WCSC21 ツツカナ アピール文書

2011 年 3 月 27 日

1 概要

今年のツツカナの一番の特徴は，指し手ごとに読む深さを学習で定めていることです．重要な手は深く，重要ではない手は浅く読む様，プロの棋譜をもとに調整しています．これにより従来の探索制御法に対し大幅な棋力の向上を確認しています．具体的には，昨年の選手権で 16/43 チームが採用していた LMR[1] という手法がありますが，この LMR を利用した場合に対して，自己対戦で勝率 0.750(225 勝 75 敗) という実験結果を得ています．

表 1 が以前に自己対戦を行ったときの結果です．比較を行ったのは 3 つのプログラムで，今回採用した手法 (NEW)，LMR を用いたもの (LMR)，指し手間で読む深さに差をつけないもの (ALL) となっています．各プログラムは指し手を読む深さ以外の処理は全て共通で，枝刈りや探索延長，評価関数等は実戦的なものを用いています．環境としては CPU Core 2 Duo E 7400 の計算機を使用し，1 手 5 秒で 300 試合を行いました．表 1 の実験結果によると，LMR は ALL に対し勝率 0.663 であり，かなり効果の高い手法と言えます．しかしその LMR に対し，NEW は更に勝率 0.750 を挙げており，十分な成果が得られたと言えます．

表 1　自己対戦の結果

組み合わせ	勝敗	勝率
NEW 対 LMR	225 勝 75 敗	0.750
NEW 対 ALL	234 勝 66 敗	0.780
LMR 対 ALL	199 勝 101 敗	0.663

2 指し手の相対消費深さを利用した探索

本章から，指し手ごとに読む深さをどのように学習したかを説明して行きます．学習というと，適当な目的関数を定め，その目的関数を最小化することでパラメータ (ここでは指し手を読む深さ) を調整するのが常套手段です．ですが，この目的関数の設定が非常に難しいものでした．学習と一口に言っても様々な課題があります．どういったサンプルから学習するのか，サンプルは準備できるか，設定した目的関数の最小化が棋力の向上に繋がるか，現実的な時間で目的関数を計算できるか等々．これらを一度に解決する良い基準がなかなか思いつきませんでした．そこで，直接指し手を読む深さを学習するのではなく，概念を一枚噛ませることで問

題を簡単にしようと考えました．例えば，有名なアルゴリズムに実現確率探索 [2] がありますが，これは指し手を指す確率を棋譜から学習し，それを読む深さに変換することで間接的に学習を行っています．同様に，探索手法を最初に定義し，そこで利用する何かを学習し，それを読む深さに変換するというプロセスを経ることにしました．

手始めに，探索手法として「指し手の相対消費深さを利用した探索」というものを考えました．根底にある考え方は [3] で述べられている事と同じ様な事です．将棋においては同じ指し手でも他の合法手との比較で読みの深さが変わり得ます．例えば，ある局面で端歩を突く手を考えているとします．先に飛車をタダで取る手が見つかっているのならば，端歩を突く手はほとんど読む必要がありません．なぜなら，大抵の場合は飛車を取る手より価値の低い手だからです．しかし，もし有力な手が見つかっていないのならば，端歩を突く手もそれなりに読む必要があります．他の手と同程度に有力だからです．この考え方をモデル化したものが指し手の相対消費深さを利用した探索です．この探索では指し手には絶対的な価値があると考え，それを絶対消費深さと呼びます．さらに指し手間で絶対消費深さの差を取ったものを相対消費深さと呼びます．本手法では一番有力な指し手の消費深さを 1 とし，それ以外の指し手は 1 番有力な指し手との比較で 1 以上の消費深さを設定します．具体的には，現在見つかっている手の中で最小の絶対消費深さを持つ手を記録しておき，今から調べる指し手とその手との相対消費深さに 1 足した値を消費深さとします．本手法を Nega Scout に実装した場合の疑似コードを図 1 に示します．ポイントとしては，オーダリングに依存しているために，動的に探索深さを制御する手法にもなっていることが挙げられます．例えば，良い手だと思っていた角を取る手が，探索を通じて悪い手だと判明してオーダリング順位が下がったとします．すると，角を取る手があるからと軽視されてきた他の手も深く読むようになります．結果，静的な知識 (角を取る手は深く読んだ方が良い等) も利用しますが，動的な情報 (オーダリング) も利用するという手法になっています．この探索における指し手の絶対消費深さの学習を行うことで，間接的に指し手を読む深さを学習します．

3 指し手の絶対消費深さの学習

学習の基本方針は，プロの棋譜で指された手を重要な手と見なし，他の手よりも相対的に深く読むように調整することです．

ある局面についてミニマックス法で探索を行うとします．通常のミニマックス法では全ての指し手の消費深さが 1 ですが，ルート局面の指し手だけ消費深さに差をつけることを考えます．例えば，ルート局面における指し手 m_1 の消費深さは 1，ルート局面におけるその他の指し手の消費深さは 2 とします．この条件のもとで残り深さを $n+1$ としてミニマックス法を呼び出したときの計算コストを考えます．ルート局面から見て，指し手 m_1 以下のリーフノード数は平均分岐数を B として B^n，その他の $B-1$ 個の指し手以下のリーフノードの数は各々 B^{n-1} と見積もることができます．そのため，リーフノード全体の数は $B^n+(B-1)B^{n-1}$ 程度であり，計算コストもこの程度になります．別の見方をすれば，指し手 m_1 以下を n 手読むために全体として $B^n+(B-1)B^{n-1}$ の計算コストを要したことになります．プロの棋譜から絶対消費深さを学習する際にも同じ見方をします．つまり，プロの指し手以下を n 手読むために全体としてどの程度の計算コストがかかるかに着目し，このコストの最小化を図ります．

始めに，問題を単純化し，学習時間を削減するための仮定として，局面 p での探索にかかる計算コストは，適当な定数 B_p を底とする指数関数 B_p^d で近似できるものとします．d は探索の深さで，例えば 3 手の読みを行うならば B_p^3 のコストを要するものとします．この仮定のもと，棋譜中の局面 p におけるコスト $C(p)$ を次

```
int negaScout(Board& board, int alpha, int beta, double depth)
{
    if (depth < 1) // リーフノードでの評価関数の呼び出し
        return evaluate(board);
    best = −∞;
    minReduction = ∞;
    board.getMoveList(moveList); // 合法手の生成
    foreach (m in moveList) {
        reduction = absReduction(board, m); // 指し手の絶対消費深さを求める
        minReduction = min(reduction, minReduction); // 最小の絶対消費深さ
        reduction = reduction - minReduction; // 指し手の相対消費深さを求める
        newDepth = depth - 1 - reduction; // 探索深さを決定する
        board.next(m); // 局面を進める
        if (最初の子) {
            v = -negaScout(board, -beta, -alpha, newDepth);
        } else {
            // Null Window Search により alpha 値を更新するか確認する
            v = -negaScout(board, -(alpha + 1), -alpha, newDepth);
            // alpha 値を更新するならば探索深さを元に戻して再探索
            if (v > alpha)
                v = -negaScout(board, -beta, -alpha, newDepth + reduction);
        }
        board.prev(); // 局面を戻す
        if (v ≥ beta) // ベータカット
            return v;
        best=max(v, best);
        alpha=max(v, alpha);
    }
    return best;
}
```

図 1　指し手の相対消費深さを利用した探索の疑似コード

のように定義します:

$$C(p) = \sum_i B_p^{D(p,t_p)-D(p,m_{pi})}. \tag{1}$$

ここで t_p は局面 p における棋譜の指し手，m_{pi} は局面 p における i 番目の合法手，$D(p,m)$ は局面 p における指し手 m の絶対消費深さを返す関数です．式 (1) は棋譜の指し手 t_p 以下を n 手読むために全体としてどの程度のコストを要するかを示しており，棋譜の指し手 t_p 以下を n 手読むコストを 1 として正規化したものです．これは 2 章で定義した指し手の相対消費深さを利用した探索において，絶対消費深さが小さい順にムーブオーダリングがなされている場合の計算コストの見積もりとなっています．学習用の棋譜の局面の全体 P に対するコスト $C(P)$ は，次のように各局面のコストの和とします:

$$C(P) = \sum_{p \in P} C(p). \tag{2}$$

学習は式 (2) を勾配法等で最小化することによって行います．

以上で大枠の説明は終わりなのですが，少し踏み込んだ話もしたいと思います．学習にあたって，局面 p における指し手 m の絶対消費深さを返す関数 $D(p,m)$ を次のように定義しました:

$$D(p,m) = \frac{s}{1 + \exp(-\sum_i w_i f_i(p,m))}.$$

ここで，$f_i(p,m)$ は局面 p における指し手 m の i 番目の特徴量，w_i はその重み，s はスケーリングのための定数です．この関数はシグモイド関数を定数倍したもので値域は 0 から s を取ります．このように値域を制限するのは，学習によって極端な値がつくことを避けるためです．指し手の特徴としては，取る手，成る手，王手，逃げる手，駒の位置関係等を用いています．また，コストとして適当な定数 B_p に何を設定すべきかはプログラムの探索に依存しますが，今回は実際に使用しているアルファベータ法の分岐数を意識して，局面 p における合法手の数の平方根としました．これには，ALL ノードにおける探索コストを最小にする狙いがあります．尚，王手のかかった局面は読み抜けが致命傷になり易い割に，合法手数が少なく探索深さの reduction の効果が薄いと考えられます．そのため reduction を行わないものとし，学習も行っていません．

参考文献

[1] An Introduction to LateMove Reductions. http://www.glaurungchess.com/lmr.html.

[2] Y. Tsuruoka, D. Yokoyama, and T. Chikayama. Game-tree search algorithm based on realization probability. *ICGA Journal*, 2002.

[3]「棋理（遠見）」開発日記. http://d.hatena.ne.jp/yos92/20091118/1258529466.