チームBarrel houseのアピール文書＠第31回世界コンピュータ将棋選手権

Barrel houseとは岡山の駅前にあるビアバーです．元々チームメイトを求めさすらって行きついたところです．マスターに許可頂いたので名前をお借りしました．その後，メンバーが変わって当初の方向性とは全く違って来ましたが，まぁ一度出した名前を変更するのもアレなのでそのままです．

プログラム名：白ビール
第28回のHefeweizenが濁った白ビールでしたが，第29回ではKristallweizenとフィルターでろ過した透き通った白ビールでした．その後Hefeweizenに戻すという形を取りました．しかしながら，欧州のチェスサイトでは命名由来から説明頂いているにも関わらず国内では白ビールとしか読んで頂けないので，もう白ビールでいいやってことです．

チームの特徴
フレッシュなチームを自認してますがどうやらおっさんチームのようです．本大会は体力勝負ではないので各方面に様々な知識や技術・勘などを働かせて今年も決勝に残れればいいかなぁと思っています。諸事情でメンバーが多忙なため具体的な策はこれから詳細を詰めていくところです．（前回も前々回もそんなアピール文だった気もしますが）

評価関数
昨年度の計測でもトップチームと劣らないものであることを確認しました．（詳しくはblogにて）
今後の調整等は未定です．

使用マシン
今年もノートパソコンを中継にクラウドの力をお借りする予定です．一昨年と昨年は同じ仕様のものを用いましたが，クラウドの方も今年は若干様子が変わっているようでベンチマーク等も未だなので具体的なことは全く決まっていません．正直あまり予算がないのですが，マシンパワーだけで負けるのもアレなので予算内で一番速いマシンを借りようと考えています．

クラスタリングについて

第５回電王トーナメントで御披露目をしたshotgunシステムの進化版であるMulti Ponderのクラスタリングを用います.

制作メンバーが本年は外れておりますが，既に実装は文書化され多くの類似実装が選手権でも見られますので改良して用いるのは問題ないと考えます.

チームDaigorilla　wcsc31

田中大吾

門倉新之助

今回使用したライブラリ

- 水匠3kai
- バーニングブリッジ
- やねうら王

評価関数

- 今回の評価関数もHKPE9を選択した。理由としては、まだまだ余地があると考えている。学習パラメーターの限界が標準型の約10倍となるため、教師の質も十分に必要だが、定跡の作成に力を入れた後、定跡を浅いDepth(14)と深いDepth(26？)の教師生成時に上手く利用した。定跡の作成方法は後ページに
- さほど良いマシーンを持ってない開発者には有利となる関数だ。→ほんとかわからんが現時点では量のほうが圧倒的に重要）（本当は自宅クラスタマシン120Cでドドーンと掘りたかったがまったく安定しなかった…）

工夫点（定跡）・現状（2/28現在R4600相当？）
工夫点・現状（4/28現在R4800相当？）最後に検証結果

- DG電竜1から水匠とDG.bookを利用し約5億程度学習
- 定跡は独自の作成したもの→初歩の定跡（自分で作成した）から約30万棋譜（DGVS水匠）を生成し、DGの評価関数で

Depth32

Max62手

Multipv 2～3

以上を条件にやねうら王で生成した。→それをまた再びDGにセットし棋譜生成をし→その棋譜を利用し再びテラショックを作成

以上を可能な限りroopした。（floodgateの棋譜もある程度利用）

これはやねうら王のテラショック定跡生成コマンドで定跡を使いながらroopすればいいんじゃね？と思いましたが。Floodgateの棋譜などもmergeしたかったため使い分けて棋譜を生成したほうが効率が良いと思った）

- 対局条件は

32t　nodes 15000000 nobook/DG_test1.book/test2.book…　投了スコア500　　62手まで　YO6.00/Suisho3kai

E5 2698v4(2ソケットを＊2利用）

工夫点（学習条件）

- HKPE9は学習が非常に難しかったEtaやlambdaの調整が非常に必要であった。
- また、データセットをしてもあまり変わらなかったため困惑した。
- とりあえずeta=0.03　lambda=0.5程度で行うも,うまくいったかは謎。

余りetaを低くすると強くならなかった(´∀｀)

とりあえず、定跡の流用した後のdepthをgensfen2019で前記の辺りのdepth（辺りと言っているのは、nodeLimitの関係で確かではないから）で掘った。

探索部…安定すればやりたかった・・・（´∀｀）

- やねうら王ライブラリを使用

- マシーンはクラスタを使用予定。

E5-2622 v3 16C（マスターノード） 2ソケット

E5-2698 v4 40C 2ソケット

E5-2698 v4 40C 2ソケット

E5-2678 v3 24C 2ソケット

合計　120C/240T　ram ddr4 ecc 未定GBを予定する

標準型NNUEで動けばNPS平均1億程度

大会本番は…

- 定跡メインの対戦になりそうだがやはり穴が大きいところもあるので嫌になったらno_bookに変更する予定。
- AWS48C　RAM192？196GB を利用している。
- 定跡評価値の上限を-75に設定するもやはり2手目34歩等指してしまう。

対水匠3kai/yo.6.00の最終検証結果

- 投了値　350
- 対局数　100
- Threads 80 USI_HASH 8192MB（E5-2698 v4 40C 2ソケット）
- ノード数平均（NPS）DG_hkpe9/yo.6.00 2500万
  水匠/yo.6.00　3100万

設定時間　一手3.5秒
ほぼ互角であった…(´∀｀)

同ノードにするとどうなることやら・・・未検証
また、短時間では3割ちょっと程度だった。

- 応援よろしくお願いします。

WCSC31 W@nderER アピール文書

Dated 2021/01/23

昨年に引き続き、入玉宣言による勝利を積極的に目指します。

elmo アピール文書 2021/3/29

１．サマリ

コンピュータ将棋開発 → 「高ノード対局用の定跡生成」と位置付ける。
定跡を抜けた後はある程度の棋力と、残り時間の差(と定跡による優位)で押し切れる
と期待して止まない。

今後、評価関数以外の開発ポイントとしてあっても良いと考えている。

２．戦略

実際のところ、時間をかけて深く局面を読むよりも、
ある程度の棋力で20手30手進めて優位となる基準で指し手を選択した方が
消費時間だけでなく、(経験的に)質の面でも優位と考えている。

定跡として必要となる膨大な局面数についても
棋力が上がるにつれて序盤進行は限定されつつあり、
何とかなるのではと思える程度にはなっている。

50手近くまで定跡進行で進められれば、それだけで自分としては成功である。

３．定跡の生成

基本的には従来のelmoの定跡生成法と同様で、あまり開発者らしいことはしていない。
・高ノードで対局し、不利無く勝ち越したものを採用する
・負けた場合は検討モードで修正し、採用する
　その際、既存の定跡が不適と判断されればそこも修正する

手動部分が大きいことを除けば、1局だけで採用するため
対コンピュータ将棋としては効率的な作成手法である。
(比較的少ない計算量で実戦的な(的中率の高い)定跡が作成可能)

※ 評価関数は前回のelmoを利用して生成

定跡の生成時に評価値や勝率等は利用していないため、
NNUEではなくディープラーニング等の手法を利用することも可能である。

４．その他改善(評価関数等)

残念ながら強くなっていない。

・学習方法の変更
評価値と勝率の関係を見ていると、評価関数によっては
勝率の高い部分/低い部分で一定程度外れ値を取るものが確認できる。
(学習用の棋譜から確認)

仮定となるが、
「学習棋譜での勝率と、評価値から算出される勝率との間にモデル上のギャップがあり、
　評価値→勝率の変換係数が不適切」なのではないかと考えた。
変換係数を学習棋譜から算出されるものと一致させることで
強制的にモデルに合うようにした。

とはいえ、既存のNNUEのパラメータを流用するために他のパラメータも修正しており
結果的に既存とほぼ等価な変換となった。が、一致率等の指標は改善した。
何か効果があったかもしれないが、残念ながら気休め程度である。

・学習棋譜生成速度の改善
やねうら王は棋譜生成用のバイナリを作る際には、専用のコンパイルオプションを用いるが、
棋譜生成速度が遅いためtournament(対戦専用)で棋譜生成をするようにした。
約50%生成速度が向上した。

・定跡ponder
ponder中に定跡にヒットしても相手時間中に思考するようにした。
定跡として不成を指せるようにした。

・stockfishの探索部の更新は取り込み予定

５．やるき(低下中)

12月くらいまで真面目に定跡作っていたが、評価関数側が全く良くならずテンションが下がる。

ちなみに先手番定跡は殆ど作っていなかったりします。

６．使用ライブラリ

・やねうら王：主に利用(一部改修して利用)
・水匠シリーズ(2～3改)：評価関数のベース＆学習棋譜生成に利用予定

第31回 世界コンピュータ将棋選手権

PAL

アピール文書

山口 祐

PALの概要

- pal/ﾊﾟﾙ/【名】<話> 仲間、友だち （例）a pen ―
- 前回大会時の工夫
  - マルチエージェント（Population Based Training）
  - 評価/探索/定跡を同時に学習、MCTS探索の実装と検証

開発者

- 山口 祐（@ymg_aq）
- 囲碁・将棋プログラム等を開発 (Github)

PALの特徴

- 2021年度より**NHK杯将棋トーナメント**の評価用AIとして採用
- **Deep Learning**を使う
- 前年のPALで生成済の評価値・候補手データを事前学習
- 探索部を書き始めたので、大会までには動くよう間に合わせたい

使用ライブラリ

- **技巧**: 探索部以外の大部分で使用

[English]

In the history of infectious diseases, a single drug has been easily resisted by mutations in the pathogen. In contrast, when multiple drugs are used simultaneously, each helps prevent the emergence of resistance to the others. Ryfamate derives from Rifamate which is the combination of two medicines for tuberculosis to retard the development of drug resistance. As with this method, Ryfamate aims to combine an NNUE alpha-beta search engine with deep learning MCTS outcomes to compensate for each other's shortcomings.

Recently, with the advent of deep learning, the opening of games has become more important. Ryfamate selects book moves with the highest hit rates based on the number of playouts instead of those with the best values. This selection not only improves the confidence of the move but also saves thinking time and guides the game to a specific pattern. As a result, Ryfamate strives to gain an advantage even against players with more computational resources.

[にほんご]

性能の良いマシンが参戦するって聞いたけど、定跡にいっぱいヒットさせて時間を稼げば、2950Xだって一世代新しい3950Xとも互角に戦えるはず。
あたいったら天才ね！

しかも、定跡で出てきやすい戦型に絞って学習させたら、二世代新しい5950Xや3090にも勝てるかも。
あたいったら最強ね！

[Library]

YaneuraOu : https://github.com/yaneurao/YaneuraOu

dlshogi : https://github.com/TadaoYamaoka/DeepLearningShogi

[第31回世界コンピュータ将棋選手権]

大将軍
（たいしょうぐん）
アピール文書

横内　健一

横内　靖尚

大将軍の概要

- 評価関数に主眼を置いた将棋プログラム。
- 評価関数の特徴としては、盤上の3駒の位置関係を考慮。

  過去には4駒の位置関係の評価関数で参加したこともありますが、結局は3駒の位置関係に落ち着きました。

大将軍の概要

- 過去の遺産をベースにしながら（いまさらながら）、評価関数の学習においては、
  - 駒の位置関係の相対位置による評価
  - 手番の学習

  を考慮。
- 3駒関係で、一般的なkppt型ではなくkkpt-kpp型

  を採用。

使用ライブラリ

- やねうら王
  - 最新のStockfishの探索ルーチン
  - わかりやすいコード（過去の改善の応用に適している）
- 水匠2
  - 評価関数の追加学習に利用予定

WCSC31 Qugiy アピール文書

森　大慶

2021 年 3 月 31 日

1　はじめに

Qugiy は今年の 1 月からフルスクラッチで開発をしている将棋ソフトです。やねうら王や技巧などをとても参考にしています。

探索部は従来の minimax 型のもので、評価関数は NNUE です。評価関数データは自前のではなく、水匠 3 改を利用する予定です。初出場なのでお手柔らかにお願いします。

2　Qugiy の特徴

2.1　飛び駒の利きをビット演算で計算

飛び駒の利きは、Magic Bitboard など表引きの手法で求めるのが一般的ですが、Qugiy ではビット演算で計算しています。

同じものをやねうら王に実装してベンチを取ると、Magic Bitboard のビルドと比べて 4% ほど高速化するようです（@Ryzen 5 3600）。Zen2 の CPU では PEXT 命令が遅いので、PEXT の実装との速度比較はしていません。

Magic Bitboard のように巨大なテーブルは必要ない（おかげで速くなる？）ですし、めんどくさい初期化も要らないので実装が楽なのもメリットです。

これらを全部公開するとやねうら王が 4% 速くなってしまうので、一部（香車の利き計算）だけ紹介することにします。

2.1.1　ビットボードレイアウト

Qugiy のビットボードのレイアウトと将棋盤は以下のように対応しています。

Bitboard と将棋盤の関係

9	8	7	6	5	4	3	2	1	
00	09	00	09	18	27	36	45	54	一
01	10	01	10	19	28	37	46	55	二
02	11	02	11	20	29	38	47	56	三
03	12	03	12	21	30	39	48	57	四
04	13	04	13	22	31	40	49	58	五
05	14	05	14	23	32	41	50	59	六
06	15	06	15	24	33	42	51	60	七
07	16	07	16	25	34	43	52	61	八
08	17	08	17	26	35	44	53	62	九
p[0]		p[1]							

いわゆる縦型ビットボードですが、縦型ならなんでも同じだろうと思っていたら、このレイアウトだと同一直線上にビットが 2 つ以上あるかを判定するときに p[0] と p[1] を or してから popcount... ができないという欠陥がありました。直すのがめんどくさいのでこのままにしてあります。このように、Qugiy のビットボードのレイアウトがやねうら王と異なっているため、以下のコードはそのままやねうら王に貼り付けても動きませんが、定数の部分を変えれば正しく動作します。

2.1.2 後手の香車の利き

このようなビットボードのレイアウトの場合、後手の香車の利きは次のように求められます。

```
template <Color C>
inline Bitboard lanceAttacks(int i, const Bitboard& occupancy) {
    if (C == BLACK) {
        ...
    }
    else if (C == WHITE) {
        __m128i mask = _mm_set_epi64x(0x3fdfeff7fbfdfeffULL, 0x000000000001feffULL);
        __m128i em = _mm_andnot_si128(occupancy.xmm(), mask);
        __m128i t = _mm_add_epi64(em, PAWN_ATTACKS[WHITE][i].xmm());
        return Bitboard(_mm_xor_si128(t, em));
    }
}
```

PAWN_ATTACKS[WHITE][i] は後手の歩がマス i にいるときの利きです。（後手の）香車であればたったこれだけで求めることができます。が、先手はこの方法で求められないのでトータルでどっちが速いかは知りません（適当）。4% の高速化の部分はおそらく飛車角の部分のおかげなので...

2.1.3 先手の香車の利き

先手の香車ですが、上のコードほど効率的ではありませんが一応ビット演算で求まります。コードは以下です。

```
if (C == BLACK) {
    __m128i mocc = _mm_and_si128(BLACK_LANCE_PSEUDO_ATTACKS[i].xmm(), occupancy.xmm());
    mocc = _mm_or_si128(mocc, _mm_srli_epi64(mocc, 1));
    mocc = _mm_or_si128(mocc, _mm_srli_epi64(mocc, 2));
    mocc = _mm_or_si128(mocc, _mm_srli_epi64(mocc, 4));
    mocc = _mm_srli_epi64(mocc, 1);
    return Bitboard(_mm_andnot_si128(mocc, BLACK_LANCE_PSEUDO_ATTACKS[i].xmm()));
}
```

BLACK_LANCE_PSEUDO_ATTACKS[i] はマス i にある香車の、盤上に何も駒がないときの利きを予め求めたテーブルです。

以上でビット演算による利き計算の紹介は終わりです。飛車角もこれの応用なので、ぜひ考えてみてください。

2.2 指し手生成の高速化

Qugiy の指し手生成の速度 (1 秒間での Capture(+ 歩の成り)、Quiet(− 歩の成り) の生成回数) は初期盤面で 15M 回/s、指し手生成祭りの局面で 8M 回/s、最大合法手局面で 4M 回/s ぐらいです。その主要な工夫をまとめます。

2.2.1 二歩判定の高速化

ビット演算ばかりで申し訳ないのですが、二歩判定もビット演算で高速化できました。全体の速度にはそこまで影響していない気がするのでコードを公開します。

```
inline Bitboard pawn_drop_mask(const Bitboard& pawns) {
    __m128i left = _mm_set_epi64x(0x40201008040201000ULL, 0x0000000000020100ULL);
    __m128i t = _mm_sub_epi64(left, pawns.xmm());
    t = _mm_srli_epi64(_mm_and_si128(t, left), 8);
    return Bitboard(_mm_xor_si128(left, _mm_sub_epi64(left, t)));
}
```

この関数は、歩のビットボードから、歩のいない筋を 1 で埋めたビットボードを返す関数です。香車の利き計算でのビット演算を応用して、複数の筋について同時に計算しています。同じ要領で複数の香車の利きを一発で求めたりできると思いますが、あまり高速化できる処理がなさそうなので使っていません。

なにげにお気に入りです。

2.2.2 SIMD による駒打ち高速化

駒打ちの生成において AVX2 命令で複数種類の駒を同時に処理すると、指し手生成祭りの局面での生成速度が 2 倍くらいになりました。上の二歩判定よりも圧倒的に効果があるのでおすすめです。

3 現時点での強さ

PVS、指し手のオーダリング、置換表での枝刈りなどの後ろ向き枝刈りに加えて、Futility Pruning, Null Move Pruning, ProbCut, Late Move Reduction などの前向き枝刈りは（条件がかなり適当ですが）実装が終わっています。一方マルチスレッドでの探索や、延長系はまだ実装していません。

Qugiy は floodgate に AisakaTaiga という名前で参加していて、現在のレーティングは 3000 程度です。作者は将棋のルールしか知らないのでどれくらいの強さなのかよく分かっていないのですが、レート的にはまだまだ弱いのでこれから強くしていきたいです。

4 選手権までに実装したい or 試したいもの

予告なく変更される場合があるので予めご了承ください。

1. Lazy SMP
2. ponder 対応
3. １手詰め
4. † Singular Extension †
5. Large Page？の利用
6. SIMD による指し手ソートの高速化
7. まともな時間制御
8. 宣言勝ち

5 最後に

ここまでご覧いただきありがとうございました！