やねうら王 PR 文書

― スーパーテラショック定跡生成手法について ―

やねうら王チーム

要　旨

近年、Deep Learning 系の将棋ソフトはプロ棋士の研究にも取り入れられてきている。Deep Learning 系の将棋ソフトは従来型の将棋ソフトに比べると序盤戦術に長けていると言える。本論文手法は、その優れた序盤感覚のソフトで大規模な定跡を自動生成しようという試みである。

第１章　従来の定跡生成手法

第１節　人間の棋譜から

コンピュータ将棋も 2010 年ごろまでは人間が定跡を手で入力していた。特にプロの棋譜は神聖視されており、プロの棋譜に出現した 32 手目までの手順をそのまま定跡としているソフトも多数存在した。GPS 将棋[2009-][1]などもプロの棋譜の手順をそのまま定跡化しているようであった。

しかしながら、プロの棋譜には 76 歩 34 歩 68 銀 88 角成(角のタダ取り)で投了というような棋譜も混じっており、プロの棋譜をそのまま定跡として採用していた Bonanza[2005-][2]がこの手順を指すということで話題になったこともある。

そこで、その棋譜の指し手が指された回数を集計し、指された回数が多かった指し手を指すことでこのような悪手を指さないようにするような試みがあった。Bonanza で言うと narrow book というエンジンオプションである。

あるいはプロの棋譜に対して自作の将棋ソフトで 1 局面 0.1 秒や探索深さ 12 など短い時間(or 深さ)で思考させ、悪い評価値の指し手だけを取り除くというハイブリッドな手法が取られることもあった。習甦[3][2014]がそのようなアプローチを取っていた。

プロの棋譜を用いる場合、インターネットで入手できる棋譜の数が限られている(4 万局程度)というのがあるし、トッププロより将棋ソフトの方が棋力が高くなった現在においては、もはや人間の棋譜は定跡生成には直接は使えないと言っていいだろう。

第２節　floodgate の棋譜から

棋譜の数で言うと floodgate(コンピューター将棋ソフトの対局サーバー)[4]の棋譜を用いるというのは魅力的である。floodgate の棋譜の数は 50 万棋譜以上存在するし、その時代時代の最新のソフトが参加しているため、floodgate の棋譜を用いればお手軽かつそれなりの精度で生成できる。2015 年ごろまでわりと流行していた手法である。

floodgate の棋譜から定跡を生成する場合、Bonanza の narrow book の考え方を踏襲し、定跡上の各局面では、なるべく floodgate で採用された回数が多い指し手を選択する。採用回数が多い指し手ほどその信頼性が高いと考えられるからである。Apery[2015][5]や、やねうら王[2014][6]の standard_book.db などに見られるようにこの方法で作成された定跡を搭載する将棋ソフトはわりと多い。

しかし採用回数ベースで定跡を生成する場合、棋譜の数が少ないとその局面での採用確率が 100%であっても信用ならないという欠点がある。例えばその局面を含む棋譜が 1 つしかない場合に、その棋譜の指し手を定跡として登録して良いのかという問題がある。

また弱いソフトの指し手が混じっている場合、それがどれほど信用できるのかという問題もある。ひどい例で言うと、定跡を抜けた直後の局面で先手必敗(評価値 -600)ということもあった。

---

[1] Wikipedia 「GPS 将棋」：https://ja.wikipedia.org/wiki/GPS%E5%B0%86%E6%A3%8B

[2] Wikipedia「Bonanza」：https://ja.wikipedia.org/wiki/Bonanza

[3] Wikipedia「習甦」：https://ja.wikipedia.org/wiki/%E7%BF%92%E7%94%A6

[4] floodgate ：http://wdoor.c.u-tokyo.ac.jp/shogi/floodgate.html

[5] Wikipedia 「Apery」：https://ja.wikipedia.org/wiki/Apery

[6] Wikipedia 「やねうら王」：https://ja.wikipedia.org/wiki/%E3%82%84%E3%81%AD%E3%81%86%E3%82%89%E7%8E%8B

あと、floodgate での対局は、序盤での持ち時間を節約するために何らか定跡を用いて対局させることが多いようで、実際の勝率は低いのにわりと採用されている序盤の指し手というのが存在していることが dlshogi の山岡の研究からわかっている。[7]

そこで、floodgate の棋譜を用いる場合、勝率ベースで指し手を選択するという方法も考えられる。その指し手を指したあと、実際に勝っているのかという点を考慮しようというわけである。この方法は、わりとうまく行くのだが、強いソフトが採用している定跡を過度に信用してしまう問題がある。例えば、floodgate には初手 38 銀に固定したソフトや、必ず振り飛車にする定跡を搭載した将棋ソフトが参加していた。おそらくそのような序盤定跡を用いた時に、普通に指すのに比べてどれくらいレートが落ちるのかを見たい開発者がいたのであろう。このような棋譜が混じっている場合、勝率ベースで定跡を生成する場合でも信用ならない指し手が紛れ込む可能性はいくらでもある。

floodgate の棋譜に対しても自作の将棋ソフトで短い時間で思考させ、明らかに悪い評価値の指し手は取り除くというハイブリッドな手法が取られることもあった。習甦の方法に倣い、やねうら王[2015]もそのような方法を用いたことがある。

しかし、floodgate の棋譜上に数回しか出現しなかった局面の指し手は信用ならないので、30 回以上出現した局面についてのみに絞って定跡化しようなどとすると、局面数はかなり減ってしまい、定跡として十分な局面数を確保できないという問題があった。

いずれにしても棋力にばらつきのある外部の棋譜に頼る以上、その信頼性を担保するのが難しく、データも無限に用意できるわけではないので定跡の局面数にも自ずと限界があった。

[7] 山岡忠夫 「floodgate の序盤 3 手の統計」 https://tadaoyamaoka.hatenablog.com/entry/2021/11/06/010026

第 3 節　人間+ソフトのハイブリッド

ある程度の棋力を有する人間が、将棋ソフトを使いながら、将棋ソフトで上位にくる指し手の変化をすべて調べ、そしてその定跡の末端の局面から対局させて勝率を計測し、形勢が互角であっても勝率が悪い局面への指し手は取り除くという、気の遠くなるような作業を経て作られた定跡がある。

このような定跡としては suimon_fan 氏が公開されている s-book_black[8]が優秀なことで有名だ。WCSC31 で優勝した elmo[2021]も手作業による定跡であり、いまだ上位ソフトの開発者でもこのような手作業での定跡生成を行っている開発者は少なくはない。

第 4 節　自己対局型自動生成

そこで出てきたのが定跡の自動生成という技術である。2015 年ぐらいから流行りだす。ただ、当時の将棋ソフトは序盤はわりと精度が低かったので、実際に序盤で思考させて、良い評価値の枝を掘っていく(思考させて定跡局面として登録していく)のは、わりと危なっかしいところがあった。

そこで、自己対局をして勝率の良いところを掘り進めていくという手法が用いられることが多かった。とりわけそのなかでも優秀だったのは shotgun 定跡[9]と呼ばれるもので、SDT5(第 5 回将棋電王トーナメント)で shotgun というソフトが採用し(shotgun は SDT5 準優勝)、その後、WCSC28 で Hefeweizen がそれを受け継いだ。[10]

[8] suimon_fan ： https://twitter.com/mztn7_fan

[9] 芝世弐 ， コンピュータ将棋における定跡生成法の一提案 ： https://ipsj.ixsq.nii.ac.jp/ej/?action=repository_uri&item_id=192055&file_id=1&file_no=1

[10] 芝世弐 ， Hefeweizen WCSC28 PR 文書 https://www.apply.computer-shogi.org/wcsc28/appeal/Hefeweizen/appeal.pdf

shotgunの方法は簡単に言ってしまえば異種ソフトをローカル環境で対局させ、勝率ベースで定跡化するというもので、相手が既知の将棋ソフトの場合、わりと高い確率でその定跡の指し手を指すようである。

Hefeweizenの手法はほぼノーメンテナンスで定跡が生成していけるということで、定跡自動生成系としてはわりと高い評価がなされているが、自己対局ベースなので生成にとても時間がかかることや既知のソフト以外のソフトだと定跡から外れやすいという欠点はあった。

生成のためにとても時間がかかるのでは、定跡に登録できる局面数が稼げないことになる。いくら精度が高く優秀であっても相手より先に定跡が尽きてしまったのでは、大会では相手の方が多くの持ち時間を残したまま終盤に突入してしまい、結果、不利になってしまう。

このように定跡DBはその登録されている指し手の質、精度はもちろんのこと、なるべく実戦で実現しやすい局面が数多く登録されていることも重要である。

第5節　Minimax探索型(テラショック化)

どんな方法で定跡を生成するにせよ、その定跡のleaf node(末端の局面)での評価値が得られているなら、定跡のゲームツリー上でMinimax探索を行えば、先後が定跡上の指し手で最善を尽くした時に到達するleaf nodeが確定する。

やねうら王では、これを**テラショック化**と呼んだ。[11]やねうら王[2017]には与えた棋譜の局面で思考させ、定跡DB上の各局面に評価値をつけるコマンドをすでに用意していたので、このコマンドを用いて、まずfloodgateの棋譜などを与え、各局面に評価値をつける。その後、その定跡DBをテラショック化することができれば、定跡ツリー上の各局面からのPV(最善応手列)が得られる。それをその各局面の最善手とした定跡DBファイルを別途出力する。これがテラショック化コマンドの概要である。

このテラショック化コマンドは、やねうら王[2019]で実装され、このテラショック化した定跡を用いてやねうら王はWCSC29で優勝した。

ところが、その大会後に判明したことだが、Minimax探索は末端の評価値がroot(定跡の探索開始局面 = 初期局面)まで伝播してくるので、ノイズに極めて弱いという性質がある。従来の将棋ソフト(非Deep Learning系の将棋ソフト)では、評価値にずいぶんノイズが乗っているようで、テラショック化には向かないという意味があった。実際、やねうら王がWCSC29で用いた定跡と前述のs-book_blackとを対局させてみたところ、定跡を抜けた局面でやねうら王の定跡側が不利な局面であった。

また、上の手法は、与えた棋譜ファイル上でMinimax探索のようなことをやっていることになるわけだが、この時、与えた棋譜の指し手しか先後ともに指さないという暗黙の仮定をしていることになる。これはじゃんけんで言うとグーが封じられているようなもので、その状態で(グー、チョキ、パーすべてを使える相手に)勝てるのかという意味もある。

テラショック化という発想自体はさほど悪くなかったのだと思うのだが、以上のように明らかな欠点も抱えており、従来型のソフトには荷が重いと言わざるを得なかった。

第6節　第一章まとめ

floodgateの棋譜から勝率の高かった指し手のみを選んで定跡化するのはさほど悪くないものの、floodgateの棋譜に出現した指し手しか指せないという意味で、グーしか出せないジャンケンにも似ている。今後、将棋ソ

[11] やねうら王ブログ　「テラショック定跡の生成手法」　：https://yaneuraou.yaneu.com/2019/04/19/tera-shock-book-generation/

フトの序盤がシビアになってくるとそんな方法は通用しなくなってくるだろう。

結局、現時点で定跡の自動生成手法で成功しているのは shotgun 定跡ぐらいのものだが、実際にローカル対局を行うので費やすリソースのわりに掘れる(登録できる)定跡の局面数に自ずと限りがある。

テラショック化は悪いアイデアではなかったが、従来の将棋ソフトには評価値にムラがある(ノイズが乗っている)ので、意図通りには機能しない。また PV(最善応手列)の周りを重点的に掘りたいのに、事前に与えた棋譜の局面しか掘れず、テラショック化してからその定跡を用いて自己対局させて、その対局棋譜を与えて思考させて、またテラショック化する、というようないイテレーション(反復)でしか定跡を掘っていけないので思考対象局面の選出に非常に手間と時間がかかる。

第 2 章　スーパーテラショック定跡生成手法

第 1 節　Deep Learning 系思考エンジンの利用

従来型の非 Deep Learning 系の将棋ソフトの序盤には評価値にムラがある(ノイズが乗りやすい)、不正確であるという弱点があった。

そこで定跡の生成には Deep Learning 系の将棋ソフトを用いることにした。現在、ソースコードが公開されている Deep Learning 系の将棋ソフトのなかで最強である dlshogi[12]の互換エンジンとして**ふかうら王**という将棋ソフトを開発した。

今回の定跡生成では、このふかうら王で 1 局面 1 秒程度思考させる。Deep Learning 系の将棋ソフト、特にふかうら王では、思考させた局面のすべての合法手に対する評価値が得られる。なので、この得られた評価値をその指し手で 1 手進めた局面での評価値であると仮定して、有望そうな局面のみを効率的に展開していくことができる。

そのあと任意のタイミングで定跡 DB に対してテラショック化を行い、最終的な定跡 DB を得る。これがスーパーテラショック定跡生成手法の骨子である。

第 2 節　思考対象局面の選出

どの局面を思考対象として選ぶかという問題がある。外部から棋譜を与えてその局面を思考させるという方法もあるが、それだと与えた棋譜の局面しか思考しないので、「自分だけがグーの出せないじゃんけん」のようになってしまう。

テラショック化した定跡を用いて、実際に対局させ、その対局棋譜上に現れた局面について再度思考させるようなイテレーション(反復)も考えられる。やねうら王[2019]ではそうやって定跡を生成していたのだが、テラショック化自体に時間が相当かかるし、ローカル対局にも時間を要する。この方法だとあまり PV まわりを掘り進められない。

そこで、定跡ツリー上で、α β 探索[13]を行い、PV(最善応手列)を取得し、その leaf node(末端の局面)を展開する(思考対象とする)ような仕組みを用意することにした。

これをここでは「(次に思考すべき局面の)選出」と呼ぶことにする。選出された局面をふかうら王の思考エンジンを用いて思考していく。これをここでは「思考」と呼ぶことにする。つまり、「選出」と「思考」。今回の定跡の自動生成はこの二つで成り立っている。

第 3 節　選出の高速化と思考との並列化

この「思考」中にも「選出」を行わなければ思考エンジンに遊び時間が出来てしまうから、「選出」と「思考」は並行して行える必要がある。また「選出」自体を高速化しないと、思考エンジンに遊び時間ができてしまう。

そのためには、

1)　思考エンジンが思考中にも次に有望そうな局面を何局面でも選出できること
2)　選出が思考より遥かに短い時間で完了すること

この二つが必須条件である。

---

[12] dlshogi GitHub : https://github.com/TadaoYamaoka/DeepLearningShogi

[13] Wikipedia : Alpha beta pruning : https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning

最終的な局面数は 1500 万局面程度を想定している。1 局面 1 秒で間断なく思考しつづける時、1 日 86400 局面思考できるから、半年で 1500 万程度であるという想定である。

この 1500 万局面の定跡をメモリ上に読み込んでいる状態においても 1 回の選出が 1 秒未満で完了する必要があると考え、今回はそれを目標に開発を行った。

そこで今回、

1)のために ranged alpha beta search という技法を開発した。

2)のために 高速化のための探索上の工夫を複数実施した。

第 4 節　ranged alpha beta search

まず、典型的な alpha beta 探索の疑似コードを示す。これはゲーム木探索の教科書などに載っている。[14]

```
// pos   : 局面
// alpha : alpha beta 探索の alpha 値
// beta  : alpha beta 探索の beta 値
Value search(pos,alpha,beta):
  foreach m in 全合法手:
    if pos を m で進めた局面が定跡 DB 上にある:
      pos.do_move(m)  // m で 1 手進める
      value = -search(pos,-beta,-alpha)
      pos.undo_move() // 1 手戻す
    else:
      value = pos の局面で m を指した時の評価値
      // 思考した時に全合法手に対応する評価値が得られている。
    if alpha < value:
      if beta <= value:
        return value # beta-cut
      alpha = value   # update alpha
  return alpha
```

[14] Wikipedia 「アルファ・ベータ法」：https://ja.wikipedia.org/wiki/%E3%82%A2%E3%83%AB%E3%83%95%E3%82%A1%E3%83%BB%E3%83%99%E3%83%BC%E3%82%BF%E6%B3%95

上記のプログラムの

foreach m in 全合法手:

の直後に、pos を m で進めた局面が思考中であるなら、この指し手 m はないもの(非合法手)として扱えば、次に良い評価値を持つ leaf node が探せるはずである。基本的な考えかたはそれだけなのだが、これを高速に行える必要がある。

やねうら王では、局面に対応する hash key として Zobrist Hashing という技法を用いている。[15]　また、やねうら王には、ある局面で指し手 m を指した局面に対応する hash key を求める関数も用意されている。

これを用いれば実際には盤面を動かさずして、指し手 m を指したあとの局面の hash key が得られる。この hash key が思考中の局面集合のなかにあれば、ないものとして扱う。つまり、さきほどの部分は、次のようなコードを書いてある。

```
foreach m in 全合法手:
  // 探索中のノードの hash key 集合に m を指した局面の hash key が含まれるなら、m をないものとして扱う。
  if searching_nodes.contains(pos.key_next(m)):
    continue
```

ranged alpha beta search の基本的なコードは以上である。

この疑似コードの searching_nodes は実際のプログラムでは C++の std::unordered_set を用いている。そのためこの集合に含むかどうかの判定は極めて短い時間で行われる。

hash key だとたまたま衝突してしまうと困るので、本来は局面を一意に識別できるものを使うべきである。

[15] Zobrist hashing ：https://en.wikipedia.org/wiki/Zobrist_hashing

そのようなものとしてはSFEN文字列[16]やそれを256bitに符号化したPackedSfen[17]などがあるが、その変換コストは馬鹿にならない。

やねうら王では、hash keyを通常使う64bitのみならず、128bit、256bitに変更する機能があるので、本手法では、128bitのhash keyを用いることにした。この場合、天文学的な確率でしかhash衝突は生じないので、定跡生成の用途で用いるならば問題ないと言えるだろう。

第5節　棋譜上の局面を掘る

本提案手法の目的は、外部棋譜を用いず、自分の思考エンジンのみでノーメンテナンスで低コスト(少ない計算量)で定跡を自動生成することにあるが、実際には、与えられた棋譜の局面を掘り進めたいこともある。

そのため、さきほどのranged alpha beta searchの疑似コードの以下の行

value = posの局面でmを指した時の評価値

の続きに以下のコードを追加する。

```
// kif_nodes:棋譜上に出現した局面のhash keyの集合
if kif_nodes.contains(pos.key_next(m)):
  value += 100
```

棋譜上に出現した局面であれば、leaf nodeの評価値に加点している。これにより、ranged alpha beta searchを行う時にこのleaf nodeは優遇される。

尤も、元があまりに悪い評価値であれば100点を加点したところで評価値としては悪いままであるから、このleaf nodeが選出されることはない。このように、与えた棋譜であっても展開するに値しないならばそこ以上は掘らないというのも自動的に上のコードでなされるので、有望な枝だけが延長される。このため大量の棋譜を与えてもそのすべての局面について思考するわけではなく、優れた局面の「選出」性能を発揮する。

第6節　千日手局面回避のテクニック

本手法では、千日手局面に到達した時のスコアは0として扱っている。通常の探索においては先手は、初期局面においてプラスの評価値がついている。後手は(先手が初手で最善手を指した場合)自分の局面でマイナスの評価値がついている。すなわち、ranged alpha beta searchでは先手は千日手を回避しながらプラスの評価値の局面に辿り着きたいし、後手はマイナスの評価値を避けるため千日手を目指すような探索をしていることになる。

この手法を実装する上で難しい点として、千日手まわりの処理が挙げられる。そこまでの経路(手順)と同じ局面に循環してしまう時、それをどのように取り扱うかという問題がある。

局面の合流を取り扱わなければ千日手に関してあまり難しい問題は発生しないのだが(Deep Learning系でMCTSを用いる将棋ソフトや囲碁のソフトはそうなっているものが多い)、定跡の生成で合流を取り扱わない場合、将棋では特に角換わりで手待ちを繰り返して同じ局面に合流することが多く、そこで組み合わせ爆発を生じてしまい、そこ以降の定跡を掘れなくなってしまう。

このため、定跡生成においては局面の合流を適切に取り扱う必要があるが、合流を取り扱うと千日手の問題が出てくる。

---

[16] SFEN文字列　「USIプロトコルとは」
: http://shogidokoro.starfree.jp/usi.html

[17] やねうら王ではSFEN文字列をハフマン符号化し、256bitにする機能が備わっている。これをPackedSfenと呼ぶ。

そのため、今回、千日手局面を ranged alpha beta search で回避するための手法を開発した。次のように行う。

「思考」済みの局面が再び「選出」された時、それは千日手手順でその局面に到達したものだと考えられる。これを banned nodes 集合と呼ぶ。PV が千日手である場合、循環局面に至る指し手 m とその局面の hash key をペアで保持しておき、その組み合わせの指し手 m を非合法手として扱えば良いのである。

この状態で「選出」を行えば、自動的にその次に有望な leaf node が得られるので、千日手局面が PV になり続けて「思考」できないという状況は回避できる。

これで「選出」時の千日手に関する問題は解決したのだが、テラショック化の時に千日手局面をどう解決するかと言う問題はある。これについては、第 9 節で詳しく述べる。

第 7 節 PV and nonPV

ranged alpha beta search は、通常のαβ探索と同様に再帰的に行われるが、ある局面の探索結果を記憶しておき、二度目にその局面に訪問した時は前回の結果を用いたい。

しかし、PV の leaf node は「選出」されたあと、「思考」が完了するまで無いものとして扱われるので、記憶していた各局面での探索結果の値が不正なものとなってしまう。

そこで、PV と NonPV という考え方を導入する。これは、チェスの Stockfish[18]というチェスのソフトなどαβ探索を行う現代の将棋・チェスのソフトで広く使われている手法である。

PV node ： PV(最善応手列)に関係する node に関する探索。枝刈りはほとんどしない。
NonPV node ： PV 以外の node に関する探索。枝刈りわりとする。

すなわち、探索する時に PV node 用の探索モードと、NonPV node 用の探索モードとの二つの探索モードがあると考えられる。PV はしっかり leaf node まで確認するが、NonPV node では枝刈りをできる限りする。PV(最善応手列)の読みが担保されていればまあいいだろうと言う考えかたである。

例えば、ranged alpha beta search で指し手 m で 1 手進めて再帰的に search 関数を呼び出すところ(次行)は、

value = -search(pos,-beta,-alpha)

以下のように、最初にまず NonPV モードで探索をし、α値を更新しそうな時だけ真面目に PV モードとして探索しなおすというコードになる。

```
value = -search<NonPV>(pos,-beta,-alpha)
if nodeType == PV && alpha < value:
  value = -search<PV>(pos, -beta, -alpha)
```

また、NonPV node では、前回その局面に訪問して探索した時の探索のスコア(best alpha)を記録しておき、それを用いて枝刈りを行う。

ただし、αβ探索では、探索窓として区間(alpha,beta) の範囲に収まる値を探索している。探索結果がこの値の範囲内に収まれば、再度訪問した時に、その値をそのまま使えるが、そうでない場合はどうなるのだろうか？これについては次節で述べる。

[18] Stockfish ： https://stockfishchess.org/

第8節 fail low/fail high

alpha beta 探索は、その局面で区間(alpha,beta)の間に収まる値のなかで最大のものを探す探索である。beta 以上の値がひとつでも見つかれば、その局面の探索を即座に中断し、search 関数はその値を返す。この状態は fail high と呼ばれ、そこで関数から抜ける枝刈りのことは beta cut と呼ばれる。

fail high が生じた時、この node のなかの指し手で beta を超えるものが一つ以上あることはわかるが、これを次回の探索の時に活かすにはどうすれば良いだろうか？

これを解決するのが、bound lower(下界)という考え方である。[19] fail high が生じた時の value(探索値)をこの局面の探索の値 search_value として保存しておく。また、同時にこの値に関するラベルとして bound lower であるとラベルをつけておく。

実際は search_value 以上の値を記録する指し手 m があるかも知れないが、前回の訪問時にすべての指し手を調べたわけではないので(beta cut したので)、そこはわからない。ただ、少なくとも search_value の値を持つ指し手 m が 1 つは存在することは言えている。

再度訪問したときに、bound lower の値として search_value が記録されていたとして、その時の探索窓の beta が

beta <= search_value

であるなら、beta cut して良いことになる。(下界による枝刈り)

同様に、前回の探索で alpha を上回る指し手 m がなかったとして、この場合 fail low と呼ばれる状態だが、この時の最大を記録した value を search_value に保管し、bound upper(上界)とラベルをつけておく。

次にこの node に訪問したとき、alpha が

alpha >= search_value

であれば、どの指し手 m でも alpha を上回らず fail low が起きる見込みが高いので即座に関数から抜け出すことができる。(上界による枝刈り)

また、alpha < search_value < beta であるような search_value の時は、search_value に bound_exact(正確にその範囲)のラベルをつけておく。

以上 3 つの枝刈りは NonPV でのみ行われる。そこで、その 3 つを合わせると以下のような枝刈りが search 関数の先頭で行えることがわかる。

```
// すでに探索済みでかつ千日手が絡まないスコアが記録されている時の枝刈り。(NonPV 限定)
if nodeType == NonPV and  search_value != VALUE_NONE:
  if    bound == BOUND_EXACT
    or (bound == BOUND_LOWER && search_value >= beta
    or (bound == BOUND_UPPER && search_value <= alpha
      return search_value
```

第9節 cyclic score

循環スコア(cyclic score)とは千日手が絡んだ探索の値のことである。いかに NonPV node であっても循環スコア(cyclic score)を信じない方が良いと思われる。

千日手は経路(そこまでの手順)に依存するので、経路が異なれば千日手にならないこともある。

そこで、千日手を検出した場合、循環スコアには cyclic フラグを立てて、子 node に cyclic フラグが立

[19] 英語では lower bound と呼ぶが、Stockfish などのソースコード上では BOUND_LOWER という定数ラベルが用いられているので、それに倣い、ここでは bound lower と書く)

っていれば、その node の探索値 search_value は cyclic score であるとみなし、それを信用しない、とすることは出来る。

ただ、こうしてしまうと組み合わせ爆発が起きるようでテラショック化が現実的な時間で終わらなくなってしまう。主な原因として次のような局面がある。

60 手目付近で矢倉に組み上げて、先後ともに角を動かして手待ちを繰り返す。角の動かし場所は 5 箇所程度あり、先後で 5×5 の組み合わせがある。cyclic score だからと言って、前回の探索値が丸々使えないのでは、こういう部分で組み合わせ爆発を容易に起こすようである。

df-pn による詰将棋にも似た問題があって、GHI 問題という有名な問題がある。df-pn に出現する GHI 問題自体は、千日手を検出した場合、そこまでの手順を hash key 化したものを置換表に記憶しておくことで、GHI 問題自体は回避できる。[20] [21]

しかし、テラショック化の時に GHI 問題の回避のためにそのように経路を保存しておいても上に書いたようにその経路自体が容易に組み合わせ爆発を起こすので無限に近い経路の数をその局面の探索結果の情報として保持する必要が出てくる。

これを回避するためには、千日手を検出した時、そこまでの経路で千日手に関与しない局面の hash key の集合(white list)と、千日手に関与する局面の hash key の集合(black list)とその時の探索の結果値をその局面の情報として保存しておく必要がある。これは実装自体が難しく、また、よほど工夫しないとメモリ空間的にも現在の PC では厳しいので、今回の採用は見送る。

そこで本手法では、cyclic score に関してはきちんと取り扱うことは諦め、NonPV では cyclic score であろうと信用するような実装にしてある。

第 10 節 本手法の効果

本論文の手法で 80 万局面ほど思考し、テラショック化を行ったスーパーテラショック定跡では、初手 76 歩に対する 34 歩を明確に悪手(評価値-243)だと認定した。[22]

2 手目 34 歩はプロの将棋においても昔からよくあるオープニング(序盤)だが、本論文の手法は、この 2 手目の 34 歩を咎めるところまできている。実際、そのあとソフト同士の対局によると先手勝率が 6 割程度であり、このオープニングは現在プロの将棋でも減りつつある。[23]

また、第一章で紹介した s-book_black とスーパーテラショック定跡とで対局させた場合、29 手目まで定跡で進行し、スーパーテラショック定跡側の定跡が先に尽きたが、その局面から最新の将棋ソフト(水匠 4)で対局を引き継ぎ、思考時間を変えて 100 戦やってみたところ 55 勝 42 敗 3 引き分けであった。s_book_black は先手専用定跡のため、スーパーテラショック定跡側は後手番である。後手番で勝ち越しているのでこれは十分な戦果であり、人間がソフトを活用し、長い時間と労力を掛けて作り上げた定跡集を、外部棋譜を一切使わず、かつ、ノーメンテナンスな自動定跡生成により打ち破ったと言えるだろう。

[20] Kishimoto, Akihiro and Martin Müller. “A solution to the GHI problem for depth-first proof-number search.” Inf. Sci. 175 (2005): 296-314.

[21] koumori-n , コウモリのちょーおんぱ 「詰将棋ソルバーにおける GHI 問題対策」 https://komorinfo.com/blog/and-or-tree-ghi-problem/

[22] やねうら王ブログ , スーパーテラショック定跡が 76 歩に 34 歩を全否定 ： https://yaneuraou.yaneu.com/2021/11/05/super-tera-shock-book/

[23] 山口 祐, ツイッター https://twitter.com/ymg_aq/status/1456283886439129089

第 11 節　本手法のまとめ

現代にふさわしい定跡生成手法として、次のような手法の誕生が望まれていた。

・完全な自動生成(ノーメンテナンス)

・少ない計算資源で掘っていける

・(floodgate の棋譜など)外部の棋譜を用いない

・(shotgun 定跡のように)対戦相手の思考エンジンを仮定しない

・思考対象局面の選出が小さな計算コストで出来る

・テラショック化のような定跡ツリー上で Minimax を行い、評価値の高い leaf node を目指す定跡が生成される

本論文で提案するスーパーテラショック定跡生成手法では、これらの要求をすべて満たすことが出来た。

また、スーパーテラショック定跡生成手法は実際にやねうら王のソースコード上に実装し、公開もしている。[24]

参考にしていただければ幸いである。

[24] やねうら王 GitHub , makebook2021.cpp : https://github.com/yaneurao/Yaneura0u/blob/e620b83d4ecb110120cf2abe4f4c806e0634c9b4/source/book/makebook2021.cpp