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

2006－HPC－107（17）
2006／8／1

証明数・反証数を用いた反復深化法における複数経路並行探索の並列化

鷹野　芙美代[†,†††]　前　川　仁　孝[††]
笠　原　博　徳[†]　成田　誠之助[†]

本稿では，評価の高い節点からと低い節点からの探索経路を並列に探索する手法である AND/OR 木階層的挟み撃ち探索において，さらに多くの経路を並行に探索する並列複数経路並行探索を提案する．AND/OR 木では評価の高い節点に解がある可能性が高いが，評価の低い節点に解があることもあり，評価の高い節点と低い節点を並行に探索することは有効である．しかし，AND/OR 木階層的挟み撃ち探索ではプロセッサ数までの経路しか並列に探索できない．そこで，並列複数経路並行探索では，各プロセッサが複数の経路を並行に探索することで，プロセッサ数が少ない場合にも多くの経路を探索でき，より高速化することができる．

Parallelization of Multi-Path Concurrent Search for Iterative Deepening using Proof and Disproof Numbers

Fumiyo Takano,[†,†††] Yoshitaka Maekawa,[††]
Hironori Kasahara[†] and Seinosuke Narita[†]

This paper proposes a parallel search algorithm named parallel multi-path concurrent search(PMPS). The algorithm searches concurrently more search paths than AOHPAS. AOHPAS is the search algorithm that searches some paths of search from a high evaluation value node and search from a low evaluation value node in parallel. However, AOHPAS searches only paths as many as processors at the same time. Therefore, PMPS searches more paths concurrently than the number of pathes that AOHPAS searches. By search of more paths in concurrently, PMPS is able to find a solution node with low evaluation value quickly. Consequently, PMPS uses less processor efficiently.

1. はじめに

AND/OR 木の探索は，探索範囲が広いと解の発見に長い時間がかかり，高速化が望まれている．AND/OR 木の有効な探索法として，節点を証明・反証するためのコストを表した評価値である，証明数・反証数[1] を用いた探索法が提案されている[2)~7)]．詰将棋問題求解の探索では，証明数と反証数双方を閾値とした反復深化法である df-pn[5),6)] が特に有効である．これらの探索法では，効率よく解を求めるため，評価値の高い，つまり証明・反証のためのコストが小さいと考えられる証明数・反証数が小さい節点から探索する．

† 早稲田大学コンピュータ・ネットワーク工学科
Department of Computer Science, Waseda University
†† 千葉工業大学情報工学科
Department of Computer Science, Chiba Institute of Technology
††† 日本学術振興会
Japan Society for the Promotion of Science

しかし，探索深さが深くなるにつれ探索節点数は指数関数的に増加し，探索時間は膨大になる．そのため，並列化も含めた探索アルゴリズムの高速化の研究が行われている[8)~12)]．並列探索の多くは，逐次探索と同様に評価の高い節点から並列に探索することで，どのような問題に対しても同等の高速化を図る．よって並列探索の場合も，評価の高い節点が解の場合には早く解が求まるが，評価の低い節点が解の場合には解を得るために時間がかかる．評価値の低い節点の解を早く求めるためには，評価値の低い節点を早期に探索すればよい．しかし，評価値の低い節点からの探索のみでは，評価値の高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．

そこで筆者らは，OR 木並列探索手法である階層的挟み撃ち探索[13),14)] を，探索木の深さが不明な AND/OR 木探索に応用した並列探索手法である，AND/OR 木階層的挟み撃ち探索を提案した[15)]．本手法は，評価の高い節点と低い節点を並列探索することで，逐次探索

での求解に長時間かかる評価の低い節点が解である問題に対して短時間で解を求めることができ，さらに逐次探索で高速に解が求まる評価の高い節点が解である場合でも逐次探索より遅くなることはない．また，使用するプロセッサ数が増えると高速化率が上昇する傾向にあり，スーパリニアスピードアップが得られることもある[15]．本手法は，多くの経路を並列に探索することで，評価の低い節点の解を早期に発見できる．しかし，本手法では 1 つのプロセッサは 1 経路のみを探索する．よって，より多くの経路を並列に探索するためにはプロセッサを増加させる必要がある．

筆者らは，1 つのプロセッサで複数の経路を並行に探索する手法として複数経路並行探索を提案し，1 経路のみの探索に比べて高速化することを確認した[16]．そこで本稿では，チップマルチプロセッサなどのように利用できるプロセッサ数が限られている場合にも多くの経路を並行に探索することができる並列複数経路並行探索を提案する．

2. df-pn

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・反証数を評価値に用いた最良優先探索である証明数探索[1]がある．証明数・反証数は，節点を true と証明もしくは反証するのにかかるコストを表す．

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) 最終的な評価が true
    $\mathrm{pn}(n)=0$ $\mathrm{dn}(n)=\infty$
  (b) 最終的な評価が false
    $\mathrm{pn}(n)=\infty$ $\mathrm{dn}(n)=0$
  (c) 最終的な評価が不明
    $\mathrm{pn}(n)=1$ $\mathrm{dn}(n)=1$
(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

証明数探索のような最良優先探索では，探索した節点を多く記憶しておく必要があるため，非常に多くの記憶領域が必要となる．そこで，証明数探索と同じ振舞を，少ない記憶領域しか必要としない深さ優先探索の反復深化法として実現する手法である df-pn が提案された[5]．df-pn は証明数と反証数を反復深化の閾値に用いる．各節点における閾値とその子節点の証明数や反証数から子節点の閾値を算出し，証明数・反証数が閾値以下であれば探索する．そして，その閾値で解が見つからなければ閾値を徐々に増加させ，探索範囲を広げながら探索する．df-pn は OR 節点では証明数が小さい子節点から展開し，AND 節点では反証数が小さい子節点から展開する．

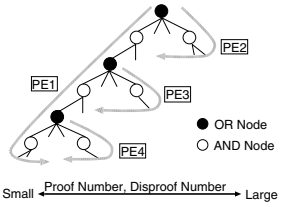

図 1 AND/OR 木階層的挟み撃ち探索
Fig. 1 AND/OR tree hierarchical pincers attack search

3. AND/OR 木階層的挟み撃ち探索

df-pn は AND/OR 木における有効な探索手法であるが，証明数の大きい節点が解の場合には，求解に長い時間がかかる．そこで筆者らは，証明数の小さい節点と大きい節点を並列に探索する並列探索手法として，AND/OR 木階層的挟み撃ち探索[15] を提案した．

図 1 に示す AND/OR 木階層的挟み撃ち探索は，証明数・反証数の小さい節点ほど左にある探索木に対し，リーダプロセッサ PE1 が左側（証明数の小さい節点）から右側（証明数の大きい節点）へと，根節点から探索する．それ以外のスレーブプロセッサ PE2 以降は，リーダプロセッサの探索経路上の節点に 1 つずつ割り当てられ，割り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に選び，選んだ節点以下を探索する．スレーブプロセッサは，割り当てられた節点の子節点を全て探索し終わると，他のスレーブプロセッサによって探索されていない節点にすぐに再割り当てされる．図 1 のように，スレーブプロセッサを階層的に割り当て，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より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．また，割り当てられた領域の探索が終了したスレーブプロセッサは，すぐに他の節点に再割り当てされ探索を続けるため，全プロセッサで同期を取らずに負荷分散ができる．

AND/OR 木階層的挟み撃ち探索は，証明数の小さい節点から探索する逐次探索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．このため，並列処理による速度向上はあまりないが，並列処理のためのオーバヘッドによる速度低下も小さい．一方，逐次探索で解の発見に長時間かかる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早期に発見することが多く，逐次探索と比べて，使用するプロセッサ数倍以上に大きく高速化することが多いと確認されている[15]．

4. 並列複数経路並行探索

AND/OR 木階層的挟み撃ち探索は，プロセッサ数，つまり並列探索する経路数を増加させると，証明数の大きい節点からの探索経路をより多く並列探索することになる．よって，証明数の大きい節点の解を発見しやすくなるため，df-pn では解の発見に長時間必要な問題で特に大きな高速化が可能となり，スーパリニアスピードアップが得られることもある．そのため，プロセッサ数を増加させずとも，プロセッサ数以上の経路を並行に探索することで，証明数の大きい節点の解を早期に発見し，高速化が可能になると考えられる．

筆者らは，プロセッサ数以上の探索経路を並行に探索する一手法として，シングルプロセッサで複数の探索経路を並行に探索する複数経路並行探索を提案し，その有効性を確認した[16]．そこで本稿では，さらにマルチプロセッサにおいてもプロセッサ数以上の探索経路を並行に探索する並列複数経路並行探索を提案する．

4.1 複数経路並行探索

逐次探索である複数経路並行探索は，**図 2** のように，AND/OR 木階層的挟み撃ち探索における各プロセッサが探索する証明数の小さい節点からの探索経路と証明数の大きい節点からの探索経路を任意の節点数ずつラウンドロビン方式で順に探索する．

AND/OR 木階層的挟み撃ち探索における各プロセッサの処理を疑似スレッドとする．このうち，証明数の小さい節点から探索する疑似スレッドはリーダ疑似スレッド（図 2 の T1），証明数の大きい節点から探索する疑似スレッドはスレーブ疑似スレッド（図 2 の T2, T3）とする．複数経路並行探索では，疑似スレッドと，疑似スレッドを管理する疑似スレッドスケジューラにより探索を行う．具体的には，疑似スレッドスケジューラが各疑似スレッドの探索順と探索量を決める．そして各疑似スレッドが，決められた節点数だけ探索する．このように探索順や探索量の決定と探索を繰り返すことによって，複数の探索経路を並行に探索する．

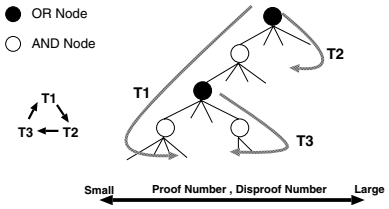

図 2 複数経路並行探索
Fig. 2 Multi-Path Concurrent Search

複数経路並行探索では，単一プロセッサで複数の経路を並行に探索するため，証明数の小さい節点が解の場合には，従来手法よりも証明数が大きい節点の探索量が多くなり，求解に時間がかかる．また，リーダ疑似スレッドの探索割合が少ないとスレーブ疑似スレッドの探索可能な節点が増えにくい．そこで，リーダ疑似スレッドの探索割合，具体的には 1 回あたりの探索量を増加させ，証明数の小さい節点からの探索割合を大きくすることで，速度低下を抑える．

複数経路並行探索は AND/OR 木階層的挟み撃ち探索と同様，証明数の小さい節点からのみの探索では解の発見に長時間かかる証明数の大きい節点が解の問題において，証明数の大きい節点から探索するスレーブ疑似スレッドによる解の早期発見が多く，証明数の小さい節点からのみの探索と比べて高速化する[16]．

4.2 複数経路並行探索の並列化

複数経路並行探索の評価結果[16] より，マルチプロセッサを使用する場合でも，プロセッサ数以上の経路を並行に探索することで高速化できると考えられる．そこで，プロセッサ数以上の経路を並行に探索するために，逐次探索である複数経路並行探索を応用し，**図 3** のように複数経路並行探索の複数疑似スレッドをそれぞれのプロセッサで並行に探索し，並列処理を行う．

図 3 に並列複数経路並行探索におけるプロセッサと疑似スレッドを示す．各プロセッサはそれぞれ疑似スレッドスケジューラを持ち，各プロセッサごとに独立して探索を行う．これにより，AND/OR 木階層的挟み撃ち探索と同様，全プロセッサで同期を取ることなく並列探索できる．図 3 は，プロセッサ 1 では疑似スレッド T1 と T2 を，プロセッサ 2 では疑似スレッド T3 と T4 を，それぞれのプロセッサの疑似スレッドスケジューラによって管理し，並行に探索する様子を示したものである．

スレーブ疑似スレッドはリーダ疑似スレッドの探索

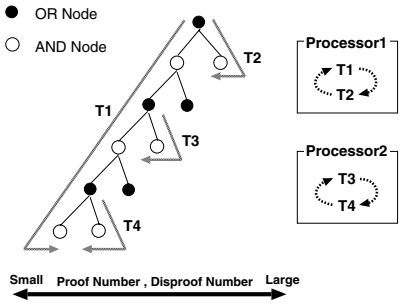

図 3 並列複数経路並行探索
Fig. 3 Parallelized Multi-Path Concurrent Search

経路上の節点を証明すると，リーダ疑似スレッドに通知する必要がある．スレーブ疑似スレッドのみを処理するプロセッサでは，スレーブ疑似スレッドは節点を証明したことをリーダ疑似スレッドへ通知し，次の節点の探索を続ける．また，リーダ疑似スレッドとスレーブ疑似スレッドを並行に処理するプロセッサでは，逐次探索の複数経路並行探索と同様，スレーブ疑似スレッドは節点を証明したことをリーダ疑似スレッドに通知し，疑似スレッドスケジューラがリーダ疑似スレッドの探索に切替える．これにより，探索する必要のない節点の探索を行う可能性を低くすることができる．

逐次探索である複数経路並行探索では，リーダ疑似スレッドの探索割合を多くし，証明数が小さい節点の探索比率を高くすることで，証明数の小さい節点からの探索ですぐに解を発見できる問題において，本来は探索しない経路も並行に探索することによる速度低下を防いでいる[16]．複数のプロセッサを使用する並列複数経路並行探索では，証明数が小さい節点を多く探索するために，各プロセッサが並行に探索する疑似スレッド数を変える．具体的には，証明数の小さい節点から探索するリーダ疑似スレッドを探索するプロセッサの疑似スレッド数を他プロセッサに比べて少なくする．これにより，証明数の小さい節点から探索する経路を，証明数の大きい節点から探索する各経路に比べて多く探索することができ，証明数の小さい節点の解の発見が遅くなる可能性が減少する．特に，1 つのプロセッサがリーダ疑似スレッドの探索のみを行う場合，そのプロセッサにおける探索は df-pn と同等になるため，df-pn より速度が大きく低下することがなくなる．

さらに並列複数経路並行探索では，証明数の大きい節点が解の問題に対して，複数経路並行探索と同様，証明数の大きい節点からの経路を多く並行探索することにより，同数のプロセッサを使用した AND/OR 木階層的挟み撃ち探索よりも高速化すると考えられる．

表 1 評価環境
Table 1 Evaluation Envairoment

CPU	PentiumIII 700MHz × 2
メモリ	1GB
OS	Linux 2.4.26

表 2 3 疑似スレッドのプロセッサへの配分
Table 2 Pseudo threads assignment using 3 pseudo threads

配分方式	Processor1	Processor2
3A	L	S1, S2
3B	L, S1	S2

表 3 4 疑似スレッドのプロセッサへの配分
Table 3 Pseudo threads assignment using 4 pseudo threads

配分方式	Processor1	Processor2
4A	L	S1, S2, S3
4B	L, S1	S2, S3
4C	L, S1, S2	S3

5. 性能評価

並列複数経路並行探索を詰将棋問題の求解を例として実装し，評価した．評価には詰将棋の問題集「将棋無双」[17] の 100 問を用いる．探索の制限時間は 18000 秒とし，評価する手法の全てで制限時間以内に解けなかった問題は評価対象としない．また，評価環境として**表 1** に示す，主メモリを共有する 2 つのプロセッサが接続されたマルチプロセッサを使用する．

本稿では，従来の探索手法で解を得るために長時間必要な問題ほど，提案手法がより高速化することを目的とする．このため，各探索手法における問題セットの総探索時間を用いて評価することで，探索時間が長い問題ほど影響が大きくなるような高速化率を計算できる．また，制限時間以内に解けなかった問題の探索時間は 18000 秒として計算する．

複数経路並行探索では，1 疑似スレッドが 1 回に探索する節点数である基準探索節点数は，1000 程度が良いことが確認されている[16]．よって，並列複数経路並行探索でも基準探索節点数を 1000 として評価する．

まず，並列複数経路並行探索において，同数の疑似スレッドで各プロセッサの疑似スレッド数が異なる場合の影響を検証する．リーダ疑似スレッドを L，スレーブ疑似スレッドを S1，S2，S3 として，各プロセッサへの疑似スレッド配分例を**表 2**，**表 3** に示す．表 2 は 3 疑似スレッドの場合の配分例を示し，3A は 1 つのプロセッサがリーダ疑似スレッドのみ，もう 1 つのプロセッサが 2 つの疑似スレッドを探索し，3B は 1 つ

表 4 3 疑似スレッドにおける疑似スレッド配分の高速化率に与える影響
Table 4 Efficient of speedup by pseudo thread assignment using 3 pseudo threads

	Speedup
3A	1.88
3B	1.20

表 5 4 疑似スレッドにおける疑似スレッド配分の高速化率に与える影響
Table 5 Efficient of speedup by pseudo thread assignment using 4 pseudo threads

	Speedup
4A	1.82
4B	1.78
4C	0.93

のプロセッサがリーダ疑似スレッドと 1 つのスレーブ疑似スレッド，もう 1 つのプロセッサが 1 つのスレーブ疑似スレッドを探索する．表 2，表 3 に示した疑似スレッド配分方式における df-pn に対する高速化率を**表 4**，**表 5** に示す．

表 4, 表 5 より，リーダ疑似スレッドとスレーブ疑似スレッドを同じプロセッサで探索しない配分方式 3A，4A は，同じプロセッサでリーダ疑似スレッドとスレーブ疑似スレッドを並行に探索する 3B，4B，4C より高速化率が高いことが確認できた．証明数が小さい節点が解の場合には，リーダ疑似スレッドの探索により早く解が求まるため，リーダ疑似スレッドが単独で処理されれば df-pn と同時間で探索できる．しかし，リーダ疑似スレッドとスレーブ疑似スレッドを同一プロセッサで並行に探索すると，これらの問題で速度低下が起こり，全体での高速化率も下がったと考えられる．

次に，2 プロセッサにおける (疑似) スレッド数の変化による影響を検証するため，AND/OR 木階層的挟み撃ち探索 (AOHPAS) と並列複数経路並行探索 (PMPS) の（疑似）スレッド数を変化させ，評価する．2 プロセッサで実行したときの df-pn に対する高速化率を**図 4** に示す．PMPS の高速化率は，疑似スレッド数 3，4 の場合は最も高速化した 3A，4A を用い，疑似スレッド数 2 の場合は 1 プロセッサに対して 1 疑似スレッドとなるため AOHPAS と同値とする．

図 4 より，AOHPAS は，2 プロセッサでの実行ではスレッド数が多くなればなるほど高速化率が低下し，4 スレッドの実行では 1.13 倍となった．証明数の大きい節点から探索するスレーブスレッドが多ければ，証明数の大きい節点の解がより早く発見されると考えられるが，それぞれのスレッドの計算資源が少なくなるため，速度が低下したと考えられる．特にリーダ疑似スレッドの探索割合が小さくなるので，df-pn では早期に発見できた解を見つけるために時間がかかる．

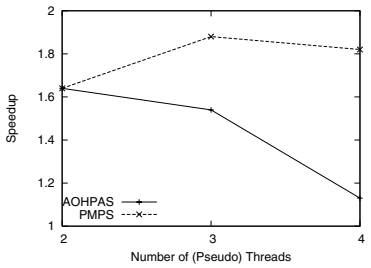

図 4 AND/OR 木階層的挟み撃ち探索と並列複数経路並行探索の（疑似）スレッド数毎の高速化率
Fig. 4 Speedup of AOHPAS and PMPS on 2 to 4 (pseudo)threads

PMPS は，疑似スレッド数 3，4 共に AOHPAS よりも高速化率が高い．PMPS では，AOHPAS のように証明数の大きい節点からの探索経路を並行に探索するのに加え，リーダ疑似スレッドが 1 プロセッサを占有するため，証明数の大きい節点の解を早期に発見でき，また，df-pn で短時間で解くことのできる問題に対しても速度低下が起こりにくい．さらに，OS のスレッドスケジューリングよりも，PMPS の疑似スレッドスケジューリングの方がオーバヘッドが小さい[16]ため，OS がスレッドを切替える AOHPAS に比べて PMPS が高速に処理することができた．

また，疑似スレッド数 3 の場合の方が，1.88 倍と，疑似スレッド数 4 の場合の 1.82 倍よりも高速化率が高い．これは，図 4 に示した PMPS では，証明数の小さい節点からの探索割合が小さくなることはないが，疑似スレッド数が増えると疑似スレッドを切替えるオーバヘッドも増加するためであると考えられる．

最後に，並列複数経路並行探索と AND/OR 木階層的挟み撃ち探索，複数経路並行探索を比較する．**表 6** に，3（疑似）スレッド用いた場合の PMPS，AOHPAS，複数経路並行探索（MPS）の df-pn に対する高速化率を示す．MPS，df-pn では使用するプロセッサ数は 1 であり，それ以外での使用するプロセッサ数は 2 である．MPS におけるリーダ疑似スレッドの探索量は，スレーブ疑似スレッドの探索量に対して，実験により効果が高いことが確認された 1.75 倍とした．

表 6 より，PMPS は MPS，AOHPAS のどちらと比較しても高速化率が高い．PMPS は MPS に対して 1.67 倍であり，同数の疑似スレッドをプロセッサ毎に分割して処理を行った効果が現れていると言える．ま

表 6 3（疑似）スレッドにおける高速化率
Table 6 Speedups on 3 (pseudo) threads

	Speedup
MPS	1.13
AOHPAS	1.54
PMPS(3A)	1.88

た，AOHPAS は 2 プロセッサで 3 スレッド実行する場合，どのスレッドがどのプロセッサに割り当てられるかは不定である．しかし，PMPS では，リーダ疑似スレッドとスレーブ疑似スレッドを分離し，プロセッサに固定することが可能である．そのため，1 プロセッサはリーダ疑似スレッドのみを実行し，証明数の大きい節点を探索しながら証明数の小さい節点の探索を相対的に増加させることで PMPS は AOHPAS よりも高速化したと考えられる．また，PMPS は df-pn と比較して 1.88 倍の高速化となった．

6. おわりに

本稿では，並列処理における各プロセッサが，それぞれ複数の探索経路を並行に探索することで，プロセッサ数よりも多くの経路を並行に探索することができる並列複数経路並行探索を提案した．並列複数経路並行探索は，プロセッサ数よりも多くの経路を並行に探索することで，少数のプロセッサでも AND/OR 木階層的挟み撃ち探索で多数のプロセッサを用いた時のように高速化できる．

評価の結果，並列複数経路並行探索を用い，探索経路の性質によって探索するプロセッサを決定することで，1 つのプロセッサが 1 つの経路を探索する並列探索，AND/OR 木階層的挟み撃ち探索よりも高速化することが確認された．

また，本稿ではプロセッサ数は 2，疑似スレッド数は 3，4 として評価を行った．今後は，プロセッサ数，疑似スレッド数がより多い場合に，疑似スレッドをどのように各プロセッサに割り当て探索すべきか，さらに検討する必要がある．

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) Feldmann, R.: *Game trees on massively parallel systems*, PhD Thesis, University of Paderborn (1993).
9) Brockington, M. and Schaeffer, J.: APHID GAME-TREE SEARCH, *Advances in Computer Chess*, Vol. 8, pp. 69–91 (1997).
10) Kishimoto, A. and Kotani, Y.: Parallel AND/OR tree search based on proof and disproof numbers, *5th Game Programming Workshop*, Vol. 99, No. 14, pp. 24–30 (1999).
11) 岸本章宏, 小谷善行: 証明数・反証数を用いた AND/OR 木探索アルゴリズムの分散メモリマシンにおける効果的な並列化法について, 情報処理学会ゲーム情報学研究報告, No.2, pp.1–8 (2000).
12) Kishimoto, A. and Schaeffer, J.: Distributed Game-Tree Search Using Transposition Table Driven Work Scheduling, *In Proc. of 31st International Conference on Parallel Processing (ICCP'02)*, IEEE Computer Society Press, pp. 323–330 (2002).
13) 甲斐宗徳, 小林和男, 笠原博徳: 階層型挟み打ち探索による PROLOG OR 並列処理手法, 情報処理学会論文誌, Vol. 29, No. 7, pp. 647–655 (1988).
14) 笠原博徳, 伊藤敦, 田中久充, 伊藤啓介: 実行時間最小マルチプロセッサスケジューリング問題に対する並列最適化アルゴリズム, 電子情報通信学会論文誌 D-I, No. 11, pp. 755–764 (1991).
15) 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).
16) 鷹野芙美代, 佐田宏史, 前川仁孝, 六沢一昭, 宮崎収兄: 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, 情報処理学会計算機アーキテクチャ研究会 ハイパフォーマンス研究会 研究報告, No. ARC-162 HPC-101, pp. 157–162 (2005).
17) 門脇芳雄: 続詰むや詰まざるや, 平凡社 (1978).