ゲーム情報学 2－1
（2000. 3. 17）

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

岸本 章宏　　小谷 善行
東京農工大学工学部情報コミュニケーション工学科
{kishi, kotani}@fairy.ei.tuat.ac.jp

要旨

我々は最近の論文で長井のPDSの非同期な並列版アルゴリズムであるParaPDSを提案した。本論文では良い負荷分散を保つためにParaPDSを修正する。マスター・プロセッサは，スレーブが前回の反復で他の仕事と比べて非常に大きな仕事をしているときはマスターの深さを延長する。

このParaPDSの修正が良い速度向上を達成することを分散メモリマシンで6題のオセロの終盤の問題を解かせることによって確認した。

A Scalable Parallel AND/OR Tree Search Based on Proof and Disproof Numbers on a Distributed Memory Machine

Akihiro Kishimoto and Yoshiyuki Kotani
Department of Information and Communication Engineering, Faculty of Engineering, Tokyo University of Agriculture and Technology
{kishi,kotani}@fairy.ei.tuat.ac.jp

Abstract

Our recent paper proposed ParaPDS, an asynchronous parallel version of Nagai's PDS. This paper modifies ParaPDS to achieve a good load balance. The master processor extends its search depth if a slave processor searches much larger pieces of work than other pieces in a previous iteration.

Modified ParaPDS achieves a good speedup by the experiment of solving 6 Othello endgames on a distributed memory machine.

1 はじめに

AND/OR 木探索は人工知能における探索問題として重要であり，詰め将棋などのゲームや定理証明など様々な応用分野がある。AND/OR 木探索の研究の目的の一つには探索アルゴリズムの高速化が挙げられる。並列計算機を用いることは高速に探索するための一つのアプローチであるが，探索アルゴリズムの並列化には次のようなオーバーヘッドがある。

(1) 探索のオーバーヘッド・・・並列探索によって逐次では探索されなかった無駄な探索節点の増加

(2) 通信のオーバーヘッド・・・プロセッサ間の情報交換による通信遅延

(3) 同期のオーバーヘッド・・・節点間の探索に依存関係があるときなど，あるプロセッサが仕事が終わるのを別のプロセッサが待つときのアイドル時間

これらのオーバーヘッドは互いに依存した関係を持っている。例えば探索節点数を減らそうとするとプロセッサ間の通信や同期のオーバーヘッドが増えるし，同期のオーバーヘッドを減らそうとすると探索節点数の増加をまねいてしまう。一般にこれらのオーバーヘッドが最小である並列アルゴリズムを提案するのは困難であり，研究者は自分の並列アルゴリズムの良さを示すために実験を繰り返してきた。

我々は以前に AND/OR 木探索において非常に有効である逐次アルゴリズム PDS[11] の並列化法である ParaPDS[7] を提案し，16 台のプロセッサを持つ分散メモリマシン上でオセロを用いて実験を行った。ParaPDS は根節点からの探索を行うマスター・プロセッサ (以下マスターと呼ぶ) とマスターから与えられた部分問題を解くスレーブ・プロセッサ (以下スレーブ) を持っているが，マスターの探索の深さが固定であったためにプロセッサ間の仕事の負荷分散が悪いという欠点を持っていた。

他の並列アルゴリズムの研究では，Brockington は [4] において並列 $\alpha\beta$ 法で良い負荷分散を保つためにマスターの探索の深さを延長した。

本論文では Brockington の負荷分散法を ParaPDS に応用する。実験は 16 台のプロセッサを持つ分散メモリマシン上で 6 題のオセロの問題を解かせることによって行い，4.88 倍の台数効果が見られた。

章の構成は次の通りである。2章では関連研究として，2 人零和ゲームの探索で代表的なアルゴリズム $\alpha\beta$ 法の並列化について述べる。3章では逐次アルゴリズムである PDS について簡単に述べる。4章では我々の提案する並列アルゴリズム ParaPDS とマスターの深さ延長法について述べる。5章ではオセロによる実験結果の分析を行い，6章でまとめと残された課題を述べる。

2 関連研究: $\alpha\beta$ 法の並列化

チェスやオセロなどのゲームの中盤の探索には $\alpha\beta$ 法を用いるのが普通である。$\alpha\beta$ 法の並列化はこれまで盛んに研究されていて，並列アルゴリズム自体の研究だけでなく，探索に適した並列マシンの設計 [6] や Cilk[2] や Multigame[16] など並列プログラミング言語による支援の研究も行われている。

$\alpha\beta$ 法のアルゴリズムの並列化には主に二つのアプローチがある。一つは探索のオーバーヘッドを減らそうとするアプローチである。Feldmann の YBWC[5] は Marsland らの PVSplit[9] を一般化した方法である。YBWC は実際のゲーム木の動的並べ変えが有効であることを前提にしたやり方であり，一番左の節点の探索は残りの節点を並列探索する前に行うことを主張している。

もう一つのアプローチは同期のオーバーヘッドを減らすアプローチである。Newborn は最初に PVSplit が持っている同期点を減らす並列アルゴリズムをチェスのプログラムで実装したが，8 台のプロセッサを持つ並列計算機における台数効果は PVSplit よりも悪かった [14]。Brockington らは Newborn の方法を一般化した非同期なアルゴリズム APHID[3] を提案し，ライブラリとして提供した。APHID はマスター・スレーブモデルとして見ることができる。APHID は深さ d の探索を行うとき，根節点から深さ d' はマスターが $\alpha\beta$ 法で探索し，スレーブが残りの深さ $(d-d')$ を反復深化法で探索する。APHID の特長は YBWC にある同期点が全く存在しないことである。さらに APHID のマスターは，良い負荷分散を保つために最近の反復で探索節点数が著しく大きな節点は深く探索してスレーブに割り当てる [4]。Brockington は APHID が YBWC よりも良い台数効果を出す可能性があることを実際のゲームを用いて示した。

我々の ParaPDS は Brockington のようなアプローチを並列 AND/OR 木探索に応用したと考えることも可能である。

3 PDS: 逐次アルゴリズム

ゲーム木探索の手法して代表的なものに最良優先探索と深さ優先探索がある。Korf が最良優先探索法である A* アルゴリズムを同じ振舞いをする深さ優先探索アルゴリズム IDA*[8] に変換して以来，様々な最良優先探索アルゴリズムが深さ優先探索アルゴリズムに直されてきた [15, 17]。詳しい説明は [11, 12] にあるが，長井の PDS はそのような研究の流れの一つであると言える。PDS は深さ優先探索法であり，最良優先探索法である Allis らの証明数探索 [1] の探索とほぼ同じ振舞いをする。[1]

PDS のアルゴリズムの特徴は次の通りである。

(1) 証明数と反証数を閾値に用いた深さ優先の反復深化法である。さらに各節点で反復深化を行なう多重反復深化法を用いている。

(2) Allis の最良優先証明節点 [1] と同じく OR 節点では証明数が最小の節点を展開し，AND 節点では反証数が最小の節点を展開する。

(3) 節点の再展開を防ぐために各節点の証明数と反証数を大規模なトランスポジション・テーブルに保持している。

PDS の特徴は次の通りである。

- 深さ優先探索であるので証明数探索に比べて少ないメモリで実装可能である。
- 同じ探索木のときに証明数探索に比べて内部節点の展開回数が少ない。

証明数とは共謀数 [10] という McAllester がミニマックス探索において評価値の信頼度を測る基準として導入した概念を Allis が AND/OR 木探索に応用したものである [1]。証明数は，AND/OR 木において根節点を証明する (先手が勝ち) ために展開しなければならない先端節点数の最小値と定義される。Allis はさらに証明数と双対な概念として反証数を導入した。反証数は根節点を反証する (後手が勝ち) ために展開しなければならない先端節点数の最小値と定義される。

ある節点 n の証明数を $\mathrm{pn}(n)$，反証数 $\mathrm{dn}(n)$ とすると，$\mathrm{pn}(n), \mathrm{dn}(n)$ は次のようにして計算される。

(1) n が未展開の先端節点のとき $\mathrm{pn}(n) = \mathrm{dn}(n) = 1$

(2) n が証明された節点のとき $\mathrm{pn}(n) = 0, \mathrm{dn}(n) = \infty$

(3) n が反証された節点のとき $\mathrm{pn}(n) = \infty, \mathrm{dn}(n) = 0$

(4) n が内部の OR 節点のとき，n の子節点を $n_1, \cdots, n_k$ とすると

$$\mathrm{pn}(n) = \min(\mathrm{pn}(n_1), \cdots, \mathrm{pn}(n_k)) \qquad \mathrm{dn}(n) = \mathrm{dn}(n_1) + \cdots + \mathrm{dn}(n_k)$$

(5) n が内部 AND 節点のとき，n の子節点を $n_1, \cdots, n_k$ とすると

$$\mathrm{pn}(n) = \mathrm{pn}(n_1) + \cdots + \mathrm{pn}(n_k) \qquad \mathrm{dn}(n) = \min(\mathrm{dn}(n_1), \cdots, \mathrm{dn}(n_k))$$

4 ParaPDS: 並列アルゴリズム

本章では提案する並列アルゴリズム ParaPDS について述べる。k 台のプロセッサを持つ計算機上で，ParaPDS は 1 台のマスターと $k-1$ 台のスレーブを持つマスター・スレーブモデルとして見ることができる。マスターは根節点からある程度の深さまで探索を行い，残りの部分問題をスレーブに割り当てる。スレーブのアルゴリズムは 3章で述べた PDS を用いており，マスターから与えられた節点を探索し，探索結果をマスターに送信する。ParaPDS ではスレーブ間で情報の交換を行わないので，スレーブ間での通信

[1] 長井らは [13] において証明数探索と同じ振舞いをする df-pn を提案している。今回の並列化法を用いて df-pn を並列化することも簡単に可能である。

と同期のオーバーヘッドがない。探索のオーバーヘッドは増える可能性があるが，マスターがスレーブに割り当てる節点の選び方を工夫することで回避する。

4.1節では深さが固定のマスターのアルゴリズムについて述べる。4.2節ではスレーブのアルゴリズムについて述べる。4.3節では分散環境における我々のトランスポジション・テーブルの実装について説明する。4.4節ではプロセッサ間になるべく等しい仕事量を与えるために行うマスターの深さ延長アルゴリズムについて述べる。

4.1 マスター・プロセッサのアルゴリズム - 深さ固定のとき

マスター・プロセッサが行うことは以下のことを根節点が証明か反証が終わるまで繰り返す。

(1) 根節点からある程度の深さまでの探索

(2) スレーブへの節点の割り当て

(3) スレーブからの探索結果の受け取り

(4) 証明数・反証数の更新

PDS や証明数探索など多くのアルゴリズムの探索の深さは可変であるが，探索の深さは基本的には深さ d' で固定であり，ParaPDS のマスターは深さ優先探索を用いる。マスターは，深さ d' の節点の探索の負荷が大きなときのみ深さを延長する。深さ優先探索を用いることの利点は上の最良優先探索や幅優先探索において必要な局面のコピーが不必要であり，上の (1)(2)(4) を単純にかつ高速に行うことができる点である。

マスターは，自身のメモリ上にトランスポジション・テーブルを持ち，テーブルの各要素にタグ情報を持っている。タグ情報には，UNCERTAIN マークという現在スレーブによって探索されていることを表すマークと，EXPAND マークという子孫節点に UNCERTAIN マークが付いていることを表すマークがある。

マスターは深さ優先に EXPAND マークの付いた節点の探索とスレーブに割り当てる節点の選択を行う。マスターが深さ d' まで木を展開したとき，現在の証明数と反証数 (トランスポジション・テーブル上にある) を計算して，親節点にその情報を反映させる。子孫の節点に UNCERTAIN マークがなければ EXPAND マークは消去する。[2]また，ある節点が証明 (反証) されたために無駄に割り当てられている節点が出てくることがある。マスターは，スレーブがそのような節点を探索しているときには探索を中止するように命令する。

マスターは深さ優先探索で AND/OR 木を探索しながら，スレーブに割り当てるべき節点の選択を行う。この選択方法には Allis の最良優先証明節点 [1] を緩めた基準を用いる。$n.\phi$ を OR(AND) 節点 n における証明数 (反証数) として，$n.\delta$ を OR(AND) 節点 n における反証数 (証明数) とする。さらに $n.\Delta\phi, n.\Delta\delta$ をそれぞれ $n.\phi, n.\delta$ に加えられる小さな正の整数として，n_i を n の子節点とする。n_i が次の条件を満たすとき，スレーブに割り当てることができる候補節点になる。

$$n_i.\delta - n.\phi < n.\Delta\phi$$

上の条件を満たしたとき，$n_i.\Delta\phi, n_i.\Delta\delta$ はそれぞれ次のように設定される。

$$\begin{aligned} n_i.\Delta\phi &:= n.\Delta\delta \\ n_i.\Delta\delta &:= n.\phi + n.\Delta\phi - n_i.\delta \end{aligned}$$

マスターは上の基準を満たすかどうかを根節点から調べる。マスターは，根節点から深さ d' の節点 n まで上の基準を満たしていて，さらに n が UNCERTAIN マークが付いていないときは n をスレーブに閾値 $n.\phi + n.\Delta\phi$ で割り当てる。

[2]EXPAND マークはスレーブに割り当てられた節点で，証明数・反証数が一時的に大きくなった節点をたどるのに必要である。マスターのトランスポジション・テーブルは PDS のテーブルとは違い，常に新しい情報に更新する。

マスターのスレーブへの割り当ては，根節点を $root$ とすると $root.\Delta\phi = root.\Delta\delta = 1$ から開始する。[3]スレーブに割り当てる仕事の数が少なく，さらに $root.\phi$ と $root.\delta$ の値が前回のマスターの探索のときと同じ値のときは $root.\Delta\phi$ か $root.\Delta\delta$ を増加させる。$root.\Delta\phi, root.\Delta\delta$ の増加方法は PDS と同じ戦略をとる。

(1) $root.\phi + root.\Delta\phi \leq root.\delta + root.\Delta\delta$ ならば，$root.\Delta\phi := root.\Delta\phi + 1$

(2) そうでなければ，$root.\Delta\delta := root.\Delta\delta + 1$

この戦略は直観的には根節点が現在証明しやすいのか反証しやすいのかを考慮して，それに適した戦略を取ることを表している。

4.2 スレーブ・プロセッサのアルゴリズム

スレーブのアルゴリズムは基本的に PDS を用いているが，一箇所だけ修正している。逐次アルゴリズムの PDS は AND/OR 木が証明か反証が終わるまで探索を続けるが，ParaPDS のスレーブはマスターが決めた閾値に至るまでしか探索を行わない。スレーブのアルゴリズムの概要は以下のようになる。

(1) マスターによって与えられた閾値 $n.\phi + n.\Delta\phi$ まで探索を行なう。

(2) 各反復の探索が終わるたびに $n.\phi$ と $n.\delta$ をマスターに返して，マスターのトランスポジション・テーブルに反映させる。

(3) 探索が終わるとマスターに UNCERTAIN マークを消すように要請する。

スレーブはマスターが根節点の証明か反証が終了するまで上のアルゴリズムを繰り返す。

スレーブがマスターから与えられた節点の探索を終了して，次の節点がまだ割り当てられていないとき，スレーブに暇な時間ができる。このようなときは，[3] と同じようにスレーブは以前にマスターから与えられた節点の探索を行う。マスターから新たな節点が割り当てられたときには直ちにその探索を中断して，その新しい節点の探索を開始する。

4.3 トランスポジション・テーブル

分散環境では自身のプロセッサのメモリ上にない情報は他のプロセッサと通信する必要があるので，トランスポジション・テーブルをどのように実装するかによって性能に大きな違いが出る。Feldmann は分散トランスポジション・テーブルという一つの大きなテーブルの一部を等しく分割して各プロセッサが保持する方法が良い結果を出すことを主張した [5]。しかし，分散トランスポジション・テーブルには通信のオーバーヘッドが非常に大きいという欠点がある。証明数・反証数の計算は，$\alpha\beta$ 法の評価関数の計算に比べて計算時間がかからないので，通信のオーバーヘッドの影響を受けやすい。[4]

ParaPDS では，局所トランスポジション・テーブルという各プロセッサが独自にトランスポジション・テーブルを持つ方式を用いている。局所トランスポジション・テーブルではプロセッサ間でテーブルの情報交換が起こらないが，探索のオーバーヘッドを増やす可能性がある。しかし，根節点からある程度の深さまでの情報をマスターのみが保持しているので，スレーブに対する上手な節点の割り当て方をマスターが行えば探索のオーバーヘッドをある程度減らすことが可能である。

k をスレーブの数として，C を全スレーブに現在割り当てられている節点の数，w_i をスレーブ i に割り当てられている節点の数，ε を小さな正の正数とする。マスターは，節点 n を次のようにして割り当てる。

(1) n が前回スレーブ i に割り当てられていて $w_i < \lfloor \frac{C}{k} \rfloor + \varepsilon$ ならば，スレーブ i に割り当てる。

(2) そうでなければ，現在マスターから与えられた節点がないプロセッサに割り当てる。

(3) (1)(2) を満たさなければ，ラウンド・ロビン方式で割り当てる。

[3] このときは最良優先証明節点と同じである。

[4] 我々は ParaPDS に実際に分散トランスポジション・テーブルを実装したが，通信のオーバーヘッドの影響で単位時間あたりの局面の展開速度が 0.1 倍程度になった。

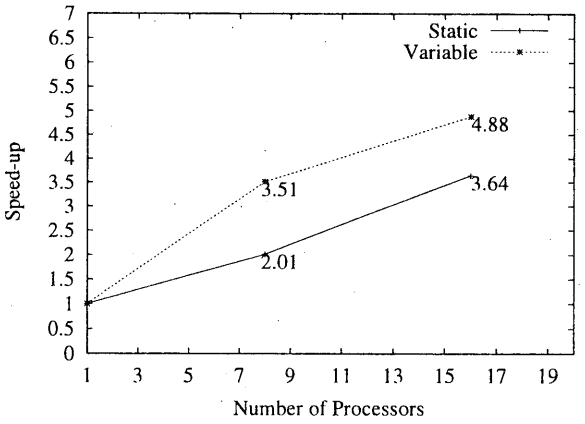

図 1: 台数効果

4.4 マスター・プロセッサの探索の深さの延長

マスターからスレーブに割り当てる節点には同じ証明(反証)数であっても，実際に証明(反証)するために展開する節点数には大きな差がある。[7] では根節点の証明数(反証数)が大きくなるにつれてスレーブに割り当てられる節点の数自体が減少し，スレーブが有効な探索を行っていなかった。

上の問題を解決する方法の一つとして，探索節点数の大きそうな節点はマスターが(さらに深く探索することで)より多くのスレーブに探索させる方法がある。我々の解決策は Brockington の方法 [4] と同じである。マスターは，最近スレーブに割り当てた節点の前回の反復における探索節点数を持っている。マスターは根節点からの探索とスレーブへの割り当ての処理が終わるたびに，探索節点数の平均値と最も探索節点数の大きかった節点 n を求める。n の節点数が平均値に比べて著しく大きなときには n の深さを延長する。このときに延長する節点の子節点の証明数と反証数の情報はマスターのトランスポジション・テーブルにはないので，延長する節点を前回に探索したスレーブに子節点の情報を要求して，マスターのトランスポジション・テーブルにその情報を書き込む。

5 実験結果

ParaPDS を分散メモリマシンである HITACHI SR-2201 上で実装した。実験は 19 マスの空白があるオセロの局面を 6 題用いて行った。我々の現在の実装では，各プロセッサは 1 秒間に約 10,000 程度の節点を展開できる。実験はマスターの探索の深さは 6 で行い，各プロセッサのトランスポジション・テーブルには 700,000 節点を登録できるようにした。プロセッサ数の増加による台数効果は以下のようにして計算できる。

$$\text{台数効果} = \frac{\text{逐次 PDS による探索時間}}{\text{ParaPDS による探索時間}}$$

台数効果のグラフを図 1に示す。グラフ中の Static が深さが固定であることを表し，Variable がマスターの深さを延長したときの結果を表す。いずれの場合もプロセッサ台数が増加するにつれて速度向上が見られるが，マスターの深さを延長したときの方がより良い向上が見られる。この結果から，マスターの深さを延長することは有効であることが言える。

さら大きな台数効果を達成するためには探索のオーバーヘッドを減らす必要がある。スレーブが逐次アルゴリズム PDS に比べてどれだけ無駄な探索をしているかを以下のようにして調べた。

$$\text{スレーブの探索のオーバーヘッド} = (\frac{\text{全スレーブの探索節点数}}{\text{逐次 PDS での探索節点数}} - 1) * 100$$

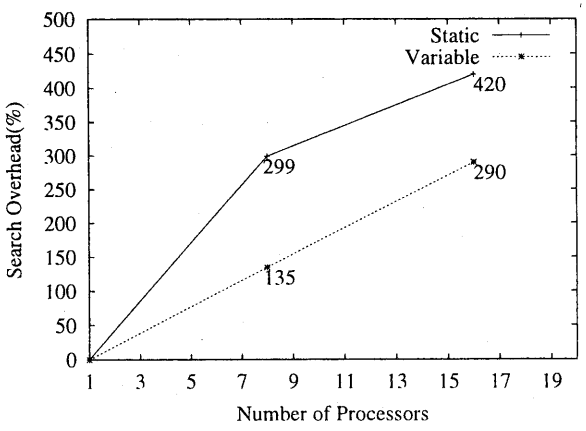

図 2: スレーブの探索のオーバーヘッド

スレーブの探索のオーバーヘッドのグラフを図2に示す。マスターの深さを延長することによって深さが固定のときに行われていた有効でない探索を減らすことができたと言えるが，マスターの深さを延長したときもプロセッサ数が増加するにつれて探索のオーバーヘッドも大きく増加している。

スレーブの探索のオーバーヘッドの増加の理由としては，スレーブのテーブルの実装が局所トランスポジション・テーブルであることよる無駄な節点の再展開が考えられる。この無駄な再展開を防ぐためには，単位時間の展開節点数を保持しながら，分散トランスポジション・テーブルのような節点情報の共有を重要な局面にのみ行う必要がある。

6 まとめと今後の課題

本論文において AND/OR 木探索を行う ParaPDS のマスターの深さを延長した。実験は分散メモリマシン上でオセロを用いて行い，マスターの深さが固定の ParaPDS よりも良い速度向上が見られた。

今後の課題としては，さらなる台数効果をだすために探索のオーバーヘッドを減らすことが挙げられる。そのためには探索に重要であると思われる節点の情報をプロセッサ間で共有する方法を開発する必要がある。

参考文献

[1] Louis V. Allis, Maarten van der Meulen, and H. Jaap van den Herik. Proof-number search. *Artificial Intelligence*, 66:91–124, 1994.

[2] Robert D. Blumofe, Christopher F. Joerg, Bradley C. Kuszmaul, Charles E. Leiserson, Keith H. Randall, Andrew Shaw, and Yuli Zhou. Cilk: an efficient multithreaded runtime system. In *Fifth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming(PPoPP'95)*, 1995.

[3] Mark Brockington and Jonathan Schaeffer. APHID game-tree search. In *Advances in Computer Chess*, volume 8, pages 69–91, 1997.

[4] Mark G. Brockington. *Asynchronous parallel game-tree search.* PhD thesis, University of Alberta, 1998.

[5] Rainer Feldmann. *Spielbaumsuche auf massiv parallen Systemen(Game trees on massively parallel systems).* PhD thesis, University of Paderborn, 1993. English translation is also available.

[6] Feng H. Hsu. *Large scale parallelization of alpha-beta search: an algorithmic and architechtual study with computer chess.* PhD thesis, Carnegie Mellon University, 1990.

[7] Akihiro Kishimoto and Yoshiyuki Kotani. Parallel AND/OR tree search based on proof and disproof numbers. In *5th Game Programming Workshop,IPSJ Symposium Series Vol. 99, No. 14*, pages 24–30, 1999.

[8] Richard Korf. Depth-first iterative deepening: an optimal admissible tree search. *Artificial Intelligence*, 27:97–109, 1985.

[9] Tony A. Marsland and Fred Popowich. Parallel game-tree search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 7(4):442–452, 1985.

[10] David A. McAllester. Conspiracy numbers for min-max search. *Artificial Intelligence*, 35:287–310, 1988.

[11] Ayumu Nagai. A new AND/OR tree search algorithm using proof number and disproof number. Booklet for the Workshop of the First International Conference on Computers and Games(CG'98), 1998. The postscript file is available at http://www.etl.go.jp/~7236/Events/workshop98/.

[12] Ayumu Nagai. A new depth-first search algorithm for AND/OR trees. Master's thesis, University of Tokyo, 1999.

[13] Ayumu Nagai and Hiroshi Imai. Proof for the equivalence between some best-first algorithms and depth-first algorithms for AND/OR trees. In *KOREA-JAPAN Joint Workshop on Algorithms and Computation*, pages 163–170, 1999.

[14] Monroe Newborn. Unsynchronized iteratively deepening parallel alpha-beta search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(5):687–694, 1988.

[15] Aske Plaat, Jonathan Schaeffer, Wim Pijls, and Arie de Bruin. SSS* = Alpha-Beta+TT. Technical Report 94-17, University of Alberta, 1994.

[16] John W. Romein, Henri E. Bal, and Dick Grune. An application domain specific language for describing board games. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '97)*, volume I, pages 305–314, Las Vegas, NV, July 1997. CSREA.

[17] Masahiro Seo. The C* algorithm for AND/OR tree search and its application to a tsume-shogi program. Master's thesis, University of Tokyo, 1995.