社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2006－GI－16 (11)
2006／6／30

コンピュータブリッジのプレイヤモデルに基づく並列ゲーム木探索

小田和 友仁　塙 敏博　上原 貴夫
東京工科大学

概要

著者等は，先に，プレイヤのモデルに基づくゲーム木探索の方法（abcアルゴリズム）とそのコンピュータブリッジへの応用について提案した．本論文では，その実装方法について述べる．実験により，基本的な実装手段としてProbing searchがOne-pass searchより高速であることを示した．さらに，ブリッジの実時間プレイを可能とするためには，並列処理が不可欠であることを示した．

Parallel Game-Tree Search based on Player Model in Computer Bridge

Tomohito Otawa　Toshihiro Hanawa　Takao Uehara
Tokyo University of Technology

Abstruct

Authors have proposed the abc algorithm, a game-tree search based on player model, and its application to Computer Bridges. This paper describes implementations of the abc algorithm. The experimental result shows that the probing search is faster than the one-pass search as the basic implementation method. It also shows that the parallel processing is inevitable for real time play of Bridge game.

1　はじめに

現在でも多くのゲームプログラムはフォン・ノイマンが1928年に発表したミニマックス戦略に基づくゲーム木探索アルゴリズムを使っている．すなわち対戦する2人のプレイヤは，ゲームの局面について同じ評価をするものと仮定している．これに対して飯田等は，同じ局面でも敵が自分と異なる評価をする可能性があると仮定して，2人の評価値に基づくゲーム木探索（Opponent-Model Search，以下OM search）について研究している[1][2]．飯田等の研究は2人によってプレイされる完全情報ゲームを念頭においている．我々は先に4人でプレイされる不完全情報ゲームであるコントラクトブリッジにおいて，同じ局面に対する各プレイヤの評価が異なると仮定し，（ダミー1人を除く）敵，味方，合計3人の評価値に基づく新しいゲーム木探索法（abcアルゴリズム）を提案した[3]．これは飯田等のOM Searchの拡張ともいえるが，不完全情報ゲーム特有の背景と有効な利用法がある．すなわち従来のコンピュータブリッジでは難しかった上級者のプレイテクニックのいくつか（敵をだますディセプティブプレイ，味方のミスを防ぐパートナーシップなど）が可能となることを示した[3]．

本論文ではabcアルゴリズムの実装方法について述べる．OM Searchにおいては One-pass方式とProbing方式が提案されている[4]．我々のabcアルゴリズムにおいては，それぞれの方式を3種の評価値をもつゲーム木探索に拡張して実装し，実験により比較した．さらに不完全情報ゲーム特有のタスクとして，多数のサンプル（可能性のあるディール）に対して上記のゲーム木探索をおこなう必要がある．そこでブリッジの実時間プレイを可能とするために，我々が先に提案した並列処理方式[5]をこのabcアルゴリズムに適用した．

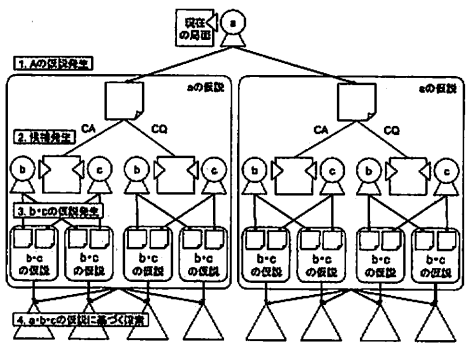


図1　abcアルゴリズムの流れ

2　abcアルゴリズム

我々は3人のプレイヤの推論を考慮した探索法として，つぎの方法を提案した[3]．

図1にabcアルゴリズムの流れを示す．

2.1　アルゴリズムの流れ

自分（ゲーム木のルートに対応するプレイヤ）を a，ダミーを除く他の2人を b，c とする．プレイヤ a がとりうる行動の候補の集合を M としたとき，つぎのようにして一つの行動（つぎに出すカード）を決定する．

[Step1] それまでのビッドおよびプレイと矛盾しないようにカードを配り，プレイヤ a から見たディー

ルの集合 $D(a)$ を作る

[Step2] 各ディール $d \in D(a)$ ごとに各行動 $m \in M$ を選んだ場合のスコア $s(m,d)$ を [Step2-1]～[Step2-3] の手順で計算する

[Step2-1] 各ディール $d \in D(a)$ ごとにそれまでのビッドおよび m を含むプレイと矛盾しないようにカードを配り，他のプレイヤ b，c の立場から見たディールの集合 $D(b,m,d)$，$D(c,m,d)$ を作る

[Step2-2] 3 つのディール $d \in D(a)$，$d_b \in D(b,m,d)$，$d_c \in D(c,m,d)$ の組合わせで，行動 $m \in M$ を選んだ結果として得られるスコア $s(m,[d,d_b,d_c])$ をダブルダミーで計算する．ただしプレイヤ a，b，c はそれぞれディール d，d_b，d_c を仮定して行動を選択するものとする

[Step2-3] 各ディール $d \in D(a)$ について $[d,d_b,d_c]$ のすべて（n 個）に対する平均値 $\sum s(m,[d,d_b,d_c])/n$ を求め，$s(m,d)$ とする

[Step3] $\sum_d s(m,d)$ が最大となるような行動 m を選ぶ

2.2 3 つの評価関数の定義

[Step2-2] では各プレイヤは異なる仮説に基づいて局面を評価するものとし，ゲーム木の同じ局面について 3 つの評価値を対応させた．我々は（a を max プレイヤと仮定し）a の評価関数 ma をつぎのように定義した．

[ma の定義]

- P が a のプレイする max ノードなら
  $ma(P) = max_i ma(P_i)$
- P が a の敵がプレイする min ノードなら
  $ma(P) = ma(P_j)$
  ただし，j はつぎのようにして決まる
  プレイヤが b の場合，$mb(P_j) = min_i mb(P_i)$
  プレイヤが c の場合，$mc(P_j) = min_i mc(P_i)$
- P が a の味方がプレイする max ノードなら
  $ma(P) = ma(P_j)$
  ただし，j はつぎのようにして決まる
  プレイヤが b の場合，$mb(P_j) = max_i mb(P_i)$
  プレイヤが c の場合，$mc(P_j) = max_i mc(P_i)$
- P がターミナルノードなら
  $ma(P) = V_a(P)$ … d に基づく静的評価

プレイヤ b，c の評価関数 mb，mc は，ミニマックス戦略に対応する．

[mb の定義]

- P が max ノードなら，$mb(P) = max_i mb(P_i)$
- P が min ノードなら，$mb(P) = min_i mb(P_j)$
- P がターミナルノードなら，$mb(P) = V_b(P)$ … d_b に基づく静的評価

[mc の定義]

- P が max ノードなら，$mc(P) = max_i mc(P_i)$
- P が min ノードなら，$mc(P) = min_i mc(P_j)$
- P がターミナルノードなら，$mc(P) = V_c(P)$ … d_c に基づく静的評価

2.3 3 つの評価値を持つゲーム木の探索

Donkers は 2 つ評価値をもつゲーム木の探索法について研究している [4]．ここでは，その One-pass 方式と Probing 方式を 3 つの評価値に拡張する．この 2 つの方式について次の例題をもちいて説明する（詳細については論文 [3] を参照されたい）．

[例題 1]

図 2 において，♡6 – ♡2 – ♡K – ♡A，♣Q – ♣3 – ♣7 とプレイが進んだとき，East は何を出すべきか．

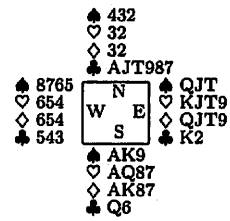


図 2 例題 1 のディールとオークション

[上級者の考え方]

知らんふりして ♣2 をプレイする．♣K のガードがなくなってしまうが，South にそれを知る余地はなく，疑う理由もない．South がもう一度フィネスをすれば，ダミーにリード権を渡せなくなり，完全に困らせることができる [6]．

[ミニマックス戦略による解]

♣K が選択される．East の視点で生成したディールの集合の例を図 3 に示す．図 4 はディール $d1$ に対するミニマックス戦略に基づく評価値（ゲーム木の一部）を表し，♣K をプレイする方が 30 点よいことがわかる．他のディールについても同様の結果になり，♣K が選択される．

[abc アルゴリズムによる解]

abc アルゴリズムをもちいた実験では，♣2 が選択された．その過程を簡単化（ディールを少なく）して説明するとつぎのようになる．ここではプレイヤ a，b，c はそれぞれ East，South，North である．

[Step1]

East の視点で生成したディールの集合を図 3 とする．

[Step2]

各ディール $d1$，$d2$ ごとに各行動 $m \in \{♣2, ♣K\}$ をとったときのスコアを計算する．

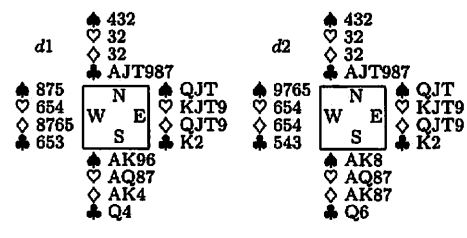


図 3 East の視点で生成したディールの集合

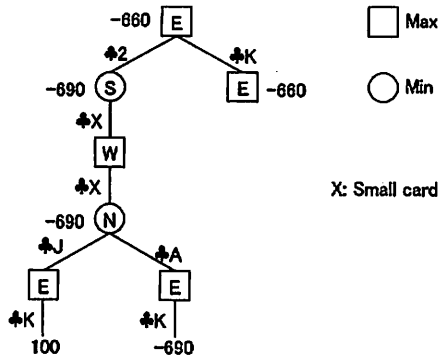


図 4　ミニマックス戦略によるゲーム木探索

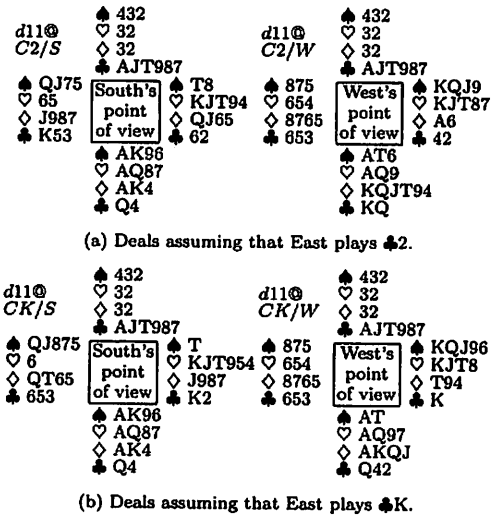


図 5　他者の視点で生成したディールの集合

[Step2-1]

ディール $d1$ について East が ♣2 を出した場合に South および West の視点で観察し生成したディールの一組を図 5(a) に示す．この場合にはディクレアラである South は ♣K が West にあると思っている．またパートナである West は ♣K が South にあると思っている．図 5(b) は East が ♣K を出した場合の例である．この場合には当然 South，West ともに ♣K が East にあることを知っている．

[Step2-2]

このステップの実装方式として，One-pass 方式と Probing 方式を以下に示す．

2.4　One-pass 方式

One-pass 方式は各ノードを 1 度しか訪れないで計算を進める方法である．この方式により探索されるゲーム木の一部を図 6 に示す．

ゲーム木のルートから左に移動する部分では，図

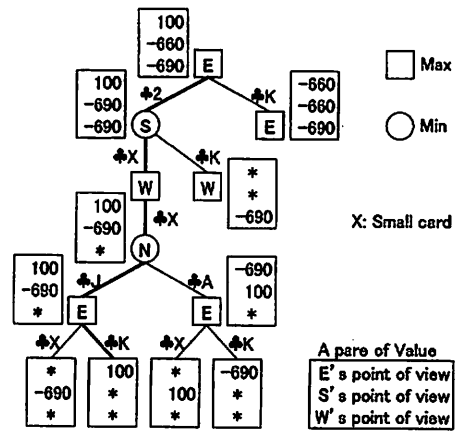


図 6　One-pass 方式によるゲーム木探索

5(a) のディールを仮定している．One-pass 方式のゲーム木では，あるノードからの移動できるノードが 3 人のプレイヤの仮定しているディールにより異なるので，そのすべてを訪れることになる．したがって，あるノード（局面）が一部のプレイヤについては到達できない（探索経過に対応するカードが仮定したハンドに存在しない）可能性がある．その場合，評価値を $*$ で表した．2.2 節の定義における $*$ の扱いは，つぎのように定義する．v は $*$ ではないとしたとき，

$$min(*, v) = v,\ min(v, *) = v,\ min(*, *) = *$$
$$max(*, v) = v,\ max(v, *) = v,\ max(*, *) = *$$

ラベル N をもつノードで，(ディクレアラである South は) North から ♣J（を出してフィネスする）か ♣A を出すかを選択する．South の視点（♣K は West と仮定）からの評価値（3 つ組みの中段）は，♣J に対して −690，♣A に対して 100 である．min プレイヤであるディクレアラは ♣J を選択する．これは max プレイヤであるディフェンスの East にとっては喜ばしい．なぜなら ♣J が選択された場合 East の視点（♣K は East にある事実を知っている）からの評価値（3 つ組の上段）は 100 であり，(通常のミニマックス戦略で) ♣A が選択された場合の評価値 −690 を大幅に上回るからである．

OM Search において飯田等は「Max ノードでの枝刈はできず，Min ノードで（$\alpha\beta$ 法でいう）β カットが可能である」ことを示している．abc アルゴリズムでも，a のプレイするノードでの枝刈は勝手にはできない．さらにプレイヤ b，c はそれぞれ独立にミニマックス戦略で探索をおこなっているので，One-pass 方式において片方のプレイヤの判断でカットすることはできない．一方，我々の設定では一部のプレイヤが到達できない（評価値が $*$ で示される）ノードがある．そこで表 1 のような $\alpha\beta$ 法のカットと $*$ の組み合わせで，全プレイヤにとって不必要な場合に枝刈をおこなった．

2.5　Probing 方式

OM Search における Probing 方式は Min プレイヤがどのノードへの移動を選択するか $\alpha\beta$ 法をもち

表 1　カットの可能な組合わせ

a の判断	b の判断	c の判断
カット or *	カット or *	カット or *

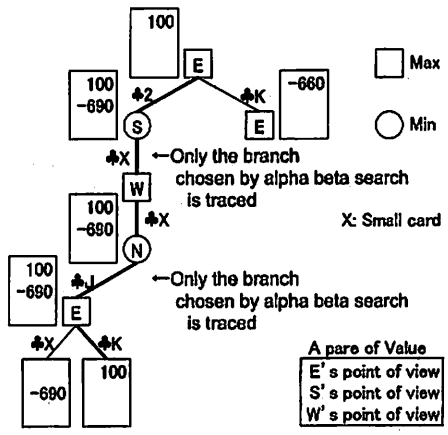


図 7　Probing 方式によるゲーム木探索

いて計算し，それに対応する Max プレイヤの値のみを保持する方式である．abc アルゴリズムでは，b あるいは c がプレイするノードに達したとき $\alpha\beta$ 法をもちいてどのノードへの移動が選ばれるか計算して，それに対応する a の計算のみをおこなう．この方式によれば，前節のようなハンドに無いカードのプレイに対応するノード（評価値 $*$）の探索をおこなう心配はない．ただしゲーム木の同じ部分を複数回探索することがある．この方式により探索されるゲーム木の一部を図 7 に示した．また，Donkers の記法に従った Probing search の擬似コードを図 8 に示す．1 行目は再帰の終了処理である．局面 h が終了条件 E に含まれるのならば，a の静的評価関数 $Va(h)$ により値を返す．2〜8 行目は局面 h が a の局面の場合である．候補の集合 L から候補 m を選んだときの局面 $h+m$ における評価値 v を再

```
AbcSearchPb(h)
1  if (h ∈ E) return [V_a(h), null]
2  if (player(h) = a)
3     L ← move(h) ; m ← firstMove(L)
4     m* ← m ; v0* ← −∞
5     while (m ≠ null)
6        [v, mm] ← AbcSearchPb(h + m)
7        if (v > v*) v* ← v ; m* ← m
8        m ← nextMove(L)
9  if (player(h) = b)
10    [w*, mm] ← AlphaBeataSearch(h, α, β, V_b(·))
11    [v*, m*] ← AbcSearchPb(h + mm)
12 if (player(h) = c)
13    [w*, mm] ← AlphaBeataSearch(h, α, β, V_c(·))
14    [v*, m*] ← AbcSearchPb(h + mm)
15 return (v*, m*)
```

図 8　Probing 方式による abc 探索

帰により求めていき，max となる評価値 v^* を求める．ここまでの処理は One-pass 方式と共通である．9〜11 行目は局面 h が b の局面である場合である．Probing ではここでまず局面 h において b が選ぶであろう候補 mm を AlphaBetaSearch 関数をもちいて求め，候補 mm を選んだときの局面 $h+mm$ について再起し，評価値 v^* を求める．12〜14 行目は局面 h が c の局面の場合であり，b の局面である場合と同様に評価値 v^* を求める．12 行目ではこうして求めた評価値 v^* とその候補 m^* を戻り値として返す．

2.6　One-pass 方式と Probing 方式の比較

One-pass 方式の評価関数と Probing 方式の評価関数をもちいてランダム木（乱数により発生させたゲーム木）を探索したときの初手の処理時間を表 2 に示す．

One-pass 方式にくらべ Probing 方式のほうがどの探索深度においても早い．

以上の説明では言及しなかったが，他のプレイヤ (b あるいは c) のカード選択時に複数の同一評価値の候補がある場合の取扱い方によりリスクが生じる．いくつかの候補の中から自分（プレイヤ a）にとって最悪のものを選んでおけば過大評価を避けられる．我々の One-pass 方式の実験では，そのように実装した．Probing 方式の実験では候補の中で最初に見つかったものを選ぶ方式も実装し，比較をおこなった．実験の結果，両者で結果に差がでるのは 15 %程度であり，後者の実行時間が前者の 3 分の 1 程度であることがわかった．3 章では実時間プレイのために多少のリスクを覚悟して後者を採用した場合のデータを示す．

2.7　応用

abc アルゴリズムが有効に働くいくつかの例を示す．

2.7.1　例題 2：パートナを迷わせない

図 9 において ◇A - ◇6 - ◇9 - ◇3，◇K - ◇7 - ◇2 - ◇4 とプレイが進んだとき，West は何をだすべきか（文献 [7] より引用）．

[上級者の考え方]

つぎに ◇ をだすと，パートナが切札で切った後 ♠ と ♣ のどちらをだしたらよいか分からないだろう．そこで，先に ♣A を取ってから ◇5 をだす．

[ミニマックス戦略による解]

♣A と ◇5 は同じに評価される．したがって偶然 ♣A を選ぶ可能性はあるが，パートナのミスを防ぐ

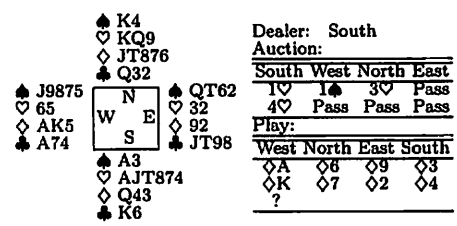


図 9　例題 2 のディールとオークション

意図は全くない．

[abc アルゴリズムによる解]

♣A が選択される．これは West がすぐ ◇5 をだしたと仮定した場合，East の視点から推論した West のハンドに ♠A があるものが生成されるからである．この場合には，パートナである East が ◇ を切札で切った後，誤って ♠ をだして失点することが [Step2-2] で計算される．West が先に ♣A をだした場合は，つぎに West が ◇5 をだすので，パートナのミスによる損はない．

2.7.2 例題 3：ディフェンダを惑わす

図 10 において West の ♣ のリードを ♣A でとったあと，ディクレアラは ◇ の何を出したらよいか（文献 [6] より引用．ただし詳細は，実験用に設定）．

[上級者の考え方]

敵にとってほしかったら連続したカードの 1 番上をだす．とらないでほしかったら，1 番下をだす．この例では，◇KQJT9 が連続しているが，◇9 をだしてディフェンスが ◇A をとりそこなうチャンスをねらう．

[ミニマックス戦略による解]

South のハンドが見えた状態でゲーム木探索をおこなうので，◇KQJT9 の評価値は同じになる．

[abc アルゴリズムによる解]

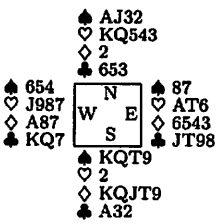
実験では（9 以下を区別しなかったので）◇T を選択した．[Step2-1] で図 11 のような仮説が生成され，[Step2-2] で West は (East が ◇QJ をもっていると思い）◇A があっても出さない可能性があると評価された結果である．

2.7.3 例題 4：シグナルを出す判断

図 12 において West の ♠ リードを ♠A でとったあと，ディクレアラは ♣6 をだした．West は何をだしたらよいか（文献 [8] より引用．ただし詳細は，実験用に設定）．

[上級者の考え方]

一般にパートナへの（カウント）シグナルの価値はディクレアラがそれから得られる利益より大きい．しかしこの例題の場合には ♣ が偶数枚であることを ♣8 でシグナルすれば，ディクレアラの推論を助けて ♣ で 4 枚とらせることになる．よって ♣2 をだす．



Dealer: South
Vul: NS
Auction:

South	West	North	East
1◇	Pass	1♡	Pass
1♠	Pass	3♠	Pass
4♠	Pass	Pass	Pass

Play:

West	North	East	South
♣K	♣4	♣9	♣A
			?

図 10 例題 3 のディールとオークション

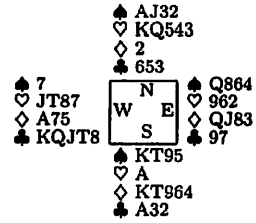
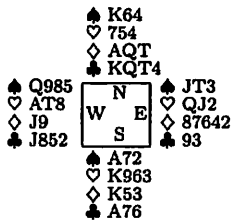


図 11 West の視点によるディールの例



Dealer: South
Vul: NS
Auction:

South	West	North	East
1NT	Pass	3NT	Pass
Pass	Pass		

Play:

West	North	East	South
♠5	♠4	♠T	♠A
?			♣6

図 12 例題 4 のディールとオークション

[ミニマックス戦略による解]

ミニマックス戦略に基づく従来の方法ではシグナルの評価はできない．商用コンピュータブリッジでは常にシグナルをだすか，全くださないかを事前に設定する．

[abc アルゴリズムによる解]

パートナも敵もカウントシグナルがだされることを予期しているモデルで生成されたディールの一例を図 13 に示した．ゲーム木探索の結果，敵に ♣ が 3 枚だと思わせる ♣2 が選択された．パートナも騙されたが悪影響はなかった [9]．

3 abc アルゴリズムの並列処理

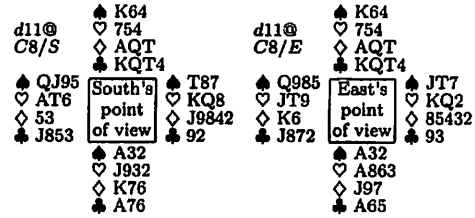
Probing を実装したコンピュータブリッジに並列処理を実装した．2.1 節のアルゴリズムにおける [Step2-2] のスコア計算 $s(m,[d,d_b,d_c])$ は完全情報ゲームとしての探索である．我々はこのゲーム木探索をタスクの単位とし並列処理を実装した．探索に必要なデータはゲーム木ごとにほぼ独立しているため，他のタスクとの間に共有情報を持つ必要がない．さらに処理順の規制もないため，非同期に分配することができる．a の仮説発生数を N_a，a の候補数を N_M，b および c の仮説発生数を N_{bc} とするとタスク数 N_T は以下の式で求められる．

$$N_T = N_a \times N_M \times N_{bc} \quad (1)$$

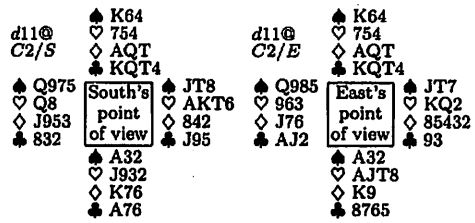
システムの構成を図 14 に示す．システムは 3 つ

表 2 ランダム木に対する平均処理時間

Depth	One-pass time(s)	Probing time(s)
12	5.91	0.53
16	170.04	7.05
20	4365.18	99.52



(a) Deals assuming that East plays ♣8.



(b) Deals assuming that East plays ♣2.

図 13　各プレイヤの視点で生成したディール

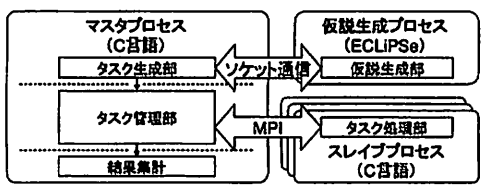


図 14　システム構成

のプロセスで成り立つ．仮説生成プロセス，マスタプロセス，そしてスレイブプロセスである．

仮説生成プロセスは 3 者の仮説生成を担当する．仮説の生成には制約論理プログラミングの手法をもちいるため，制約論理プログラミング言語である ECL^{i}PS^e で実装した．マスタプロセスとはソケット通信により同期をとる．

マスタプロセスはタスク生成とタスク分配，結果集計を担当する単一のプロセスである．スレイブプロセスはマスタの管理に従いタスクの処理に専念するプロセスであり，複数が並列に動作する．

タスクの管理にはタスクプールによる動的スケジューリングを実装した．実装にはクラスタの管理や，クラスタ間のメッセージ通信が不可欠である．そこでマスタプロセスとスレイブプロセスは MPI ライブラリである LAM をもちいて C 言語で実装した．

3.1　MPI ライブラリ

MPI(Message Passing Interface) は分散メモリ環境における並列プログラミングのためのライブラリの仕様である．我々は MPI の仕様に準拠した実装ライブラリである LAM(Local Area Multicomputer)[10] をもちいた．LAM は UNIX のデーモンとして常駐し，これを介して通信するため高速な通信が可能である．MPI の提供するプログラミングモデルは SPMD(Single Program Multi Data stream) である．すなわち全ノードがひとつのプログラムを読み込み実行する．ただし各ノードに割り当てられたランクと呼ばれる識別 ID により処理を分岐することは可能である．

我々はマスタとスレイブのプロセスに分けるためにランクを利用した．ランクが 0 のクラスタをマスタ，それ以外のクラスタをスレイブとする．

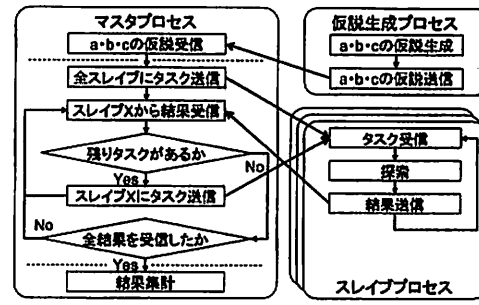


図 15　カード決定のフローチャート

3.2　仮説生成プロセス

図 15 のフローチャートに沿って流れを説明する．仮説生成プロセスは 2.1 節のアルゴリズムにおける [Step1]〜[Step2-1] における a, b, c の仮説と候補を発生し，これをまとめてマスタプロセスに送信する．

3.3　マスタプロセス

マスタプロセスは仮説生成プロセスから a, b, c の仮説と候補を複数受け取ると，これらを組にしてタスクプール（配列）に保持する．

続いてタスク管理に移る．まず全スレイブに対して 1 つずつタスクを送信する．処理が終了し，結果を送信してきたスレイブに対して，タスクプールにタスクが残っているならば次のタスクを渡す．全タスクを送り出し，その結果が帰ってくるまでこれを繰り返す．

スレイブから受け取る結果は候補のカード情報と，そのスコア $s(m,d)$ とする．全タスクの結果がそろったところで，2.1 節の [Step3] に従い集計に移る．結果として受け取ったカードの候補ごとにスコアの総和 $\sum_d s(m,d)$ を計算し，最大となるようなカードの候補 m を最善手として決定する．

3.4　スレイブプロセス

スレイブプロセスは 2.1 節の [Step2-2] を担当する．マスタから送られてきたタスクに含まれる 3 者の仮説を元にプレイの経過からすでに出されたカードを取り除き，現在の局面まで更新する．さらにタスクに付随する候補のカードをプレイした場合のスコア $s(m,d)$ を完全情報ゲームとして探索する．得られたスコアと候補のカード情報をマスタに送信する．その後は再度マスタからのタスク受け取り待ちのループに入る．

4　評価実験

100Mbps の LAN で接続された PC をもちいて評価をおこなった．並列用のすべての PC は表 3 の A

表 3　スペック

項目	A	B
CPU	Pentium4 2.0GHz	Pentium4 3.40GHz
RAM	512MB	1024MB
OS	FreeBSD 5.0	WindowsXP
言語	gcc 3.2.1	ECL[i]PS[e]5.8
Library	LAM/MPI 6.5.9	

表 4　12 枚から 1 枚の選択に要する時間

Depth	Slaves	Total Time(s)	Parallel Time(s)	Sequential Time(s)
12	64	8.64	1.33	7.31
12	32	9.52	2.22	7.29
12	16	11.48	4.18	7.30
12	8	15.59	8.29	7.31
12	4	23.84	16.54	7.31
12	2	40.34	33.03	7.31
12	1	73.23	65.93	7.30
16	64	18.02	10.73	7.29
16	32	25.73	18.44	7.30
16	16	42.66	35.37	7.28
16	8	77.00	69.70	7.30
16	4	145.89	138.59	7.30
16	2	284.04	276.71	7.34
16	1	560.69	553.38	7.31
20	64	87.47	80.14	7.33
20	32	151.81	144.50	7.31
20	16	287.06	279.77	7.29
20	8	558.95	551.65	7.29
20	4	1102.56	1095.28	7.28
20	2	2196.96	2189.69	7.27
20	1	4383.37	4376.10	7.26

に示したスペックを持ち，LAM の実行環境が整っている．1 台をマスタとし，64 台までをスレイブとしてもちいる．LAM の機能によりマスタから各クラスタにおけるプロセスの起動を制御することができる．また仮説生成プロセスの実行用に表 3 の B に示したスペックを持つ 1 台の PC を用意した．

a の仮説発生数は 16，b と c の仮説発生数は c とした．

4.1　一手の処理時間と台数効果

セカンドリード（2 トリック目の最初に出すカードの選択）にかかる処理時間を表 4 に示す．深度が深くなるにつれ，処理時間が指数関数的に増加することが確認できる．またスレイブ数を増加することにより処理時間が減少することも確認できる．

処理時間から割り出した台数効果のグラフを図 16 に示す．台数効果はスレイブ 1 台での処理時間を各スレイブ台数での処理時間で割った値である．

Depth が浅くなるほど台数効果があがらない．これは並列化できない逐次処理が並列効果のボトルネックとなるというアムダール則の顕れである．表 4 によればセカンドリードにおける逐次部の処理時間は 7.3 秒程度かかる．逐次部の処理時間のほとんどは仮説生成のための時間であり，生成すべき仮説発生数に比例する．生成すべき仮説発生数 N_D は次の式で求められる．

$$N_D = N_a + N_a \times N_M \times N_{bc} \qquad (2)$$

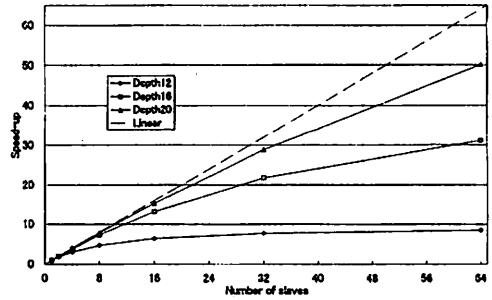


図 16　セカンドリードにおける台数効果

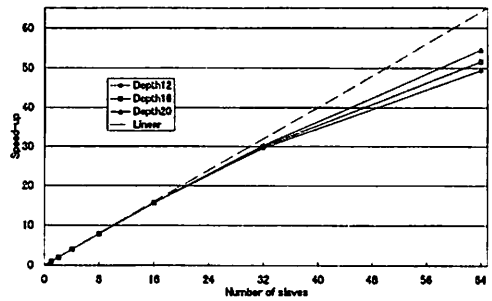


図 17　セカンドリードにおける並列部の台数効果

このうち N_a，N_{bc} はパラメータとして実行時に与える値である．逐次部の処理時間を抑えるのにこのパラメータを小さくするのであれば，得られる結果の妥当性とのトレードオフを考慮する必要がある．

逐次処理部の影響を省いた台数効果を見るため，並列処理部のみの処理時間を測定し，台数効果を割り出した．結果を図 17 に示す．どの深度もほぼ理想的な台数効果を保っている．

4.2　タスク毎の処理時間のばらつき

タスク毎の処理時間を表 5 に示す．深度 20 では最大 66 秒のタスクもあり大きなばらつきがみられた．しかし図 17 に示したスレイブ 64 台の並列処理では動的スケジューリングの効果によりうまく分散され，台数効果には響かなかった．ただしさらに台数を増やしたときに，このばらつきはネックとなる．

4.3　ゲーム進行と処理時間

ブリッジではゲームの進行により探索条件が大きく変動する．それに伴い処理時間も変動する．ゲーム進行とともに変化する処理時間を調べるため，我々のコンピュータブリッジに 4 人分をプレイさせ，各プレイヤの手番における処理時間を測定した．深度は 20 とした．ゲーム進行に伴う処理時間の変化を図 18 に示す．またゲーム進行に伴う候補数の変化を図 19 に示す．

リードプレイヤ（トリックの最初にカードを出すプレイヤ）は手札の全カードが候補となる．以降のプレイヤは基本的に最初に出されたカードのスートを出さなければならないため候補が絞られる．よっ

表 5　探索時間の区分ごとのタスク数

Range	Depth12	Depth16	Depth20
0 ～ 1	768	603	35
1 ～ 2	0	135	114
2 ～ 3	0	21	132
3 ～ 4	0	5	102
4 ～ 5	0	3	79
5 ～ 6	0	0	51
6 ～ 7	0	1	56
7 ～ 8	0	0	49
8 ～ 9	0	0	30
9 ～ 10	0	0	23
10 ～ 11	0	0	16
11 ～ 12	0	0	15
12 ～ 13	0	0	11
13 ～ 14	0	0	13
14 ～ 15	0	0	7
15 ～ 16	0	0	5
16 ～ 17	0	0	3
17 ～ 18	0	0	2
18 ～ 19	0	0	3
19 ～ 20	0	0	4
20 ～	0	0	18

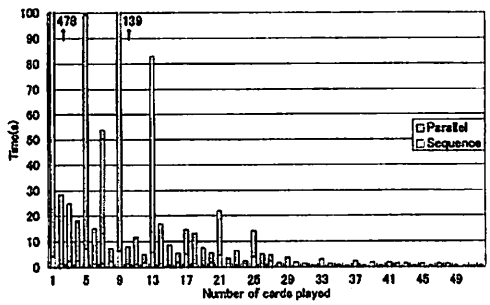


図 18　ゲーム進行と処理時間

てリードプレイヤのカード選択はそれ以降に比べ処理時間がかかる傾向にある．特にプレイ開始直後のリードは処理時間がかかる．しかしスタンダードリードという経験則に基づく約束に従うことでリスクを犯すことなく候補を絞り込むことができる．

序盤ではカードの候補数が多いため，処理時間がかかる．また出されたカード枚数が少なく確定しない情報が多いため，タスク毎の処理時間のばらつきによるオーバーヘッドも大きいと考えられる．出されたカード枚数が増加するにつれて各プレイヤの手札枚数が減少するため，出しうるカードの候補数が減少し処理時間も短くなる．後半戦は人間とリアルタイムに対戦する上で現実的な処理時間に抑えることができた．

5　おわりに

先に提案した abc アルゴリズムの実装方法について検討した．Probing 方式の探索，不完全情報ゲーム向の並列処理方式が高速化に効果があることが確認できた．また今後の改良により実時間プレイが可能となる見通しを得た．具体的な改良点としては，ディール生成部の並列化，プロセッサ数の増加，

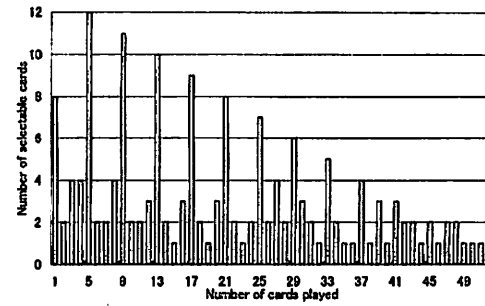


図 19　ゲーム進行と候補数

探索におけるトランスポジションテーブルの利用，ヒューリスティックな方法による候補の絞込みなどがある．

参考文献

[1] Iida, H., Uiterwijk, J.W.H.M., Herick, H.J. van den, and Herschberg, I.S.: Potential application of opponent-model search, Part1, ICCA Journal, Vol.16, No.4, pp.201-208 (1993).

[2] Iida, H., Uiterwijk, J.W.H.M., Herick, H.J. van den, and Herschberg, I.S.: Potential application of opponent-model search, Part2, ICCA Journal, Vol.17, No.1, pp.10-14 (1994).

[3] 小田和友仁，上原貴夫: コンピュータブリッジにおけるプレイヤのモデルとゲーム木探索, 情報処理学会論文誌, 投稿中 (2006).

[4] Donkers, H.H.L.M.: Searching with opponent models, SIKS Dissertation Series No.2003-13 (2003).

[5] 小田和友仁，上原貴夫: コンピュータブリッジの並列処理, 情報処理学会論文誌, 採録決定 (2006).

[6] ビクター・モロー, ニコ・ガードナー著, 難波田愈訳: カードプレイテクニック, 日本コントラクトブリッジ連盟 (1998).

[7] 桜井恒夫: ディフェンスのシグナルとカードプレイ, エスアイビー・アクセス (1998) 改訂版 (2002).

[8] Bird, D. and Smith, M.: Defensive Signal, Bridge Technique Series, Master Point Press(2000).

[9] 小林紀之, 小田和友仁, 上原貴夫: コンピュータブリッジによるカウントシグナル, 情報処理学会論文誌, Vol.45, No.6, pp.1704-1714 (2004).

[10] Trustees of Indiana University, "LAM/MPI Parallel Computing", http://www.lam-mpi.org