3 選手権優勝記
―激指の技術的改良の解説―

鶴岡 慶雅
北陸先端科学技術大学院大学

ぼろ負けの 2009 年選手権

2009 年の選手権決勝リーグで，激指は 2 勝 5 敗の 6 位という結果に終わる．選手権の決勝は 8 つのプログラムによるリーグ戦なので試合数はわずかに 7 局しかない．そのため，最終的な順位は運に左右される要素が大きく，普段はその結果をあまり気にすることはないのだが，この年は違った．単に大幅に負け越したというだけでなく，将棋の内容という点で，明らかに上位プログラムと大きな差がついていた．

将棋では，高段者同士の対戦になると，単純な駒得などによって攻めが成功するようなことはほとんどない．そのかわり，陣形の乱れであるとか，駒の働き具合であるとか，なんとなくとらえどころのない要素によってわずかに形勢の差が生じ，その差が徐々に積み重なることで最終的な勝敗につながることになる．従来，コンピュータは，駒の損得のようなはっきりと数値化しやすい要素に関しては正確に評価することが可能だが，駒の働き具合のような，漠然とした要素を評価することは得意ではなかった．

ところが，2009 年の選手権決勝リーグでは，はっきりと状況が変わっていた．優勝した GPS 将棋をはじめ，上位のプログラムの指し手は，プロ棋士が指した将棋といわれてもまったく違和感のないレベルの指し手になっていた．相手の陣形のわずかな乱れをついてぎりぎりの攻めをつないでいく様子は，激指を含めた旧世代のプログラムとは，形勢判断，あるいは大局観と呼ばれるものの点で，次元の違うものになっていた．

読みと形勢判断

人間が将棋を指す上で重要なのは，未来の展開を深く精確に予測する「読み」と読んでいる局面の有利不利を正しく評価する「形勢判断」である．このことはコンピュータ将棋にもそのまま当てはまり，前者を実現するのが，「探索アルゴリズム」，後者を実現するのが「評価関数」といわれるものである．探索アルゴリズムとしては，深さ優先探索，反復深化，アルファ・ベータ枝刈りの 3 点セットを基本として，これにさまざまな工夫を加えることで多くの将棋プログラムが効率的な探索の実現を試みている．

激指は，探索アルゴリズムとして，「局面の実現確率」に基づく手法を用いており，狭くて深い読みを特徴としている．具体的には，個々の合法手がどれぐらいの確率で指されるのかを，プロ棋士の大量の棋譜とロジスティック回帰モデルを利用して推定し，その確率が高い局面を優先的に探索するという手法である．個々の手を表現する際に，指し手に関係する盤面の局所パターンなどのさまざまな特徴量を用いることで，プロ棋士が頭の中で読むべき手を選択するときの，パターン認識的な「直感」をコンピュータで再現したかったというのがこの手法を開発した動機の 1 つである．この手法を用いることで，選手権の 25 分切れ負けといったような短い時間の将棋でも，20 手以上先の局面まで読むことが可能

になっている．

評価関数というのは，局面の形勢判断をコンピュータで行うための関数で，任意の与えられた局面に対して，どちらがどれだけ有利なのかを数値化する関数である．将棋の形勢判断において重要な要素は，駒の損得と駒の働きといわれるが，評価関数では，そのような要素をなんらかの方法で定量化することを試みるわけである．簡単な方法の１つとしては，たとえば，歩の価値は 100 点，金の価値は 600 点などとあらかじめ決めておき，自分の駒と相手の駒の価値の合計の差を計算すれば駒の損得に関しては数値化できることになる．駒の働きを数値化しようとするとだいぶ難しくなり，自玉や相手玉との位置関係や，盤上の他の駒との連携の具合なども考慮しなくてはならなくなる．考慮しなければいけない状況も多様なうえに，割り振る点数にも微妙な調整が必要だ．このようなパラメータの調整は非常に手間のかかる作業だが，かつては完全に手作業で行われており，将棋プログラム開発における作業の多くの割合を占めていた．

評価関数の自動学習を実用レベルで初めて成功させたのは，Bonanza（ボナンザ）の作者の保木氏である．保木氏は，評価関数のパラメータチューニングの問題を，プロ棋士の大量の棋譜を学習データとした指し手選択の教師付き学習の問題として定式化することで，評価関数の完全自動学習に成功した．Bonanza は 2006 年の選手権で優勝し，その後多くのプログラムが保木氏の手法にならい，手作りの評価関数から自動学習された評価関数へと舵をきることになる．

ただ，個人的には，これまで多大の労力をかけて作ってきた激指の評価関数に思い入れがあったのと，自動学習によってプログラムの指し手の個性が失われてしまうことが嫌で，激指では評価関数の自動学習を導入してこなかった．それでも，探索アルゴリズムの工夫や評価関数の手作業での細かいチューニングなどで，2008 年の選手権まではなんとか乗り切ってきたが，2009 年の選手権では，このままでは上位に残れなくなることをはっきりと思い知らされることになる．それが冒頭で述べた昨年の選手権での惨敗である．自動学習された評価関数と，手作業の積み重ねで作られた激指の評価関数には，特に駒の働きを正確に評価する能力という点で非常に大きな差がついていた．激指も評価関数の自動学習に踏み切ることは避けられなくなった．

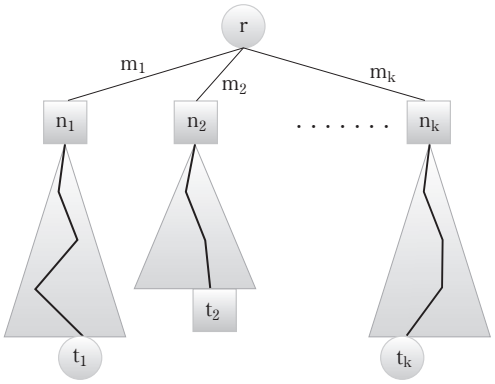


図 -1　探索の末端ノードを利用した評価関数の学習

評価関数の自動学習

Bonanza の評価関数の学習手法についてはすでに各所で紹介されているが，後半の解説のために，あらためてその手法についてここで簡単に説明することにする．詳しく知りたい読者には，オリジナルの保木氏の文献[1]にくわえて，棚瀬氏，金子氏による解説[2), 3)]をお勧めする．

手法の肝を一言で言うと，「プログラムが，学習データ中のそれぞれの局面において，『正解』手となるべく同じ手を指すように評価関数のパラメータをチューニングする」ということである．プロ棋士の棋譜を学習データに用いた場合，正解手というのはプロ棋士が実際に指した手，ということになる．

図 -1 に手法の解説のための概念図を示す．これは，学習データ中のある局面 r において，k 手の合法手 $m_1, ..., m_k$ が存在することを示す．ここで，正解手は m_1 であるとする．ルート局面 r から合法手 $m_1, ..., m_k$ を指すことによって実現される局面を $n_1, ..., n_k$ とし，さらに，それぞれの局面から，「浅い探索」を行った結果得られる読み筋（principal

variation）の末端の局面を $t_1, ..., t_k$ とする．

ここで，ノード x における評価関数の値を $y(x)$ と書くことにすると，上述したような「プログラムが正解手と同じ手を指す」ことを実現するためには，1 以外の任意の j について，

$$y(t_1) > y(t_j)$$

となっていればよいことになる．Bonanza 式の学習では，これを学習データ全体を考慮して最適化するため，$y(t_1), ..., y(t_k)$ の相互の大小関係を定量的に評価する損失関数を定義し，学習データ全体として，その損失関数の値の和を最小化するようにパラメータを最適化する．もし仮に，損失関数を定義するのに $y(t_1), ..., y(t_k)$ ではなく $y(n_1), ..., y(n_k)$ を用いるのであれば，この問題はごくありふれた教師付き学習の問題と同一であり，適当な損失関数を用いれば，単純な凸関数の最適化問題として勾配法などを利用して解くことができる．しかし，Bonanza 式の学習の場合，$y(n_1), ..., y(n_k)$ ではなく，間に探索というプロセスがはさまった $y(t_1), ..., y(t_k)$ を利用するため，パラメータの最適化をどのように行えばよいかという問題は簡単ではない．なぜなら評価関数のパラメータを変化させた場合には，読み筋もそれに応じて変わってしまうからである．ところが保木氏は，勾配法をベースとした手法を用いることで，現実的には損失関数の値が減少するようにパラメータ調整を行うことができることを示し，評価関数の完全自動学習を成功させた．

オンライン学習による評価関数の自動学習

Bonanza で成功した評価関数の自動学習は，昨年優勝の GPS 将棋をはじめとして数多くのプログラムに実装されており，その効果は実証済みであったが，いくつか問題がなかったわけではない．激指でその学習手法を導入しようとしたときに個人的にいちばん大きな問題だと感じていたのは，学習に時間がかかりすぎることだ．学習時には，学習データ中の数百万もの局面のそれぞれにおいて，すべての合法手に対して，先に述べたような「浅い探索」を実行して読み筋を求める必要がある．さらに，パラメータが収束するまでにはその処理全体を何十サイクルも繰り返す必要があるため，1 回の学習に 2 週間〜3 カ月かかるといわれていた．

学習にかかる時間は実行時の速度にはまったく関係がないため，それほど重要なことではないと考える人もいるかもしれないが，実際には将棋プログラムを開発するうえで大きな影響を及ぼす．激指は，評価関数に使用する特徴量として，さまざまなタイプの特徴量を用いている．考えられるさまざまな特徴量の中から，評価関数の精度を向上させるうえで効果が高く，しかも実行時の速度を大きく低下させない特徴量を選び出すためには，数多くの試行錯誤のプロセスが不可欠である．そのためには，(1) ある特徴量を実装，(2) パラメータを最適化，(3) 性能評価，ということを繰り返す必要があるが，学習（パラメータの最適化）にかかる時間が長くなると開発の効率が致命的に落ちてしまう．

もう 1 つの問題は，学習の際に行う「浅い探索」では，正解手の意味を理解できないことが多いという点である [2]．当時，「浅い探索」としては「全幅探索 2 手＋静止探索」という程度の深さの探索が普通であったが，このようなごく浅い探索では人間が深い狙いを持って指した手の意味を汲み取ることはできない．たとえば，プロ棋士が何か深い狙いを持って大駒を捨てる手を指した場合，浅い探索では駒を捨てた効果が分かる局面に到達しないため，コンピュータとしては，単純にその駒の価値が低いから捨てられたのだというように学習してしまう．そのような誤解に基づく学習を避けるためには，「2 手＋静止探索」のようなきわめて浅い探索ではなく，もう少しまともな深さの探索をする必要があると思われた．

上述のどちらの問題に対処するにしても，学習にかかる計算コストを削減することが最重要の課題だと考え，激指に評価関数の自動学習を導入するにあたって，まずはそれを実現することに注力した．

さて，機械学習の学習コスト削減法として最もよく知られている手法の 1 つに，オンライン学習と

呼ばれる手法がある[4]．バッチ学習と呼ばれる通常の学習手法の場合，パラメータをどのように更新するべきかは，学習データ全体を考慮して計算される．別な言い方をすれば，損失関数の勾配を学習データ全体を用いて計算し，それに基づいてパラメータの更新を行う．それに対してオンライン学習では，学習サンプルを1つ観測するたびにパラメータの更新を行う．たとえば確率的勾配降下法（stochastic gradient descent）と呼ばれるオンライン学習法では，損失関数の勾配を，ランダムに抽出された学習サンプル1つだけを用いて近似し，即座にパラメータを近似された勾配の最急降下方向に動かしてしまう．一見ずいぶんと乱暴な手法に見えるが，適当に学習係数を制御することで正しい解に収束することが保証されている．また，バッチ学習と比較してはるかに頻繁にパラメータの更新を行うことができるため，現実的な収束に要する時間はバッチ学習とくらべてかなり短くなることが多い．

激指の評価関数の自動学習では，確率的勾配降下法とよく似た，平均化パーセプトロン（averaged perceptron）と呼ばれるオンライン学習アルゴリズム[5]をベースとしている．学習アルゴリズムは非常に単純である．図-1に関する説明で，「1以外の任意のjについて，

$$y(t_1) > y(t_j)$$

となっていればよいことになる」と述べたが，激指の評価関数の学習では，ランダムシャッフルされた学習データ中の各局面で，なるべくその条件が満たされるようにパラメータの更新をその場で行ってしまうというだけである．いま，ある局面において上式の条件が成り立たない指し手の集合をMとすると，評価関数のパラメータ，すなわち重みベクトル$\mathbf{w}$は以下の式で更新される．

$$\mathrm{w} \leftarrow \mathrm{w} + \frac{1}{|M|}\sum_{j \in M}(\phi(t_1) - \phi(t_j))$$

ただし，$\phi(t_j)$は局面t_jを表現する特徴ベクトルとし，その局面の評価値は，

$$y(t_j) = \mathrm{w}^T\phi(t_j)$$

として，特徴ベクトルと重みベクトルとの内積で計算されるものとする．

激指の評価関数の学習は，このパラメータ更新処理を，プロ棋士の棋譜30000棋譜からなる学習データのすべての局面に対して1度実行するだけである．ただし，最終的に利用する評価関数のパラメータは，学習途中の各局面でのパラメータを平均化したものを用いる．この平均化処理は，平均化パーセプトロンの名前の由来ともなっているが，学習データに対する過学習を防ぎ，ノイズに強い学習を行ううえで欠かせない処理である．

これが激指の評価関数の学習の基本アルゴリズムであるが，学習の効果と効率を高めるためにいくつかの工夫を行っている．1つは，単純に正解手の評価値が候補手の評価値を上回るようにするだけではなく，ある程度のマージンを持って上回るようにしていることである．これは，将棋の場合，中終盤の局面になると，正解の一手以外は大悪手になることが多いという特性を学習に活かしたいためである．マージンの大きさは，序盤の局面では小さく（歩の価値の1/10程度），終盤の局面では大きく（歩の価値の3倍弱）なるようにしている．

もう1つの工夫は，学習の速度を向上させるため，個々の局面において，すべての合法手を考慮するのではなく，ランダムに選択された16手についてのみを考慮して上記の計算を行っているという点である．通常，平均化パーセプトロンで学習する場合，すべての解（将棋の評価関数の学習でいえばすべての合法手）を考慮する必要はなく，最善解（ルート局面で探索した結果見つかった最善手）のみを考慮すればよいはずなのだが，将棋の評価関数の学習でも同様のことを行おうとしたがうまくいっていない．ランダムに16手を選択するというのは妥協の産物ともいえる．

学習にかかる時間は，それぞれの合法手に対して行う「浅い探索」の深さに依存するが，基本深さが6手の場合で8時間程度，8手にした場合でも2日間で済んでいる．通常は6手読みで学習させているが，この場合一晩で学習の結果が得られるので，開発を効率よく行うことができる．従来のバッチ学習の場

合，普通は同じような計算コストの処理を最低でも数十回繰り返す必要があるので，学習時間の差は圧倒的だ．

復活

2009年秋ごろから本格的に激指の開発を再開し，上記の評価関数の自動学習を含め，探索アルゴリズムの効率向上など，さまざまな改良を重ねた．改良の効果を測定するのは簡単なことではないのだが，現在は，GPS将棋チームの人たちが運営するfloodgateという将棋プログラム同士が自動で対戦できるサーバがあり，非常に便利になっている．floodgateでは，他のさまざまなプログラムとの対局を30分に1局のペースで行うことができ，1日で50局近くの対戦データを取ることができる．さらに，対戦結果をもとに個々の将棋プログラムの強さが「レーティング」というスコアとして自動的に計算されるようになっているため，改良の効果をレーティングの向上という形ではっきりと観察することができる．

激指の場合，2009年の秋に本格的な改良を開始してから，2010年の選手権までに，レーティングにして400点ぐらいは強くなったのではないかと思われる．おそらくそのうちの300点ぐらいは，評価関数の自動学習によって達成されたものである．レーティングにして400点というのはきわめて大きな向上で，これまでは，1年間で100点向上すれば万々歳という感じだっただけに，この1年での激指の棋力の伸びは非常に大きかった．

最終的に，2010年の選手権用の8コアマシン(Intel Xeon 5590 × 2)での激指のfloodgateのレーティングは3000点近くに達した．この3000点という数字が，人間の棋力でいうところのどの程度に対応するのかははっきりとは分からない．ただ，いくつかの断片的な情報を総合すると，floodgateのレーティングは人間用のインターネット将棋道場である「将棋倶楽部24」のレーティングと比較して300点ぐらい厳しいと推測されるので，仮に激指が将棋倶楽部24で指したとすると，持ち時間の短い設定であれば3300点ぐらいになるのではないかと考えられる．

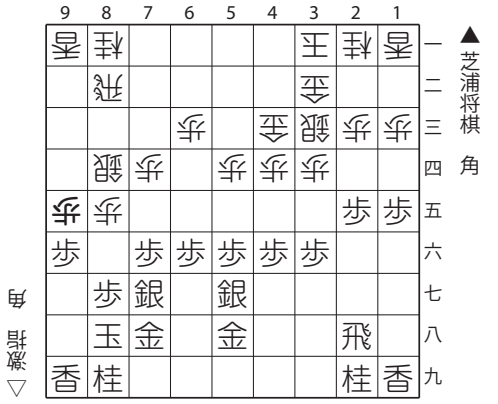


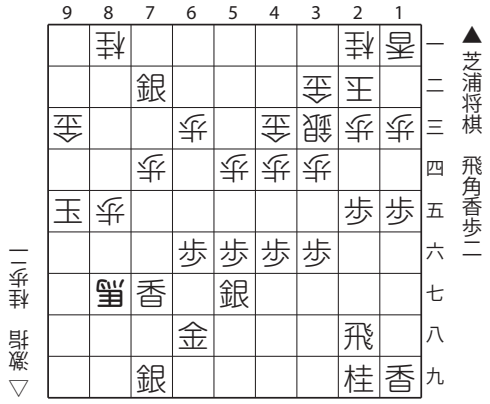
図-2　2次予選　対芝浦将棋

選手権での将棋

今年の選手権での激指の将棋について簡単に振り返る．2次予選で激指は，ボナンザライブラリを使用した芝浦将棋に敗れるのだが，その負け方がなかなか興味深かった．

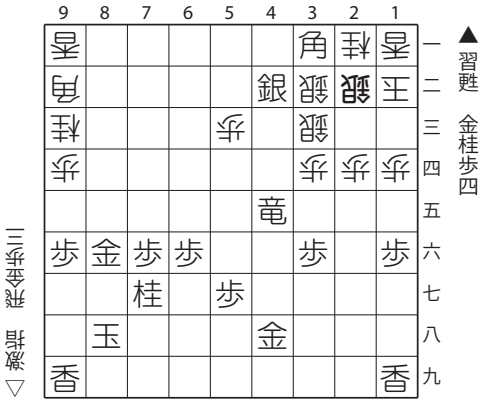
図-2は後手の激指が△9五歩と仕掛けた局面だ．素人目にも棒銀が成功しているように見えるが，実際この局面は，プロ棋士レベルの視点から見ても後手が優勢であるらしい．ところが実戦は，この37手後の局面（**図-3**）で，先手に▲8八香という巧妙な受けの手が存在して激指は負けることになる．

△9五歩と仕掛けた局面からは後手には変化の余地がほとんどないようだ．仮にそうだとすると，△9五歩と仕掛けた手が悪手ということになってしまうが，それを避けるためには先に述べたように37手後の相手の受けの好手が正しく読めている必要がある．しかし現在の激指で読めるのはたかだか20数手先までであり，とてもそこまで深く読むことはできない．あるいは，評価関数をもっと精密にするというアプローチもあるのかもしれないが，プロ棋士の形勢判断で「仕掛けて後手よし」という局面なのだから，それを上回る精度の評価をコンピュータで



【78 手目△8 七馬　まで】

図 -3　2 次予選　対芝浦将棋



【118 手目△2 二銀打　まで】

図 -4　決勝リーグ　対習甦

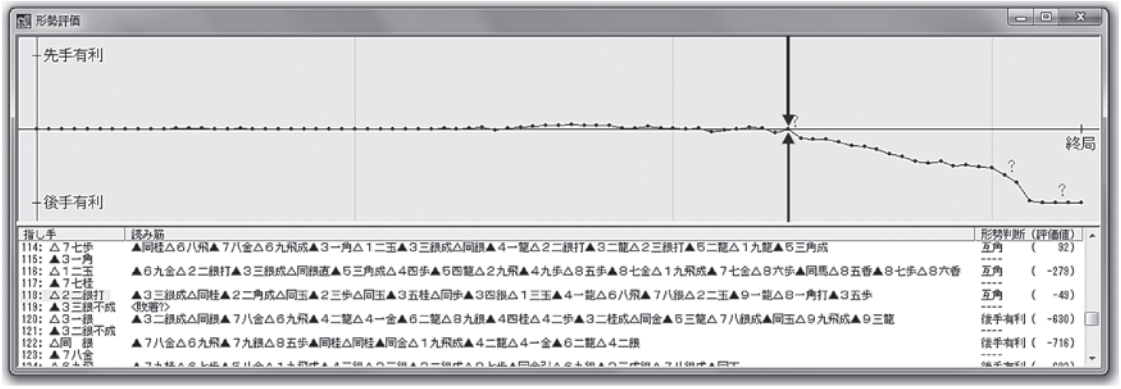


図 -5　決勝リーグ　習甦（先手）vs　激指（後手）の形勢の推移

行うのは無理なのではないか，という気もする．

図 -4 は，決勝リーグでの，今回準優勝したプログラム「習甦」との将棋である．図はちょうど激指が△2 二銀と受けたところだ．ここで習甦が▲3 三銀成と指せばまだまだ互角の戦いが続いたようだが，実際は▲3 三銀不成と指し，激指の△3 一銀を許して逆転してしまった．**図 -5** は，激指から見て，この将棋で形勢がどのように推移したのかを示すグラフである．折れ線グラフが形勢の推移を表していて，上にいくほど先手有利，下にいくほど後手有利を示す．中盤のあたりでは，折れ線がわずかに先手有利側に触れており，激指が苦しい展開であったことが分かる．▲3 三銀不成を指した局面は，図中の 2 本の矢印で指されている点が対応している．クエスチョンマークがついているのは激指がその手を悪手だと考えていることを示している．図の下半分にはその局面前後での激指の読み筋が表示されているが，激指は，▲3 三銀不成のところでは，相手の指し手は▲3 三銀成だと予想していて，その後の展開は，ほぼ互角（マイナス 49 点：歩の価値の半分程度の差）だと考えていることが分かる．コンピュータ将棋に長いことかかわっていると，この図のグラフを見ても分かるように，将棋というゲームは，良い手を指して勝つというゲームではなく，悪い手を指すことで負けるゲームなのだということを実感する．

コンピュータ将棋と情報科学

激指では以前から探索アルゴリズムの部分で機械学習を利用していたのだが，評価関数にまで機械学習を導入したことで，完全に機械学習に魂を売ってしまったようで悔しい気がしなくもない．YSS の作者の山下氏が，自動学習された評価関数による YSS の将棋を見て，「自分のプログラムではなくな

ったような気がする」と嘆息していたのには同感だ．

ただ，情報科学に関する研究という意味では，コンピュータ将棋が，機械学習という情報科学の最もホットな研究分野の１つと結びつくことで，さまざまな興味深い知見が得られるようになってきたことは喜ぶべきことである．保木氏による研究成果をはじめとして，コンピュータ将棋発のいろいろな研究成果が，今後，情報科学の他の分野で活かされることを期待したい．また同時に，職人芸的な手作業の積み重ねがあまり必要なくなることで，汎用的な情報科学の研究対象としてのコンピュータ将棋の魅力がさらに高まっているといえるのではないだろうか．

参考文献

1) 保木邦仁：局面評価の学習を目指した探索結果の最適制御，第 11 回ゲームプログラミングワークショップ（GPW-06），pp.78-83 (2006).
2) 棚瀬　寧：棚瀬将棋の技術背景，情報処理，Vol.49, No.8, pp. 987-992 (Aug. 2008).
3) 金子知適：最近のコンピュータ将棋の技術背景と GPS 将棋，情報処理，Vol.50, No.9, pp.878-884 (Sep. 2009).
4) Bottou, L. : Online Learning and Stochastic Approximations, In Saad, D., editor, Online Learning and Neural Networks, Cambridge University Press, pp.9-42 (1998).
5) Collins, M. : Discriminative Training Methods for Hidden Markov Models : Theory and Experiments with Perceptron Algorithms, In Proceedings of EMNLP, pp.1-8 (2002).

（平成 22 年 5 月 31 日受付）

鶴岡　慶雅（正会員） tsuruoka@jaist.ac.jp

2002 年東京大学大学院博士課程修了．博士（工学）．英国マンチェスター大学研究員等を経て 2009 年より北陸先端科学技術大学院大学准教授．機械学習を用いた自然言語処理，テキストマイニング等に関する研究に従事．

4 大規模クラスタシステムでの実行 ―GPS 将棋の試み―

田中 哲朗
東京大学情報基盤センター

金子 知適
東京大学大学院総合文化研究科

大規模クラスタ化の動機

大関や横綱に昇進した力士は直後の場所で活躍できないことが多いとよく言われる．昇進直後はスポンサーなどに呼ばれたり，取材を受けるのに忙しく稽古の暇が取れないことが原因であると言われるが，世界コンピュータ将棋選手権でも同様の傾向があることが知られている．

連続優勝は 2000 ～ 01 年の IS 将棋以降はなく，ここ 5 年は，

2004 年優勝 YSS → 2005 年 4 位
2005 年優勝激指 → 2006 年 5 位
2006 年優勝 Bonanza → 2007 年 4 位
2007 年優勝 YSS → 2008 年 4 位
2008 年優勝激指 → 2009 年 6 位

と優勝ソフトのシード権 (3 位以内) 獲得失敗が続いていた．

力士と違って，コンピュータ将棋の場合は「稽古不足で弱くなる」ことはないが，年々レベルアップする中では「伸びが小さい」だけでもすぐに抜かれてしまうからであろう．

GPS 将棋は昨年の第 19 回世界コンピュータ将棋選手権で初優勝を遂げた．運にも恵まれたが，2003 年の初参加以来，実現確率探索，機械学習による評価関数のパラメータ調整など有望そうな技術を取り入れるとともに独自の工夫も取り入れていった地道な努力が実を結んだ優勝と言えるだろう [1]．

最近難しくなってきた連続優勝を達成すべく，この 1 年間，GPS 将棋に評価関数の精度の向上，詰将棋の並列化などさまざまな改良を加えてきた．しかし，コンピュータ将棋の連続対戦場である Floodgate[☆1] では，GPS 将棋は Bonanza, 激指等の他の強豪プログラムに対しては負け越すことが多かった．

- 比較的遅いマシン（Opteron 248, 2.2GHz）で走らせた逐次プログラムを Floodgate に常駐させている．
- 評価関数の調整などは一手 30 秒の持ち時間で行っているので，持ち時間 1 局 15 分の Floodgate での強さとはマッチしていない可能性がある．

などの事情はあるが，他の強豪プログラムが急激な勢いで進歩していることは明らかだった．

そこで，GPS 将棋の棋力の大幅な向上を実現するために，従来から用いていたマルチスレッドによる並列化に加えて大規模クラスタ環境での並列化を適用することを試みた．本稿では並列化の詳細と，作成したシステムによる世界コンピュータ将棋選手権参加で得られた知見を述べる．

クラスタ上でのゲーム木探索

主記憶を共有するマルチコアマシン上でスレッドを複数用いて，ゲーム木探索を効率的に行う手法としては PVS（Principal Variation Splitting）[2] およびその改良のさまざまな手法が考案され，コンピュータ将棋の分野でも広く使われるようになっている．

☆1 http://wdoor.c.u-tokyo.ac.jp/shogi/floodgate.html

一方，主記憶を共有しないクラスタ環境のゲーム木探索への適用は遅れている．これは，マルチスレッドによる並列化と比較して，以下のような不利な点があるためと考えられる．

- **通信遅延**
  プロセッサと主記憶上との通信と比較して，別プロセッサ間の通信はスループットが数桁小さいが，通信遅延の方はさらに差が大きい．そのため，全体の性能を落とさないためには，細かい通信ではなく，まとまった単位での通信を行う必要がある．
- **トランスポジションテーブル共有の困難**
  合流がある木探索の場合は，トランスポジションテーブルを共有しないと同じノードを重複して探索する可能性がある．トランスポジションテーブルの参照，更新は高頻度で発生するので通信のみで実現するとオーバヘッドが大きい．
- **負荷分散の困難**
  通信遅延が大きいため，実行すべきタスクを持たないプロセッサがあり，別のプロセッサに実行可能なタスクが余っていても，タスクの割当ては瞬時には行われない．
- **不要な探索の即時中断の困難**
  枝刈りの結果，あるノード以下の木を探索することが不要になることもあるが，探索の中断と別のタスクへの割当てが瞬時に行われない．

研究レベルでは，これらの弱点を克服するための有望なアルゴリズムがいくつか発表されているが，多くのプログラムが開発されているコンピュータチェスの世界でもクラスタ並列化は一般的にはなっていない．

一方，将棋プログラムの疎結合並列計算機上の実行例は少ないながらも存在している．1997 年 2 月に開催された第 7 回コンピュータ将棋選手権には「スーパー将棋」が SR2201 というスーパーコンピュータの 8 プロセッサ構成で参加している．これは root の子供のみを並列に探索するという簡単な並列化を行ったもので，予選 2 勝 5 敗という結果に終わっている．

また，東京大学大学院情報理工学研究科の戦略ソフトウェア創造人材養成プログラムの一貫で 720 台のノート PC を接続して，将棋プログラム「激指」を使った並列探索を試みた事例がある．これはある程度のスピードアップは得られたものの，並列化のオーバヘッドが大きく，1 台で実行する逐次プログラムよりも遅い結果しか得られなかったし，一局を通じて対戦可能なプログラムは作られなかった [3]．

複数のマシンを「逐次で行う木探索を高速に実行する」目的以外に利用した例としては，第 7，8 回コンピュータ将棋選手権に登場した S1.2，S1.3 がある．これは 2 台の計算機を用意して 1 台には通常探索を実行させ，残りの 1 台には詰みの有無だけを探索させるというものである．また，第 19 回世界コンピュータ将棋選手権での文殊 [4] および，第 20 回世界コンピュータ将棋選手権での「Bonanza Feliz」で使われた合議アルゴリズムも複数マシンの有効な利用法として注目を集めた．

なお，第 20 回世界コンピュータ将棋選手権では「ボンクラーズ」もクラスタ環境での並列化を実行している．Web ページによる解説☆2 を見る限りでは，本稿で述べる GPS 将棋のクラスタ並列化よりは本格的な並列化を試みているようである．大規模クラスタ環境で実行したときにどの位強くなるのか今後の展開を期待したい．

GPS 将棋のクラスタ並列化

一般には台数に比例するスピードアップを並列化の目標とすることが多いが，今回は，台数の平方根に比例するスピードアップしか実現できない並列化モデルを用いた．

木探索に関しては，「実質的なスピードアップがマシン数の平方根になる高速化は容易」と言われることが多い．これは，以下の考察に基づくものである．

- 木のルートから決められた深さまですべてのノー

☆2 「インサイド・ボンクラーズ」http://aleag.cocolog-nifty.com/blog/2010/01/post-6445.html

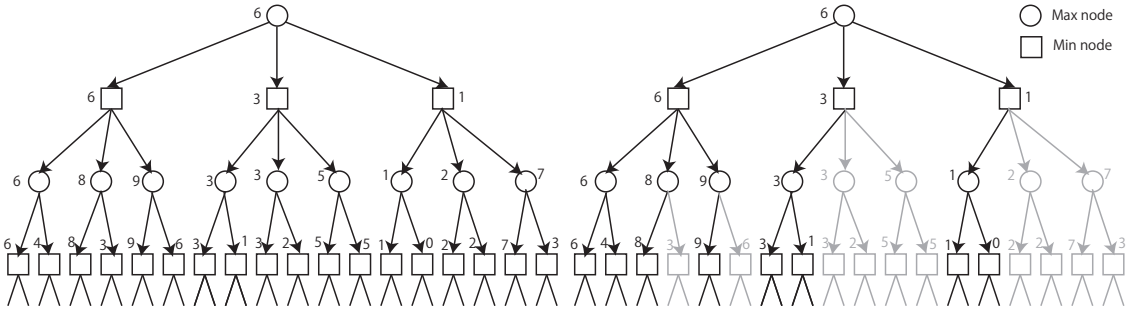


図-1　単純なミニマックス法の並列実行とアルファベータ法との比較

ドを展開して，それより深い木探索部分を別プロセスで並列に実行するという方法で容易に並列化することができる．

- このとき，決められた深さまでの展開ノードはその深さまでの単純なミニマックス法の探索ノード数と等しい．
- 理想的なアルファベータ法の探索ノード数は単純なミニマックス法の探索ノード数の平方根となる．

図-1 の左側に，この方法で深さ3まで展開してその下を並列に探索する木の例を示す．この図では深さ3の18個のノードを別プロセスの逐次プログラムで探索することになる．

この木は，左の子供ほど良い手になるように並べているが，この順でアルファベータ法で探索を行ったときに探索する部分を図の右側の黒で，不要になる部分を灰色で示す．この図では，18個のノードのうち8個のノードしか有効な探索をしていないということが分かる．展開を打ち切る深さが深くなるにつれて，有効な探索を行うノードの数はノード数の平方根に収束するので，無駄になる割合が大きくなっているわけである．

実際のゲーム木の探索では，

- ゲームの局面は木ではなく合流やサイクルのあるグラフだが，逐次探索ではトランスポジションテーブルにより合流を扱える．トランスポジションテーブルを共有せずに並列化するだけでは合流は扱えない．
- アルファベータ法では適切な探索ウィンドウ（アルファベータ法におけるα値，β値の組）が設定

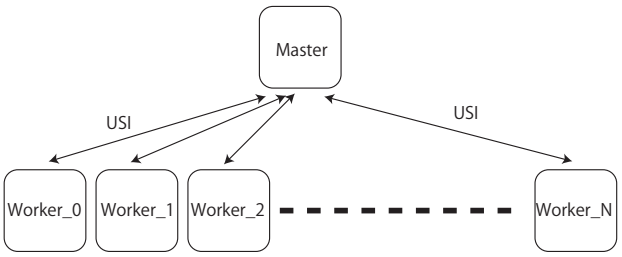


図-2　マスタ・ワーカーモデルによる実現

されているが，この並列化では探索ウィンドウが$[-\infty, \infty]$となっているので，そのノード以下の探索ノード数が増える．

ことがあるので，台数の平方根のスピードアップが得られない可能性はある．一方で，実際の探索が理想的なアルファベータ法ではないために，台数の平方根以上のスピードアップが実現される可能性もある．

このモデルに基づく並列化であるが，実際に将棋に適用するのはそれほど自明ではない．平均分岐数80の将棋でトップレベルで全幅探索を用いると，深さ2で展開しても80^2=6400プロセスが必要になるためである．

本稿における並列化では，実装を容易にするために**図-2** のようなマスタ・ワーカーモデルを用いるとともに，必要なプロセス数を抑えるための工夫を行っている．以下に概要を述べる．

- 一手ごとの探索時間は局面の進行度と持ち時間，経過時間，手数などから決定する．
- 短い時間（制限時間1秒）で，Multi PV（rootでPV以外のノードの探索のウィンドウ幅を広めにして，PVよりも少し劣る手も求める）で探索し

（以下では presearch と呼ぶ），そのときの rank（順序）に応じて，子供ノードにリソース（ワーカー群）の数を調整して割り当てる．

- presearch 中に別ワーカーを使って並列に詰み探索も行う．
- 残り時間が少ない場合は実現確率のみで候補手を生成する．
- 親から渡される「ワーカー群」のサイズが 1 になったら 1 ワーカーで残り時間いっぱいそのノードを探索させる．
- 「残りの手」は 1 ワーカーで探索を行う．

以上に対応するマスタ部分の擬似コードを**図 -3** に示す．

擬似コード中で関数 distribute として現れている子供へのワーカー群の割当は root では，$\frac{1}{4}$，$\frac{1}{4}\times\frac{3}{4}$，$\frac{1}{4}\times\left(\frac{3}{4}\right)^2$，...非 root では $\frac{1}{2}$，$\left(\frac{1}{2}\right)^2$，$\left(\frac{1}{2}\right)^3$，... のように rank に応じて台数を減らす形で行っている．**図 -4** に木の探索の際にワーカーを割り当てていく様子を示す．

これは，**表 -1** のように，presearch での rank が高い手が最終的に選択される確率が高く，rank に応じて選択確率が指数関数的に下がってきているという観察に基づいている．

```
int search(board,workers,time_left){
  if workers の数が 1
    return workers[0].search(board,time_left)
  end
  if time_left が 10 秒以上
    # ks に候補手を入れる
    Thread.new(ks = workers[0].presearch(board))
    Thread.new(cm = workers[1].has_checkmate(board,1 秒 ) )
    Thread.join()
    if cm が checkmate
      return +INF
    end
  else
    # 時間がないので実現確率のみで候補手生成
    ks = workers[0].gen_moveprobabiity(board)
  end
  if ks が 0 手
    # 負けの場合もあるが一応時間をかけて読んでみる.
    return workers[0].search(board,time_left-1 秒 )
  else if ks が 1 手で forced move
    return
      -search(board.do_move(ks[0]),time_left-1 秒 ,workers)
  end
  d=Array.new(ks.size+1)
  # workers( ただし workers[0] 以外 ) を子供たちに分ける.
  d=workers.distribute(ks)
  v=Array.new(ks.size+1)
  # その他の手を読む
  Thread.new(v[ks.size]=
    workers[0].search_other(ks,board,time_left-1 秒 ))
  for i=0 to ks.size
    Thread.new(v[i]=
      -search(board.do_move(ks[i]),d[i],time_left-1 秒 ))
  end
  Thread.join()
  return max(v) # 子供の手で最大の値を返す
}
```

図 -3　マスタプログラムの擬似コード

マスタ・ワーカー構成で実現する場合，

- マスタとワーカープログラムの起動，マスタ・ワーカー間の通信はどのようにして行うか？
- マスタとワーカーをそれぞれどのようなプログラミング言語で実装するか？

などの実装上の選択肢がある．

今回は，

- ワーカープログラムはクラスタ並列用に開発せずに，GPS 将棋の USI プロトコル GUI 用エンジンプログラムである gpsusi をそのまま用いる．
- マスタとワーカーの間の通信は同一マシン内では pipe，別マシンでは ssh を使ったストリーム通信上の USI プロトコルに基づく通信．

とした．

ここで使われた USI（Universal Shogi Interface）[☆3] はチェスプログラムの GUI プログラムと思考プログラムの間の通信に使われる UCI（Universal Chess Interface）を参考に Tord Romstad 氏が提案した将棋プログラムの GUI プログラムと思考プログラムの間の通信プロトコルであり，以下のような特徴がある．

- 通信に使われるのは 7-bit ASCII のみ．
- 通信内容は行単位で parse 可能．短くするために

☆3　http://www.glaurungchess.com/shogi/usi.html

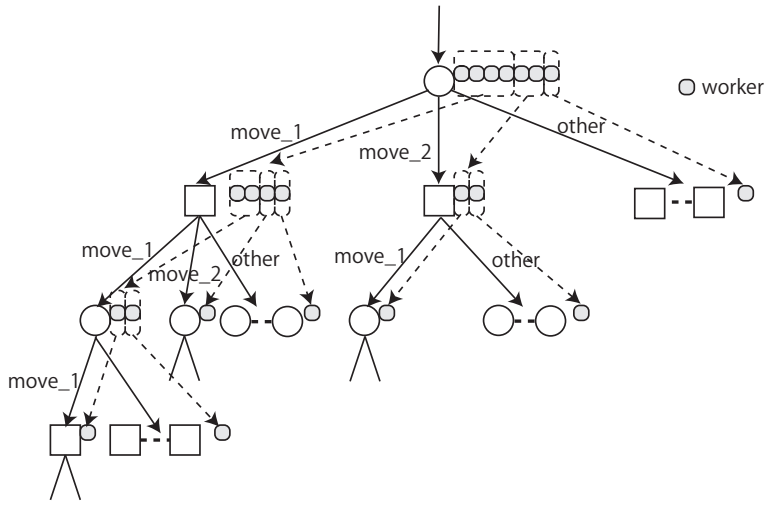


図 -4　rank による子供へのワーカー群の割当て

順位	0	1	2	3	4	5	その他
選択確率 (%)	46.3	21.4	12.0	6.65	4.70	2.78	5.93

表 -1　presearch による順位と最終的な選択確率

エンコーディング上工夫されている.

- 非同期 (asynchronous) な通信を必要とするコマンドも存在.

USI プロトコルに対応した GUI プログラムとしては,「将棋所」☆4,「プチ将棋」☆5, BCMShogi ☆6 など多数存在する. 特に「将棋所」は独自の USI 拡張をしていて, GPS 将棋もそれの最低限部分を実装した実行プログラム gpsusi を使っていた. 今回はそれに加えて, gpsusi にいくつかの拡張コマンドを実装した.

マスタプログラムは Ruby で記述した. これは以下の理由による.

- Floodgate の将棋サーバで使われている将棋盤クラス等が再利用できる.
- 実験を行うには, コンパイル時間のかからず, USI の parse が自然に書けるインタプリタ言語が適している.
- マルチスレッドプログラミングが容易に行える.

☆4 http://www.geocities.jp/shogidokoro/
☆5 http://www.geocities.jp/shogi_depot/
☆6 http://home.arcor.de/Bernhard.Maerz/BCMShogi/

一方で,

- スクリプト言語の中でも実行速度は遅い.
- 変数名のミス等はコンパイル言語ではコンパイル時に検出されるが, インタプリタなので, 実行するまで検出されない.

などのデメリットも当然ある. マスタプログラムの Ruby プログラムのうち, 今回のクラスタ並列化で新たに書いたコードは 1500 行程度である.

評価

予備実験として, Xeon X5365 (3GHz) × 2 = 8 core のサーバを使って,

- 8 スレッド使ったスレッド並列版
- 1 スレッド 8 ワーカーを使ったクラスタ並列版

を指定局面から 30 秒の秒読み先読みなしで 40 回戦わせたところ, 前者が 26 勝 14 敗と勝ち越した. 一方,

- 4 スレッド使ったスレッド並列版
- 1 スレッド 8 ワーカーを使ったクラスタ並列版

で同じ条件で対戦させたところ, 20 勝 20 敗と互角の結果を得た.

〈4〉大規模クラスタシステムでの実行

このことから，マシン 1 台ごとにスレッド並列版をワーカーとして 1 つ動かして，それをまとめたクラスタ並列構成にするという方針がまとまった．

今回は各ワーカーの能力が異なるヘテロな環境で実行したが，

- あるノードを探索する際には，まず，1 番速いワーカーで presearch を 2 番目に速いワーカーで詰み探索を行う．
- 複数の候補が見つかったあとで，「その他の手」は 1 つのワーカーでしか実行できないので，一番速いワーカーを使う．他の候補手には，2 番目に速いワーカーからラウンド・ロビンで割り当てていく．

というアルゴリズムで割当てを行った．

第 20 回世界コンピュータ選手権には，以下の構成で参加した．

- マスタ Xeon X5365（3GHz）8core 8 thread, ruby プログラムのみを動かす．
- ワーカー
  - Xeon X5570 (2.93GHz) × 2, 8core, 16 thread
  - Xeon X5470 (3.33GHz) × 2, 8core, 8 thread
  - Opteron 2376 (2.3GHz) × 2, 8 core, 8 thread
    4 台
  - Opteron 280 (2.4GHz) × 2, 4 core, 4 thread
  - Core 2 duo (2GHz), 2 core, 2 thread 307 台

構成の中で，最も台数の多い 307 台のマシン（Apple 社 iMac）の設置されている演習室 2 室のうちの 1 室の写真を**図 -5** に示す．省電力のために端末は入力がないまま一定時間経過すると，ディスプレイスリープ状態に入る（ssh で使用する分には問題ない）はずだが，図を見ると何台かはディスプレイ表示状態になっている．

この構成での動作を確認するために，人間用の問題集「ラクラク次の一手 2」[5] の問題 216 問を一手 20 秒で解かせてみた．結果は 216 問中正解が 196 問となり，Xeon X5570 × 2 のスレッド並列版で一手 30 秒の制限時間で解かせたときの正解数である 186 問よりも改善されていた．

図 -5　使用した演習室の 1 つ

次に，Xeon X5570 × 2 のスレッド並列版と上記構成から該当マシンのみを除いたクラスタ並列版を，持ち時間 25 分で指定局面からの連続対戦をさせたところ，途中でバグが出たのを除くと 13 連勝となった．

iMac は他のサーバ機と違って誤り訂正機能のないメモリを使っているし，台数も多いので，計算の二重化や故障時の再構成を考慮することが望ましい．サーバとの通信プログラムは思考プログラムと分けて，思考プログラムのプロセスがエラー等で落ちたときに，通信プログラムが再起動するような仕組みを組み込む構想はあったが，リモート参加用に新たに作った対戦プログラムにそこまで組み込む時間がなかったので，314 台のうち 1 台でも異常動作したらその場で負けが確定するという厳しい状況で決勝戦に望んだ．

結果的に，7 局無事に落ちずに対戦を終えることができたのは，運が良かったとしか言いようがない．**表 -2** に示すように，結果は 5 勝 2 敗で 3 位に終わった．

選手権の 7 局での読み筋の中の PV（Principal Variation）の長さ（静止探索，詰み探索を含まない通常探索で到達するノードの深さ）をグラフにしたものを**図 -6** に示す．4 つのグラフはそれぞれ以下を示す．

WCSC2010　今回のクラスタ版による 7 局の PV の長さ

	1回戦	2回戦	3回戦	4回戦	5回戦	6回戦	7回戦
対戦相手	芝浦将棋	激指	YSS	習甦	Bonanza Feliz	ボンクラーズ	大槻将棋
GPSの手番	後手	先手	後手	先手	後手	先手	先手
勝敗	勝	勝	勝	負	勝	負	勝

表-2　世界コンピュータ将棋選手権決勝でのGPS将棋の対戦

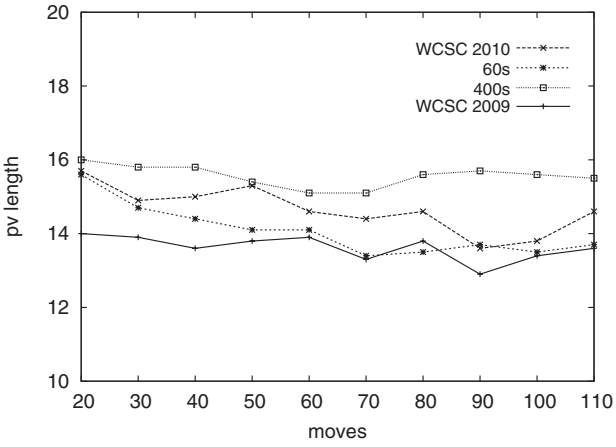


図-6　PVの長さの比較

【68手目3三桂　まで】

図-7　対ボンクラーズ戦

60s　Xeon X5570 × 2のスレッド並列で一手につき，60秒打ち切りで7局の各局面を考えさせたときのPVの長さ(選ばれた手はWCSC2010とは一致しない)

400s　Xeon X5570 × 2ののスレッド並列で一手につき，400秒打ち切りで7局の各局面を考えさせたときのPVの長さ（選ばれた手はWCSC2010とは一致しない)

WCSC2009　昨年度の決勝7局（Xeon X5570 × 2のスレッド並列)のPVの長さ

WCSC2010は最大35秒の探索でも，WCSC 2009より平均一手位PVが長いこと(厳密には違う局面を読んでいるので直接比較はできない)，60sよりはPVが長いことが分かる．一方，400sよりも探索したノードの総数が数倍多いはずだが，無駄なノードを多数探索しているため，PVの長さは400sには及んでいない．

明らかな失着としてプロ棋士から指摘を受けた局面について読みの内容を確認してみる．

図-7は第6戦ボンクラーズ戦の68手目の局面である．この局面は駒得もあり7一角成としてGPS将棋が優勢と言われていた．ここでは，60s, 400s共に7一角成を選ぶが(400sはその後で4三金2四飛の展開を読んでいる)，クラスタ版は2四飛車と仕掛ける手を選んで一気に形勢をそこねてしまった．

クラスタ版はPVとして「2四飛, 2三歩，3四飛」で評価値は431を返していた．400sの「7一角成，4三金右，2四飛」の読み筋と合わせて，クラスタ版の読みを解析して，ワーカー1つで担当するまで木を展開してみると，以下のようになった．

```
第1候補 7一角成
 + 第1候補 8六歩
 | 第2候補 9一飛
 | 第3候補 4三金右
  + 第1候補 8二馬
  | 第2候補 7五歩
  | 第3候補 7七金右
  | その他　2四飛車 △ 365
第2候補 3五歩
第3候補 6四歩
第4候補 1六歩
第5候補 9六歩
第6候補 7五歩
第7候補 3三桂成
第8候補 3三桂不成
第9候補 7七銀
第10候補 2四飛
 + 第1候補 2三歩
  + 第1候補 2九飛車
  | 第2候補 3三桂成
  | その他　3四飛車 ○ 431
 + 第2候補 2三銀
 + 第3候補 2三金
 + その他　3一玉
...
その他 6九金
```

このように，2四飛はrootで第10候補の手であり，「2四飛, 2三歩，3四飛」は探索深さが十分でないところで，得られた評価値を使っていることが

分かる．平均してある程度深く読めても，勝負どころで探索深さが足りないと致命的なミスになるということで，将棋というゲームの怖さを改めて実感した．

今後の大規模クラスタシステム

本稿で述べた大規模クラスタシステムは，「探索効率を極限まで高める」というものではなく，「簡単な枠組みでリソースをつぎ込めばつぎ込むほど強くなることを実証する」のが目的で試作されたものである．結果的には2敗して優勝を逃したことで，目的を果たせず残念な結果に終わった．ただ，今回の結果で判明した弱点も，探索深さが足りないときに再探索を行うなどの改良である程度は克服できると期待される．

gpsusi以外であっても，USIにgpsusi同様の拡張をした思考プログラムであれば，マスタ部分を利用して同等の並列化が行えるはずである．また，今回のクラスタ並列化は通信量(および頻度)が少ないので，一般参加型の分散コンピューティングで適用できる可能性もある．

今回作成したシステムは，マスタ部分のRubyプログラム，ワーカー部分のgpsusi共にGPS将棋のソースレポジトリ☆7から入手可能である．手元に空いているクラスタがある方はぜひお試しいただきたい．

参考文献

1) 金子知適：コンピュータ将棋の新しい波：3. 最近のコンピュータ将棋の技術背景とGPS将棋，情報処理，Vol.50, No.9, pp.878-886 (Sep. 2009).
2) Marsland, T. A. and Popowich, F.：Parallel Game-tree Search, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol.PAMI-7, No.4, pp.442-452 (1985).
3) 金田憲二：多数の遊休PC上での分散ゲーム木探索，第1回大域ディペンダブル情報基盤シンポジウム，http://web.yl.is.s.u-tokyo.ac.jp/~kaneda/pub/kaneda-coe04-abst.pdf (2004).
4) 伊藤毅志：コンピュータ将棋の新しい波：4. 合議アルゴリズム「文殊」 単純多数決で勝率を上げる新技術，情報処理，Vol.50, No.9, pp.887-894 (Sep. 2009).
5) 日本将棋連盟書籍編：ラクラク次の一手2 基本手筋集，日本将棋連盟 (2003).

(平成22年5月31日受付)

☆7 http://gps.tanaka.ecc.u-tokyo.ac.jp/cgi-bin/viewvc.cgi/trunk/gpsshogi/?root=gpsshogi

田中　哲朗（正会員） ktanaka@tanaka.ecc.u-tokyo.ac.jp

1965年生．1987年東京大学工学部卒業．1992年同大学院博士課程修了．博士（工学）．現在同大情報基盤センター准教授．2009年よりゲーム情報学研究会主査．

金子　知適（正会員） kaneko@graco.c.u-tokyo.ac.jp

東京大学大学院総合文化研究科助教．2008年よりゲームプログラミングワークショップ共同プログラム委員長．