新規節点で固定深さの探索を併用する df-pn アルゴリズム

金子知適[†] 田中哲朗[††]
山口和紀[††] 川合慧[†]

本稿では，df-pn アルゴリズムの拡張として，末端節点で 1 から 3 手の固定深さの探索を行うことを提案し，実戦で表れる詰の発見を題材に探索が効率化されることを示す．AND/OR 木の探索において df-pn は現在最良のアルゴリズムと考えられており，探索節点数の観点では極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深さ優先探索の方が効率の良い実装が可能である．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 に提供することを行った．また，固定深さの探索の末端では，王手を生成することなしに一手詰みを判定する関数と，全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 (H') を実装して利用した．この組み合わせにより，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．

実験の結果，df-pn で求めた棋譜に表れる詰み局面に対して，平均探索時間がオリジナルの df-pn の約 $\frac{1}{3}$，局面表に登録する節点の数は平均約 $\frac{1}{5}$ という大きな効率の向上を実現したことを示した．また，評価関数 (H, cost) を使う df-pn$^+$ に対しても，受け方の手番の末端で 2 手読みを用いることで，df-pn$^+$ の約 $\frac{2}{3}$ という効率の向上を実現したことを示した．

Df-pn with Fixed-Depth Search at Frontier Nodes

TOMOYUKI KANEKO ,[†] TETSURO TANAKA ,[††] KAZUNORI YAMAGUCHI [††]
and SATORU KAWAI [†]

This paper proposes a new AND/OR tree search algorithm by combining df-pn algorithm with fixed-depth search. In the proposed algorithm, fixed-depth search is performed at each new frontier node of df-pn algorithm, in order to efficiently find checkmate or to compute proof and disproof numbers of the node according to the search tree expanded by fixed-depth search. This combination is expected to be effective because df-pn is a very efficient algorithm as to the number of node expansion and because fixed-depth search is more efficient in execution time for node expansion than df-pn. We also developed two special functions for use in fixed-depth search. One is for finding one-ply checkmate without move generation, and the other is an evaluation function to estimate proof and disproof numbers considering all attack moves without move generation.

Our experiments on checkmate search in Shogi showed that the proposed algorithm was more efficient than df-pn by about three times in execution time and five times in memory use. Also, compared to df-pn with evaluation functions (df-pn$^+$), it was more efficient by about 1.5 times in execution time.

1. はじめに

詰将棋解答プログラムは，df-pn アルゴリズム[21] に代表される技術の進歩により大きく進歩してきたが，実戦での利用を考えるとさらなる効率化が必要とされている．現実の詰将棋プログラムは，詰や不詰の証明に必要な局面の 10 から 100 倍の局面を探索している[19]．このことは，もし証明に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できることを意味している．不要な探索を減らす方法としては，詰みやすさを評価する評価関数を作成し末端接点の証明数と反証数を適切に初期化する方法 (df-pn$^+$[6]) がある．しかし，二手以上先を予測する評価関数を調整することは現在のところ様々な困難がある[18]．

そこで本研究では，詰探索を効率化するために df-pn の末端接点で固定深さの探索を行い，証明数と反証数

† 東京大学大学院総合文化研究科
Graduate School of Arts and Sciences, The University of Tokyo
{kaneko,kawai}@graco.c.u-tokyo.ac.jp
†† 東京大学情報基盤センター
Information Technology Center, The University of Tokyo
{ktanaka,yamaguch}@mail.ecc.u-tokyo.ac.jp

を求めることを提案する．単純な評価関数でも探索と組み合わせることで証明数と反証数の予測は正確になるため，評価関数の設計が容易になると期待できる．また，df-pn は探索節点数の観点では極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実行時間に関しては深さ優先探索の方が効率の良い実装が可能である．両者を組み合わせることで，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向上が期待できる．本研究ではさらに，高速な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') を用いることでより効率をあげている．

実際に，提案したアルゴリズムは棋譜に表れた詰み局面に対して，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．

次節で関連研究について紹介した後，3.1 節で証明数と反証数及び，df-pn 探索と df-pn$^+$ 探索を説明する．続いて，4 節で df-pn と固定深さの探索の併用方法について提案し，5 節で実験結果を報告した後に，6 節で結論を述べる．

2. 関 連 研 究

近年発展している df-pn アルゴリズムは，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良したアルゴリズムで，300 手以上の詰将棋を全て解くという成果をあげている[21]．また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 問題が生じる可能性があるが，df-pn における対策[4] も提案されている．他に詰将棋を解くアルゴリズムとしては，PN*[7),12)] が df-pn より先に考案され，成果をあげてきた．こちらも有力な手法であるが，反復深化で反証数を閾値にするという点で df-pn の方が効率が良いとされている[21]．そこで，本研究では，df-pn アルゴリズムをベースにした．

さらに効率を向上させる工夫として，df-pn に評価関数を組み合わせたアルゴリズム (df-pn$^+$) が提案されており，オセロの終盤の探索で良い評価関数を用いると探索ノード数を $\frac{1}{6}$ にできるという効果が確認されている[6]．また，評価関数を用いる場合は既に展開した節点の再探索が増える性質があるため，子節点の証明数や反証数の平均値に基づく大き目の閾値を用いる方法も提案されている[2]．

一方で，評価関数を設計することの難しさから詰将棋に関しては，df-pn$^+$ はそれほど広まっていない．☆過去の筆者らの詰将棋の評価関数に関する研究[18] 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らなかった．他のアルゴリズムと組み合わせた詰将棋の評価関数としては，三輪ら[16),17)] 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),15)]，df-pn$^+$ 探索を用いて実戦の局面を対象に充分な量の実験を行なった研究は著者らの知る限りではない．本研究では，単純な評価関数と固定深さの探索と組み合わせることで，複雑な評価関数を用いなくとも，優れた効率が得られることを示している．

将棋において，一手詰を判定する関数は山下[10]，佐野ら[14] が文献中で提案している他，多くの将棋プログラムが利用している．本研究で用いた一手詰判定関数は，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ため非常に効率の良い実装になっている．

3. 証明数と反証数と df-pn アルゴリズム

3.1 証明数と反証数

まず，df-pn アルゴリズムの振舞いを制御する変数である証明数と反証数について説明する．証明数と反証数は節点毎に定義される値で，探索木の状態により決まる次のような意味を持つ:[21]

証明数 ある節点が詰であることを証明するために，詰であることを証明する必要がある先端節点の数の最小値．

反証数 ある節点が不詰であることを証明するために，不詰であることを証明する必要がある先端節点の数の最小値．

この証明数と反証数は，次に展開する節点を決めるために用いられる．すなわち，ルート節点から攻方では証明数の小さい節点を，受方では反証数の小さい節点をたどった末端の局面が順次展開される．これは pn 探索のアルゴリズムであるが，df-pn 探索でも計算方法は異なるものの同じ節点が展開される．

その際に，証明数と反証数を定義通りに計算することは困難であるため，木の探索を仮定して**表 1** のように再帰的に計算する．合流のあるゲームでは，木であるという仮定が成りたたないためこの計算方法では効率が悪くなる．この問題に関しては一般的な解決法はまだなく，特別なケース毎に対応が提案されている[8]．さらに，詰将棋ではサイクルが存在するため，ノードの深さを保持して上流への合流はカウントを遅らせるなどの対策[3] も必要である．

☆ 「東大将棋」では駒得具合を考慮して証明数を予測しているという (岸本氏との個人的な会話).

表 1　証明数・反証数の計算方法

節点の種類		証明数	反証数
先端	詰	0	∞
	不詰	∞	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$^+$ 探索では，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 (例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), df-pn$^+$ 探索は，証明に必要な節点のみを展開するため効率が良くなる．但し，そのような評価関数を作ることは容易ではない．

H の他に，df-pn$^+$ アルゴリズムでは cost という別の評価も導入している．こちらは指手毎に保持する値で，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．df-pn 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 と同じ振る舞いとなる．

4. 固定深さの探索と df-pn の組み合わせ

提案手法では，df-pn アルゴリズムの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 アルゴリズムの動作に反映させる．具体的には，詰や不詰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[13]も計算する．

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時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．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探索

攻方の手番での固定深さの探索を行う場合は，df-pn アルゴリズムの攻方の手番の新規節点で指手 (王手) を生成する前に，1 手または 3 手の探索を行う．詰みや不詰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 の閾値を越えていれば，その節点の探索を終了させ，そうでなければ通常通りに df-pn 探索を続行する．

節点を訪問してから節点全体に対する証明数と反証数を計算する点で，この探索は 3.2 節で説明した評価関数 (H) とは異なる．予備的な実験の結果，指手を生成後にそれぞれの指手に対して固定深さの探索を行うよりも，指手の生成前に探索を行う方が効率的であった．このことは，固定深さ探索により詰や不詰が見つかったり証明数や反証数が df-pn の閾値を越え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探索

受方の手番での固定深さの探索を行う場合は，df-pn アルゴリズムの受方の手番の新規節点で指手 (王手回避) を生成した後に，それぞれの指手について，2 手 (王手回避の後に一手詰みがあるかどうか) の探索を行う．詰みや不詰の場合はそのまま扱い，そうでない場合に展開した木について証明数と反証数を計算することは，攻方の手番で固定深さの探索を行う場合と同様である．

この探索は，それぞれの指手毎に証明数と反証数を求めるため，3.2 節で説明した評価関数 (H) の一種と捉えることができる．予備的な実験では，攻方の手番で固定深さの探索を行う場合と異なり，指手の生成前に探索を行うよりも，指手を生成後にそれぞれの指手に対して固定深さの探索を行う方が効率的であった．この理由は，固定深さの探索が攻方の手番で終わり，王手の生成をせずに一手詰判定関数を呼ぶだけであるため，不詰の判定ができないためと考えられる．王手を全て生成すれば不詰の判定が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．☆

4.3 一手詰判定関数の設計

続いて，本研究で用いた一手詰判定関数について説明する．本研究で実装した関数は，実際の王手を生成せず，また局面を動かさずに一手詰の存在を判定するため非常に効率が良いという特徴がある．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は考えていない．また，移動王手と駒打では移動王手を先に判定する．これは証明駒を最小にするためである．

☆ 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 駒打用の表の例

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) 駒を打つ候補のマス (玉以外の利きがなく，攻方の利きがある空白)

(2) 玉が移動可能なマス (攻方の利きがなく，受方の駒もない)

(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 つ以上あるような，空白または受方の駒のある)

4.3.1 駒打による一手詰の判定

上記の (1) と (2) の組み合わせ (合わせて 16 bit) に関して，(1) のどこかに駒打を行い (2) を塞ぐのに必要な持駒の種類をあらかじめ求めて表に保存しておく．判定時にはその表と持駒を比較して，必要な持駒のどれもなければ駒打による一手詰みは存在しない．**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様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，右の表の一行目の様に金を所持している必要がある．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 (表二行目)．何を持っていても詰まない場合には，表三行目のように表を引くだけで判定される．

必要な持駒のどれかを所持していた場合は，駒打により利きを遮って詰まない例外[10] かどうかを盤面を見ながら判定する．その際も，駒の種類と打つ場所から自分の利きを塞ぎうる場所の表をあらかじめ求めておいて利用することで効率的に判定する．

4.3.2 移動による一手詰の判定

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) を全て塞げるかどうかをビット演算により求める．全て塞げる見込みがある場合には，移動により利きを遮って詰まない例外かどうか詳細な判定を行う．その際には，移動した駒の直接の利きのあるマスを直接求めておき，それと (3) の情報を元にビット演算を行い判定を効率化する．また，移動の場合は自殺 (移動により自玉が王手状態になる反則) に注意を払う必要がある．

4.4 一手後の証明数と反証数の評価関数 (H') の設計

一手詰判定関数により一手詰み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必要がある．単純には (1,1)，あるいは王手を生成して (1, 王手の数) とすることもできる．しかし，どちらの場合でも，全ての節点の証明数が 1 となり局面の詰みやすさを反映できない．また後者の場合は，王手を生成するコストがかかるという問題点がある．

そこで，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して，全ての王手のなかで証明数の最小値を求める関数 H' を作成した．この関数 H' は，全ての王手を考慮する点が，指手毎に証明数や反証数を予測する H とは異なる．

4.4.1 証明数の予測

攻方の手番での証明数は，各王手の後の局面に関して王手を受ける指手の数の最小値に相当する．ここで，王手を生成せずにそのような予測をするために，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の理想的な王手がかかった場合の逃げる場所の数を予測することとした．

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取る極めて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 (そのような着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) の組み合わせ (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) の組み合わせと盤の状況から，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: 全ての (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: 全ての (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: 全ての (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

一手詰判定関数では，駒打や移動により利きが遮られることで最小自由度が増える可能性を検査する必要があったが，この予測では admissible であ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挙げられる．しかし，そのような可能性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 (但し 0 の場合は 1) とした．これは有効 (取り返されない) 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実戦の局面を題材に詰探索の実験を行った．

5.1 GPS 将棋

提案手法の効果を測定するために，オープンソースプログラムである GPS 将棋[20] の詰将棋関数を題材に実験を行った．GPS 将棋の詰将棋関数は，df-pn に評価関数を組み合わせた df-pn$^+$ アルゴリズム[6] で，攻方の手番で玉の逃げられそうなマスの数を予測する (H)，ただ捨ての王手は展開順序を後にする (cost) などの基本的な評価が入っている．cost の具体的な値は**表 2** に掲載する．

5.2 実験に用いた局面

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，一つの棋譜からは最初に見つかった一つのみを採用した．その結果，1,000 棋譜から 777 局面を得た．この際，評価関数の影響を排除するために，局面の収集においては H や cost を使わずに，df-pn アルゴリズムを用いた．同様に，不詰の局面の例として，965 局面を収集した．不詰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が証明できず，10,000 節点以内の探索で不詰が証明できるという条件で収集した．

なお，どの設定の探索でも，df-pn には局面の優越関係，証明駒[13]，GHI 対策[4]，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(tm) Processor 252 (Turbolinux AMD64 8.0) のマシンで行った．

表 3 固定深さ探索の接点あたりの平均所要時間

深さ	H’	詰みのある局面	詰みのない局面
1	無	1381	1183
1	有	1738	1659
3	有	1590	1741

表 4 深さ固定探索が返す証明数と反証数の平均値

深さ	H’	詰みのある局面		詰みのない局面	
		証明数	反証数	証明数	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

5.3 固定深さ探索の性能

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，接点あたりの所要時間の平均値を求めた．探索深さ 1 及び 3 の結果を**表 3** に掲載する．なお単位はマシンサイクルである．

一手詰みがあるかどうかを判定するだけならば，2GHz の CPU ならば毎秒 200 万局面近く行うことができてかなり高速である．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 1.5 倍程度の時間がかかることが分かる．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かったため掲載しなかった．

5.4 評価関数 H’ の性能

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，深さ固定探索が返す証明数と反証数の平均値を求めた．**表 4** 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ことが分かる．

5.5 詰探索の性能の改善

続いて，詰探索の性能の改善を示すために，提案手法の有無，評価関数の有無の設定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，速度 (マシンサイクル数) と局面表に登録した局面の数などを測定し，平均を取った．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．

5.5.1 詰みのある局面

まず，詰みのある局面に対しては，**表 5** 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

表 2 GPS 将棋の df-pn$^+$ における cost

	と	成香	成桂	成銀	馬	龍	金	歩	香	桂	銀	角	飛
攻方	2	4	4	4	8	8	8	1	4	4	4	6	6
受方	0	-2	-2	-2	-3	-3	-2	0	-1	-1	-2	-2	-2

表 5 評価関数を無効にして集めた 777 局面に対する探索効率

手法	所要時間 ($\times 10^4$)	節点数 (df-pn)	節点数 (局面表)	失敗
df-pn	3869	7029	5429	1
df-pn / a1	2312	4370	3573	
df-pn / a3	(*) 1205	1355	(*) 906	
df-pn / d2	1382	2008	1680	
df-pn / a1 + p2d2	2562	4500	3994	1
df-pn / a3 + p2d2	1294	1224	880	
df-pn / d2 + p2d2	1426	2014	1725	
df-pn$^+$	1540	2987	1755	1
df-pn$^+$ / a1	1202	2284	1362	1
df-pn$^+$ / a3	1499	1590	(*) 710	
df-pn$^+$ / d2	(*) 1004	1608	1014	
df-pn$^+$/ a1 +p2d2	1132	1936	1348	
df-pn$^+$/ a3 +p2d2	1425	1379	745	
df-pn$^+$/ d2 +p2d2	958	1427	1060	

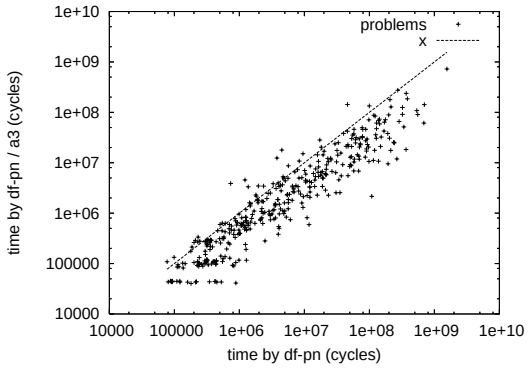

図 2 df-pn / a3 と df-pn の所要時間の散布図

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全ての手法で解けた 775 題を対象とした．解けなかった問題の数は各手法毎に表の「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 (*) で印した．なお，平均所要時間は桁が大きいため一万で割った値を記載している．

5.5.1.1 固定深さの探索の効果

各行がアルゴリズムに対応し，初めの “df-pn” はオリジナルのアルゴリズムである．次の “df-pn / a1” は提案手法であり，攻方の末端設定で 1 手読みを行うように拡張した df-pn アルゴリズムである．同様に，“df-pn / a3” は攻方の局面で 3 手読みを，“df-pn / d2” は受方の局面で 2 手読みを行う．

提案手法は，どれもオリジナルの df-pn より所要時

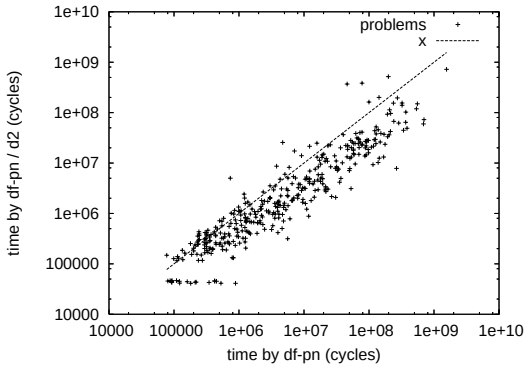

図 3 dfpn / d2 と df-pn の所要時間の散布図

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 6 倍近いの効率化が達成されていることが読みとれる．**図 2** 及び**図 3** 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．どちらの図でもほとんどの問題が $y = x$ 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．

5.5.1.2 閾値拡張の効果

次の三行の “p2d2” がついたものは，子節点の平均値に基づく閾値の拡張[2] を有効にしたものである．但し，無制限に閾値を拡張すると問題によっては遅くなる場合が目立ったため，拡張の上限を 2 で制限して実験を行った．この拡張は，この問題セットに対しては，それほど目立った効果はなく，多少の改善または若干の副作用となっている．

5.5.1.3 評価関数 (H, cost) の影響

次の七行は，5.1 節で説明した GPS 将棋の評価関数 (H と cost) を併用したものである．実験結果から，評価関数を用いると探索はさらに効率化されることが分かる．節点数は攻方の末端節点で 3 手読みを行う “df-pn$^+$ / a3” が最も効率的であるが，平均探索時間は受方の末端節点で 2 手読みを行う “df-pn$^+$ / d2” の方が速い．また，閾値の拡張 (p2d2) は適用しない場合と比較して，安定して効率が良くなっている．

5.5.2 詰のない局面

表 6 に，不詰の局面で実験した結果を掲載する．この実験では，どの手法も 965 局面全てを不詰と判定した．所要時間は，攻方で 1 手読みを行う手法 (a1) が効

表 6 不詰の 965 局面に対する探索効率

手法	所要時間 ($\times 10^4$)	節点数 (df-pn)	節点数 (局面表)	失敗
df-pn	1607	3383	2171	
df-pn / a1	(*) 1493	3216	2067	
df-pn / a3	2569	4060	(*) 1497	
df-pn / d2	2108	3554	2386	
df-pn / a1 + p2d2	1530	3019	2189	
df-pn / a3 + p2d2	2392	3324	1465	
df-pn / d2 + p2d2	2326	3742	2429	
df-pn$^+$	1693	4106	1571	
df-pn$^+$ / a1	(*) 1674	4022	1533	
df-pn$^+$ / a3	2526	4569	(*) 1219	
df-pn$^+$ / d2	2002	4100	1510	
df-pn$^+$/ a1 + p2d2	1617	3394	1558	
df-pn$^+$/ a3 + p2d2	2313	3741	1241	
df-pn$^+$/ d2 + p2d2	1835	3506	1540	

表 7 df-pn$^+$/ a1 で収集した 805 局面に対する探索効率

手法	所要時間 ($\times 10^4$)	節点数 (df-pn)	節点数 (局面表)	失敗
df-pn	1669	2988	1730	3
df-pn / a1	1418	1416	(*) 645	4
df-pn / a3	1250	1965	1329	1
df-pn / d2	(*) 1034	1487	1060	1
df-pn / a1 + p2d2	3448	6275	4958	3
df-pn / a3 + p2d2	1864	2613	2116	1
df-pn / d2 + p2d2	1935	1858	1176	2
df-pn$^+$	1136	2077	1233	
df-pn$^+$ / a1	(*) 1058	1602	995	
df-pn$^+$ / a3	1249	1161	(*) 626	
df-pn$^+$ / d2	5896	9679	7322	
df-pn$^+$/ a1 +p2d2	1825	2007	1258	
df-pn$^+$/ a3 +p2d2	3493	5890	5083	
df-pn$^+$/ d2 +p2d2	1864	2549	2084	

表 8 df-pn$^+$/ a3 で収集した 782 局面に対する探索効率

手法	所要時間 ($\times 10^4$)	節点数 (df-pn)	節点数 (局面表)	失敗
df-pn	6445	11119	8752	8
df-pn / a1	4133	7362	5946	1
df-pn / a3	(*) 1722	1824	(*) 1220	1
df-pn / d2	2313	3171	2728	
df-pn / a1 + p2d2	4198	7196	6240	1
df-pn / a3 + p2d2	2264	2029	1413	1
df-pn / d2 + p2d2	2334	3143	2647	1
df-pn$^+$	1544	2850	1723	1
df-pn$^+$ / a1	1288	2111	1345	
df-pn$^+$ / a3	1166	1272	(*) 632	
df-pn$^+$ / d2	(*) 994	1502	1007	
df-pn$^+$/ a1 + p2d2	1222	1943	1422	
df-pn$^+$/ a3 + p2d2	1138	1036	606	
df-pn$^+$/ d2 + p2d2	1051	1491	1149	

表 9 df-pn$^+$/ d2 で収集した 789 局面に対する探索効率

手法	所要時間 ($\times 10^4$)	節点数 (df-pn)	節点数 (局面表)	失敗
df-pn	3869	7029	5429	8
df-pn / a1	2312	4370	3573	7
df-pn / a3	(*) 1205	1355	(*) 906	
df-pn / d2	1383	2008	1680	1
df-pn / a1 + p2d2	2562	4500	3994	5
df-pn / a3 + p2d2	1294	1224	880	
df-pn / d2 + p2d2	1426	2014	1725	2
df-pn$^+$	1540	2987	1755	
df-pn$^+$ / a1	1202	2284	1362	1
df-pn$^+$ / a3	1490	1590	(*) 710	
df-pn$^+$ / d2	(*) 1004	1608	1014	
df-pn$^+$/ a1 + p2d2	1132	1936	1348	1
df-pn$^+$/ a3 + p2d2	1425	1379	745	
df-pn$^+$/ d2 + p2d2	958	1427	1060	

果的であった．このことは，一手詰判定関数の利用は不詰の証明にも役に立つことを示している．また，評価関数 (H, cost) を使わない方が若干効率が良かった．これは，5.2 節で説明した局面の収集の際に，評価関数を用いなかったことが原因として考えられる．

5.5.3 様々な設定で収集した詰みのある局面

最後に，提案手法の中でどの手法が最も効果的であるかを調査するために，局面を変えて実験した結果を示す．前節までで使用していた局面は，5.2 節で説明したように評価関数を使わない df-pn で収集したが，本節の実験ではそれに替えて，df-pn$^+$/ a1, df-pn$^+$/ a3, df-pn$^+$/ d2 の 3 種類の手法で収集した局面を用いた．その際に，使用する棋譜に重なりがないようにしている．それらの局面に対する探索の結果を，**表 7**, **表 8**, **表 9** に示す．平均値は今まで同様に，全ての手法で解けた局面に対して集計し，それぞれ 797 題，774 題，775 題であった．

df-pn$^+$/ a1 で収集した局面 (表 7) に対しては，df-pn$^+$/ a1 または df-pn / d2 の効率が良く，df-pn$^+$/ d2 の効率が特に悪いという結果となっている．また，閾値の拡張 (p2d2) はほとんどの手法で性能が悪化している点がこの問題セットの特徴的なところである．

df-pn$^+$/ a3 で収集した局面 (表 8) と，df-pn$^+$/ d2 で収集した局面 (表 9) に対しては，df-pn$^+$/ d2 が最も速く，メモリ使用量の観点からは，df-pn$^+$/ a3 が最も良い．また，評価関数の利用 (H, cost) はほとんどの手法で性能が向上している．

総合すると，df-pn$^+$/ a1 は不詰以外の全ての問題セットに対して，オリジナルの df-pn より成績が向上しており，有効であると言える．df-pn$^+$/ d2 及び df-pn$^+$/ a3 は全体として df-pn$^+$/ a1 よりもさらに効率が良いが，どちらが良いかは問題セットにより異なる．メモリ使用量の観点からは安定して df-pn$^+$/ a3 が優秀である．また，df-pn$^+$/ d2 は有力な手法であるが，df-pn$^+$/ a1 で収集した局面に対して効率が悪く，この性質が何に由来するかは今後検討する必要がある．

6. おわりに

詰探索を効率化するために，現在最良のアルゴリズムと言われている df-pn アルゴリズムに組み合わせて，末端で 1 手から 3 手の固定深さの探索を行うことを提案した．df-pn は探索節点数の観点では極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，末端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利用することで，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．

実験の結果，df-pn で求めた棋譜に表れる詰み局面に対して，平均探索時間がオリジナルの df-pn の約 $\frac{1}{3}$，局面表に登録する節点の数は平均約 $\frac{1}{5}$ という大きな効率の向上を実現したことを示した．また，評価関数 (H) を使う df-pn$^+$ に対しても，受け方の手番の末端で 2 手読みを用いることで，df-pn$^+$ の約 $\frac{2}{3}$ という効率の向上を実現したことを示した．

最後に，提案した拡張のうちどれが最良かを調べるために，棋譜に表れる詰み局面の収集方法を変化させたところ，一手読みの拡張は安定して df-pn より優れていたが，二手読み三手読みの拡張に関しては，問題により一手読みより良い場合と悪い場合があることが分かった．

参考文献

1) L. V. Allis, M. van der Meulen, and H. J. van den Herik. Proof-number search. *Artificial Intelligence*, 66:91–124, 1994.
2) A. Kishimoto. *Correct and Efficient Search Algorithms in the Presence of Repetitions*. PhD thesis, University of Alberta, Mar. 2005.
3) A. Kishimoto and M. Mueller. Df-pn in go: An application to the one-eye problem. In *Advances in Computer Games 10*, pp. 125–141. Kluwer Academic Publishers, 2003.
4) A. Kishimoto and M. Mueller. A solution to the ghi problem for depth-first proof-number search. In *7th Joint Conference on Information Sciences (JCIS2003)*, pp. 489–492, 2003.
5) A. Nagai. A new and/or tree search algorithm using proof number and dispr oof number. Complex Games Lab Workshop, 1998.
6) A.Nagai and H.Imai. Application of df-pn$^+$ to Othello endgames. In *Game Programming Workshop in Japan '99*, pp. 16–23, Oct. 1999.
7) M.Seo, H.Iida, and J.W. Uiterwijk. The pn*-search algorithm: Application to tsume-shogi. *Artificial Intelligence*, 129(1-2):253–277, 2001.
8) 柿木. 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, 17, 2005.
9) 久米. 将棋倶楽部 24 万局集. ナイタイ出版, 2002.
10) 山下. YSS–そのデータ構造，およびアルゴリズムについて. 松原（編）, コンピュータ将棋の進歩 2, 第 6 章, pp. 112–142. 共立出版, 1998.
11) 野下. 詰将棋を解くプログラム T2. 松原（編）, コンピュータ将棋の進歩, 第 3 章, pp. 50–70. 共立出版, 1996.
12) 脊尾. 共謀数を用いた詰将棋の解法. 松原（編）, コンピュータ将棋の進歩 2, 第 1 章, pp. 1–21. 共立出版, 1998.
13) 脊尾. 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, Oct. 1999.
14) 佐野, 橋本, 長嶋, 飯田. 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, Nov. 2004.
15) 田中, 飯田, 小谷. 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.
16) 三輪. Svm を用いた将棋の詰みの予測とその応用. Master's thesis, 東京大学新領域創成科学研究科基盤情報学専攻, Feb. 2005.
17) 三輪, 横山, 近山. Svm を用いた将棋の詰みの予測とその応用. 第 9 回ゲームプログラミングワークショップ, pp. 143 – 150, Nov. 2004.
18) 金子, 田中, 山口, 川合. 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.
19) 金子, 田中, 山口, 川合. 効率的な詰将棋探索のための評価関数. 第 11 回 ゲーム情報学研究会, pp. 3–8, 2004.
20) 田中, 副田, 金子. 高速将棋ライブラリ OpenShogiLib の作成. 第 8 回ゲームプログラミングワークショップ, Nov. 2003.
21) 長井, 今井. 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, 43(6):1769–1777, 2002.