人　工　知　能　99−14
（１９９５．　３．　７）

C*アルゴリズムによるAND/OR木の探索および詰将棋プログラムへの応用

脊尾 昌宏

東京大学 理学部 情報科学科

概要

人工知能研究の一分野である問題解決において、与えられた問題をいくつかのより簡単な部分問題に分解して解く過程は、AND/ORグラフの探索としてモデル化できる。AND/OR木の探索で解が一意的な問題や任意の解を求めればよい問題に対して、高速に解を求めるC*アルゴリズムを開発した。これはMcAllesterにより提案された共謀数という概念に基づいており、人間の思考に近い探索手法で効率的に解を求める縦型の反復深化法である。多重反復深化法やハッシュによる動的評価などの手法により高速化している。またC*を詰将棋を解くプログラムとして実装し、「続詰むや詰まざるや」の約200題を実際に解かせる実験を行ない、人間の専門家を凌ぐ解答能力があることを確認した。

The C* Algorithm for AND/OR Tree Search and its Application to a Tsume-Shogi Program

Masahiro SEO

Department of Information Science, Faculty of Science, University of Tokyo
7-3-1, Hongo, Bunkyo-ku, Tokyo 113, Japan

email: ms-3@is.s.u-tokyo.ac.jp

ABSTRACT

An AND/OR graph represents the problem-solving process of reducing the given problem to more simple ones in a field of Artificial Intelligence. This paper presents the C* algorithm which solves fast the AND/OR tree search problems having a unique solution or when any solution suffices. It is a depth-first iterative-deepening method, based on conspiracy numbers proposed by McAllester, and solves the problems efficiently by human-like search. Such enhancements as multiple iterative deepening and dynamic evaluation by hashing are combined with it. We also implemented the C* as a Tsume-Shogi (the mating problem of Japanese Chess) solver whose ability is far superior to human experts from the numerical results of solving about 200 problems in the book "Zoku-Tsumuya-Tsumazaruya".

1 はじめに

コンピュータチェスはこれまで世界の数多くの研究者によって研究されてきた [8]。初期の段階では組合せ爆発を避けるためにチェス固有の知識を利用して探索の候補手を絞る前向き枝刈りと呼ばれる手法が用いられており、実力は初級者レベルにとどまっていた。しかしハードウェア技術の進歩に伴いすべての手を読む全幅探索が可能になり、反復深化法、静けさ探索、ハッシュ表、キラーヒューリスティックなどによる探索法の進歩およびスーパーコンピュータや並列計算機などのチェス専用マシンの開発により人間の世界チャンピオンレベルの実力に達して [8]、１９９４年８月にはついにチェス・ジーニアス２というプログラムがチャンピオンのカスパロフを負かしてしまった。

一方、日本ではコンピュータ将棋の研究がさかんになりつつある [9]。将棋とチェスの最大かつ本質的な違いは、将棋では持駒が使えるということである。このために分岐因子に大きな差が生じて、将棋では最善手の決定に十分な深さまで探索できないことが多い。ここで分岐因子というのは、１つの局面における合法手の数の平均値のことであり、チェスの分岐因子が約３５であることは以前から知られている。将棋の分岐因子については、これまで経験的に大きいと見なされてきただけであったが、最近になって約８０であることがプロ棋士の棋譜の分析により分かった [10]。

また詰将棋を解くプログラムの研究は９０年代に入って急速に進歩し、１７手詰以下のいわゆる短編と呼ばれる詰将棋に関しては人間の専門家を越える解答能力を示すようになった [14]。９２年に野下により開発されたＴ２では、ハッシュ法による同一局面の判定やいろいろな詰将棋特有の工夫により２５手詰以下の問題はほぼ解けるようになった [4]。Ｔ２を含めこれまでの詰将棋プログラムは詰手数による縦型の反復深化法を用いており、組合せ爆発により詰手数の長い問題はほとんど解くことはできなかった。しかし９２年に伊藤が開発したプログラムItoでは横型の最良優先探索を用いることにより、長手数の問題でもある程度解けるようになった [4]。Itoでは探索木の深さに関係なく詰みそうな局面から優先的に探索していくという、より人間らしい探索手法を用いている。また詰将棋の分岐因子が５程度であることが本研究の実験から分かった [17]。これは将棋やチェスに比べるとかなり少ないが、単純な全幅探索では１７手を越える中編長編の問題を解くことはできない。

以上のような状況を踏まえると、全幅探索が時間的に不可能な問題に対しては、根節点の値を決めるために重要な局面から優先的に探索していく方法を取らなければならないと考えられる。つまり良さそうな手や意味のある手をより深く読んでいくという、人間らしい思考をモデル化することが重要である。最近になって、このための手法として共謀数という概念を使った探索が研究されてきている [1] [2] [11] [16]。

今回は共謀数を使ったAND/OR木の探索アルゴリズムC*を開発し、詰将棋プログラムとして実装し、実際に問題を解かせる実験を行なった。既存の詰将棋プログラムの欠点としては、Ｔ２では手数の長い問題に対して計算時間がかかりすぎ、Itoでは変化の複雑な難問に対して記憶領域が足りなくなる、といった問題がある [4] [7]。本研究では、この２つの欠点を克服するプログラムの開発を目指した。

2 AND/OR木と詰将棋の関係

人工知能研究の一分野である問題解決において、与えられた問題をより簡単な部分問題に分解して解く過程は、AND/OR木の探索として一般化できる。詰将棋の探索空間はAND/OR木と見なすことができる。根節点 (root node) は与えられた問題局面を表していて、OR節点である。先手（攻め方）の手番の局面はOR節点に対応し、後手（玉方）の手番の局面はAND節点に対応する。詰め上がりの局面は、継続節点を持たないAND節点に対応し、解的 (solved) であるという。逆に先手番の局面でまったく王手をかけることができなくなった局面は、継続節点を持たないOR節点に対応し、非解的 (unsolvable) であるという。非終端OR節点は、継続節点のいずれかが解ければ解的になり、継続節点すべてが非解的ならば非解的になる。また非終端AND節点は、継続節点すべてが解ければ解的になり、継続節点のいずれかが非解的ならば非解的になる。探索の途中でまだ解的であるか非解的であるか定まっていない節点は、未解的 (unsolved) であるという。ここでポテンシャル解決木 (potential soluton tree) および解決木 (solution tree) の定義をしておく。

[定義1] ポテンシャル解決木 pst とは以下の条件を満たす部分木である。

1. 根節点は pst に含まれる。
2. 非終端 OR 節点 n が pst に含まれるならば、n の継続節点の1つだけが pst に含まれる。
3. 非終端 AND 節点 n が pst に含まれるならば、n の継続節点すべてが pst に含まれる。

[定義2] 解決木とは、その根節点が解的なポテンシャル解決木である。つまり先端節点 (tip node)、すなわち継続節点を持たない節点がすべて解的な終端 AND 節点であるポテンシャル解決木のことである。

詰将棋の解は解決木の形で表現できる。詰将棋の解決木は、作意手順だけでなく変化手順もすべて表現しているものである。通常、詰将棋は玉方が最善の手を尽くした作意手順を解答することになっているが、これはある解決木の根節点から終端節点までの詰手順のうち、もっとも長い詰手順に相当する。解決木は普通複数個あって、これらの解決木に共通に含まれている詰手順のうち、もっとも長い詰手順が作意手順になる。共通に含まれる詰手順が存在しない詰将棋は、余詰になる。作者が意図した解決木と異なる解決木を探索した場合、変化別詰を解答する可能性がある。

ところで、詰将棋の変種としてばか詰と呼ばれる問題があり、マニアの間では人気がある。ばか詰とは先手と後手が協力して後手玉を詰める問題である。ばか詰の探索空間は OR 木であり、解は無数に存在するので、最短手順の解を求めるルールになっており、共謀数を使ったアルゴリズムとは全く別の手法で解くことになる。

3 共謀数について

C*アルゴリズムは共謀数 (conspiracy numbers) という概念を基にしている。共謀数というのは、McAllester により提案されたもので、ミニマックス木の根節点のミニマックス値をある別の値に変えるために値を変えなくてはならない終端節点数の最小値のことである [11]。別の値に変わるための共謀数が大きいほど、根節点の値は安定であると言える。ある敷居値以上の共謀は起こらないと考えて、根節点の取りうる値が1つになるまで探索を続けるのが共謀数を使ったミニマックス木の探索アルゴリズムである。共謀数の詳細およびそれを使ったミニマックス木の探索については、参考文献を参照されたい [11][16]。

AND/OR 木における共謀数は、与えられた問題（根節点）を解くために解かなければならない部分問題（先端節点）数の最小値と定義される [1][2]。すなわちポテンシャル解決木のコストをまだ解けていない先端節点数と定義したとき、ある節点を解くための共謀数は、その節点を根とするポテンシャル解決木の最小コストになる。AND/OR 木の節点 n における共謀数 $CN(n)$ の計算法を以下に示す [1][2]。

1. n が終端 AND 節点の場合、$CN(n) = 0$
   n が終端 OR 節点の場合、$CN(n) = \infty$
   n が非終端先端節点の場合、$CN(n) = 1$
2. n が非終端 OR ノードの場合、n の継続節点を $n_1, ..., n_k$ とすると、
   $CN(n) = \min_{1 \le i \le k} CN(n_i)$
3. n が非終端 AND ノードの場合、n の継続節点を $n_1, ..., n_k$ とすると、
   $CN(n) = \sum_{i=1}^{k} CN(n_i)$

次節以降では共謀数を使った AND/OR 木の探索手法について述べる。

4 共謀数を使った探索の AO*としての一般化

AND/OR 木の最適解を求めるためのアルゴリズムとして、AO*アルゴリズムが古くから研究されている [12] [13]。最良優先探索として有名な A*アルゴリズムは、木がすべて OR 節点からなる特殊な場合の AO*と見なすこともできる。ここでは、共謀数を使った探索を AO*という一般的な探索手法の枠組でとらえることを考えてみる [2]。与えられた問題の解は根節点における最小コストの解決木として表現される。2つの節点 n_i, n_j 間の枝のコストを $c(n_i, n_j)$ で表すと、探索木内の n を根節点とする最適解決木のコストのヒューリスティックな評価値 $h(n)$ は、以下のように定義される [12]。

1. n が終端 AND 節点ならば、$h(n) = 0$.

2. n が終端 OR 節点ならば、$h(n) = \infty$.

3. n が未解的先端節点ならば、$h(n)$ は n を根節点とする最適解決木のコストのヒューリスティックな評価値となる。

4. n が継続節点 $n_1, ..., n_k$ をもつ非終端 OR 節点ならば、$h(n) = \min_{1 \le i \le k} [c(n, n_i) + h(n_i)]$.

5. n が継続節点 $n_1, ..., n_k$ をもつ非終端 AND 節点ならば、$h(n) = \sum_{i=1}^{k} [c(n, n_i) + h(n_i)]$.

上の定義で、3. の $h(n)$ の値を 1 として、4. と 5. のすべての枝のコストを 0 と定義すれば、探索木のすべての節点 n において $h(n)$ の値は $CN(n)$ と等しくなる。つまり共謀数を使った AND/OR 木の探索は、AO*アルゴリズムにおいて未展開の非終端節点のヒューリスティック関数の値を一様に 1 に定めて、枝のコストを 0 にしたものとして一般化できる。適当な静的評価関数を使って、先端節点の共謀数を評価することも当然考えられるが、そのようなアプリケーションに依存した手法を使わないのが、共謀数を使った探索の特徴の一つである。別の言い方をすれば、共謀数はヒューリスティックな評価関数を利用しない手法であると言える。

$h(n) = 1$ と評価された未解的先端節点 n をさらに展開していった結果、n が解的とならば $h(n) = 0$ となり、n が非解的ならば $h(n) = \infty$ となる。したがって共謀数は真のコストを過大評価 (overestimate) することがあり、通常ならば実行可能性 (admissibility) が保証されないところだが、求まった解のコストは常に 0 であるから、その意味で実行可能性は明らかに成立する。もし枝のコストを適当な値（例えば 1）に定めれば、共謀数が真のコストを過大評価しないようにすることが可能である [2]。このとき詰将棋で言えば、長手順への迷い込みが抑制され、短手数の問題を高速に解くことに重点を置いたアルゴリズムになり、長手数の問題に対しては性能が低下する。しかし本研究では、解の長さによる制約を一切受けない詰将棋プログラムを作るため、枝のコストは 0 にした。

また同一コストのポテンシャル解決木の展開の選択および同一ポテンシャル解決木内の展開節点の選択を深さ優先の順序にすることにより、AO*を縦型の反復深化法にできる。枝のコストを 0 にする利点として、反復深化法の再展開による再計算の無駄が少なくなる、ということがある。各反復ごとに展開される節点数が十分大きくなるからである。AO*を縦型の反復深化法にした場合、常に最小コストのポテンシャル解決木を展開していることにはならない。しかし根節点以外の OR 節点でも同様に反復深化法を使えば、常に最小コストのポテンシャル解決木の先端節点を展開していることになる。

次節で述べるハッシュによる動的評価法は、節点を展開してそのコストが変化する毎に、その節点から根節点までのコストを更新する、AO*で一般的に採られている更新手続きに対応する。

5 C*アルゴリズム

共謀数を使った探索は本来最良優先探索であり、メモリが大量に必要になる欠点がある [11] [16]。そこで共謀数のしきい値を反復毎に増やしていく縦型の反復深化法を用いた [6] [15]。また根節点だけでなく、すべての OR 節点でも同様に反復深化法を使う多重反復深化法 (multiple iterative deepening) により効率化した。AND 節点を展開したときの継続節点の共謀数の数え方としては、新たに生成した未解的 OR 節点は 1 とカウントするのだが、以前生成されて共謀しきい値 N で解けなかったことがハッシュ表に登録されている OR 節点については N とカウントする、ハッシュによる動的評価法 (dynamic evaluation by hashing) を用いた。具体的なプログラムを次に示す。

```
comment 根節点における反復深化;
procedure ITERATE:
    comment 根節点の共謀数の初期化;
    CN(r) ← 1
    while
        if (EXPAND(r, CN(r)) = 1) then
            return

comment 先端節点の展開手続き;
procedure EXPAND(n, CN(n)):

1. comment ハッシュ表の節点 n の登録データを
   探す;
   if FINDTABLE(n, c) = 1 then
      begin
         if c = 0 then return 1
         else if c ≥ CN(n) then return 0
      end
```

```
2. comment 節点 n を展開してすべての合法手を
   生成する;
   GENERATE(n)
   if n が継続節点を持たない then begin
      if n が AND 節点 then begin
         PUTINTABLE(n, 0)
         return 1
      end
      if n が OR 節点 then begin
         PUTINTABLE(n, ∞)
         return 0
      end
   end

3. comment ハッシュ表によるサイクルの検出;
   PUTINTABLE(n, CN(n))

4. comment ハッシュ表を使って各継続 OR 節点
   の共謀数をカウントする;
   if n が AND 節点 then begin
      for 各 n_i do begin
         if FINDTABLE(n_i, c) = 1 then
            begin
               if c = 0 then
                  n_i を継続節点から除去
               else CN(n_i) ← c
            end
         else CN(n_i) ← 1
      end
      comment 和がしきい値を越えた場合;
      if Σ_{i=1}^{k} CN(n_i) > CN(n) then begin
         CN(n) ← Σ_{i=1}^{k} CN(n_i)
         PUTINTABLE(n, CN(n))
         return 0
      end
      else
         CN(n) ← CN(n) − Σ_{i=1}^{k} CN(n_i)
   end

5. if n が OR 節点 then begin
      comment 継続 AND 節点の再帰呼出し;
      for 各 n_i do begin
         CN(n_i) ← CN(n)
         if EXPAND(n_i, CN(n_i)) = 1 then
            begin
               PUTINTABLE(n, 0)
               return 1
            end
      end
      comment CN(n) の更新と登録;
      CN(n) ← min_{1≤i≤k} (CN(n_i))
      PUTINTABLE(n, CN(n))
      return 0
   end

6. if n が AND 節点 then begin
      comment 継続 OR 節点の再帰呼出し;
      for 各 n_i do begin
         CN(n_i) ← CN(n) + CN(n_i)
         comment 多重反復深化法;
         while do begin
            comment n_i が解けた場合;
            if EXPAND(n_i, CN(n_i)) = 1 then
               break
            comment n_i が解けなかった場合;
            if CN(n_i) > CN(n) then begin
               comment CN(n) の更新と登録;
               CN(n) ← Σ_{j=i}^{k} CN(n_j)
               PUTINTABLE(n, CN(n))
               return 0
            end
         end
      end
      PUTINTABLE(n, 0)
      return 1
   end

comment ハッシュ表の探索手続き;
procedure FINDTABLE(n, c):
   if 節点 n の情報が登録されている then begin
      c ← table(n)
      return 1
   end
   else
      return 0

comment ハッシュ表への挿入手続き;
procedure PUTINTABLE(n, c):
   if 節点 n の情報が登録されている then
      if c = 0 or c ≥ table(n) then
         table(n) ← c
```

else

$table(n) \leftarrow c$

解が求まるまでの根節点での反復回数は、解決木内のAND節点の継続節点数の最大値以上で、かつ解決木内の終端AND節点数以下であることは自明である。共謀数を反復深化法で使った場合、反復回数はAND節点における継続節点の順序付けに依存する。一般に解決木内のAND節点ではすぐに解ける簡単な継続節点から探索し、それ以外のAND節点ではすぐに解けないことが判明する継続節点から探索するのが、解が求まるまでの反復回数が少なく、解以外の部分を探索する無駄が少ない。また以前の反復であまり探索していない継続節点から探索すると、節点の再展開率が少なくなって効率的である。AND節点での最適な順序付けについては、まだ問題が残されている [12] [13]。

6 節点間の依存関係について

ここでは共謀数を使ったアルゴリズムを詰将棋プログラムとして実装する上での問題点について考えてみる。共謀数を使った探索では、分割された部分問題は互いに独立であるということを基本的な仮定としており、それぞれの部分問題が解ける確率が互いに独立なとき、その部分問題数が多いほどそれらがすべて解ける確率は低いので、共謀数が少ないポテンシャル解決木から展開していくことは理にかなっている。しかし実際にはそのような仮定は成立しないのが普通である。ここでは節点間の依存関係と効率的な探索法について、一方の節点が他方を優越している場合と、優越関係はないが互いに類似している場合に分けて考察してみる。

節点間の優越関係（dominance relation）は次のように定義される [3]。P_2がP_1よりも良い解を持つことができない場合、P_1, P_2間の関係は$P_1 D P_2$と表現され、節点P_1は節点P_2を優越すると言う。つまりAND/OR木において優越関係$P_1 D P_2$は、もしP_2が解ければP_1は必ず解けることを表している。また逆にP_1が解けないことが分かれば、P_2は必ず解けないことを意味する。節点を展開する前にハッシュ表を利用して優越関係を調べることにより、無駄な探索を省くことができる。そのためには、例えばP_1を展開するより先にP_2を展開するといった工夫が必要である。

詰将棋において2つの局面P_1, P_2間に優越関係が成立するのは、2つの局面の（持駒以外の）盤面の状態がまったく同じで、かつ局面P_2の先手の持駒が局面P_1の先手の持駒の部分集合になっている場合である。この場合、局面P_2が解ければ、P_1は少なくともP_2を解くのと全く同じ手順で解ける。将棋のルールで打ち歩詰めが禁じ手になっていることから、これ以外の優越関係はまったく成立しない。詰将棋を解く場合、優越関係がもっとも威力を発揮するのは、玉方の無駄合を探索する場合である。一見して無駄な合駒のように見えても実は意味があって、それが最善の応手であることが稀にあるので、無駄合もすべて探索しなければならない。合駒の種類は最大7通りあり、遠くから王手をかけた場合は合駒を打つ場所がたくさんあって、しかもそれらの多くは無駄な合駒であるから、うまく処理しないとそれだけで膨大な探索量になってしまう。我々のプログラムでは、まず最初に無駄合以外の応手をすべて探索して、それらがすべて解ければ、無駄合を後手玉に近い位置から順に探索するようにしている。その結果、後に示す実験において、解決木の終端節点の総数が生成節点数よりもはるかに多いという珍現象もいくつか見られた。また無駄合と予想される手は共謀数としてカウントしない、無駄合以外の合駒については1種類だけカウントする、など実際の解けにくさの度合をなるべく正確に共謀数に反映させる工夫をした。

次に2つの局面P_1, P_2が類似している場合を考えてみる。いくら2つの局面が似ていても優越関係が成立しなければ、一方の局面の探索結果からもう一方の局面の探索を省く、といったことはできない。しかし局面P_1が解けた場合、もう一方の局面P_2およびその子孫の局面に対して、P_1を解いて得られた結果を利用することにより、探索が効率化できる場合が多い。これはコンピュータチェスにおける技法の一つであるキラーヒューリスティックの一種である。詰将棋でこの手法が有効になる例として合駒の問題がある。先手番の局面を展開するときに、その局面に至る以前のN手目に合駒があり、N手目の別の種類の駒による合駒のうちすでに解けたものがあって、かつその合駒で置き換えた局面がハッシュ表に登録されていれば、登録された局面での最善手を最初に探索するという方法を採った。また玉方の不成の応手に対しても、同様のヒューリスティックを用いた。

これらの工夫をプログラムに組み込むことにより、計算時間が短縮されることが確認できた。また解が木でなくグラフになっている場合、同一局面での共

謀数が同一ポテンシャル解決木内で２重にカウントされるので、問題の実際の難易度を反映しないことがある。これは局面間に優越関係や類似関係がある場合についても同様である。このような場合の対処法は今後の研究課題である。

7　実験結果

「続詰むや詰まざるや」は江戸時代から昭和までの名作と言われている代表的な詰将棋２００題およびその解答と解説が掲載されている市販の問題集である [5]。その大部分の問題は、現在の新聞や週刊誌等に掲載されている詰将棋と比較するとはるかに難解である。またこの問題集には４１人の作者による最短１１手詰から最長８７３手詰までのさまざまなタイプの問題が選ばれており、詰将棋プログラムの性能を評価するのに適切なテスト問題であると言える。この問題集に対する他の著名な詰将棋プログラムの実行結果としては、Ito は約１３５題、Ｔ２は約７０題を解いている [4][7]。我々のプログラムはC言語で実装されており、実験はサンのスパークステーション２０上で行なった。掲載されている合計２０３題（第３０番は小問４題からなる）のうち、不詰問題や逃れ図式、中将棋の駒の入った問題を除いた１９７題について、各問制限時間３０分で解かせてみた。その結果、１７９題の問題が解けて、解答率は９１%に達した。詰手数別の解答率を表１に掲載し、各問題を解くのにかかった時間を表２に分類した。また制限時間内に解けなかった問題の番号およびそれらの詰手数は表３に列挙した。なお詰手数をあらかじめプログラムに教えることはしていないのは当然であるし、詰手数から探索の深さを制限することもしていない。これらはすべて同一プログラムによる結果である。

表１より、他の詰将棋プログラムでは顕著であった詰手数が長くなることに伴う解答率の低下がほとんど見られないことが分かる [4] [7]。解けなかった問題の多くは、第５０番前後および第１８０番以降に集中しており、これらは有名な将棋図巧と将棋無双の代表作および昭和の名作と言われているものである。また解けなかった問題のほとんどは、その解答解説の内容および筆者が過去に実際に解いてみた経験から察すると、人間の解答者にとっても非常に難解な問題のようである。解けた問題については、解決木の終端局面数も出力するようにしており、この

詰手数	問題数	解答数	解答率 (%)
11–19	29	29	100
21–29	45	43	96
31–39	39	34	87
41–49	24	22	92
51–99	44	41	93
101–873	16	10	63
合計	197	179	91

表 1: 詰手数別の解答率

計算時間（秒）	問題数
0–1	21
1–10	44
10–100	67
100–1000	38
1000–1800	9
合計	179

表 2: 解けた問題数の各計算時間による分類

数が大きいほど解答時間がかかる傾向があった。したがって解けなかった問題の多くはおそらく解決木の終端局面数が非常に大きな問題であると思われる。つまり解決木の終端局面間の依存関係が大きいということであり、この辺りに共謀数を使ったアルゴリズムの限界がありそうで、さらに工夫が必要とされるところかもしれない。

最後に、解けなかった問題１８題の中から、超長手数作品と言われる、寿（第５５番、６１１手詰）と新扇詰（第２００番、８７３手詰）を選んで、制限時間を２時間に延長して解かせてみたところ、２題とも２時間以内に解くことができた。この難問２題

番号	14	26	40	48	49	50
手数	35	25	35	119	163	69
番号	51	53	55	84	105	119
手数	41	85	611	123	25	31
番号	139	181	184	193	199	200
手数	57	33	45	35	103	873

表 3: 制限時間内に解けなかった問題とその詰手数

が解けたことから、他の１６題についても同様に制限時間を延長することにより、かなりの問題を解くことが可能であると思われる。

また問題ごとに、全展開局面数に対する再展開局面数の割合（再展開率）および分岐因子を計算した。再展開率は平均２割程度であり、多重反復深化法の再展開による再計算の無駄は少ないことが実験的に示せた。また詰将棋の分岐因子は５程度（約5.2）であることが分かった。各問題の計算時間、生成局面数、展開局面数、反復回数、解の葉数、分岐因子などのデータは紙数の関係で掲載できないので、興味をお持ちの方は[17]を参照していただきたい。

8 おわりに

AND/OR木の探索問題に対して、共謀数を使ったアルゴリズムC*を開発し、詰将棋プログラムとして実装し、実際に問題を解かせる実験を行なった。「続詰むや詰まざるや」の１７９題を各３０分以内で解くことができた。さらに、有名な寿（６１１手詰）と新扇詰（８７３手詰）を各２時間以内で解くことができた。さらに詰将棋の分岐因子が実験結果から計算され、５程度であることが分かった。

参考文献

[1] Allis, L.V., Meulen, M., and Herik, H.J., "Proof-number search," *Artificial Intelligence, Vol.66,* 1994, pp.91–124.

[2] Elkan, C., "Conspiracy Numbers and Caching Searching And/Or Trees and Theorem-Proving," *IJCAI-89 Proceedings of the Eleventh International Joint Conference on Artificial Intelligence,* 1989, pp.341–346.

[3] Ibaraki, T., "The Power of Dominance Relations in Branch-and-Bound Algorithms," *J. ACM, Vol.24, No.2,* 1977, pp.264–279.

[4] 伊藤琢巳, 野下浩平：詰将棋を速く解く２つのプログラムとその評価, 情報処理学会論文誌, Vol.35, No.8, 1994, pp.1531–1539.

[5] 門脇芳雄：続詰むや詰まざるや, 平凡社, 東洋文庫, 1978

[6] Korf, R.E., "Depth-First Iterative-Deepening: An Optimal Admissible Tree Search," *Artificial Intelligence, Vol.27,* 1985, pp.97–109.

[7] 小谷善行：コンピュータ将棋協会資料集, Vol.6, コンピュータ将棋協会, 1992.

[8] Levy, D., and Newborn, M., コンピュータチェス 世界チャンピオンへの挑戦, サイエンス社, 1994.

[9] Matsubara, H., "Shogi (Japanese Chess) as the AI research target next to chess," *ETL technical report ETL-TR-93-23,* Electrotechnical Laboratory, 1993.

[10] 松原仁：ゲームとしての将棋のいくつかの性質について, 情報処理学会人工知能研究会, *Vol.96, No.3,* 1994, pp.21–30.

[11] McAllester, D.A., "Conspiracy Numbers for Min-Max Search," *Artificial Intelligence, Vol.35,* 1988, pp.287–310.

[12] Nilsson, N.J., "Problem Solving Methods in Artificial Intelligence," *McGraw-Hill, New York,* 1971.

[13] Nilsson, N.J., "Principles of Artificial Intelligence," *Tioga, Palo Alto, CA,* 1980.

[14] 野下浩平：詰将棋を解くアルゴリズムについて, 電子情報通信学会コンピューテーション研究会, *COMP91-56,* 1991.

[15] Reinefeld, A. and Marsland, T.A., "Enhanced Iterative-Deepening Search," *IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol.16, No.7,* 1994, pp.701–710.

[16] Schaeffer, J., "Conspiracy Numbers," *Artificial Intelligence, Vol.43,* 1992, pp.67–84.

[17] Seo, M., "The C* Algorithm for AND/OR Tree Search," *Master Thesis,* 1994, to appear.