WCSC29 詳細アピール文書（PAL）

山口祐

1　開発動機

将棋ソフトの開発要素として主に探索、評価関数、定跡がある。一般的には、まず (1) 探索部の調整を行い、(2) 評価関数を学習し、最後に (3) 定跡を作成するという順番で実施される。しかしこれらの調整・学習は本来は密接に関係しており、例えば探索部の枝刈りやオーダリングに関するパラメータは、学習時の思考時間（探索ノード数）や評価関数によって最適値が異なると指摘されている。また、評価関数も特定の序盤局面に誘導することによって擬似的にレーティングが向上することがあり、評価関数の学習毎に局面が偏らないように定跡手順を作成し、それ以降の局面を適切に学習させる必要がある。

理想的には (1)(2)(3) を繰り返せばよいが、開発・計算コストが著しく増大する。そこで本ソフトではこれら 3 要素を並列的に学習させる Trinity Learning（三位一体学習）を提案し、学習効率を向上させることを試みた。具体的には学習中の探索部のパラメータ調整、学習時の局面選択におけるモンテカルロ木探索の導入とその定跡への転用を実装した。

2　開発過程

線形評価関数を採用する将棋ソフトの探索部は、チェスエンジンと同様に $\alpha\beta$ 法が主流となっている。探索部は指し手のオーダリングや枝刈りを、短時間の自己対戦で計測しながら調整していくのが一般的である。

これに対し、深層ニューラルネットワーク（DNN）で採用される評価値ベースのモンテカルロ木探索（Value-based Monte Carlo Tree Search、以下 MCTS）では、探索は方策関数の学習結果として自動的に調整される。また、DNN の方策関数は（雑に言うと）$\alpha\beta$ 法のプログロムでは探索済みの巨大な置換表にも相当し、初期局面から学習することで自然に定跡を包含することになる。したがって、線形評価関数でもこれに類する学習が実現すれば、探索・評価・定跡を同時に向上できると考えられる。

表1: 探索手法の特徴

	$\alpha\beta$ 法	DNN+MCTS
探索速度	速い	遅い
枝刈り	主に後向き	前向き
探索の改良	主に手調整	強化学習
定跡	別途作成	方策関数に内包

まず線形評価関数に MCTS を導入することを試みた（注：選手権には不採用）。KPPT 等の評価関数はそれ自体で深い探索に対して一手一致率が 40% 近くあるため、方策としても有望である。そこで YaneuraOu[*1] に通常の評価関数とは別にもう一つの評価関数（= 方策関数）を読み込み、局面評価時に同時に差分計算を行うようにした。さらに置換表ではなく探索木を直接保持し、末端ノード (s_t) で全合法手（子ノード $a_i|s_t$）を評価し、方策関数 P の親ノードと子ノードの差を softmax 関数にかけたものを方策確率 p として親ノードに登録するようにした。

$$p_i = \frac{exp(P(a_i|s_t) - P(a_{t-1}|s_{t-1}))}{\sum_j exp(P(a_j|s_t) - P(a_{t-1}|s_{t-1}))} \quad (1)$$

[*1] https://github.com/yaneurao/YaneuraOu

探索時には方策確率と評価値の統計値に基づき行動関数を計算し、どのノードを探索するかを決定する。子ノードに対して実際に探索した回数の分布 y_i に方策関数を近づけるように、損失関数をクロスエントロピー ($L = -\sum y_i log(p_i)$) で規定できる。指し手 a_i によって出現する特徴に対し、L を最小にするよう方策関数を更新する学習部を実装し、WCSC28 の PAL の評価関数（KPPT）に Apery[*2]の評価関数で生成した教師データで追加学習させたものを用いて、方策関数を学習させた。

次に MCTS 探索とは別に、Stockfish 型の探索部を用いた場合の探索・評価関数・定跡の一体学習を試みた。探索パラメータの調整には Population Based Training[*3]を用い、一定の幅でパラメータをランダム初期化した 16 エージェントを並列に学習させながら、エージェント間のレーティングを計測し、レーティングが低いものを高いものに順次代替しながらパラメータを調整させた。

評価関数の学習では、一般的には棋譜や定跡からランダムな指し手を数手加えた局面から学習データを生成するのに対し、初期局面からの単一の探索木を学習を通じて保持し、末端ノードを順次学習の開始局面として学習データを生成する手法を採用した。ノード展開は探索深さ 12、multiPV=8 で最大の評価値から 200 以内の手を子ノードとして登録する。指し手は MCTS と同様、子ノードの中から評価値の平均値に探索回数によるペナルティを付与した行動関数によって選択され、末端ノードの評価値を順次上流のノードに伝播させ更新するようにした。これにより、棋譜や定跡から外れた局面の学習データを効率的に生成できる他、有力な手については優先的に探索が割り振られることが期待される。また、探索木は、有力な指し手ほど探索回数が多くなるので、適当な閾値を設定することで任意の局面の定跡として転用することができる。

Trinity Learning の学習は探索深さ 8-10 で生成した 2 億局面毎に 1 世代とし、評価関数の学習～レーティング計測～エージェント更新を KPPT および NNUE 型評価関数[*4]で 40 世代程度行った。学習環境は AWS EC2 c5.18xlarge 32 インスタンスを用いた。

3　結果・考察

学習方法と、前年度の PAL の評価関数・探索部とのレーティング差を表2に示す（表中のカッコ内は 95% 信頼区間）。レーティング計測は互角局面集[*5]から 20 ～ 60 手目の局面をランダムに選択し、1 スレッド 1 秒で 1000 対局程度行った。

表2: 開発手法とレーティング差

学習方法	レーティング差
MCTS 探索	-265 (± 29)
TL + KPPT	+70 (± 22)
TL + NNUE	+139 (± 24)

MCTS 探索は探索部のみの変更であるが、$\alpha\beta$ 法より明らかに弱くなった。原因して、方策関数の学習が進まなかったことが挙げられる。一手一致率は 40% 程度になったものの、必然手の方策確率が（特に終盤で顕著に）低くなった。このことから、線形関数だけでは方策確率を十分に表現できない可能性が示唆される。

一方、Trinity Learning で一体的に探索部・評価関数を学習させた結果、KPPT ・ NNUE 型評価関数ともに有意にレーティングが向上した。大会ではよりレーティングが高い NNUE 型評価関数を使用した。（追試については実施していないが、Population Based Training の性質上完全な再現は難しいと考えられる）

定跡については、特に相掛かり・角換わりで不利になる局面に飛び込む変化が散見された。そこで採用する探索回数の閾値を大きくし、角換わりを中心に 30 手程度までに抑えたものを選手権では使用した。（一部の変化については手動で延長して登録）

[*2] https://github.com/HiraokaTakuya/apery
[*3] https://arxiv.org/abs/1711.09846
[*4] tanuki-(WCSC28) を初期値に使用
[*5] https://github.com/tttak/ShogiGokakuKyokumen/