2006－GI－15（7）
2006／3／7

ゲーム木探索における２つの局面表の交互使用

長嶋 淳 [1], 橋本 剛 [1], 飯田 弘之 [1,2]

[1] 北陸先端科学技術大学院大学　[2] 科学技術振興機構さきがけ研究２１

概 要

ゲーム木探索において，ハッシュ表として実装される局面表を利用して探索を効率化することは広く行なわれている．局面表は探索開始時にクリアされることが多く，そのため探索中に集められた情報はその次の手の探索に活用されることは少ない．本稿では２つの局面表を交互に使用して，前回探索時の情報を有効利用する方法を提案する．この手法を我々の将棋プログラム Tacos に実装し，探索が効率化されたことを実験によって確認した．また，探索の効率化に伴い Tacos の強さが向上したことも確認した．

Alternative use of two transposition tables for game tree search

Jun Nagashima[1], Tsuyoshi Hashimoto[1], Hiroyuki Iida[1,2]

[1]Japan Advanced Institute of Science and Technology
[2]PRESTO, Japan Science and Technology Agency

Abstract

This paper concerns the use of transposition tables in game-tree search. The use of transposition tables enables a computer to efficiently search a game tree. Such a transposition table is used only during the current search. This paper proposes a method that uses alternatively two different transposition tables. One is for a game-tree search in the current position, while another one is for the search in the position where the player is to move next. We implement this method on computer shogi program Tacos to evaluate. The results of experiments performed show that the proposed method improves the search efficiency as well as the playing strength.

1　はじめに

ゲームで探索を行っていると，別手順で同一局面にたどり着くことがある．例えば，図１では，Ｆの局面になるにはA→B→FとA→C→Fの２つの手順がある．A→B→Fの手順で先に探索していれば，A→C→Fの手順でFの局面になったときには再度探索することなく，A→B→Fで探索した時の結果を使いたい．これを実現するために，局面表 (transposition table) が使われる．局面表は一般的にハッシュ表で実装される．局面をもとにハッシュキーを作成し，探索結果の評価値を格納する．評価値以外にその局面の最善手などを登録し，反復深化を行うことで探索を効率化するといったことがよく行われている．ハッシュ表の要素にどのような情報を持たせるか，また多数の探索局面に対してどのような局面をハッシュ表に残すかについて様々な工夫が行われている．

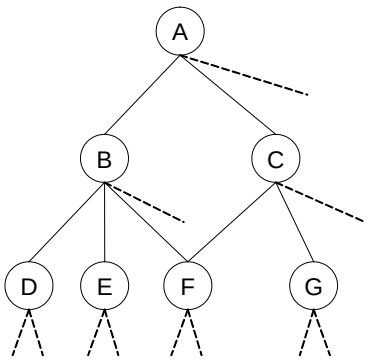

図 1: 同一局面にたどり着く例

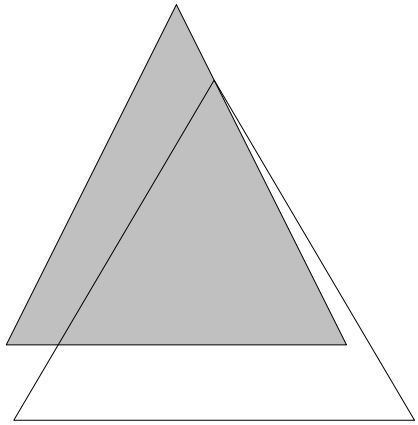

図 2: 探索木

将棋などの２人ゲームをプレイするプログラムでは，探索開始時に局面表をクリアすることが多いようである．これは，１つ前の自分の手番の探索時に得た情報を捨てていることになる．しかし，クリアしないとハッシュ表が満杯になって新しい情報が格納できなくなったり，置き換える場合でも頻繁に置き換えが発生しその分性能が低下することが起こりうる．

本稿では，２つのハッシュ表を交互に利用することにより，前回探索時の情報を利用する方法を述べる．

2　関連研究

探索局面数がハッシュ表の容量を上回る場合，全ての探索局面をハッシュ表に残すことはできず，探索の効率化のためには有望そうな局面を残したい．このため，ハッシュアドレスが衝突したときに既に格納されている情報とこれから格納しようとしている情報を比較し，より有望そうなものを残す方法が考案されている．

Breuker[1] や Lazar[2] の研究では，以下の７つの置き換え方式を比較している．

- *deep*：より深く探索した結果を残す．
- *new*：新しく格納しようとしている方を残す．
- *old*：既に格納されている方を残す．
- *big1, bigAll*：探索木の大きい方を残す．
- *twoDeep, twoBig*：１つのハッシュ要素に２局面格納できるようにし，*new* と *deep* 又は *big1* を組み合わせて使う．

また，チェスプログラムの Crafty では *twoDeep* に似た方式が利用されており，ハッシュ表の各要素に３局面を格納できるようになっている [3]．このうち１局面分がより深く探索した結果を残すのに使われ，残りの２局面分は新しく格納しようとしている局面を残すのに使われている．このように様々な工夫が行われている．

3　前回探索時の情報の利用

局面表の利用にあたり，探索前に格納されている情報をクリアすることが多い[1]．対局中であれば，クリアすることにより１つ前の自分の手番の探索時に得た情報を捨てていることになる．図 2 において，前回探索時に探索した木が灰色の部分，今回探索する部分が白い部分だとする．このとき，二つの木の重なる部分の情報を局面表に残しておけば，探索の効率化が図れる．

しかし，クリアすることにはいくつかの理由が考えられる．ハッシュアドレスが衝突した場合に置き換えを行わず別の場所に格納を試みる場合では，前回探索時の情報が残っていると空いている場所が少なくなり，衝突が何度も起きたり新しい情報を入れる場所がなくなってしまう．置き換えを行う場合でも，クリアする場合に比べて置き換えが頻繁に起こるために置き換えのコストが余計にかってしまう．

有効そうな情報を残し，無用そうな情報のみを探索前に選択的に消すことも考えられる．しかし，ハッシュ表の要素１つずつを有効そうかどうかをチェックしながらクリアするのは大きなコストがかかり，あまり実用的ではない．

そこで今回，前回探索時の情報を残しつつハッシュ表のクリアに余分なコストがかからないようにするため，２つのハッシュ表を交互に利用することを提案する．以下に要点と，図 3 に C++ 風のプログラムを示す．

- 局面表として，２つの全く同じ作りになっているハッシュ表を用意する．
- ハッシュ表への参照として，前回探索用と今回探索用の参照を定義する．
- 対局開始時に両方のハッシュ表をクリアし，前回探索用と今回探索用にそれぞれのハッシュ表を割り当てる．
- 探索開始時に，前回探索用のハッシュ表をクリアする．クリア前には前々回の探索時の情報が入っている．
- クリア後，前回探索用と今回探索用の参照を入れ替える
- 探索時のハッシュ参照では，今回探索用を先に調べ，見つからない場合は前回探索用を調べる．
- 局面探索後にハッシュ表に情報を残す場合は，今回探索用に書き込む．

通常の実装と比較して違う点は，探索前にハッシュ表をクリアする際に参照を入れ替えるところと，参照時に今回探索用で見つからなかった場合に前回探索用を調べに行く点の主に２点のみであり，実装は非常に容易である．一方この方法では，前回探索時のハッシュ表をそのまま保存しておくため，２倍のハッシュ領域が必要となってしまう．しかし，メモリが十分ある場合には，この方法により効率化が期待できる．

[1]将棋プログラム YSS では [4] によると，”オープンアドレス法を用いて 16 回まで再ハッシュを行なう．”，”ハッシュ領域を 75%まで使い切った時点でハッシュ表は満杯と判定して，以下の登録は行なわない”，ということである．クリアに関して特に記述はされていないが，この方式では探索開始時にハッシュ表のクリアが必要である．但し，この記述は 1998 年のものであるので，最新の YSS でこの方式になっているかは不明である．また，将棋プログラム激指では [5]，相手の手番のときに相手の手を予測して探索を行なっている．相手の手の予測が当たって予測中の場合，”現在のハッシュテーブルをクリアしないで，通常の探索を開始する．”とある．やはり，通常の探索を開始する前にハッシュ表をクリアすると明記されているわけではないが，予測手が当たった場合はクリアしないということは，恐らく通常の探索を開始する前にはクリアしているものと思われる．

```
Hash hash[2];    // ２つの全く同じ作りになっているハッシュ表
Hash *pZenkai;    // 前回探索用 (pZenkai) のハッシュを指す
Hash *pKonkai;    // 今回探索用 (pKonkai) のハッシュを指す

void initGame() // 対局開始時の処理
{
    hash[0].crear();    // 両方のハッシュ表をクリアする
    hash[1].crear();
    pZenkai = &(hash[0]);    // それぞれのハッシュ表を割り当てる
    pKonkai = &(hash[1]);
}

void initSearch() // 探索開始前の処理
{
    pZenkai→crear();    // 前々回の探索時の情報が入っているハッシュ表をクリア
    Hash *temp = pZenkai;    // 前回探索用と今回探索用の参照を入れ替え
    pZenkai = pKonkai;
    pKonkai = temp;
}

int search(Board *pBoard) // αβなどの探索本体 (Board:ゲームの盤面を表す)
{
    HashElem *pElem;    // HashElem:ハッシュの要素
    pElem = find(pKonkai, pBoard);    // 先に今回探索用から検索 (失敗時は NULL が戻ってくる)
    if(pElem == NULL)    // 今回探索用で見つからなかった場合
    {
        pElem = find(pZenkai, pShogiban);    // 前回探索用から検索
    }
    // 以下，ハッシュ表に情報があった場合の処理
    if(pElem ≠ NULL)
    …

    // 末端処理，再帰呼び出しなど
    …

    // 探索結果をハッシュ表に書き込んで戻る
    update((pKonkai, pElem); // ハッシュ表に要素を追加/更新 (pElem に結果が入っているとする)
}
```

図 3: 交互利用部分の擬似コード

4　実験

前節で提案した２つのハッシュ表の交互利用を，我々の将棋プログラム Tacos に実装した．Tacos では局面表としてオープンアドレス法のハッシュ表を用いており，衝突が発生した場合は置き換えを行わず，再ハッシュを行っている．ハッシュ表がある程度埋まった場合には探索を打ち切っている．ハッシュ要素は局面を表す 64bit のコードをキーとし，評価値や最善手などを登録している．普段は要素数 2^{22} 個 (192MB) のハッシュ領域を確保している．この設定では，Athlon 64 X2 4800+の PC 上で探索させた場合，35 秒程度探索を行うとハッシュが埋まって探索が打ち切られる．また，探索アルゴリズムは実現確率探索 [6] を実装しており，反復深化で実現確率の閾値を徐々に増やして探索を行っている．Tacos に関するその他の情報は [7] に記載されている．

以下，提案手法を実装する前の Tacos を Tac-1，実装後を Tac-2 と呼ぶことにする．ハッシュ表以外の実装に関しては，Tac-1 と Tac-2 は同じものである．この２つのプログラムを用い，局面を探索させる実験と自己対戦による２種類の性能評価を行った．どちらの実験も Athlon 64 X2 4800+の PC 上で行なった．

4.1　局面探索による性能評価

まず，Tac-1 と Tac-2 それぞれにテスト局面を探索させ，探索ノード数や探索時間の平均を比較する実験を行った．テスト局面として，インターネット上の将棋対戦サイトである将棋倶楽部２４ (http://www.shogidojo.com/) の棋譜集 [8] から，五段以上同士の対局を無作為に 100 局選び，30 手から 80 手の間の１局面を各局から取り出した．探索条件は，探索時間を固定した場合 (10 秒／25 秒)，反復深化の反復回数を固定した場合 (初回の反復深化では実現確率探索の確率閾値を 0.5 とし，反復するごとにを 0.5 倍ずつにしている．10 回 ／ 12 回)，探索ノード数を固定した場合 (150 万ノード／ 300 万ノード) の６種類について行った．それぞれの条件のもと，探索時間や探索ノード数などを測定した．Tac-1 はテスト局面をそのまま探索し，Tac-2 はテスト局面の２手前の局面を同条件で探索した後にテスト局面を探索した．結果を表 1 に示す．

探索時間を固定した場合の結果では，探索ノード数以外に大きな差は見られなかった．この条件で Tac-2 の探索ノード数が減少したのは，ハッシュ参照時に局面が見つからなかった場合には前回探索用ハッシュ表を見に行くという処理が加わるため，探索速度が低下したことが原因だと考えられる．一方，反復回数を固定した場合でも Tac-2 の探索ノード数が減少しているが，これは探索が省略されたり探索順序が改善されることによる探索の効率化が原因と考えられる．また，探索ノードを固定した場合の Tac-2 の反復回数が若干多くなっているのも，探索が効率化されたことを表している．この他，最大探索深さや反復回数に差が出ていないものがある．これは Tac-2 の探索が効率化される一方で探索速度が低下したため，差がでなかったと考えられる．

4.2　自己対戦による性能評価

次に，Tac-1 と Tac-2 による自己対戦を行った．Tacos には探索時に乱数を用いて指し手を変えるといったことは行っていないため，初期局面から対局を行うと毎回同じ展開になってしまう．これを避けるため，定跡書やプロの対局で現れた序盤から中盤の互角の局面を 100 局面用意し，そこから対局を開始した．それぞれの局面について先手後手を入れ替えて対局し，計 200 対局を行った．思考時間は１手に 10 秒と，１手に 25 秒使う２種類の条件下で行った．千日手及び 300 手を超えた場合は引き分けとし，0.5 勝 0.5 敗の扱いとした．結果を表 2 に示す．

表 1: 局面探索の結果

	深さ	反復	ノード	時間
Tac-1 (１手 10 秒)	18.5	12.27	184.7	-
Tac-2 (１手 10 秒)	18.6	12.24	179.8	-
Tac-1 (１手 25 秒)	20.2	13.14	445.8	-
Tac-2 (１手 25 秒)	20.1	13.24	425.7	-
Tac-1 (反復 10 回)	15.3	-	52.0	9.27
Tac-2 (反復 10 回)	15.3	-	46.7	8.70
Tac-1 (反復 12 回)	18.6	-	275.8	25.46
Tac-2 (反復 12 回)	18.4	-	273.3	25.52
Tac-1 (150 万局面)	18.2	12.03	-	18.25
Tac-2 (150 万局面)	18.5	12.10	-	18.20
Tac-1 (300 万局面)	19.5	12.76	-	27.89
Tac-2 (300 万局面)	19.4	12.82	-	28.23

深さ：探索した最大深さ　　反復：探索時の反復深化の反復回数
ノード：探索ノード数 (単位:万)　時間：探索時間 (単位:秒)

表 2: 自己対戦の結果

	Tac-1	-	Tac-2
１手 10 秒	83	-	117
１手 25 秒	86.5	-	113.5

200 戦した場合，112 勝以上すると 95%以上の確率で強くなったと言える．１手 10 秒でも 25 秒でも，Tac-2 は Tac-1 に 112 勝以上しているため，提案手法により 95%以上の確率で Tacos が強くなったと言える．また，探索時間が短い条件の方が Tac-2 がより多く勝っている．これは，前回探索時の情報が使えるために，短い時間でも精度のよい探索が行えたことを表している．

5　まとめ

今回我々は，前回探索時の情報を残して探索を効率化するために，２つのハッシュ表を交互に利用する方法を提案した．この手法を我々の将棋プログラム Tacos に実装し，性能を向上することができた．特に短い時間で探索を行う場合，前回探索時の情報を利用して探索が効率化され，精度のよい探索結果が得られることがわかった．

提案手法では２つの全く同じ作りのハッシュ表を必要とするため，ハッシュ表に割り当てるメモリ領域が通常の２倍必要となる．しかし，実装は非常に簡単であり，十分なメモリが確保できる場合には効果が期待できる．

6　今後の課題

局面探索の実験では，探索が効率化される一方で探索速度が低下することがわかった．ハッシュ表に局面が見つからない場合に毎回前回探索時のハッシュ表を見に行くことが速度低下を招く原因だが，探索の末端近くでは前回探索時のハッシュ表でも局面が見つからないことが多いのではないかと予想される．このため，探索深さによっては前回探索時のハッシュ表を見に行かないといった工夫を行い，できるだけ探索速度が低下しないよう改良したい．

謝辞

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

参考文献

[1] D. M. Breuker, J.W.H.M. Uiterwijk and H.J. van den Herik: Replacement Schemes for Transposition Tables. ICCA Journal, Vol 17, No.4, pp.183-193, 1993.

[2] S. Lazar: Analysis of Transposition Tables and Replacement Schemes. Department of Computer Scienceand Electrical Engineering, University of Maryland, Baltimore County, 1995.

[3] Robert M. hyatt: Crafty のソースファイル,
ftp://ftp.cis.uab.edu/pub/hyatt/src/crafty-20.1.zip

[4] 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, 共立出版, pp.112-142, ISBN: 4320028929, 1998.

[5] 鶴岡 慶雅: 将棋プログラム「激指」, アマ４段を超える コンピュータ将棋の進歩４, 共立出版, pp.1-17. ISBN: 4320120744, 2003.

[6] 鶴岡 慶雅, 横山 大作, 丸山 孝志, 近山 隆: 局面の実現確率に基づくゲーム木探索アルゴリズム, 第６回ゲームプログラミングワークショップ (GPW2001), pp.17-24, 2001.

[7] 橋本 剛: 将棋プログラム TACOS のアルゴリズム, アマトップクラスに迫る コンピュータ将棋の進歩５, 共立出版, pp.33-66, ISBN: 4320121546, 2005.

[8] 久米 宏: 最強の棋譜データベース将棋倶楽部２４, 成甲書房, ISBN：4880861693, 2004.