Vol.2010-HPC-126 No.27
2010/8/4

分散並列モンテカルロ木探索フレームワークの提案

美添一樹[†1]　石川裕[†1,†2]

モンテカルロ木探索を並列化するためのフレームワークを提案する．逐次でのモンテカルロ木探索の実装を持っている利用者に対し，容易に分散並列環境を活用する手段を提供することを目的とする．並列化手法として Transposition table Driven Scheduling を用いることにより，高い並列度でも有効なフレームワークが作成できると期待される．

A Proposal for Distributed Parallel Monte Carlo Tree Search Framework

KAZUKI YOSHIZOE[†1] and YUTAKA ISHIKAWA[†1,†2]

We propose to implement a framework for Monte Carlo Tree Search (MCTS) on distributed parallel systems. The objective is to facilitate the parallelization of existing sequential Monte Carlo Tree Search programs. It is expected to be effective for highly parallel distributed systems by using Transposition table Driven Scheduling as the parallelization technique for the framework.

1. はじめに

探索アルゴリズムは並列化の実装が困難であることが多く，分散環境での並列化のためにライブラリなどがあれば有用である．例えば alpha-beta 探索を用いたゲーム木探索のための分散並列化ライブラリが存在し[2]，逐次での alpha-beta 探索の実装を少ない労力で並列化することが可能である．

本論文ではモンテカルロ木探索の並列化のための分散並列木探索フレームワークを提案する．逐次版のモンテカルロ木探索を実装している利用者が，逐次版のプログラムの部品を再利用して並列計算機を容易に活用できるようにすることが目的である．

モンテカルロ木探索 (Monte Carlo Tree Search, MCTS) は，それまで有効なアルゴリズムが知られていなかった囲碁においてコンピュータの棋力を飛躍的に向上させ，注目を集めた[19]．囲碁の棋力を向上させただけではなく，他の多くのゲーム (多人数ゲーム，二人ゲーム，一人ゲームを含む) にも有効であることが示されつつあり[15),16),20]，さらにプランニング[12]，バイオメトリックシステムのセキュリティ評価[17] 等においても性能を発揮することが報告されており，高性能かつ汎用性の高い探索アルゴリズムとして期待できる．そのため，モンテカルロ木探索の並列化を支援するフレームワークを作成することは有用である．

モンテカルロ木探索の並列化手法として，現在はルート並列化などの単純な並列化手法が主に研究されているが，これは並列度が高まった際に性能向上が得られるかどうか不明確である．そこで，提案するフレームワークでは並列化手法として Transposition table Driven Scheduling (TDS) を用いる．これは Iterative Deepening A* (IDA*)[11),21] 探索などの並列化で高性能を発揮することが知られている手法である[13),14]．

本論文では，新たな並列探索アルゴリズムとして TDS-モンテカルロ木探索 (TDS-MCTS) を提案しそのアルゴリズムについて述べる．また，TDS-モンテカルロ木探索を用いた分散並列モンテカルロ木探索フレームワークを提案する．

2. 背景

2.1 モンテカルロ木探索

2.1.1 確率分布に基づく探索

代表的な木探索アルゴリズムとしては，最短経路を求める A*探索[21] や二人ゼロ和完全情報ゲームを解くための alpha-beta 探索[9] などがある．多くの探索アルゴリズムはグラフ構造の中を移動しながら，最善の評価値を持つ節点を発見することを目的としている．

対して，モンテカルロ木探索[4),5),10] は探索木の各節点が単純な評価値ではなく確率分布を持つことを想定した探索アルゴリズムである．特に，今まで有効なアルゴリズムが知られていなかった囲碁に対して有効であることで注目を集めた[19]．

実際には探索中に一番有望な末端節点を選択し，そこからランダムシミュレーションを行い，その結果として確率分布を得ることによって探索を行う．統計的な基準に基づいて根

†1 東京大学大学院
Graduate School of The University of Tokyo
†2 東京大学情報基盤センター
Information Technology Center, The University of Tokyo

Vol.2010-HPC-126 No.27
2010/8/4

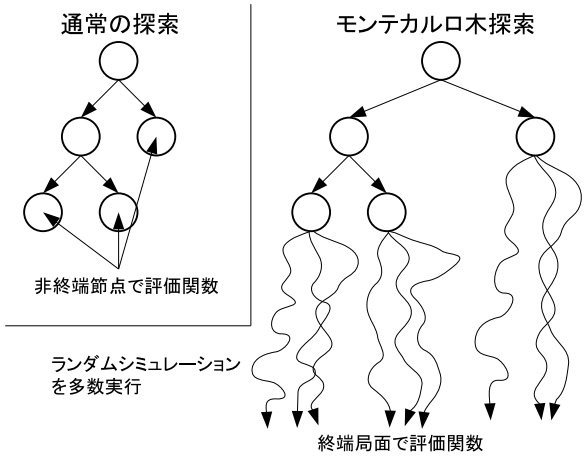

図 1 モンテカルロ木探索と通常の探索アルゴリズムの違い

節点から順番に一番有望な子節点を選択して末端節点まで到達し，そこからランダムシミュレーションを行う．ここで用いられるシミュレーションのことを，ゲーム AI の世界ではプレイアウト (playout) という造語で呼ぶ．

図 1 に示すように，通常の探索アルゴリズムは非終端節点で評価関数を呼び，節点の評価値を得る．そのためには高速で正確な評価関数が用意されている必要がある．対してモンテカルロ木探索では，一般的にはランダムに終端節点まで局面を進めてから評価を行う．終端節点での評価は正確であることが期待されるため，途中の節点を正確に評価できない問題に対しても有効である．囲碁で用いられているランダムシミュレーション (プレイアウト) の例を図 2 示す．左側の局面を評価するために，双方のプレイヤーにランダムに終局までプレイさせて結果を得る．

囲碁にはオセロ，チェス，将棋などの他の多くのゲームと異なり，正確かつ高速な評価関数を作成するのが難しいという性質がある．そのため，min-max 探索 (alpha-beta 探索) が有効であるはずの二人ゼロ和完全情報ゲームであるにも関わらず，コンピュータの強さはアマ初段以下にとどまっていたが，2006 年のモンテカルロ木探索の提案以降急激に棋力が向上し注目を集めた[19]．

2.1.2 有望な確率分布とは

モンテカルロ木探索で問題になるのは，有限の計算時間の中で，できるだけ有望な確率分布を発見するための統計的な基準をどのように定めるかということである．これに対して，Multi-Armed Banit Problem の理論[1] を背景とした UCB1 (Upper Conficence Bound 1) という方式を用いた，UCT (Upper Confidence bound applied to Trees) アルゴリズムが広く用いられている[10]．

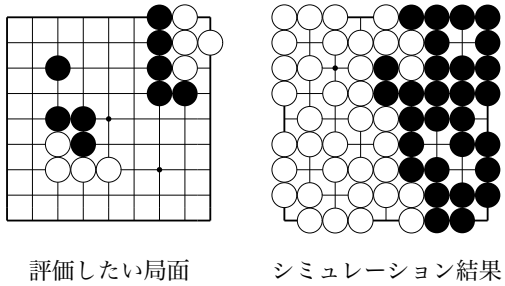

図 2 囲碁におけるランダムシミュレーション (プレイアウト) の例

ある節点において，もっとも有望な子節点を選択することを考える．その節点以下で行われたシミュレーションの総数を t とする．また，ある子節点以下で行われたシミュレーションの回数を s とし，その子節点から得られた確率分布の平均を μ とする．この場合にその子節点の UCB1 値は以下のような式で示される．

$$\mu + \sqrt{\frac{2 \ln t}{s}}$$

すべての子節点についてこの値を計算し，最大の値を持つ子節点でシミュレーションを行う．

図 3 にモンテカルロ木探索の擬似コードを示す．doSimulation(**node** *n*) がランダムシミュレーションを行う関数であり，この関数を探索対象に応じて交換することにより多くの問題にほぼ同等のアルゴリズムを適用することができる．また，selectBestChild(**node** *n*) を変更すれば UCB1 値以外の基準を用いることが可能である．

2.1.3 alpha-beta 探索とモンテカルロ木探索の並列化比較

A*探索や alpha-beta 探索では，探索中に初めて訪問する節点で評価関数を呼ぶ．対して，モンテカルロ木探索では評価関数の代わりにランダムシミュレーションを行う．

囲碁におけるランダムシミュレーション (プレイアウト) とは，与えられた局面から乱数に基づいて終局まで手を進めることである．実装に依存することなので一概には言えない

```
struct tree_node {
  //節点を一意に特定するためのデータ構造
  ...
}
struct entry {
  //ハッシュ表に格納するデータ
  tree_node n;
  child_node children[MAX_CHILDREN];
  //その他，累積評価値と累積シミュレーション回数など
  ...
}
struct child_node {
  //子節点の累積評価値と累積シミュレーション回数など
  ...
}

double UCTSearch(tree_node n) {
  entry e = LookUpHashTable(n);
  if (e.simulationNum >= threshold) {
    doSimulation(n);
  }

  tree_node best_child = SelectBestChild(n);
  double value = UCTSearch(best_child);
  return value;
}
```

図 3 モンテカルロ木探索 (UCT アルゴリズム) の擬似コード

が，現在用いられている手法の場合，初期局面から終局まで 1 回ランダムシミュレーションを行うためにかかる時間は，Core2Duo 2.0 GHz 程度の CPU の 1core を用いた場合，19 路盤では 1 ミリ秒程度のオーダーである．これは例えば将棋における評価関数の実行時間と比較すると 1000 倍程度のオーダーで長い．

探索中に展開される探索木の節点の数は評価関数やランダムシミュレーションの呼び出し回数に比例するので，結果として，alpha-beta 探索とモンテカルロ木探索を比較すると同程度の実行時間あたりに展開される節点の数は後者の方が圧倒的に少ないことになる．また，それぞれのランダムシミュレーション同士は自明に並列である．

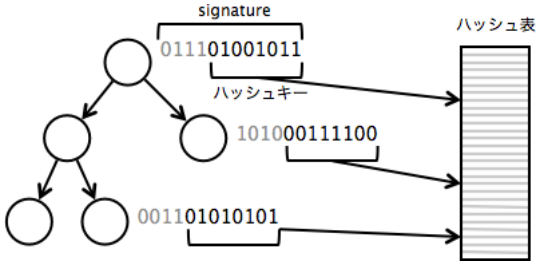

図 4 通常のハッシュテーブル

以上の要因により，alpha-beta 探索よりもモンテカルロ木探索の方が自明に並列な部分が多いため，並列化がより容易であると期待できる．

2.2 Transposition table Driven Scheduling (TDS)

2.2.1 ハッシュ表に基づいた並列探索手法

Transposition table Driven Scheduling (TDS) [13),14)] は 1999 年に Romein らによって提案された探索の並列化手法である．Transposition Driven Scheduling とも呼ばれる．Iterative Deepening A* (IDA*) 探索[11)] で 15 パズルやルービックキューブなどを解く実験では 128 ノードのクラスタを用いてほぼ 128 倍の性能向上を得るなど，非常に効率的な並列化手法として知られている．

木探索の並列化手法は，多くの場合探索木に着目した並列化を行う．子節点をそれぞれ別の計算ノードに割り当て，それぞれの部分木を並列に探索する手法である．それに対して TDS はハッシュ表を基準に並列化を行う．TDS の仕組みを説明するために，まず探索におけるハッシュ表の役割について説明する．

2.2.2 探索におけるハッシュ表 (Transposition Table) の役割

木探索 (グラフ探索) の実装においては，以下のような目的でハッシュ表を利用することが有効であることが知られている．

- 過去の探索の履歴を用いた性能向上手法 (反復深化，history heuristic など)
- 合流を検知することにより無駄な探索を減らす

探索に用いられるハッシュ表は Transposition Table (遷移表) と呼ばれる．

図 4 に探索におけるハッシュ表の利用方法を示す．各節点ごとに，Zobrist hash [18)] などのハッシュ関数を用いて 64bit から 128bit 程度の signature を計算する．得られた signature のたとえば下位 n bit をハッシュ表にアクセスする際のインデックスとして用いる．

Vol.2010-HPC-126 No.27
2010/8/4

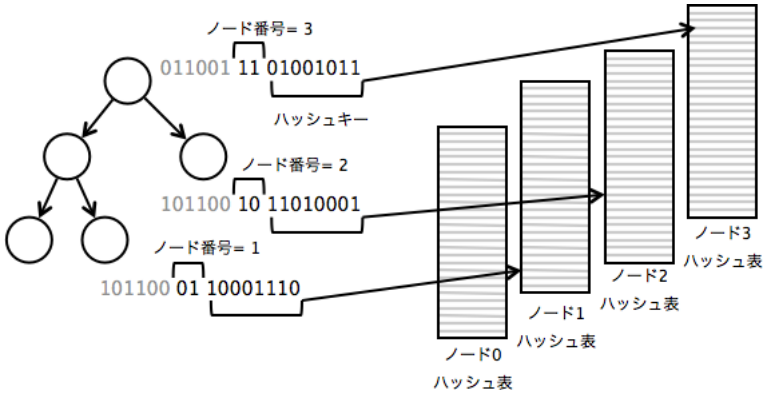

図 5 TDS におけるハッシュテーブル

分散環境において並列探索を実装する際にはハッシュテーブルをどのように保持するかが問題となる．上述のような子節点を計算ノードに振り分ける手法の場合，負荷分散の予測が外れた場合には各計算ノードが持つハッシュ表の内容を交換する必要が生じ通信量が増大するため，実装の工夫が必要である．

2.2.3 ハッシュ値による Transposition Table の分散配置

TDS ではハッシュ表を，signature の値に応じて各計算ノードに分散して配置する．一種の Distributed Hash Table を用いる手法であると言える．単純な方法としては，探索木の各節点の signature のうちの数 bit を，割り当てられた計算ノードを示す番号として用いる．各計算ノードは，自分に割り当てられた節点の情報のみを自分のハッシュ表に保持する．

図 5 はこの方式を図示したものである．子供の節点は (多くの場合) 異なる計算ノードに割り当てられているため，一段深く探索するごとに他の計算ノードに通信を行う必要がある．

一見すると通信回数が多く非効率な方式のように見えるが，実際には以下のような重要な利点がある．

- 通信はすべて 1 対 1 であり，ブロードキャストは必要としない．
- 通信を行っている間に他の計算を行うことが可能である．
- 探索木のサイズが十分に大きければ，自動的に負荷分散が行われるためハッシュ表のデータを交換する必要がない．

3. TDS によるモンテカルロ木探索の並列化フレームワークの提案

探索アルゴリズムの並列化は実装にかかる労力が大きいため，逐次版の探索アルゴリズムの部品を用いて並列探索を実現するフレームワークを用意することは探索アルゴリズムの実装者にとって有用であると期待される．実際に alpha-beta 探索の並列化ライブラリとして Asynchronous Parallel Hierarchical Iterative Deepening (APHID) [2] が公開されている．

本論文ではモンテカルロ木探索で同様のフレームワークを提供することを目的とする．並列化の手法としては Transposition table Driven Scheduling (TDS) を用いる．

3.1 TDS の動作

すべての計算ノードは基本的に同じループを回る．ループの前半では，他の計算ノードからのメッセージを受ける．後半では受け取ったメッセージの中のジョブを処理する．ジョブには子節点の探索を命令する SEARCH ジョブと，親節点へ結果を送信する REPORT ジョブの 2 種類がある．

この動作を図 6 に示す．各計算ノードは各自のハッシュ表とジョブキューを持つ．また，対応する擬似コードを図 7 に示す．図の中のそれぞれの番号は以下の説明に対応している．

- **[1]** 通信用のバッファを確認し，もしジョブが届いていればキューに格納する．
- **[2]** ジョブキューにジョブがあれば 1 つ読み込む．
- **[3]** ハッシュ表を確認し，担当する節点の情報を確認する．SEARCH ジョブなら S に，REPORT ジョブなら R1 へ進む．
- **[S]** もしその節点のシミュレーション回数が閾値より小さければ [S1a] へ，閾値以上なら [S1b] へ進む．
- **[S1a]** 現在のノードからランダムシミュレーションを行う．
- **[S2a]** ランダムシミュレーションの結果をハッシュ表に格納する．
- **[S3a]** 親節点に REPORT ジョブを送る．
- **[S1b]** 現在のノードから一番有望な子節点を選択する．
- **[S2b]** ハッシュ表の情報を更新する．
- **[S3b]** 子節点の探索を要求する SEARCH ジョブを送信する．
- **[R]** 受け取った探索結果を元に、現在の節点の情報を更新する。
- **[R1]** ハッシュ表を確認し，節点の情報を格納する．
- **[R2]** 現在の節点の情報を更新し，さらに親節点へ結果を送信する．

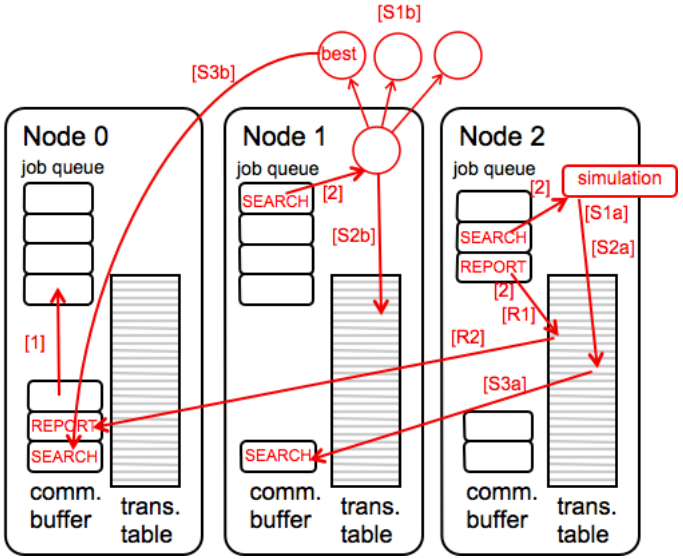

図 6 TDS の動作

3.2 フレームワークの利用方法

利用者が用意する必要があるものは以下の通りである．

- **struct tree_node**: ゲームのルールに応じて，節点のデータを定義する．逐次版とほぼ同じでよい．必須．
- **double** doSimulation(**tree_node** *n*): 節点 *n* を引数としてシミュレーションを行い，評価値を返す関数．逐次版とほぼ同じでよい．必須．
- **tree_node** selectBestChild(**tree_node** *n*): 節点 *n* の子節点の中から一番有望な子節点を選び，その子節点の tree_node を作って返す関数．逐次版とほぼ同じでよい．必須．
- **job_packet**: ジョブのデータ構造．必要に応じて項目を追加する．
- **entry**: ハッシュ表のエントリのデータ構造．必要に応じて項目を追加する．

囲碁プログラムで用いられているその他のテクニックにも対応するために，実際にはさらに柔軟に拡張可能なようにする予定である．また，並列度が大きくなった場合などには，性能向上のために調整すべきパラメータが多数存在する．利用者から設定可能とするか，自動的に適切な値をフレームワーク側で設定するようにするなどの対応が考えられるが，これは

```
void TDS_MCTS () {;
  do {
    job_packet job;
    if (ReceiveJob(&job) true) {                          [1]
      EnqueueJob(job);
    }
    if (PopJobQueue(&job) true) {                         [2]
      tree_node n = job.tree_node;
      entry e = LookUpHashTable(n);                       [3]
      switch(job.type) {
      case SEARCH:
        if (e.simulationNum <= threshold) {               [S]
          double value = doSimulation();                  [S1a]
          WriteToHashTable(e, value);                     [S2a]
          SendResult(HomeProcessor(n.parent), value);     [S3a]
        } else {
          tree_node best_child = SelectBestChild(n);      [S1b]
          WriteToHashTable(e, value);                     [S2b]
          SendNode(HomeProcessor(best_child), best_child); [S3b]
        }
        break;
      case REPORT:
        double value = UpdateValue(n);                    [R]
        WriteToHashTable(e, value);                       [R1]
        SendResult(HomeProcessor(n.parent), value);       [R2]
        break;
      }
    }
  } while (!notFinished());
}
```

図 7 Pseudo-code of TDS-MCTS

今後の検討課題である．

4. 関 連 研 究

4.1 探索並列化のフレームワーク

Brockington らによって提案された Asynchronous Parallel Hierarchical Iterative Deepening

(APHID) [2] は alpha-beta 探索の並列化手法であり，この手法に基づいた並列探索ライブラリが提供されている．利用者が評価関数やゲーム木を作成する関数を実装すれば分散並列環境で並列に alpha-beta 探索を動作させることができるようになっている．性能向上はゲームや環境に依存するが，オセロプログラムを並列化した実験では 64 ノードで 25 倍程度の速度向上を得ている．

4.2 モンテカルロ木探索の並列化

囲碁プログラムにおいて広く用いられている手法はルート並列化と呼ばれる手法である[3),6)]．これは単に異なる乱数を用いた探索を各計算ノードで行い，定期的に結果を集計することによって性能向上を得ようとするものである．モンテカルロ木探索では逐次の探索でも長時間続けて探索するよりも，同じ時間を時分割して異なる乱数で複数回探索し，結果を合計する方が性能が良くなるという報告がされている．これがルート並列化が効果を発揮する理由の一つだと推測される．

合議とは 2008 年頃から将棋プログラムの並列化手法の一つとして小幡らにより提案された手法である[22)]．異なる将棋プログラムを複数実行し，その結果の多数決や楽観的合議 (一番評価値の高い手を返したプログラムの手を採用する手法) を用いて対戦を行うものである．これを囲碁の並列モンテカルロ木探索に応用したところ，上記のルート並列化を上回る性能を上げたことが副島らによって報告されている[23)]．

公表されている実験結果は最大 16 から 64 ノード程度での並列化にとどまるものの，囲碁プログラム MoGo の開発チームによって最大 3,000 コアを用いての人間との対戦も行われた．これは文献[3)] の方式を元にした並列化だと思われる．

これらの手法は少なくとも数十コアの並列度までは効果があることが知られているが，100 ノード以上の計算ノードを利用した場合に実際に性能向上が得られるのかどうか不明確なままである．

4.3 TDS による探索の並列化

Transposition table Driven Scheduling による IDA* 探索の並列化は Romein らによって提案された[13),14)]．128 ノードのクラスタを用いることにより，問題によっては IDA* 探索の性能を 128 倍以上に向上させることに成功している．(台数以上の性能が発揮された理由は，ハッシュ表に用いたメモリサイズが逐次探索の場合と比較して 128 倍になったためである．)

岸本らは alpha-beta 探索に TDS を応用した TDSAB を提案し[8)]，64 ノードを用いて 20 倍から 25 倍程度の性能向上を達成した．これは alpha-beta 探索の並列化としては十分効率が良いと思われる．また，岸本らは A* 探索を用いたプランニングのソルバを TDS を用いて並列化し，既存の並列プランニングソルバを大きく上回る速度性能を達成した[7)]．

5. ま と め

逐次版のモンテカルロ木探索の実装者が分散並列環境を利用することを容易にするために，モンテカルロ木探索の分散並列化のためのフレームワークを提案した．並列化手法としては Transposition table Driven Scheduling を用いる．本論文では提案するフレームワークに用いる並列探索アルゴリズムである TDS-モンテカルロ木探索 (TDS-MCTS) について述べた．今後は本フレームワークを分散並列環境上に実装し，仮想的なゲーム木について性能向上の測定を行うことを計画している．

参 考 文 献

1) Auer, P., Cesa-Bianchi, N. and Fischer, P.: Finite-time analysis of the multi-armed bandit problem, *Machine Learning*, Vol.47, pp.235–256 (2002).
2) Brockington, M. and Schaeffer, J.: APHID: Asynchronous Parallel Game-Tree Search, *Journal of Parallel and Distributed Computing*, Vol.60, pp.247–273 (2000).
3) Chaslot, G., Winands, M. and vanden Herik, H.: Parallel monte-carlo tree search, *In Proc. Computers and Games 2008 (CG 2008)*, Lecture Notes in Computer Science, Vol. 5131, Springer, pp.60–71 (2008).
4) Coulom, R.: Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search, *Proc. 5th International Conference on Computer and Games*, pp.72–83 (2006).
5) Gelly, S., Wang, Y., Munos, R. and Teytaud, O.: Modification of UCT with patterns in Monte-Carlo Go, Technical Report 6062, INRIA (2006).
6) Kato, H. and Takeuchi, I.: Parallel Monte-Carlo Tree Search with Simulation Servers, 第 13 回ゲーム・プログラミングワークショップ (GPW2008)，pp.31–37 (2008).
7) Kishimoto, A., Fukunaga, A. and Botea, A.: Scalable, Parallel Best-First Search for Optimal Sequential Planning, *In Proc. 19th International Conference on Automated Planning and Scheduling (ICAPS-2009)*, pp.201–208 (2009).
8) Kishimoto, A. and Schaeffer, J.: Distributed Game-Tree Search Using Transposition Table Driven Work Scheduling, *In Proc. 31st International Conference on Parallel Processing (ICPP'02)*, pp.323–330 (2002).
9) Knuth, D.E. and Moore, R.W.: An analysis of Alpha–Beta Pruning, *Artificial Intelligence*, Vol.6, No.4, pp.293–326 (1975).
10) Kocsis, L. and Szepesvári, C.: Bandit Based Monte-Carlo Planning, *17th European Conference on Machine Learning (ECML 2006)*, Lecture Notes in Computer Science, Vol.4212, Springer, pp.282–293 (2006).
11) Korf, R.E.: Depth-first iterative-deepening: An optimal admissible tree search, *Artificial In-*

telligence, Vol.27, No.1, pp.97–109 (1985).
12) Nakhost, H. and Müller, M.: Monte-Carlo exploration for deterministic planning, *In Proc. of the 21st international jont conference on Artifical intelligence (IJCAI'09)*, pp.1766–1771 (2009).
13) Romein, J.W., Bal, H.E., Schaeffer, J. and Plaat, A.: Analysis of Transposition-Table-Driven Work Scheduling in Distributed Search, *IEEE Trans. Parallel Distrib. Syst.*, Vol.13, No.5, pp. 447–459 (2002).
14) Romein, J. W., Plaat, A., Bal, H. E. and Schaeffer, J.: Transposition Table Driven Work Scheduling in Distributed Search, *In 16th National Conference on Artificial Intelligence (AAAI'99*, pp.725–731 (1999).
15) Schadd, M. P.D., Winands, M. H.M., vanden Herik, H.J., Chaslot, G. M. J.B. and Uiterwijk, J. W. H.M.: Single-Player Monte-Carlo Tree Search, *In Proc. Computers and Games (CG2008)*, pp.1–12 (2008).
16) Sturtevant, N. R.: An Analysis of UCT in Multi-player Games, *In Proc. Computers and Games (CG2008)*, pp.37–49 (2008).
17) Tanabe, Y., Yoshizoe, K. and Imai, H.: A Study on Security Evaluation Methodology for Image based Biometrics Authentication Systems, *In Proc. of IEEE Third International Conference on Biometrics: Theory, Applications and Systems (BTAS09)* (2009).
18) Zobrist, A.: A new hashing method with applications for game playing, Technical report, Department of Computer Science, University of Wisconsin (1970).
19) 美添一樹：モンテカルロ木探索 : コンピュータ囲碁に革命を起こした新手法，情報処理， Vol.49, No.6, pp.686–693 (2008).
20) 佐藤佳州，高橋大介：モンテカルロ木探索によるコンピュータ将棋，第 13 回ゲーム・プログラミングワークショップ (GPW2008)，pp.1–8 (2008).
21) 松本啓之亮，森　直樹，黄瀬浩一：知能システム工学入門，コロナ社 (2002).
22) 小幡拓弥，杉山卓弥，保木邦仁，伊藤毅志：将棋における合議アルゴリズム：既存プログラムを組み合わせて強いプレイヤを作れるか?，第 14 回ゲーム・プログラミングワークショップ (GPW2009)，pp.51–58 (2009).
23) 副島佑介，岸本章宏，渡辺　治：モンテカルロ木探索の root 並列化とコンピュータ囲碁での有効性について，第 14 回ゲーム・プログラミングワークショップ (GPW2009)，pp.27–33 (2009).