オセロの定石読切りと問題点に関する考察

上田徹, 橋本剛, 橋本隼一
北陸先端科学技術大学院大学

概　要

2007 年に J. Schaeffer らによってチェッカーの結論が「引分け」であると解かれ，大きな話題を呼んだ．本論文ではチェッカーに次ぐ題材としてオセロを取り上げ，df-pn と経路分枝数探索 (BNS) を用い読切りを行っていく．まず問題となってくるのが証明数探索の持つ「2 重カウント問題」であり，これに対しては長井の方法を実装した．次に探索性能の比較を行い，その結果 df-pn の探索速度が BNS よりも遅くなっていた．原因となったのがオセロの DAG の頻度で，DAG が多いため長井の方法だと短手数のうちは有効であっても長手数になるとデメリットの方が大きくなる事が分かった．BNS は探索速度が df-pn よりも速いが総探索ノードは遥かに多く長手数になると df-pn よりも読切りにかかる時間が長くなる事が分かった．オセロの読切りには既存の方法ではなく両者の長所を併せ持つ探索法が必要であると言える．

Solving an Opening Book of Othello and Consideration of Problem

Toru Ueda, Tsuyoshi Hashimoto, Junichi Hashimoto
Japan Advanced Institute of Science and Technology

Abstract

Checkers was solved by J. Schaeffer et al in 2007 and the conclusion is draw. This news aroused lots of comments. Othello is taken up as a theme that follows Checkers in this paper, and solving problems by using Df-pn and Branching Number Search. Here the most important problem is ”Double Counting ” of Proof Number Search. We implemented Nagai ’s method as this measure. Next, we compared Df-pn to BNS. From the result, Df-pn is slower than BNS, that caused by DAG frequency of othello. In this paper, we show that Nagai’s method performs well for short problems, but the demerit becomes significant in largers the large number of DAGs in othello being the main issue. On the other hand, BNS works faster but search more nodes than DFPN, making it much slower in larger problems. Considering these results, solving othello requires a new method that searches a low number of nodes like DFPN with the speed of the BNS.

1　はじめに

2007 年，Schaeffer らによりチェッカーが解かれ [1]，大きな話題となった．日本において，チェッカーと同程度にシンプルで，かつ人気のあるゲームと言えばオセロであろう．現時点で 4 × 4，6 × 6 オセロは解かれているが，8 × 8 オセロはまだ解かれていない．8 × 8 オセロは Checker よりも難しく合法な局面数は Checker の 5×10^{20} に比べ 10^{28} と言われており [2]，読切りに成功したら大きな話題を呼ぶ事が予想される．

本研究では最終目標を 8 × 8 オセロの完全な読切りとするが，まずはオセロの定石を解く事を試み，またその際の問題点について考察する．現在最強のオセロプログラムは Zebra[3] であり，残り 30 マスの読切りが可能である [4]．しかし，定石と呼ばれるもののうち最も石が置かれているものでも 35 マスが残っており [5]，解を求めるには不十分である．

我々は [1] でも利用された df-pn，その改良として提案されている BNS を用いた読切りを試みた．BNS は比較的新しいアルゴリズムであるため df-pn との比較が十分に行われているとは言い難い．また df-pn の弱点として知られる「2 重カ

ウント問題」に関してもさらなる考察が求められる．

2　関連研究

2.1　ゲームの読切りの歴史

比較的小さなゲームの幾つかは現在までに解かれている．92 年に Qubic が証明数探索を用いて解かれ，93 年に GoMoku の先手必勝が，98 年には Connent Four の最善応手手順がそれぞれ明らかにされている．また 93 年には，6 × 6 オセロが Joel Feinstein によって解かれており，結果は後手必勝であった [6]．中でも最近 Schaeffer らによって解かれたチェッカーはこれまでに解かれたゲームの中でも最も大きな計算複雑性を持つものであり，世界中で大きく取り上げられた．

Schaeffer らが用いた探索アルゴリズムは大きく分けて *Seeding*，*Proof Tree Manager*，*Proof Solver* からなる [1]．Seeding は文献などから得た最善と考えられる手順をいう．Proof Tree Manager は Seeding として与えられた手を基準に PNS で探索を行い，ゲームを解く為に必要と思われる局面を生成し Proof Solver へ渡す．Proof Solver は与えられた局面をまず α β で探索をし，一定の条件を満たした局面について df-pn による探索を行い結論を求める．

このアルゴリズムには証明数を用いた探索 (PNS，df-pn) が含まれており，証明数探索一般に付随する GHI 問題や 2 重カウント問題への対策が取られていなくてはならない．しかし，文献 [1] を見る限り，GHI 問題に関する記述はあっても 2 重カウントに関する記述は無い．2 重カウント問題を認識していなかったか，無視できる程度であったかは判断できないが，仮に探索法を証明数探索のみに統一し 2 重カウントの対策を取っていればもっと短期間で解かれていた可能性がある．

2.2　*df-pn* と *BNS*

df-pn とは 01 年に長井が証明数探索 [7] を基に考案した AND/OR 木を解く探索法で，深さ優先探索法でありながら最良優先と同等の振る舞いをする探索法である [8]

一般に証明数を用いた探索法は厳密な意味での木を想定しており，手の循環や同一局面への合流が容認されているゲーム理論的な木に適用した場合，「GHI 問題」や「2 重カウント」が発生することが指摘されていた．このうち前者は岸本により解決されており [9]，そもそもオセロにおいては発生しない．

一方「2 重カウント問題」[8] とは証明数探索において AND ノードの証明数を子ノードの証明数の和によって定義する事に起因する問題で，同一局面に別手順で至る事が出来る場合に，ルートノード近くの証明数が指数的に増えてしまう現象を言う．プログラムによっては証明数が容易にオーバーフローし，あるいは探索時間を大幅に引き延ばす原因となる．

この問題に対し，いくつかの解決策が提案されてきたが，完全な解決には至ってない．柿木の方法 [10] は詰将棋での玉の受け手を利用するものであり，将棋以外のゲームには応用できない．長井の方法 [8] は著者自身が「詰将棋の性質を利用した ad-hoc な解法」と述べているが，オセロへの応用が可能ではないかと考え検証を行う事にする．結果については後述する．

岡部の方法は BNS として知られる [11]．岡部は「2 重カウント問題」が証明数を利用する限り避けられない問題であると考え，新たに経路分枝数と呼ばれる概念を導入し，AND ノードの経路分枝数を選択した子ノードの経路分枝数と定義する事により 2 重カウントの影響を受ける事の無い探索法を提案した．

しかし，探索の基本性能は df-pn の方が探索に使う情報が多いため性能が良い可能性が高く，どちらの探索法がオセロの読切りに適しているかは決定できない．

3　*df-pn* と *BNS* の比較

前節で述べたようにオセロの読切りを行う方法としては，

・df-pn + 長井による 2 重カウント回避法

表 1: 対策の有無での探索性能の違い

	2 重カウント対策あり			対策なし		
残り	nodes	time(s)	solved	nodes	time(s)	solved
10	1.3	0.3	50/50	1.5	0.6	50/50
11	3.3	0.5	50/50	3.5	1.1	50/50
12	11	1.3	50/50	6.6	1.9	50/50
13	19	2.2	50/50	55	11	50/50
14	75	7.1	50/50	46	12	49/50
15	100	8.7	50/50	115	36	47/50
16	383	29	50/50	280	54	41/50
17	380	31	50/50	642	1447	34/50
18	2560	153	50/50	1015	225	24/50
19	3472	176	50/50	1280	321	19/50
20	10379	419	50/50	972	231	9/50
21	6187	258	18/18	×	×	0/18
22	6991	251	5/5	×	×	0/5
23	8452	326	5/5	×	×	0/5
24	7025	362	5/5	×	×	0/5
25	3022	187	3/3	×	×	0/5
26	4271	213	3/3	×	×	0/5
27	2173	129	3/3	×	×	0/5
28	1372	71	3/3	×	×	0/5
29	3851	216	3/3	×	×	0/3
30	5368	254	3/3	×	×	0/3
31	6540	417	2/2	×	×	0/2

※ 表のデータの単位は nodes(× 1000)，time(s).

3.2 *BNS* との探索性能の比較

表 2: BNS の読切り結果

残り	nodes	time(s)	solved
10 手	1.5	0.2	50/50
11 手	3.7	0.3	50/50
12 手	12	0.6	50/50
13 手	22	0.7	50/50
14 手	79	1.6	50/50
15 手	117	2.2	50/50
16 手	583	16	50/50
17 手	666	11	50/50
18 手	3713	65	50/50
19 手	7308	135	50/50
20 手	32876	645	50/50
21 手	76287	1538	23/23
22 手	80171	1483	5/5
23 手	60978	1380	3/3
24 手	5031	102	3/3
25 手	8421	169	3/3
26 手	15821	316	3/3
27 手	2405	48	3/3
28 手	9356	195	3/3
29 手	6391	127	3/3
30 手	16498	349	2/2

※ 表のデータの単位は nodes(× 1000)，time(s).

・BNS

の 2 つの選択肢がある．我々はまず 2 重カウント対策がオセロでどの程度有効に働くかを検証し，次に df-pn と BNS の性能を比較した．また，プログラムには共通で SmallTreeGC と SmallTreeReplacement[8] を実装した．

テスト局面として残り手数毎に 50 局面をランダムに選び，CPU:Pentium4 3.40GHz，メモリ:2GB のマシンを使用した．なお以下に示す実験中残り 20 手以上のものについては時間的な制約から十分なデータが取れていない．

表のデータの単位は nodes(× 1000)，time(s).

3.1 2 重カウント対策の有効性

2 重カウントへの対策として長井の手法を採用し対策を行わないものと比較した．[8] に従い未探索ノードは証明数 (pn) は 1 とした．また，証明数が 32767 を超えた場合はオーバーフローを起こしたと見なし読みを打ち切った．実験の結果を表 1 に示す．それぞれの探索時間を比べると 2 重カウント対策を行った方が読み切るまでの時間が早いのがわかる．また読切り成功数では対策ありの方が対策なしに比べ明らかに成功数が上回っている．

次に 2 重カウント対策を行った df-pn と BNS の探索性能を比較した．諸条件は前節と同様に設定した．実験の結果を表 2 に示す．探索速度で見た場合，BNS の方がやや良い結果を見せているが，総探索ノードでは df-pn の方が少なくなっている．

4 考察

実験結果について特筆すべき点はどちらの探索法もシンプルな実装 (df-pn+[8] のような改良は加えてない) で 20 手以上の読切りに成功している点である．df-pn に関して言えば 31 手の読切りにも一部成功しており，オセロプログラム Zebra と同等かそれ以上であると言える．

4.1 探索速度

df-pn と BNS の探索速度にこれほどの差が出るとは予測していなかった．df-pn は証明数・反証数を，BNS は分枝数をそれぞれ利用するものの，この部分のプログラムにはほとんど差がない．しかし df-pn には 2 重カウントを回避するた

めの改良が加えてあり，この箇所が速度差に影響していると考えられる．

そこで追加実験としてオセロ読切り中でのDAG の発生頻度を調べた．結果，DAG を検知する関数を呼び出す回数が残り手数が 17 手の時点で，DAG を検知する関数を呼び出す回数が残り手数が 17 手の時点で，平均で 2 万回を超えていた．これにより df-pn の探索速度の低下が起こり表 1，2 からも総探索ノードは BNS より少なくても，探索時間が 2 倍以上かかっている事が見て取れる．これは長手数の読切りを行うとするとさらに DAG は増え df-pn の探索性能はさらに低下するものと考えられる．以上の事から長井の手法は短手数の読切りの場合，有効な手法であるが長手数となってくると効果以上にデメリットの方が大きくなると考えられる．

4.2 *GC(Garbage Collection)*

もう 1 つ気付いた点は GC にも影響がある事である．現在，ハッシュには SmallTreeGC[?] が実装してあり，トランスポジションテーブルの登録数が一定を超えるとエントリーの中から不要な情報を消しているが，2 重カウントの基点となっている局面の情報は消す事が出来ないためテーブルに残している．この消す事の出来ない情報によってエントリの占有率が上がると，GC で重要度の高いエントリ情報を消す事になってしまい探索に影響を及ぼす事が考えられる．

こういった影響を受けるため長井の 2 重カウント対策は非常に効果的ではあるが，効果があるのは短い手数を解く時だけで，長手数を解くとなるとデメリットの方が大きくなってくる．よって，df-pn を 2 重カウントをふまえた上の探索に改良した方が効果的であると考える．

5 まとめと今後の予定

df-pn でオセロ読切りを行う際，2 重カウントは対応策を実装したとしても

・探索速度の低下

・GC

といった点に大きな影響がある事が分かった．この点をふまえ df-pn の改良を行う．現在考えているのは，AND 節点での証明数の扱いの変更である．通常 AND 節点での扱いは

・pn = 子節点の pn の和

となっているが，この点を改良する事を考えている．例えば，今考えているもので

・$pn = MAX$(子節点の pn) + 非選択子節点の数

の様な改良を考えている．

読切りに関しては現在もデータは取り続けており今後も継続して行っていく．

参考文献

[1] J.Schaeffer, Y.Bjornsson, N.Burch, A.Kishimoto, M.Muller, R.Lake, P.Lu, S.Sutphen, "Checkers is Solved", Science vol. 317, no. 5844, pp.1518-1522.

[2] "コンピュータリバーシ研究会", http://dais.main.jp/reversi/index.html .

[3] "Zebra", Gunnar Andersson, http://radagast.se/othello/ .

[4] "The FFO endgame test suie", http://www.radagast.se/othello/ffotest.html.

[5] "定石を覚えよう，オセロ", http://www5f.biglobe.ne.jp/ nats/jyousekitop.htm.

[6] "Perfect play in 6 × 6 Othello from two alternative starting positions", http://www.feinst.demon.co.uk/Othello/6x6sol.html

[7] L.V. Allis, M. van der Meulen, and H.J. van den Herik,"Proof Number Search", Artifical Intelligence66, pp.91-124, 1994.

[8] A.Nagai, "Df-pn Algorithm for Searching AND/OR Trees and Its Applications", PhD thesis, University of Tokyo(December 21,2001)

[9] Akihiro Kishimoto and Martin Müller, "A General Solution to the Graph History Interaction Problem", Nineteenth National Conference on Artificial Intelligence (AAAI'04), pages 644-649, AAAI Press, 2004,

[10] 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", 2004

[11] 岡部文洋, "経路分岐数を用いた詰将棋解図について", 第 10 回 ゲームプログラミングワークショップ, pp.9-16, 2005