詰将棋における df-pn$^+$ 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

金子知適[†] 田中哲朗[††]
山口和紀[††] 川合慧[†]

本稿では，実戦での詰将棋探索を効率的にすることを目的に，将棋の棋譜にあらわれる局面を対象に詰みやすさを予測する評価関数について提案し，df-pn$^+$ 探索と組み合わせて効果を確認した．詰将棋では df-pn 探索が効果的であることが知られているが，それに評価関数を組み合わせた df-pn$^+$ 探索は，評価関数設計の難しさからあまり使われていなかった．そこで，本稿では機械的な方法で評価関数を作ることを試みた．具体的には，df-pn$^+$ 探索の評価関数の一つである，局面の証明数と反証数の初期値を設定する関数に着目し，その関数の調整方法として，訓練例の局面について df-pn 探索で指定のノード数を探索した後の証明数と反証数をあらかじめ求め，評価関数にそれらの値を予測させる方法を提案した．

実際に，何種類かの特徴量を用いた評価関数について，実戦の棋譜に表れた約 30 万局面を用いてパラメータを最小二乗法手法により自動的に調整し，別の実戦に表れた約千局面を対象に実験を行った．その結果，作成した評価関数を用いた df-pn$^+$ 探索は，評価関数を用いない df-pn 探索に比べて，効果的に詰や不詰を発見できることを確認した．最も良い評価関数の場合，平均して半分以下の探索ノード数で詰を発見した．

Evalution Functions for df-pn$^+$ in Shogi based on Prediction of Proof and Disproof Numbers after Expansion

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

Evaluation functions for checkmate search in Shogi-game are proposed, that are suitable for df-pn$^+$ search. They estimate initial values of proof number and disproof number in a newly expanded state in the search. We trained evaluation functions so that they predict proof numbers and disproof numbers of a given state after expansion by normal df-pn search of specified number of nodes. The parameters in our evaluation functions were statistically determined by means of least mean squares, by using states appeared in real game records. Our experiments showed that df-pn$^+$ search combined with the proposed evaluation functions were actually more efficient than df-pn search without any evaluation function, for proving or disproving states appeared in real game records. With the best evaluation function, the number of node expanded was reduced to half on average.

1. はじめに

長編詰将棋を解けるようになるなど詰将棋解答プログラムは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．強い将棋プログラムを作るためには，何手も進んだ局面での長手数の詰を正確に認識する必要があり，そのためには探索中に何度も詰将棋探索を行う必要がある．しかし，現状の詰将棋解答プログラムの効率は充分ではないため，探索の末端付近ではごく短い詰しか読むことができていない．

本稿では，詰みやすさを評価する詰将棋専用の評価関数を作成することで，詰将棋の解答を効率化することを試みた．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[12]．もし詰みやすさを評価する良い評価関数があれば証明に不要な局面の探索を省くことができ，大幅に探索局面数 (探索コスト) を減らすことができると

† 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

いうアイデアである．

詰みやすさを評価する評価関数として，何ノードか展開した後の証明数や反証数を予測する関数を提案し，棋譜にあらわれた局面を訓練例に用いて評価関数のパラメータを自動的に調整した．詰将棋を解くすぐれた手法である df-pn 探索[15] に，作成した詰将棋用の評価関数を組み合わせたところ，棋譜から収集した詰む局面や詰まない局面に対して詰や不詰を証明するまでの探索ノード数を大幅に減らすことができた．

次節で関連研究について紹介した後，3 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 節でそれらを予測する評価関数について提案し，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 探索は，pn 探索[1] と同じ振舞いを保ちつつ深さ優先に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おり[15]，また，詰碁でも効果が確認されている[4]．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 探索における対策[5] も提案されている．また，大きな問題を解くために必要な局面表の GC[14] についても研究されている．

さらに，df-pn 探索に評価関数を組み合わせたアルゴリズムとして df-pn$^+$ 探索が提案されている[6]．なお，次の節で説明するが，df-pn 探索の評価関数は，通常の評価関数とは異なり専用のものである．その応用例としては，オセロの終盤で勝ち負けを探索する際に and-or 木と考えて df-pn$^+$ を用いると，df-pn 探索と比較して探索ノード数を約 $\frac{1}{6}$ にできたことが確認されている[6]．オセロでは終盤で信頼できる通常の意味の評価関数が存在している[3] ため，長井の研究ではそれを df-pn$^+$ 用の評価関数に変換している．一方，指将棋の終盤の評価関数はそれほど正確ではないため，本研究では指将棋の評価関数とは別に，df-pn$^+$ 用の評価関数を独自に設計した．

詰将棋においては，主に評価関数を設計することの難しさから df-pn$^+$ は一般にはそれほど広まっていない.「東大将棋」では駒得を考慮して証明数を予測している．☆金子らは証明木の大きさを教師例として評価関数を訓練する手法を提案しているが，探索効率の向上はそれほど大きくない[12]．このことは局面から詰不詰ならびにその証明木の大きさを予測することの難しさを示していると考えられる．本研究では予測する数値を，一定ノード展開した時点での証明数，反証数とすることによって，パラメータ調整を簡単にすることに成功した．

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，様々なアイデアが提案されているが，df-pn$^+$ 探索を用いたものや実戦の局面を対象に充分な量の実験を行い効果をあげた研究は著者らの知る限りではない．他の探索手法と組み合わた研究としては，着手の種類による評価や，玉の自由度，玉の危険度などの評価したものがある[9),11]．本稿の評価関数の特徴量を検討するにあたって参考にした．

他に詰将棋を解くアルゴリズムとしては，PN*探索[7),10] が早く考案され，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 探索の方が効率が良いとされている[15] ことと，探索手法と評価関数との組み合わせ方が既に議論されている[6] ことを考慮して．本稿では df-pn 探索を採用した．

3. 詰将棋探索

3.1 証明数と反証数

まず，df-pn 探索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．証明数と反証数はノード (状態) 毎に定義される値で次のような意味を持つ:[15]

証明数 あるノード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ノードの数の最小値.

反証数 あるノードが不詰であること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ノードの数の最小値.

この証明数と反証数は，次に展開するノードを決めるために用いられる．すなわち，ルートノードから攻方では証明数の小さいノードを，受方では反証数の小さいノードをたどった末端の局面が順次展開する．このアルゴリズムをそのまま実装すると最良優先探索の一種となり，pn 探索と呼ばれる．一方，df-pn 探索は，pn 探索を反復深化の技術を使って深さ優先に変更したものであるが，展開されるノードは pn 探索と同じになるように設計されている．

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という問題があるが一般的な解決法はま

☆ ACG03 での岸本氏との個人的な会話による．

表 1 証明数・反証数の計算方法 (df-pn)

ノードの種類		証明数	反証数
先端	詰	0	∞
	不詰	∞	0
	不明	1	1
内部	攻方	min(子ノードの証明数)	$\sum$(子ノードの反証数)
	受方	$\sum$(子ノードの証明数)	min(子ノードの反証数)

だない．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[4] が必要である．

3.2 評価関数の利用

一方，df-pn$^+$ 探索では 2 種類の評価関数を使い，証明数，反証数の計算方法を**表 2** のように変更する[6]．まず，先端ノードの (証明数，反証数) を (1,1) とする代わりに，ノード毎の予測値 (h_proof，h_disproof) を用いる)．以降この予測関数を h と表記する．また，各辺 (指手) に対して cost を設定し，ノードの証明数，反証数を計算する際に，子ノードの証明数，反証数にそれぞれ指手の cost_proof, cost_disproof を加えて用いる．この二つの値を設定する関数を cost と呼ぶ．指手の種類に応じて (cost_proof, cost_disproof) を設定することにより，駒をただで捨てる手は後回しにするなどの効果を実現することができる．

この二つ関数を本稿では評価関数と呼ぶ．ここで h の予測が正しければ，df-pn$^+$ 探索は，証明に必要なノード以外を一切展開しないため効率が良くなる．そのような評価関数 h を作ることを本研究の目標とした．次節で作成方法を具体的に説明する．もう一つの評価関数である cost は調整が難しいことが知られているため[6]，本稿では学習の対象にはしなかった．

4. 評価関数の設計とパラメータの調整

4.1 評価関数作成の枠組

ここでは評価関数 (h) の作成方法として，df-pn 探索で一定のノード数 (ps と表記する) を探索した後の証明数と反証数の予測をさせることを提案する．表 1 の定義からこの予測がうまくいけば，各リーフノードにつき ps ノードの探索の節約になるため，ps 倍の効率化が可能となる．また，このように設定するメリットとして，訓練例が簡単に準備できるため機械的なパラメータ調整が可能である点があげられる．展開する数 ps の適切な値は分からないため，10, 20, 40 の 3 通りを試した．

具体的には，以下の手順で評価関数を作成する．

(1) 訓練例用の局面を準備する

(2) 各局面について，以下の手順で適切な証明数と反証数のペアを計算する:

(a) df-pn 探索で探索し，指定のノード数 (ps) を展開したところで打ち切る．

(b) 表 1 の定義に従って証明数と反証数を計算する．

(3) 局面から，証明数と反証数のペアを予測するように，評価関数のパラメータを調整する．

表 3 駒の価値

歩	香	桂	銀	角	飛
100	400	400	550	800	950
馬	龍	金・他の成駒			
1150	1300	600			

簡単のために，評価関数は線形結合とし，また，証明数を予測する関数と反証数を予測する関数を別々に作成した．さらに，訓練例を準備する都合から本稿では受方の手番の評価関数のみを対象とした．続いて用いた特徴量と，訓練例について順に説明する．

4.2 特 徴 量

ここでは以下の 3 種類の特徴量を用いた．

4.2.1 24 近傍 (e)

- 玉の 24 近傍のそれぞれのマスについて守備駒の有無． 値は 0, 1．盤の外は 1.
- 玉の 24 近傍のそれぞれのマスについて攻撃側の利きの数．盤の外は 0.
- 玉の 24 近傍のそれぞれのマスについて守備側の利きの数．盤の外は 0.
- 駒の種類毎に攻撃側の持駒の数.
- 駒の種類毎に守備側の持駒の数.

4.2.2 改良版 24 近傍 (n)

- 玉の 24 近傍のそれぞれのマスについて駒の種類. 値は 0, 1.
- 玉の 24 近傍のそれぞれのマスについて攻撃側の利きの数．盤の外は 0.
- 駒の種類毎に攻撃側の持駒の数.
- 駒の種類毎に守備側の持駒の数.
- 玉の位置．値は 0, 1.
- 駒の損得．各駒の評価値は GPS 将棋[13] で使われているものである．具体的な値は**表 3** に掲載する．

4.2.3 改良版 24 近傍二次 (n2)

改良版 24 近傍 (n) の二次の項を取ったものである．但し同じ項目同士は二乗せずにそのままの値を用いた．これは駒の損得の値が正負に分布していて，二乗すると符号が失われてしまうためである．

4.3 局面の準備

パラメータ調整に用いる局面は，実戦の棋譜から集めた．これは実戦での詰将棋探索を効率化すること

表 2 証明数・反証数の計算方法 (df-pn$^+$)

ノードの種類		証明数	反証数
先端	詰	0	∞
	不詰	∞	0
	不明	h_proof	h_disproof
内部	攻方	min(子ノードの証明数+指手の cost_proof)	$\sum$(子ノードの反証数+指手の cost_disproof)
	受方	$\sum$(子ノードの証明数+指手の cost_proof)	min(子ノードの反証数+指手の cost_disproof)

表 4 ノード数制限の範囲で詰や不詰が反面した局面の数

ノード数制限	詰	不詰	全体
10	14898	7056	275994
20	15254	13781	301642
40	19568	19800	294012

表 5 指定ノード展開後の証明数と反証数の分布

ps	証明数		反証数	
	平均	分散	平均	分散
10	4.17	5.34	5.50	10.49
20	5.64	20.14	9.46	51.13
40	8.71	98.42	13.60	115.22

を目的としているためである．具体的には将棋倶楽部 24☆1で指された 40,000 棋譜を使用した[8]．棋譜に表れる局面のうち，王手のかかっている局面を対象として，df-pn を用いて詰将棋探索を行い，ps ノード展開後の証明数と反証数を用いた．その際，連続した局面で同じ証明数，反証数を持つ局面は，玉の周囲が同じ状況である可能性が高いため，最初の一つのみを採用した．詰将棋探索では，df-pn アルゴリズムに，優越関係の利用[15]，サイクル対策[4]，GHI 対策[5] を組み込んだものを用いた．また詰将棋特有の工夫として，中合の後回しや，飛角歩が成らない指手は打歩詰以外は読まないなどをいれている．☆2また，初手から 50 手までの局面は詰が少なく，また重複も多いと予想されるため使用しなかった．

証明数，反証数の計算にあたって指定のノード数 (ps) を展開する前に詰が判明した場合は，便宜的に (証明数，反証数) = (1, ps) として扱った．同様に不詰の場合も，(ps, 1) として扱った．訓練例の局面では，そのような局面は少なかった．☆3**表 4** に具体的な数値を掲載する．また，詰，不詰が判明しなかった局面で証明数や反証数が ps を越えた場合は ps で打ち切った．展開後の証明数と反証数のヒストグラムを**図 1** 及び**図 2** に示す．重ならないために ps 毎に少しずらして描いている．また平均と分散を**表 5** に示す．展開ノード数

☆1 http://www.shogidojo.com/

☆2 プログラムは将棋ライブラリの一部として利用可能である[13]．

☆3 なお，最終的に詰む局面と不詰の局面の割合では後者の方が多かった．

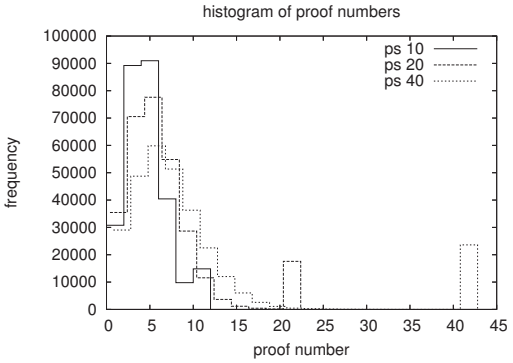

図 1 展開後の証明数の分布

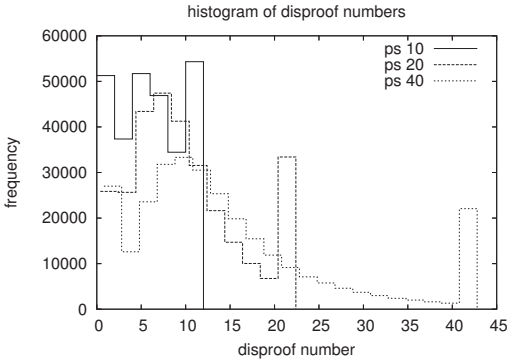

図 2 展開後の反証数の分布

(ps) が多いほど，平均，分散ともに上昇している．また，証明数と反証数を比較すると，反証数の方が分散が平均，分散ともに大きい傾向がある．

5. 評価関数の効果

5.1 パラメータの調整結果

前節で説明した評価関数について，それぞれの線形結合中の重みを最小二乗法で推定した．実際の計算には，共役勾配法[2] を用いた．☆4

調整の結果を**表 6** に掲載する．表の ps は展開するノード数，r は相関係数，mse は平均自乗誤差を表す．

☆4 http://netlib2.cs.utk.edu/linalg/html_templates/Templates.html

表 6 パラメータ調整の結果

評価関数 (h)		証明数		反証数	
特徴量	ps	r	mse	r	mse
e	10	0.64	3.0	0.67	5.7
	20	0.58	13.4	0.63	25.1
	40	0.51	76.5	0.64	68.3
n	10	0.70	2.6	0.70	5.3
	20	0.63	12.3	0.67	22.7
	40	0.56	70.1	0.68	62.0
n2	10	0.81	1.9	0.81	1.9
	20	0.76	9.1	0.78	18.0
	40	0.71	52.1	0.82	46.4

表から証明数，反証数とも，e, n, n2 の順に正確さが向上し，特徴量の種類を増やすことが効果的であることが読みとれる．また，展開するノード数が増えるほど予測が難しいことが確認できる．

5.2 詰将棋探索での効果

5.2.1 使用したテスト例と評価関数

訓練例とは別の棋譜から収集した 1024 局面でテストした．収集方法は訓練例と同様に，王手のかかっている局面を選び，40 ノードを探索した時点で同じ証明数，反証数になるような連続した局面は除いた．

評価関数は h については，前節で説明したものと，評価関数がない場合に相当する必ず 1 を返す関数 (null と表記) を用いた．本稿では受方の手番の h のみ作成したために，攻方の手番については (証明数, 反証数) の初期値を (1, 1) とした．また，cost については，評価関数がない場合に相当する必ず 0 を返す関数 (null と表記) と GPS 将棋[13] で用いているヒューリスティックなコスト (piece と表記) の 2 種類を用いた．GPS 将棋のヒューリスティックは，攻方の場合は駒をただで捨てる手にコストをかけることで後回しにし，受方の場合はただで取れて駒に負のコストをかけることで優先的に読むものである．具体的な値は**表 7** に掲載する．

5.2.2 探索ノード数の分析

評価関数の種類を変えながら df-pn$^+$ 探索で展開する局面を 40 万ノードまでに制限する条件で探索したところ，詰と不詰が判明した局面の数は**表 8** の結果となった．評価関数を用いた場合には，用いなかった場合と比べて，詰や不詰が判明した局面がやや増えている．

続いて，どの評価関数の組み合わせでも詰や不詰が判明した局面について，探索ノード数を測定して比較した．結果を**表 9** に示す．表で uniq はユニークな探索局面の数を，total は重複した局面も含めて探索したノード数を意味する．各問題毎の測定値に関して，その平均値を記載した．上半分が cost を使わない場合，下半分が使う場合である．

表 8 テスト局面の詰と不詰

評価関数			詰	不詰	不明
h		cost			
null		null	287	668	45
e	10	null	289	674	37
n	10	null	290	671	39
n2	10	null	289	671	40
e	20	null	288	670	42
n	20	null	289	669	42
n2	20	null	288	671	41
e	40	null	287	667	46
n	40	null	288	665	47
n2	40	null	290	666	44
null		piece	289	667	44
e	10	piece	290	671	39
n	10	piece	290	671	39
n2	10	piece	291	671	38
e	20	piece	292	671	37
n	20	piece	291	670	39
n2	20	piece	291	673	36
e	40	piece	289	668	43
n	40	piece	291	667	42
n2	40	piece	293	668	39
(共通)			284	658	

表 9 探索ノード数の比較

評価関数			詰		不詰	
h		cost	uniq	total	uniq	total
null		null	7913.7	10919.2	7979.4	13004.4
e	10	null	4366.8	6486.9	5323.9	12528.8
n	10	null	4098.1	6070.5	5132.5	12152.9
n2	10	null	2854.9	4364.7	5220.9	12492.6
e	20	null	3786.7	5680.1	5138.2	13500.3
n	20	null	3447.9	5116.5	4982.2	13002.3
n2	20	null	2799.5	4247.2	4905.0	13319.3
e	40	null	4284.4	6702.8	5367.6	15930.9
n	40	null	3832.7	6043.3	5337.9	15741.5
n2	40	null	3356.6	5353.7	5261.7	15767.2
null		piece	3900.1	6916.4	6124.6	15200.4
e	10	piece	2664.6	4834.2	4468.3	12661.8
n	10	piece	2419.8	4393.8	4459.7	12641.3
n2	10	piece	2186.8	4024.4	4255.7	12075.3
e	20	piece	2408.6	4185.9	4460.7	13312.2
n	20	piece	2294.5	4066.4	4111.3	12317.2
n2	20	piece	1899.4	3398.4	4046.5	12379.5
e	40	piece	2371.8	4162.4	4577.6	14528.1
n	40	piece	2364.5	4288.2	4406.6	14208.2
n2	40	piece	2120.5	3819.2	4379.4	14129.6

全体として，評価関数を使う方が使わない場合と比べて良い成績となっている．特に詰の場合で，20 ノードの展開予測を行う n2 の評価関数が，h を使わない場合と比べて半分以下のノード数で詰を発見している．一方，不詰の場合では，ユニークな局面数は減るものの全体の探索ノード数の減少は詰の場合よりも少なく，評価関数によっては逆に増えているものもある．これは繰り返し探索する局面の割合が高まっていることを意味するが，原因については今後の分析が必要である．

表 7 GPS 将棋のコスト

	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

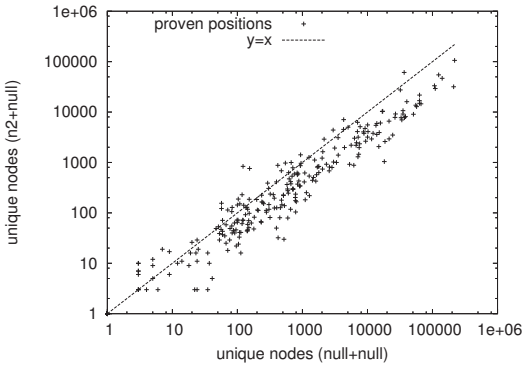

図 3 評価関数 (n2,ps=20) の有無による探索局面数 (詰)

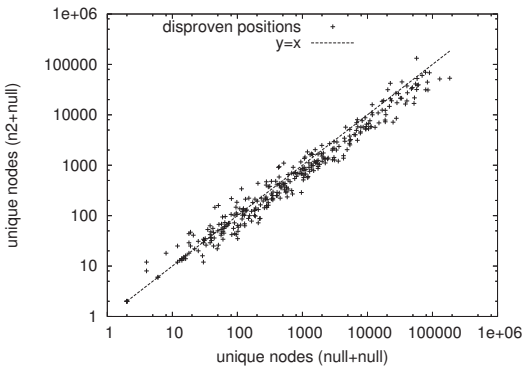

図 4 評価関数 (n2,ps=20) の有無による探索局面数 (不詰)

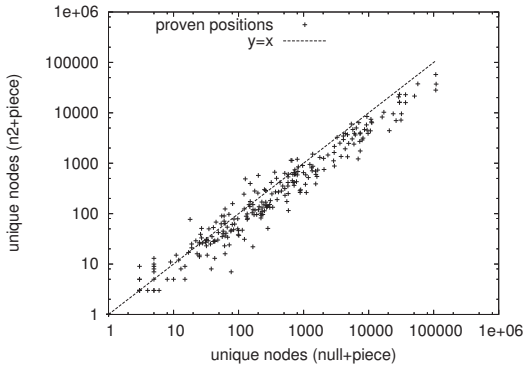

図 5 評価関数 (n2,ps=20) の有無による探索局面数 (コスト利用) (詰)

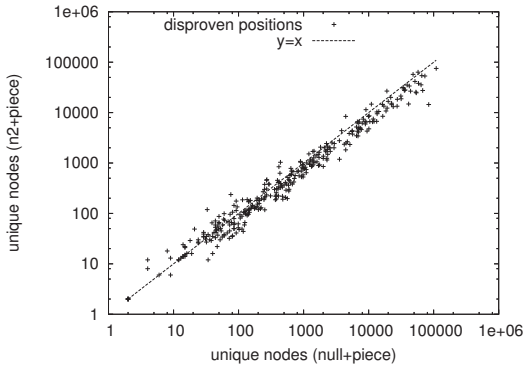

図 6 評価関数 (n2,ps=20) の有無による探索局面数 (コスト利用) (不詰)

ヒューリスティックな cost との組み合わせに関しては，cost を用いない場合でも組み合わせた場合でも，作成した評価関数 h はうまく働くことが確認できた．

さらに，もっとも良かった評価関数 n2 (ps=20) について，評価関数を使わない場合と比較するために，各局面についての探索ノード数の散布図を描いた．**図 3** と**図 4** がコストを使わない場合で，**図 5** と**図 6** がコストを使う場合である．横軸が評価関数を使わない場合の探索ノード数で縦軸が使う場合のものである．両軸とも対数で描いた．コストを利用した場合も，利用しない場合も，直線 y=x の右下への分布が多いために，全体的に探索ノード数が減少していることを確認できる．

5.3 各評価関数の探索速度

続いて，評価関数を用いた場合の探索速度について報告する．まず 1 局面あたりの探索速度を測定した．

表 10 評価関数の種類と詰将棋探索の速度 (clocks/position)

評価関数	速度
e	8247.0
n	9353.5
n2	32405.5
null	6152.7

テスト例の中の一題を例題に，各評価関数を用いて探索した時の詰を発見するまでの所要時間を測定し，探索した局面数で割ることによって，1 局面あたりの所要時間をクロック数を求めた．総所要時間には詰将棋内部で呼ばれるシミュレーションにかかる時間が含まれるため正確な速度ではないが，シミュレーションはそれほど使われないためだいたい近いと考えられる．総探索局面は，評価関数により異なるが 10 万局面前後であった．測定には，Opteron 2.2GHz (Turbolinux AMD64 8.0) のマシンを用いた．結果を**表 10** に示す．

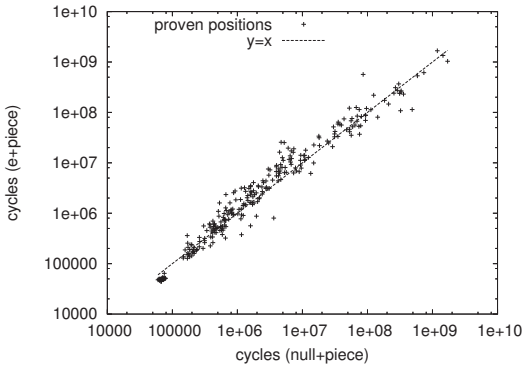

図 7 評価関数 (e) の有無による探索時間 (詰)

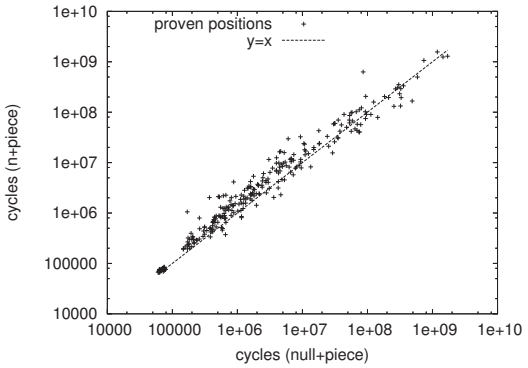

図 8 評価関数 (n) の有無による探索時間 (詰)

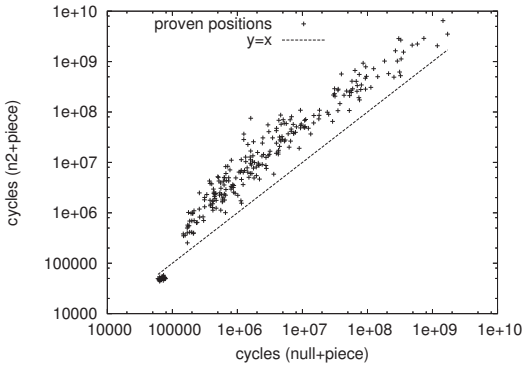

図 9 評価関数 (n2) の有無による探索時間 (詰)

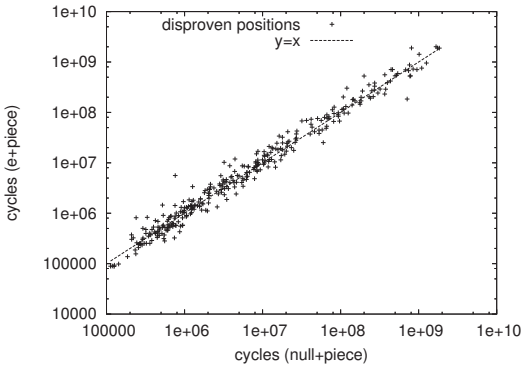

図 10 評価関数 (e) の有無による探索時間 (不詰)

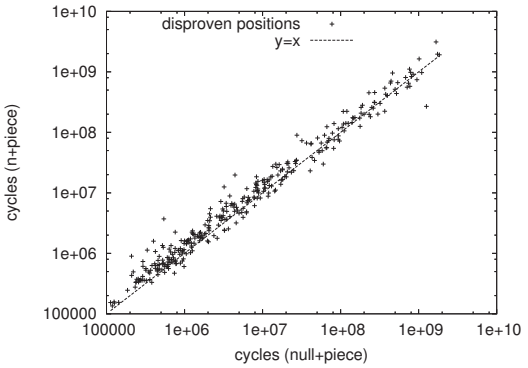

図 11 評価関数 (n) の有無による探索時間 (不詰)

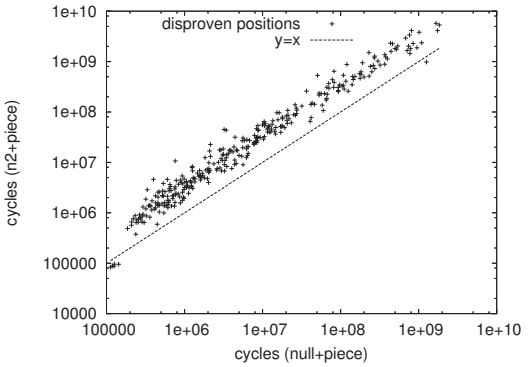

図 12 評価関数 (n2) の有無による探索時間 (不詰)

測定結果から，評価関数を用いた場合の速度の低下は，評価関数を用いない場合に比べて 1.3 倍から 5 倍程度で抑えられることが分かった．

続いて，詰，不詰を証明するまでの探索時間全体について調べるため，評価関数の有無による探索時間の散布図を描いた．評価関数 h については，前節の実験で成績が良かった ps=20 の，e, n, n2 の 3 種類を扱った．また cost を利用した方が成績が良いため，cost を利用する場合のみ対象とした．横軸が評価関数を使わない場合の探索時間で縦軸が使う場合の時間をクロックで測定したものである．両軸とも対数で描いた．

評価関数 e, n については，詰，不詰どちらの場合でも，ほぼ直線 y=x の近くに分布しており，ほぼ同じ時間がかかっていると言える．これは探索ノード数の減

少による効果と 1 ノードあたりの探索時間の増大したことの影響が相殺していることを示している．評価関数 n2 については，詰，不詰どちらの場合でも，一部の例外を除いて直線 y=x よりもはっきり左上に分布しており，評価関数を用いた方が探索時間がかかるという結果になった．これは探索ノード数の減少の効果を，1 ノードあたりの探索時間の増大したことの影響が上回ったことを示している．

現在の実装は重みに浮動小数を用いるなど最適化が充分でないため，評価関数 e, n については，より効率的な実装を行い速度を向上させることで実戦での利用に充分実用的な評価関数にすることができると考えている．また，評価関数 n2 については速度にして 5 倍程度の開きがあるため実戦での活用は難しいが，探索ノード数の削減効果が大きいことから難しい問題や長編詰将棋などメモリの制限が厳しい場合の探索に有効であると考えられる．

6. おわりに

本稿では詰将棋探索を効率的に行うための評価関数を考案し，df-pn$^+$ 探索と組み合わせた結果を報告した．評価関数の作成方法としては，評価関数を用いない df-pn 探索で指定の数のノードを展開した時の証明数と反証数を予測する方法を提案し，実際に評価関数を作成した．評価関数のパラメータは，実戦の棋譜に表れた局面を訓練例として調整した．

そして，訓練例とは別の棋譜に表れた詰，不詰の局面に対して探索を行い，詰及び不詰を証明する効率を調べたところ，作成した評価関数を用いた探索では，用いない場合と比較して探索局面数にして 2 倍程度効率が向上することが確認された．

今後の課題としては，評価関数の評価速度の向上のための効率的な実装や特徴量の選別があげられる．また，今回は受方の手番のための評価関数のみを扱ったが，攻方の手番の評価関数を作成すればさらに探索を効率化できると予想される．さらに本稿では実戦での詰将棋にテーマを絞ってパラメータ調整を行なったが，長編詰将棋のための評価関数についてはまだ研究がない．実戦用の評価関数がそのまま適用できるのか，あるいは専用の評価関数が必要かどうかは興味深いテーマである．

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) R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. V. der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. SIAM, Philadelphia, PA, 1994.

3) M. Buro. Improving heuristic mini-max search by supervised learning. *Artificial Intelligence*, 134(1–2):85–99, Jan. 2002.

4) 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.

5) 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.

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) 久米. 将棋倶楽部 24 万局集. ナイタイ出版, 2002.

9) 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.

10) 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.

11) 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.

12) 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.

13) 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.

14) 長井, 今井. 詰将棋を解くプログラムにおける効率的なハッシュの利用法について. アルゴリズム研究会, No. 79, pp. 21–26. 情報処理学会, 2001.

15) 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.