測する評価関数，第 9 回ゲームプログラミングワークショップ，pp.14–21 (2004).
18) 長井　歩，今井　浩：df-pn アルゴリズムと詰将棋を解くプログラムへの応用，情報処理学会論文誌，Vol.43, No.6, pp.1769–1777 (2002).

(平成 22 年 1 月 25 日受付)
(平成 22 年 9 月 17 日採録)



金子　知適（正会員）
1997 年東京大学教養学部卒業．2002 年東京大学大学院総合文化研究科博士課程修了．博士（学術）．2002 年東京大学院総合文化研究科助手．2007 年助教．思考ゲーム，知識処理に興味を持つ．

田中　哲朗（正会員）
1965 年生まれ．1987 年東京大学工学部計数工学科卒業．1992 年東京大学大学院博士課程修了．博士（工学）．東京大学工学部助手，東京大学教育用計算機センター助教授を経て，現在は東京大学情報基盤センター准教授．日本ソフトウェア科学会，ACM 各会員．

山口　和紀（正会員）
1978 年東京大学理学部数学科卒業．1981 年東京大学理学部助手．1985 年理学博士．1989 年筑波大学電子情報工学系講師．1992 年東京大学教養学部助教授．1999 年東京大学情報基盤センター教授．2007 年東京大学総合文化研究科教授．モデリング全般に興味を持つ．ACM 会員．

川合　　慧（正会員）
1967 年東京大学理学部物理学科卒業．1980 年東京大学理学部情報科学科講師．1984 年東京大学教育用計算機センター助教授．1988 年東京大学教養学部教授．1996 年東京大学総合文化研究科教授．2007 年放送大学教授，理学博士．研究分野グラフィクス，プログラミング，ユーザインタフェース．電子情報通信学会，ソフトウェア科学会，ACM 各会員．