一手詰み判定関数の実装および終盤探索への応用

佐野 晶彦[1], 橋本 剛[2], 長嶋 淳[3], 飯田 弘之[4,5]

[1] 静岡大学情報学研究科 [2] 静岡大学工学部 [3] 静岡大学理工学研究科 [4] 静岡大学情報学部
[5] 科学技術振興事業団さきがけ研究21「機能と構成」領域

E-mail: {cs0042, hasimoto, cs8066, iida}@cs.inf.shizuoka.ac.jp

概要

コンピュータ将棋では終盤の探索において詰みの有無がとても重要となる．近年，詰み探索は目覚ましい進歩を遂げているがまだまだ処理に時間がかかりすべてのノードに対して行っているわけにはいかない．そのため，詰み探索のできなかったノードに一手詰みがあった場合それを見逃してしまい読みの質に問題が生じてしまう．そこで，この問題に対処しかつ高速な探索を実現させるため一手詰み判定関数を用いて詰み探索を行った．その結果，末端ノードで制御を加えて一手詰み判定関数を使用することにより終盤用問題をより多く解くことができた．また，詰み探索のプロセスに一手詰み判定を加えることでより高速な詰み探索を実現させることができた．

An enhancement for endgame search using one ply mating search

Akihiko Sano [1], Tsuyoshi Hashimoto [2], Jun Nagashima[3], and Hiroyuki Iida[4,5]

[1] Graduate School of Information, [2] Faculty of Engineering,
[3] Graduate School of Science and Engineering,
[4] Faculty of Information, Shizuoka University
[5] "Information and Systems", PRESTO, Japan Science and Technology Corporation

abstract

In Computer Shogi, the existence of mate becomes very important in endgame search. Although mating search has accomplished remarkable progress in recent years, it can't carry out to all nodes because processing still takes time. Therefore, when there is one ply mate in the node which cannot search mate, a problem will arise in the quality of search in order to overlook it. Then, in order to cope with this problem and to make high-speed search realize, one ply mating search is used for mating search. Consequently, more problem for endgame was solved by using one ply mating search under control in the end node. Moreover, more high-speed mating search was created by adding one ply mating search to the process of mating search.

1 はじめに

コンピュータ将棋では主に終盤において詰み探索が行われている．詰み探索とはその名の通り詰みか否かを探索するものであり，各探索ノードで用いられる．ただし，処理が重く探索時間も限られているのですべてのノードに対して行っているわけにはいかない．そのため，思考制限時間の10分の1だけ探索する[1]，探索の深さが浅く，かつ玉が詰みやすいと推定された場合のみ探索する[2]などの方法がとられている．しかし，探索ノード数を減らしてしまうと一手詰みがあっても見逃してしまう恐れが生じる．そこで，これをできるだけ避けるため末端ノードでの一手詰み判定関数の使用を試み，その結果について考察する．また，詰み探索では当然高速に探索を行うことが必要となってくる．そこで，一手詰み判定関数の詰み探索本体への使用を試みる．しかし，この方法は既に有岡によって行われており[3]，良い結果が出なかったとされている．だが，いろいろと工夫してみることで良い結果を出せないか実証も兼ねて実験し，その結果と問題点について述べる．

2 一手詰み判定関数の実装

実装に関してはより速く一手詰みか否か判定することを主眼において行った．そのため，局面を進めずに判定できるようにした．また一手詰みをかける手（以後，一手詰み手）の統計をとり，一手詰み手が指される頻度の多いマス，一手詰み手になりやすい駒種から判定するようにした．統計の結果を表1に示す．また，判定にはビットパターンを使いより速く判定できるようにした[4]．具体的に説明すると，まずすべての駒種についてそれぞれの位置ごとのビットパターンを用意しておく．例えば，図1の0の位置おける金のビットパターンは11100001となる．金を0の位置に指すことで位置1，2，3，4に利きが生じるためである．そして相手玉近傍のビットパターンを求め，王手駒とのANDをとることで詰み(=0)か否かを判断することができる．図2では相手玉近傍のビットパターンは10001001となる．ただし，この方法では完全に詰みか否かを判定することはできない．例えば図3のような局面で銀を3ニに移動して王手をかける場合，ビットパターンによる方法では00001000（玉のビットパターン）× 11110111（銀のビットパターン）＝0であり一手詰みとなる．しかし，実際には銀が飛車の利きをさえぎっており1ニに逃げられるため一手詰みではない．そのため，明らかに一手詰みでない手の排除用に使用した．更に，時間がかかりかつ使用頻度の低い判定を後に回し使用頻度の高い判定を先に行うことで，明らかに一手詰みでない手の判定をすぐに終えられるようにした．具体的には次のような順序で判定する．

1. 一手詰み手を生成できるマスか（例，味方の駒がないか，相手の利きがないか等）
2. 移動できる駒があるか
3. その駒を移動することで王手がかかり，かつ相手玉が移動できなくなるか

また，一手詰み判定にかける時間を短縮するため一手詰みとなりやすい手のみ判定するようにしたバージョンも作成した．

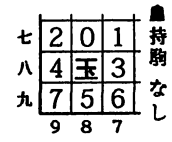

図1: 各ビットの位置

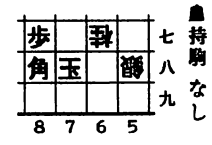

図2: 玉近傍の様子

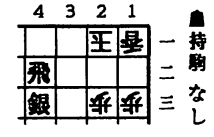

図3: ビットパターン判定の例外

表 1：一手詰み手の統計

相手玉から見た位置	王手種類	駒種	相手玉から見た方向	総数
近傍	打ち	金	真ん前	133995
近傍	打ち	金	右	87020
近傍	打ち	銀	真ん前	73138
近傍	打ち	金	左	61855
近傍	打ち	金	右前	59742
近傍	打ち	銀	右前	43962
近傍	打ち	金	左前	42428
近傍	打ち	銀	左前	34473
近傍	打ち	銀	右後ろ	31798
近傍	打ち	香	真ん前	27474
近傍	打ち	金	真後ろ	26190
近傍	移動	龍	右	23735
近傍	打ち	飛	真ん前	21572
近傍	打ち	飛	右	20532
近傍	移動	金	右	20241
近傍	打ち	飛	左	19812
近傍	移動	龍	右後ろ	19806
近傍	移動	馬	右	19333
近傍	移動	馬	左	16886
近傍	打ち	銀	左後ろ	16690
近傍	移動	と	右	16598
近傍	移動	龍	真後ろ	15691
近傍	移動	金	右前	15536
近傍	打ち	角	右前	15436
近傍	移動	金	真ん前	15023
近傍	移動	全	右	13676
近傍	移動	龍	左	13620
近傍	打ち	角	右後ろ	13175
近傍	移動	馬	右後ろ	12464
近傍	打ち	桂	左桂	12170
近傍	移動	歩	真ん前	11916
近傍	打ち	飛	真後ろ	11695
近傍	移動	龍	右前	11495
近傍	移動	と	左	11344
近傍	移動	馬	真ん前	10858
近傍	移動	龍	左後ろ	10651
近傍	移動	馬	右前	10411
近傍	移動	龍	真ん前	10397
近傍	移動	全	真ん前	9991
近傍	移動	全	右前	9928
近傍	打ち	角	左前	9746
近傍	移動	馬	真後ろ	9120
近傍	移動	金	左	9050
近傍	打ち	桂	右桂	8713
近傍	打ち	角	左後ろ	8554
近傍	移動	金	左前	8178
近傍	移動	龍	左前	7648
近傍	移動	と	真ん前	7573
近傍	移動	と	右前	7249
近傍	移動	圭	真ん前	6636
近傍	移動	馬	左前	6453
近傍	移動	と	左前	6446
近傍	移動	全	左前	6180
近傍	移動	銀	右前	5767
近傍	移動	馬	左後ろ	4848
近傍	移動	全	左	4697
近傍	移動	銀	真ん前	4551
近傍	移動	圭	右	4508
近傍	移動	圭	右前	4494
近傍	移動	全	真後ろ	4473
近傍	移動	圭	左	4446
近傍	移動	金	真後ろ	4243
近傍	移動	圭	左前	4045
近傍	移動	銀	左前	3735
近傍	移動	銀	右後ろ	3625
近傍	移動	銀	左後ろ	3563
近傍	移動	香	右	2353
近傍	移動	桂	左桂	2251
開き王手	移動	—	—	1817
近傍	移動	杏	左	1787
近傍	移動	桂	右桂	1777
近傍	移動	杏	左前	1639
近傍	移動	と	真後ろ	1348
近傍	移動	香	真ん前	1305
近傍	移動	杏	右前	1268
近傍	移動	飛	真ん前	1244
近傍	移動	杏	真ん前	1181
近傍	移動	圭	真後ろ	1172
遠方	移動	龍	右	760
遠方	移動	龍	左	631
近傍	移動	飛	右	339
近傍	移動	杏	真後ろ	270
近傍	移動	飛	左	269
近傍	移動	角	右前	200
近傍	移動	角	左前	129
近傍	移動	角	右後ろ	101
遠方	移動	馬	右後ろ	69
遠方	移動	龍	真後ろ	65
遠方	移動	角	左前	47
遠方	移動	馬	左後ろ	41
近傍	移動	飛	真後ろ	40
遠方	移動	馬	右前	29
遠方	移動	馬	左前	17
遠方	移動	角	右前	11
近傍	移動	角	左後ろ	6
遠方	移動	飛	真ん前	6
遠方	移動	龍	真ん前	5
遠方	移動	香	真ん前	4
			計	863943

3 一手詰み判定関数を用いた終盤探索

詰み探索はとても重い処理であるため，すべてのノードに対して行っているわけにはいかない．そのため，使用条件を設けて探索するノード数を減らす必要がある．しかし，それに伴って生じる問題の一つに一手詰みがあってもそれを発見できず見逃してしまうということがある．その対策として終盤通常探索の末端ノードにおいて詰み探索を行っている時間がない場合，代わりに処理の軽い一手詰み判定関数を行うようにする．だが，この方法では末端ノードに一手詰みがほとんどない場合無駄な探索時間が多く発生してしまう．そこで，詰みやすさの指標となる玉の安全度を使用条件として用いた．なお，玉の安全度とは玉の移動できるマスの数を意味し，我々のプログラムではこれに盤の端近くでの補正と玉周りのマスへの利きの制圧度による補正を若干加味して計算されている．また，詰みに近づくほど値は小さくなる [5].

3.1 実験

一手詰み判定関数を用いた終盤探索を行うものとそうでないものとに四段の終盤問題 [6]100 問を 1 問 10 秒で解かせ，その解答数を比較した．また 100 の一手詰み局面における相手玉の安全度をとったところ，-1300 付近の値が多かった (図 4)．そのため，相手玉の安全度が-1300 未満のとき一手詰み判定を行うようにした．また，安全度をどの値に設定したときにもっとも良い結果が出るかも実験した．なお，我々のプログラムでは歩を 100 点，飛車を 1000 点としている．マシンの性能は CPU:Pentium4 3.0GHz，メモリ:504Mbyte，OS: WindowsXP である．結果を表 2 に示す．表中の一手詰み (一部) というところは一手詰みになりやすい手のみ判定するバージョンの結果である．これを見ると，使用条件を安全度-1700 未満としたときにもっとも解答数が多いということがわかる (図 5)．末端ノードで一手詰みとなる確率と一手詰み判定にかかる総時間とのバランスがもっとも良かったからであると考えられる．また使用条件を設けなかったもの，安全度を高く設定したものについては解答数は少なかった．一手詰み判定が行われた確率に比べて一手詰みであった確率が低く無駄な時間が多く生じていたためであると考えられる．安全度を低く設定したものについては高い確率で一手詰みが発見されていた．しかし，一手詰み判定が行われる確率がとても低く解答数は思ったほど伸びなかった．このように終盤探索において末端ノードで一手詰み判定関数を使う場合，現実的には使用条件を設けざるを得ない．

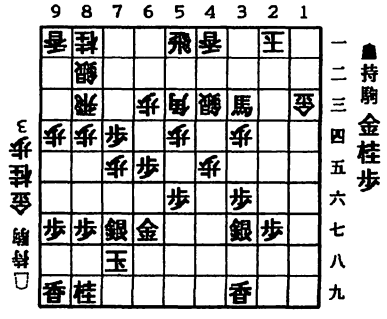

図 4: 後手玉の安全度が-1300 のときの局面

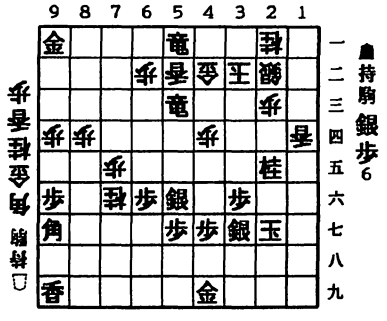

図 5: 後手玉の安全度が-1700 のとき局面

表２：終盤探索実験の結果

	使用条件 (安全度)	解答数	使用確率 (%)	詰み確率 (%)
一手詰み判定使用	−	20	90	19
	-1000	23	36	33
	-1300	23	19	43
	-1500	22	12	41
	-1700	24	8	57
	-2000	23	4	66
	-2300	21	2	67
一手詰み (一部)	-1000	21	38	26
	-1300	24	38	26
	-1500	22	12	51
	-1700	24	7	50
	-2000	23	4	57
	-2300	22	2	65
不使用	−	22	−	−

4 一手詰み判定関数を用いた詰み探索

詰み探索は終盤の探索においてとても重要な位置を占めており頻繁に使用される．しかし，非常に処理が重いため特に高速化すべきものの一つである．そこで，詰み探索への一手詰み判定関数の利用を試みる．これにより，一手詰み局面における探索時間を短縮しその分の時間を他のノードにあてることで単位時間内に探索できるノード数を増やすというねらいがある．この方法では当然，多くの一手詰みでないノードが探索の対象となり無駄な時間が生じてしまう．しかし，一手詰み局面で短縮できた時間によりそれをカバーし，全体的

にみて探索時間を短縮できる可能性がある．また，実装に関しては王手生成の直前に組み込むという方法をとる．

4.1 実験

詰み探索に一手詰み判定関数を組み込んだものとそうでないものとに22の平均18手の詰め将棋を解かせ，費やした時間を比較するという方法をとった．実験に使用したマシンの性能は前実験と同様CPU：Pentium4 3.0GHz，メモリ：504Mbyte，OS：WindowsXPである．実験結果を表3に示す．結果から一手詰み判定関数を組み込んだものがもっとも早く問題を解き終わっていることがわかる．つまり，実験前の予想通り一手詰みの局面で短縮できた時間が大きく，一手詰みでない局面で少々時間を無駄にしてもそれを十分にカバーできるということが言える．また，一手詰み判定になんらかの制御を加えることにより更に良い結果が出ると思われる．なお，安全度は計算コストが高いため詰み探索では求めておらず使用条件としては使えなかった．

表3：詰み探索実験の結果

	総探索時間 (s)
一手詰み使用	30.000
一手詰み使用（一部）	30.984
不使用	39.797

5 まとめ

終盤探索において処理の重い詰み探索をすべてのノードに対して行う場合，多くの無駄な時間が生じてしまう．そのため，様々な方法で探索ノード数を減らす工夫がされている．しかし，探索ノード数を減らすことにより一手詰みを見逃してしまう恐れが生じる．そこでこの問題の解決するため，末端ノードで詰み探索を行う時間がない場合代わりに処理の軽い一手詰み判定を行うようにした．だが結果としては，一手詰みのない局面においても一手詰み判定を行ってしまうためその分探索時間を浪費してしまい良い結果が出なかった．そこで対策として詰みやすさの指標となる相手玉の安全度を一手詰み判定関数の使用条件とした．その結果，一手詰み判定を行わないバージョンと比べ四段の終盤問題を2%多く解けるという結果を得ることができた．また，詰み探索の高速化は終盤探索においてとても重要になる．そこで，一手詰み判定関数を詰み探索にも利用し一手詰み局面での探索時間を短縮することで全体としての探索速度を上げようと試みた．22の平均18手の詰み問題を解かせた結果，一手詰み判定を行わないものと比べ約13%速く解くことができるという結果を得た．一手詰みのない局面で無駄にした時間より一手詰みのある局面で短縮できた時間の方が多かったためであると言える．

6 今後の課題

一手詰み判定関数を利用した詰み探索，終盤探索では一手詰みのない多くの局面においても一手詰み判定を行ってしまう．しかし，人間は「この局面には一手詰みがありそうである」「この局面では明らかに一手詰みは存在しない」と瞬時に判断できる．そのため，今後はコンピュータ将棋にも人間のような高度な判断をさせ，一手詰み判定関数をより効果的に使えるようさせていきたい．

謝辞

本研究は日本学術振興会特別研究員奨励費(#2267, #1452)による助成を受けている．

参考文献

[1] 有岡雅章: 将棋プログラムKFEndにおける探索, コンピュータ将棋の進歩4, pp.18-40, 共立出版 (2003)

[2] 金沢伸一郎: 金沢将棋のアルゴリズム, コンピュータ将棋の進歩3, pp.15-26, 共立出版 (2000)

[3] 有岡雅章: 将棋プログラムKFEndのHP http://www31.ocn.ne.jp/ kfend/index.html

[4] 山崎二三雄, 瀬野訓啓, 飯田弘之, 小谷義行: 将棋の先読みをしない1手詰めの計算法, 第42回 (平成3年前期) 全国大会講演論文集, 2巻, pp.267-268, 情報処理学会 (1991)

[5] 橋本剛, 作田誠, 飯田弘之: 必至問題を解くプログラムとその評価, 人工知能学会誌論文誌, 6号a, 16巻, (2001)

[6] 濱田浩寛: 四段の終盤, pp.5-203, 毎日コミュニケーションズ (2001)