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

2005－ARC－162（27）
2005－HPC－101（27）
2005／3／9

証明数・反証数を閾値とした反復深化法の複数経路同時探索による高速化

鷹野　芙美代†　佐田　宏史†　前川　仁孝†
六沢　一昭†　宮崎　収兄†

本稿では，PDS の複数の探索経路を同時に探索することにより，証明数の大きい節点に存在する解も高速に発見する逐次探索手法を提案する．証明数を用いた探索では，証明数の小さい節点が解である可能性が高いため，一般に証明数の小さい節点から探索する．そのため，証明数の大きい節点が解であった場合に多くの時間を必要とする．そこで証明数の大きい節点と小さい節点の複数経路を交互に探索することで証明数の大きい節点が解である場合でも早く解を見つけることができる．加えて探索時の情報を用いて各経路の探索順と探索量を動的に変更することでさらに効率良く探索できる．本稿では証明数などを用いて探索順・探索量を動的に変更し，評価した．

Speedup by Simultaneous Multi-Path Search in Iterative Deepening Using Proof Number and Disproof Number

Fumiyo Takano,† Hiroshi Sata,† Yoshitaka Maekawa,†
Kazuaki Rokusawa† and Nobuyoshi Miyazaki†

This paper proposes a fast sequential algorithm searching multiple paths simultaneously. A node with a small proof number has a high probability of a solution. Search algorithms using proof numbers searches only from a node with a small proof number. Though, it takes a long computation time if a solution is a node with a large proof number. The proposed algorithm searches alternately nodes with a large proof number and nodes with a small proof number. Additionally, it changes the amount of search and the order of search in each search path dynamically by using runtime information while searching. Finally, this paper evaluates the algorithm using proof numbers to change the amount of search and the order of search.

1. はじめに

AND/OR 木の探索アルゴリズムとして，探索木の全ての節点を探索する深さ優先探索や，有効だと思われる節点のみを探索する最良優先探索などがある．最良優先探索は，深さ優先探索よりも探索空間が大きい問題を解くことができる可能性が高いが，探索するために多くの記憶領域が必要となる．そこで近年，少ないメモリ量で最良優先探索のような探索順を実現するために，反復深化法が用いられている．AND/OR 木探索の節点の評価値としては，節点を証明・反証するためのコストを表した証明数・反証数[1] が有効であるとされている．これらを用いた深さ優先探索として，証明数と反証数のうちの片方もしくは双方を閾値として反復深化する探索法が提案されている[2)~7)]．その中でも詰将棋の探索では，証明される部分木の探索と反証される部分木の探索を効率良く行う，証明数と反証数双方を閾値とした反復深化法 PDS[4)] が有効であるとされている[8)]．PDS は証明するためのコストが小さい証明数の小さい節点から探索する．そのため，証明数の大きい節点に解が存在する場合は，求解に長時間必要となる．このような場合には証明数の大きい節点も早期に探索した方が良い．しかし，証明数の大きい節点からの探索のみでは，証明数が小さい節点が解である多くの場合に対して高速に解が得られない．

そこで本稿では，証明数の小さい節点と大きい節点を同時に探索するため，PDS の探索木における複数の探索経路を交互に探索する複数経路同時探索を提案する．本手法は，証明数の大きい節点が解である場合には早く解を発見し，証明数の小さい節点が解である場合にも PDS よりも解の発見があまり遅くならないことを目的とする．複数経路を同時に探索するためには，複数のプロセッサに各探索経路を割当て，並列処理す

† 千葉工業大学 情報工学科
Department of Computer Science, Chiba Institute of Technology

る手法が考えられる．このような並列探索手法としてAND/OR 木階層的挟み撃ち探索（AOHPAS）[9] が提案されている．逐次的に複数経路を同時に探索するには，AOHPAS における各プロセッサの探索を交互に行えば良い．しかし，複数の経路を交互に探索するだけでは，並列探索時に隠蔽されていた無駄な節点を探索する影響が顕著に現れる．そこで，逐次探索による複数経路同時探索では，探索時の節点情報を用いて各経路の探索順や探索量を動的に変更する．これにより探索する必要のない節点の探索を省き，解の存在する可能性の高い節点を重点的に探索することができる．

以降，2 章で証明数と反証数を用いた PDS，3 章では提案手法である複数経路同時探索について述べる．4 章では詰将棋の求解を例として複数経路同時探索の評価を行い，5 章でまとめと今後の課題を述べる．

2. PDS

AND/OR 木は全ての子節点が true とならなければ true とならない AND 節点と，子節点のうちどれか 1 つでも true となれば true となる OR 節点からなる．AND/OR 木の探索法の一つに，証明数探索[1] がある．証明数探索は，節点を証明や反証するのに必要なコストを表す証明数・反証数を評価値として用いる最良優先探索である．証明数はある節点を証明するために true であることを示さなければならない最小節点数を表し，反証数は反証するために false であることを示さなければならない最小節点数を表す．

節点 n の証明数を $\mathrm{pn}(n)$，反証数を $\mathrm{dn}(n)$ とすると，$\mathrm{pn}(n)$ と $\mathrm{dn}(n)$ は以下のように計算される．

(1) 節点 n が先端節点
  (a) 最終的な評価値が true
    $\mathrm{pn}(n)=0 \quad \mathrm{dn}(n)=\infty$
  (b) 最終的な評価が false
    $\mathrm{pn}(n)=\infty \quad \mathrm{dn}(n)=0$
  (c) 最終的な評価が不明
    $\mathrm{pn}(n)=1 \quad \mathrm{dn}(n)=1$
(2) 節点 n が内部節点
  (a) n が OR 節点
    $\mathrm{pn}(n)$=子節点の pn の最小値
    $\mathrm{dn}(n)$=子節点の dn の和
  (b) n が AND 節点
    $\mathrm{pn}(n)$=子節点の pn の和
    $\mathrm{dn}(n)$=子節点の dn の最小値

証明数が小さいほど証明しやすいと言えるので，OR 節点では証明数最小の子節点を選ぶ．また，反証数が小さいほど反証しやすいと言えるので，AND 節点では反証数が最小となる子節点を選ぶ．

証明数探索などの最良優先探索は，探索した全節点を記憶する必要があり，多くの記憶領域が必要となる．そこで，反復深化する深さ優先探索により証明数探索とほぼ同じ振舞をする PDS が提案されている．PDS は，証明数と反証数を閾値とし，証明数と反証数が閾値以内の節点を探索する．証明数と反証数の閾値を 1 から開始し，徐々に増やしながら反復深化する．根節点の証明数（反証数）が 0 となれば証明（反証）されたこととなり探索を終了する．根節点の証明数も反証数も 0 にならなければ，証明数または反証数の閾値を増やし再探索する．PDS では，OR 節点は証明数の小さい子節点から展開し，AND 節点は反証数の小さい子節点から展開する．また，反復深化するので同じ節点を何度も探索するため，各節点の情報をトランスポジションテーブルに保存し，節点の再展開を防ぐ．

3. 複数経路同時探索

証明数の小さい節点は，証明するのに必要なコストが小さく解に近い場合が多い．しかし，証明数の小さい節点からのみの探索では，証明数の大きい節点が解である場合，解を得るまでに長い時間を必要とする．このような場合には証明数の大きい節点を先に探索した方が早く解が求まる．しかし，AND/OR 木では証明数が小さい節点が解であることが多く，探索前には探索木のどこに解が存在するかは分からないため，証明数の大きい節点からのみの探索は非現実的である．

そこで本稿では，PDS の探索木において証明数の大きい節点からと証明数の小さい節点からの複数の探索経路を同時に探索することで，証明数の大きい節点の解も早く発見する複数経路同時探索を提案する．

以下，証明数の大きい節点と証明数の小さい節点を並列に探索する手法 AOHPAS，AOHPAS を逐次探索に応用した複数経路同時探索の探索法と動的に探索経路を変更するための方法について述べる．

3.1 AND/OR 木階層的挟み撃ち探索

複数の経路を同時に探索するためには，各探索経路に複数のプロセッサを割当て並列に探索する手法が考えられる．このような並列探索の一つとして，証明数の小さい節点からの探索と大きい節点からの探索を並列に行い，さらに証明数の小さい節点を多くのプロセッサで探索する AOHPAS が提案されている．AOHPAS では，**図 1** のように，1 つのリーダプロセッサが探索木の左側（証明数の小さい節点）から右側（証明数の大きい節点）へと，根節点から PDS で探索する．それ以外のスレーブプロセッサは，リーダプロセッサの

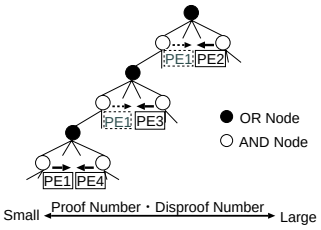

図 1 AND/OR 木階層的挟み撃ち探索
Fig. 1 AND/OR Tree Hieralchical Pincers Attack Search

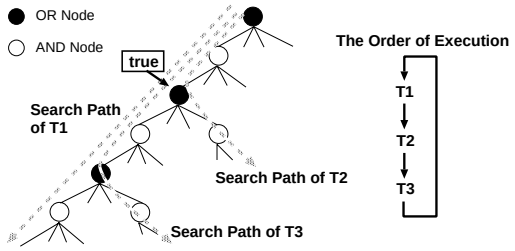

図 2 疑似スレッドスケジューリング
Fig. 2 The Pseudo Thread Scheduling

探索経路上の OR 節点に 1 つずつ割当てられ，割当てられた節点の子節点のうち根とする節点を証明数の大きい節点から証明数の小さい節点の順番に選び，その節点以下を PDS により探索する．スレーブプロセッサは，割当てられた節点の子節点を全て探索し終わると，すぐに他の未探索の節点に再割当てされる．図 1 のように，プロセッサを階層的に割当て，探索木の左右から挟み撃つように並列に探索することで，証明数の小さい節点以下を多くのプロセッサで探索しながら，証明数の大きい節点も並列に探索することができる．

AOHPAS では，PDS でもすぐに解を発見できるような証明数の小さい節点が解である問題に対しては，証明数が小さい節点から探索するリーダプロセッサが解を発見するため，並列処理による速度向上の効果はあまりなく，探索時間は PDS の探索時間とほぼ同じとなる．しかし，逐次の PDS で探索するまでに長時間かかるような証明数の大きい節点が解である問題に対しては，証明数の大きい節点から探索するスレーブプロセッサが解を発見することが多く，PDS と比べて使用するプロセッサ数倍以上に高速化する可能性が高いことが確認されている[9]．このような場合には，各プロセッサの探索を逐次処理により交互に行うだけでも，ほとんどの問題において PDS よりも高速化する．

3.2 AOHPAS の逐次的探索

AOHPAS は，証明数の小さい節点が解である場合，全プロセッサの総探索節点数は PDS よりも増加するが，探索時間は PDS とほぼ同じになる．1 つのプロセッサで交互に AOHPAS の探索を行うと，探索節点数は並列探索とほぼ同値となり，探索節点の増加分だけ PDS よりも探索時間が長くなる．また，全経路の探索を同じ比重で順番に探索すると，解ではなさそうな節点も多く探索してしまい，既に探索の必要がないと確定できた節点も探索する可能性がある．

そこで，複数経路同時探索では AOHPAS における各プロセッサの探索経路を疑似的なマルチスレッドを用いて交互に探索し，探索時の節点の情報を用いて，各経路の探索順や探索量を動的に変更する．これにより探索する必要のない節点の探索を省き，解の存在する可能性の高い節点を重点的に探索することができる．

以下，AOHPAS における各プロセッサの探索を疑似スレッド，疑似スレッドの探索量を決定するものをスケジューラと呼ぶ．AOHPAS と同様に，リーダ疑似スレッドが PDS で探索し，スレーブ疑似スレッドがリーダの探索経路上に割当てられる．スケジューラは，探索時の情報によって，次に探索する疑似スレッドとその疑似スレッドの上限探索量を決定する．疑似スレッドは決められた量の前回の探索に続く探索を行う．

3.3 動的探索経路変更

本節ではスケジューラによる疑似スレッドの選択法，疑似スレッドの探索量を決める展開節点数，証明数・反証数，リーダ疑似スレッドの優先について述べる．

3.3.1 スケジューリングの最適化

疑似スレッドのスケジューリングは，基本的にラウンドロビン方式とする．しかし，リーダ疑似スレッドの探索経路上の節点をスレーブ疑似スレッドが証明すると，リーダ疑似スレッドの探索経路が大きく変わる可能性がある．よって，証明された節点以下に再割当てをしても無駄であり，その節点以下を他の節点が探索することも無駄である．そこで，スレーブ疑似スレッドが割当てられた節点を証明した場合には，探索量が上限探索量に満たなくとも他の節点に再割当てせずに探索を中止し，ラウンドロビン順とは関係なくリーダ疑似スレッドへ切り替えることで最適化する．

図 2 を用いて最適化スケジューリングについて説明する．図 2 は，リーダ疑似スレッド T1 の探索経路上の節点を T2，T3 が探索している．この時，T2 が割当てられた節点を証明すると，リーダ疑似スレッド T1 の証明された節点以下の探索と T3 の全探索は必要ない．この場合，T2 は他節点に再割当てされずに探索を終了し，次に T3 ではなく T1 に切り替えることにより，T2 と T3 の無駄な探索を省くことができる．

3.3.2 展開節点数による探索量の計数

疑似スレッドを実時間により切替えるとオーバヘッドが大きい．探索は節点の展開とトランスポジションテーブルの参照からなる．よって，展開節点数とトランスポジションテーブル参照数を用いて探索量を計数する．トランスポジションテーブル参照のコストは節点展開のコストに比べて小さいため，探索量を式 (1) により計数する．疑似スレッドは探索量がスケジューラの決めた上限探索量を越えたら探索を停止する．

$$\text{探索量} = \text{展開節点数} + \text{TT 参照数}\times\text{重み} \quad (1)$$

上限探索量が多いとスレーブ疑似スレッドの無駄な探索が増える可能性がある．逆に，上限探索量が少ないと，スケジュールのオーバヘッドが大きい．よって，最適なトランスポジションテーブル参照の重みと，以降の節で述べる基準展開節点数を決定する必要がある．

3.3.3 証明数と反証数による上限探索量の動的変更

複数経路同時探索は，PDS の探索経路以外も探索する．複数経路を等比率で探索すると証明数の大きい節点の探索が多くなり，PDS よりも探索効率が悪化する．そこで，複数経路の探索を PDS の探索に近づけることで効率良く探索する．そのため，節点の証明数により疑似スレッドの上限探索量を変化させ，証明数の小さい節点の探索は長くし，大きい節点の探索は短くする．上限探索量は式 (2) により計算する．ここで $\mathrm{pn}(n_i)$ は疑似スレッド i に割り当てられた節点 n_i の証明数，base は基準展開節点数，all は全疑似スレッド数とする．リーダ疑似スレッド 1 の $\mathrm{pn}(n_1)$ のみ上限探索量計算時に探索している節点の証明数を用いる．

$$\text{上限探索量} = \frac{\sum_{j=1}^{all}\mathrm{pn}(n_j)}{\mathrm{pn}(n_i)}\times \text{base} \quad (2)$$

また，PDS はスレーブ疑似スレッドを割当てる OR 節点では証明数の小さい節点から探索するが，AND 節点では反証数の小さい節点から探索する．そこで，節点の証明しやすさだけではなく反証しやすさも考慮に入れる場合，節点 n_i の反証数 $\mathrm{dn}(n_i)$ を用いて上限探索量を式 (3) により計算する．

$$\text{上限探索量} = \frac{\sum_{j=1}^{all}(\mathrm{pn}(n_j)+\mathrm{dn}(n_j))}{\mathrm{pn}(n_i)+\mathrm{dn}(n_i)}\times \text{base} \quad (3)$$

しかし，証明数の小さい節点における探索の比重を大きくすると，証明数の大きい節点の上限探索量が少なくなり PDS に近づくため，複数経路同時探索の利点である証明数の大きい節点に解があった場合の解の早期発見率が下がる可能性もある．また，証明数の小さい節点の探索が少ないと，PDS では早く見つけることができる解の発見が遅くなる可能性が高くなる．

3.3.4 リーダ疑似スレッドの探索の優先

3.3.3 節の上限探索量の動的変更に対し，さらに PDS の探索に近づけるために，リーダ疑似スレッドの上限探索量を他の疑似スレッドに対して増加させる．これにより，証明数の大きい節点からの探索を行いながら，PDS のように証明数の小さい節点の探索を優先することができる．複数経路同時探索のリーダ疑似スレッドの探索の割合を他の疑似スレッドの ∞ 倍とすると，PDS の証明数の小さい節点からのみの探索と等価である．また，1 倍とすると AOHPAS の逐次的な実行と等価となる．しかしリーダ疑似スレッドの探索量を増加させ過ぎると，証明数と反証数による上限探索量の決定法と同様に，スレーブ疑似スレッドの上限探索量が少なくなり証明数の大きい節点にある解を発見しにくくなる可能性がある．

4. 評 価 実 験

詰将棋を解くプログラムに対して複数経路同時探索の有効性と各経路の上限探索量の動的変更法について評価した．評価には詰将棋の問題集「続詰むや詰まざるや」[10] の前半 50 問を用いた．探索の制限時間は 3600 秒とし，PDS，複数経路同時探索ともに制限時間以内に解けなかった問題，PDS の探索時間が 10 秒未満の問題については評価の対象としない．そのため計 21 問が評価の対象となった．

以下，スレーブ疑似スレッドが割当てられた節点を証明した場合のスケジューリング最適化の効果，疑似スレッド切替えの基準となる節点数とトランスポジションテーブル参照数の考慮について，また，証明数と反証数による疑似スレッドの上限探索量の動的変更やマスタ疑似スレッドの探索優先率について評価を行う．

評価に用いた高速化率は式 (4) により計算する．PDS では解くことができないが，複数経路同時探索により解くことができた問題は PDS の探索時間を 3600 秒とし，また逆に PDS では解くことができたが複数経路同時探索では解くことができない問題は複数経路同時探索の探索時間を 3600 秒として高速化率を計算した．

$$\text{高速化率} = \frac{\text{PDS の探索時間}}{\text{複数経路同時探索の探索時間}} \quad (4)$$

4.1 最適化スケジューリングの効果

図 3 にスレーブプロセッサが割当てられた節点を証明した場合の最適化を行った optimized と，最適化せず常にラウンドロビンで切替えた normal の PDS に対する高速化率の相乗平均を示す．両手法とも疑似スレッドの 1 回の上限探索量は 1000 節点とした．

図 3 より，疑似スレッド数が少ない時は両手法の高

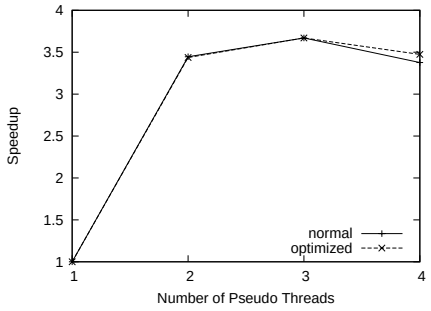

図 3 最適化スケジューリングの効果
Fig. 3 The Efficiency of Optimized Scheduling

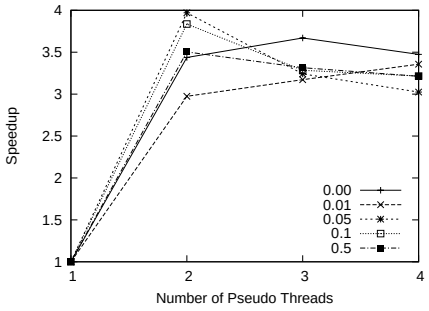

図 5 トランスポジションテーブルの参照
Fig. 5 Reference a Transposition Table

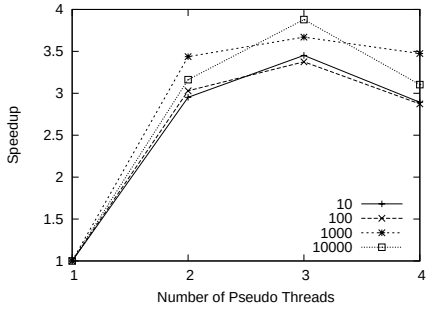

図 4 展開節点数
Fig. 4 The Number of Search Nodes

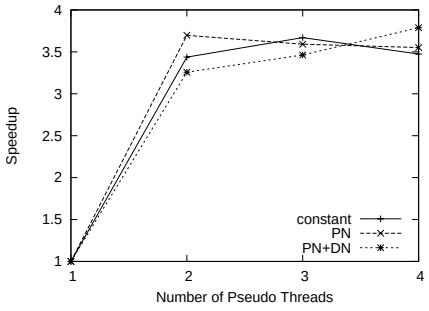

図 6 証明数と反証数による上限探索量の決定
Fig. 6 Decision of Amount of Search by Proof Number and Disproof Number

速化率はほぼ同じであるが，疑似スレッド数が 4 の時は最適化したスケジューリングの方が若干高速化率が高い．これは疑似スレッドが割当てられた節点を証明し，その節点以下に他の全ての疑似スレッドが割当てられていた場合，ラウンドロビンでは，(疑似スレッド数-1) ×上限探索量分の必要ない探索を全て行うためである．スレッド数の増加に伴いこの無駄な節点の探索は増加する．しかし疑似スレッド数 4 でもあまり変化が見られないのは，数千節点多く探索しても，探索時間にはほとんど影響がないためであると考えられる．

4.2 基準展開節点数

各疑似スレッドの上限探索量の基準となる，基準展開節点数を評価する．**図 4** に展開節点数を 10, 100, 1000, 10000 とした場合の高速化率の相乗平均を示す．

図 4 より，疑似スレッド数 3 では節点数 10000 の高速化率が高いが，平均すると節点数 1000 の高速化率が高い．これは，1 回の展開節点数が 10 や 100 などのように少ないと，スケージューリング回数が多くなるのでオーバヘッドが大きくなり，逆に展開節点数が多いとスレーブ疑似スレッドの一度に探索する量が増え，逐次探索では必要なかった節点を多く探索してしまう可能性が高いためである．

また，各疑似スレッドの探索量として，展開節点数のみではなくトランスポジションテーブルの参照も考慮するべきかを検証する．このとき探索量は式 (1) によって計数する．トランスポジションテーブルの参照を考慮しない，つまり重みが 0 の時と，参照の重みを 0.01, 0.05, 0.1, 0.2 とした時の高速化率の相乗平均を**図 5** に示す．基準展開節点数は 1000 とした．

図 5 より，疑似スレッド数 2 では重みが大きくても高速化率が高いが，全体では重みを付けない方が高速化率が高い．重みを 0.01 よりも小さくすることも考えられるが，トランスポジションテーブルの参照時間は節点の展開時間に比べ非常に短いため，疑似スレッドの探索量として考慮しなくても良いと考えられる．

4.3 証明数と反証数による上限探索量の動的変更

各疑似スレッドにおける上限探索量を証明数のみを用いて式 (2) により計算した PN と，証明数と反証数の双方を用いて式 (3) により計算した PN+DN，上限探索量は動的に変更せず一定の constant の高速化率の相乗平均を**図 6** に示す．基準展開節点数は 1000 であり，探索量の計数にトランスポジションテーブルの参照は考慮しない．

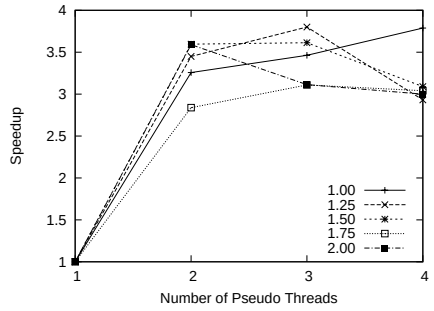

図 7 リーダ疑似スレッドの優先率
Fig. 7 Priority Ratio of the Leader Pseudo Thread

図 6 より，PN，PN+DN，constant の 3 つとも高速化率はほぼ変わらないことがわかる．また，疑似スレッド数 2 では PN，疑似スレッド数 3 では constant，疑似スレッド数 4 では PN+DN と，疑似スレッド数によって最適な上限探索量を決定するパラメータが異なるという結果が得られた．

4.4 リーダ疑似スレッドの優先率

リーダ疑似スレッドの優先率を検証するため，前節の証明数と反証数を用いた動的上限探索量決定に加え，リーダ疑似スレッドの上限探索量を 1.00 倍，1.25 倍，1.50 倍，1.75 倍，2.00 倍とした時の高速化率の相乗平均を**図 7** に示す．1.00 倍は図 6 の PN+DN と同一となる．基準展開節点数は 1000 であり探索量の計数にトランスポジションテーブルの参照は考慮しない．

図 7 より，リーダ疑似スレッドの優先率は 1.25 倍から 1.50 倍程度の場合に高速化率が高く，それ以上に高くしてもあまり効果がないことが分かる．これはリーダ疑似スレッドの優先率を上げたことによって証明数の大きい節点以下に存在した解を発見するのが遅くなるためだと考えられる．

4.5 複数経路同時探索による効果

複数経路同時探索では，各疑似スレッドの探索量をいくつかのパラメータを組み合わせた値によって動的に変更する．疑似スレッド数は，少ないと証明数の大きい節点の探索をあまりできないが，多過ぎても必要ない節点を多く探索してしまう．そのため，図 3 から図 7 より疑似スレッド数 3 の時が比較的高速化率が高い．しかし，図 3 から図 7 の中では，図 5 の疑似スレッド数 2 のトランスポジションテーブル参照の重みを 0.05 とした場合が最も高速化率が高い．よって，使用疑似スレッド数やトランスポジションテーブルの参照の割合，リーダ疑似スレッドの優先率などの動的上限探索量決定のパラメータやその値の組合せ方によってはさらなる高速化が得られる可能性もある．

5. おわりに

本稿では，AND/OR 木探索手法 PDS の探索木において，複数の探索経路を交互に探索することによって，評価の悪い節点が解である場合にも速く解を見つけることができる複数経路同時探索を提案した．複数経路同時探索は探索時の節点の情報を用いて各経路の探索順や上限探索量を動的に変化させることによって効果的に探索する．複数経路同時探索を評価した結果，PDS と比べて高速することが確認できた．今後の課題として，各パラメータの効果的な組合せを調査し，さらに効率的に探索ができるようなパラメータを考案することが挙げられる．また，リーダ疑似スレッドの優先率の動的な変更や，使用する疑似スレッド数の動的な増加，使用する疑似スレッド数によって各パラメータの値を変化させることも考えられる．

参考文献

1) Allis, L. V., van der Meulen, M. and van den Herik, H. J.: Proof-Number Search, *Artificial Intelligence*, Vol. 66, pp. 91–124 (1994).
2) 脊尾昌宏: 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, 情報処理学会 人工知能研究報告, No. 99, pp. 103–110 (1995).
3) Seo, M., Iida, H. and Uiterwijk, J. W.: The PN*-search algorithm: Application to tsumeshogi, *Artificial Intelligence*, Vol. 129, pp. 253–277 (2001).
4) Nagai, A.: A new AND/OR tree search algorithm using proof number and disproof number., *Complex Games Lab Workshop*, pp. 40–45 (1998).
5) Nagai, A. and Imai, H.: Proof for the Equivalence between Some Best-First Algorithms and Depth-First Algorithms for AND/OR Trees, *IEICE TRANS.*, No. 10, pp. 1645–1653 (2002).
6) 長井歩, 今井浩: df-pn アルゴリズムの詰将棋を解くプログラムへの応用, 情報処理学会論文誌, Vol. 42, No. 6, pp. 1769–1777 (2002).
7) Nagai, A. and Imai, H.: Application of df-pn+ to Othello Endgames, *Proceedings of Game Programming Workshop '99*, pp. 16–23 (1999).
8) 作田誠, 飯田弘之: 高性能な AND/OR 木探索アルゴリズムの詰め将棋問題による比較, 静岡大学情報学研究, Vol. 5, pp. 15–22 (1999).
9) 鷹野芙美代, 関根敦史, 佐田宏史, 前川仁孝, 六沢一昭: AND/OR 木における証明数・反証数を用いた階層的挟み撃ち探索, 情報処理学会論文誌, Vol. 45, No. SIG 11(ACS 7), pp. 280–289 (2004).
10) 門脇芳雄: 続詰むや詰まざるや, 平凡社 (1978).