

修士論文
コンピュータ囲碁における Root 並列化について

東京工業大学
大学院情報理工学研究科
数理・計算科学専攻
副島 佑介
08M37189
平成 21 年度 修士論文
指導教官 佐々政孝 教授

平成 22 年 1 月 29 日

概要

コンピュータ囲碁では、モンテカルロ木探索を利用する方法が最も有効であり、モンテカルロ木探索の並列化は、囲碁プログラムの強さを改善する手法の一つである。現在主流となる並列化手法には Root 並列化と Tree 並列化が存在する。本研究では強豪囲碁プログラム Fuego を用いて、Root 並列化と Tree 並列化の両手法の再評価を行った。結果 Tree 並列化の方が先行研究に比べて、より性能が高い並列化手法であることが示された。

次に、従来の Root 並列化手法であった総和制と、今回提案するコンピュータ囲碁における Root 並列化手法の合議制との 2 つの Root 並列化を Fuego に実装し、その性能を調査した。その結果、合議制 Root 並列化の方が総和制に比べより性能が高い事が示された。

最後に、大規模な CPU コア数を用いた場合の Root 並列化の効果の予測を行った。64CPU コアを用いた実験で、Chaslot らの先行研究とは異なり、Root 並列化の有効性だけでなく限界も示すことができた。一致率を用いた実験では、Root 並列化の限界点および、効果の特徴などを発見する事ができた。

目次

第 1 章	序論	7
1.1	人工知能とゲーム	7
1.2	並列化の重要性	8
1.3	本論文の貢献	8
1.4	本論文の構成	9
第 2 章	関連研究	10
2.1	ゲーム木探索	10
2.1.1	ゲーム木探索	10
2.1.2	モンテカルロ囲碁	13
2.1.3	モンテカルロ木探索	15
2.1.4	モンテカルロ木探索の理論的背景	15
2.2	モンテカルロ木探索の改善	18
2.2.1	囲碁の知識を用いたプレイアウトの改善	18
2.2.2	RAVE	18
2.3	モンテカルロ木探索の並列化	19
2.3.1	モンテカルロ木探索のオーバーヘッド	19
2.3.2	Leaf 並列化	20
2.3.3	Tree 並列化	21
2.3.4	Root 並列化	22
2.3.5	先行研究における各並列化手法の比較	24
第 3 章	本研究で実装した Root 並列化	26
3.1	Fuego (対象囲碁プログラム)	26
3.2	総和制の欠点	28
3.3	合議制	28
3.3.1	コンピュータ将棋における合議制	28
3.3.2	コンピュータ囲碁における合議制の提案	30
3.3.3	実装方法	30
3.4	総和制と合議制の差	31
第 4 章	実験と評価	33
4.1	実験環境	33
4.2	実験条件	33
4.3	実験の概要	34
4.4	対戦実験による Root 並列化の有効性	35
4.5	Root 並列化と Tree 並列化との比較	37

4.6	Root 並列化と Tree 並列化の融合	38
4.7	総和制と合議制の比較	40
4.8	総和制と合議制の着手の性質	41
4.9	一致率による Root 並列化の有効性	44
4.9.1	一致率比較実験	44
4.9.2	より多くのプロセスを用いた Root 並列化の一致率	46
4.9.3	最善的 Root 並列化の一致率	46
4.9.4	一位度別一致率	48
4.9.5	局面段階数別一致率	54
第 5 章	結論	56
5.1	まとめ	56
5.2	今後の課題	58

目次

2.1	3×3ゲームのゲーム木探索	10
2.2	MiniMax 木探索	12
2.3	プレイアウト	14
2.4	モンテカルロ木探索のイメージ図	17
2.5	モンテカルロ木探索	18
2.6	Leaf 並列化のイメージ図	20
2.7	Tree 並列化のイメージ図	21
2.8	Root 並列化のイメージ図	23
2.9	Root 並列化における集計方法の具体例	24
3.1	fuego	26
3.2	fuego のモジュール図 文献 [5] 参照	27
3.3	コンピュータ将棋における合議制 (Bonanza を利用)	29
3.4	合議制のイメージ図	31
3.5	合議システム	32
4.1	9路盤:Root 並列化 (総和制・合議制 4~64 コア) vs 逐次 Fuego	35
4.2	9路盤:Root 並列化 (総和制・合議制 4~64 コア RAVE なし) vs 逐次 Fuego (RAVE なし)	36
4.3	9路盤:Root 並列化 (総和制・合議制 4~64 コア) vs 逐次 Mogo	36
4.4	19路盤:Root 並列化 (総和制・合議制 4~64 コア) vs 逐次 Fuego	37
4.5	19路盤:Root 並列化 (総和制・合議制 4~64 コア) vs 逐次 Mogo	38
4.6	9路盤:Root 並列化 (合議制 64 コア) vs Tree 並列化 (1~8 スレッド)	39
4.7	19路盤:Root 並列化 (合議制 64 コア) vs Tree 並列化 (1~8 スレッド)	39
4.8	合議制が F_{tree} の着手と一致した局面 (白番)	43
4.9	総和制が F_{tree} の着手と一致した局面 (黒番)	43
4.10	9路盤:F80s との着手一致率: 逐次 Fuego, Root 並列化 (合議制・総和制), Tree 並列化	45
4.11	19路盤:F80s との着手一致率: 逐次 Fuego, Root 並列化 (合議制・総和制), Tree 並列化	46
4.12	19路盤:F80s との着手一致率: Root 並列化 (合議制 64-512 コア)	47
4.13	19路盤:F80s との最善的一致率: Root 並列化 (合議制 1-512 コア)	47
4.14	9路盤 ($0 \leq \text{一位度} < 0.2$) の局面における一致率	49
4.15	9路盤 ($0.2 \leq \text{一位度} < 0.4$) の局面における一致率	49
4.16	9路盤 ($0.4 \leq \text{一位度} < 0.6$) の局面における一致率	50
4.17	9路盤 ($0.6 \leq \text{一位度} < 0.8$) の局面における一致率	50
4.18	9路盤 ($0.8 \leq \text{一位度} < 1.0$) の局面における一致率	51

4.19	19路盤 ($0 \leq \text{一位度} < 0.2$) の局面における一致率	51
4.20	19路盤 ($0.2 \leq \text{一位度} < 0.4$) の局面における一致率	52
4.21	19路盤 ($0.4 \leq \text{一位度} < 0.6$) の局面における一致率	52
4.22	19路盤 ($0.6 \leq \text{一位度} < 0.8$) の局面における一致率	53
4.23	19路盤 ($0.8 \leq \text{一位度} < 1.0$) の局面における一致率	53
4.24	9路盤 局面段階数別の合議制一致率	54
4.25	19路盤 局面段階数別の合議制一致率	55
5.1	19路盤 局面段階数別一致率	60
5.2	19路盤 局面段階数別一致率	61
5.3	19路盤 局面段階数別一致率	61
5.4	19路盤 局面段階数別一致率	62
5.5	19路盤 局面段階数別一致率	62
5.6	9路盤 局面段階数別一致率	63
5.7	9路盤 局面段階数別一致率	63
5.8	9路盤 局面段階数別一致率	64
5.9	9路盤 局面段階数別一致率	64
5.10	9路盤 局面段階数別一致率	65

表 目 次

2.1	探索空間 (局面数) の推定	11
2.2	10000 シミュレーションにおける, Single-Run 並列化と Multiple-Runs 並列化の GnuGo3.6 との対戦勝率比較 (T.Cazenave et al. 2006)	24
2.3	13 路盤・一手 1 秒での, Root 並列化, Tree 並列化の GnuGo3.7.10 との対戦勝率比較 (G.Chaslot et al. 2008)	25
2.4	9 路盤 Root 並列化, Tree 並列化の GnuGo3.7.10 との対戦勝率比較 (G.Chaslot et al. 2008)	25
4.1	9 路盤 合議制・総和制の勝率比較 (%) (8 × 8Root 並列化)	39
4.2	19 路盤 合議制・総和制の勝率比較 (%) (8 × 8Root 並列化)	40
4.3	9 路盤 8 × 8Root 並列化& 64 コア Root 並列化における合議制 vs 総和制の勝率比較 (%)	40
4.4	19 路盤 8 × 8Root 並列化& 64 コア Root 並列化における合議制 vs 総和制の勝率比較 (%)	40
4.5	9 路盤 64 コア 200 試合の棋譜 (着手の不一致数 3.4% (187/5500 局面))	42
4.6	9 路盤 8 × 8 200 試合の棋譜 (着手の不一致数 2.3% (159/6759 局面))	42
4.7	19 路盤 64 コア 200 試合の棋譜 (着手の不一致数 8.7% (2277/26064 局面))	42
4.8	19 路盤 8 × 8 200 試合の棋譜 (着手の不一致数 4.9% (1314/27032 局面))	43
4.9	図 4.8 での総和制と合議制の各着手の情報 (上位 5 位まで)	44
4.10	図 4.9 での総和制と合議制の各着手の情報 (上位 5 位まで)	44
4.11	全局面における一位度の分類具合	48

第1章 序論

1.1 人工知能とゲーム

探索問題はコンピュータサイエンスにおける問題を解決する為に重要な役割を持っており、また探索アルゴリズムは様々な実用的な問題に生かす事が可能である。ゲームは人工知能における探索アルゴリズムの研究のよい対象として長年見なされてきている。理由としては、まず、ゲームは研究者に強く、明確なモチベーションを与える事が出来る。例えば、人間の名人に勝ちたいという目標などがそれに当たる。次に、ゲームはルールと勝ち負けが非常にシンプルである。その為、研究者にとっては、他の実世界の対象よりも評価がしやすい。

過去 50 年、研究者達はオセロやチェッカー、チェスなどのコンピュータゲームに膨大な労力を費やしてきた。チェッカーは 1994 年に世界チャンピオンに [12]、オセロは 1996 年に世界チャンピオンに完勝した [17]。チェスにおいては、1997 年に IBM の Deep Blue が当時の世界チャンピオン Kasparov を破る快挙を果たした [11]。将棋では 2009 年現在プロ 4 段のレベルにまで到達している。その中で、囲碁は現時点で最も優れたコンピュータでも 19 路盤においてアマチュア 3,4 段程度である。強いコンピュータ囲碁の実現は、どのゲームと比較しても最も難しい課題とされ、人工知能の最も難しい挑戦の一つであると思われる。

囲碁の歴史はおよそ四千年におよび、今でも世界中で多くの人を魅了し続けている。そのルールは単純ではあるが、その複雑さは、70 年代後半から始まった優れたコンピュータ囲碁の実現を、幾度となく阻んできた。囲碁は多くの面で他のプログラム (チェスや将棋) と異なっている。まず、探索空間の大きさが挙げられる。典型的な囲碁盤の大きさである 19 路盤では、その探索空間は 10^{171} にも及ぶ。次にその局面の評価のし難さが挙げられる。チェスや将棋のように、駒の損得や王手、チェックメイトのように明確な局面の評価基準が存在しない。囲碁には人間独特の評価が必要であり、それをコンピュータで実現する為に GnuGo のようなパターンで全ての場合を評価する方法が用いられてきたが、その強さは弱いアマチュアレベル程度に留まっていた。

近年、囲碁におけるこのような問題点に進展があった。それがモンテカルロ木探索である。モンテカルロ木探索は、その局面からゲームの終局までランダムに合法手を選択し局面を展開するランダムシミュレーション (プレイアウト) を繰り返し、その終局面の勝敗結果を用いて、局面の評価を行い探索木を探索する手法である。この手法のアイデアは終局面の評価は、途中局面の評価に比べはるかに易しいという点である。またその為に評価関数は使わない。よって、他の新しいゲームや知識表現の困難なゲームにおいても有効な手法と言える。最近のトップクラスのコンピュータ囲碁ではこの探索手法が用いられており、その有効性が示されている。

1.2 並列化の重要性

モンテカルロ木探索を更に改良する方法として、より大きな探索木の構築と、プレイアウト数の増加が挙げられる。より大きな探索木の実現はより多くの着手を探索することができ、より有望な局面でプレイアウトを行う事ができる。また、プレイアウト数の増加によって、局面の優劣の評価がより正確になる。

従来、コンピュータゲームはマシンの性能の向上と共に進化してきた。より高速なマシンは、より多くの探索をより深く行うことを可能にする為である。しかし近年はシングルコアマシン当たりの CPU 速度の向上率は以前より低い為、こうしたハードウェアの改善による恩恵を受けにくくなってきている。よって、前述した改良を、逐次で行う事はより困難になってきている。また、それとは対称的に、近年コンピュータのマルチコア化が進んできている。誰でもマルチコアを常備するコンピュータが手に入る時代になった。以上の流れから、モンテカルロ木探索を改良する方法として並列化が重要かつ主流となってきた。

各プロセス(またはスレッド)が探索木を共有する Tree 並列化 [6, 7] は、代表的な並列化法である。この方法では、より大きな木を構築できるが、木の共有の際に生じるオーバーヘッドのため、CPU コア数の増加による探索速度改善の効率が低下するという欠点がある。

一方、Root 並列化 [1] は、プロセス間で独自にモンテカルロ木探索を行うため、CPU コア数の増加に伴い、探索速度も上昇する。しかし、並列化を行っても逐次に比べて各プロセスの探索木が改善されないため、Tree 並列化の方が、より有望な局面でプレイアウトを行っている可能性がある。

代表的な囲碁プログラムで利用されている手法は、Tree 並列化であるが [7, 5]、Chaslot らは、Root 並列化は Tree 並列化よりも有効であると報告している [2]。しかし、文献 [2] では、Root 並列化がスーパーリニアな効果を得ているが、Root 並列化の方がなぜ有効であるかを示す具体的な実験データが示されていない。また、Root 並列化について具体的にどの程度まで効果があるかを示した論文も今の所存在していない。

1.3 本論文の貢献

本論文の貢献は大きく分けて以下の 3 点になる。以下、それぞれについて説明する。

Tree 並列化と Root 並列化の性能の再評価

並列木探索が、CPU コア数以上の性能を発揮する場合には、逐次探索に改良の余地がある場合が多い。このため、Chaslot らの Root 並列化の結果 [2] でも同様のことが生じている可能性がある (2 章関連研究を参照)。例えば、Chaslot らのプログラムには、逐次モンテカルロ木探索の性能を大幅に改良する RAVE [22] が実装されていない。また、Chaslot らの実験では、16CPU コアまでしか利用しておらず、より多くの CPU を利用したときにも Root 並列化が台数効果を引き続き得られるかどうかは、研究課題の一つである。そこで今回我々は Root 並列化と Tree 並列化の効果差の再評価を行った。

本研究で述べる Root 並列化は、合計 64 コアの CPU から構成される分散メモリ環境で、Fuego を利用して実装している。Fuego は現状で最も強い囲碁プログラムの一つであるだけでなく、様々なオプションがあり、RAVE なしのバージョンを実行することもできる。

また、性能のよい Tree 並列化も実装されている為、Chaslot らの実験結果を再評価するのに理想的なプログラムである。

総和制と合議制の比較

本研究では、Root 並列化の手法として先行研究で提案されていた総和制だけでなく、コンピュータ将棋で有望とされている合議制 [25] に基づく Root 並列化をコンピュータ囲碁において実装した。両手法の効果を、対戦結果や、着手の性質等で評価し比較をした。その結果、囲碁プログラムにおいても合議制がより有力な手法であることがわかった。

大規模な数の CPU コアを用いた場合の Root 並列化の効果の予測

囲碁における Root 並列化の効果が、どの程度、またはどの様であるかについて言及した論文はほとんどない。本研究では Root 並列化の台数効果について、最大 512 プロセスまでを用いて

1. 自己対戦や他のプログラムとの対戦勝率
2. 強いプログラムとの着手の一致率

という基準で実験・計測し、Root 並列化の効果の限界点、特徴について調べた。

1.4 本論文の構成

以降、本論文の構成は次のようである。

- 2 章: 関連研究
モンテカルロ木探索と、その並列化手法についての先行研究について述べる。
- 3 章: 提案手法
本研究の提案手法について述べる。
- 4 章: 実験と評価
本研究の実験方法、および実験結果と評価について述べる。
- 5 章: 結論
本研究の結論と、今後の課題について述べる。

第2章 関連研究

本章では関連研究として、2.1章ではゲーム木探索について、従来手法とUCTモンテカルロ木探索について説明する。そして2.2章ではモンテカルロ木探索の様々な並列化手法について説明し、2.3章でコンピュータ将棋における合議制について説明する。

2.1 ゲーム木探索

2.1.1 ゲーム木探索

本節ではゲーム木探索について説明する。

ゲーム木探索とは、コンピュータプレイヤーがある局面からの最善手を求める為の手法として古くから用いられている探索手法である。ゲーム木探索では、局面を節点(節点)として表現する。現在の局面をルート節点、局面からの合法手を枝、親節点から合法手によって推移可能な局面を子節点として用いる事で、ゲームの将来の局面を木構造によって表現したものである。簡単なゲーム(例えば3×3ゲーム(図2.1))であれば、初期局面から全ての実現可能な局面を展開した後に、プレイヤーにとって有望な局面となるように手を選択していけばよい。しかしより複雑なゲームでは、実現可能な局面(以下これを探索空間と呼ぶ)は非常に膨大な数になってしまいゲーム木を完全に作成できない。

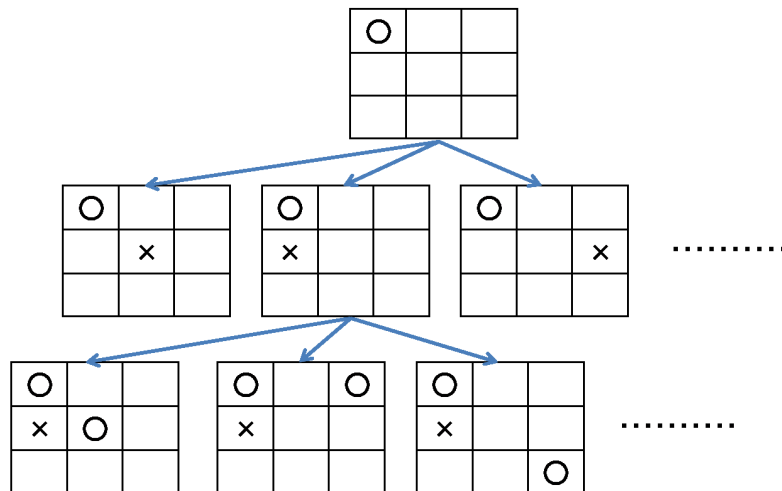


図 2.1: 3×3 ゲームのゲーム木探索

探索空間とは、あるゲームに対し、ゲーム木で最終局面まで表す際に必要な大きさ、つまり全局面の数にあたる。例として表 2.1 に主なゲームの探索空間の大きさの概算を示す。

複雑なゲームでは、表から分かるように、全局面を展開する事は現実的に不可能である¹。例えば、最も複雑なゲームとして囲碁が挙げられる。囲碁では探索空間が 10^{171} にも及び、勝敗が決するまでの探索を行う事は到底不可能である。

表 2.1: 探索空間 (局面数) の推定

ゲーム	局面数
チェッカー	10^{20}
オセロ	10^{28}
チェス	10^{50}
将棋	10^{71}
囲碁 (9 路盤)	10^{38}
囲碁 (19 路盤)	10^{171}

よって実際のゲーム木探索では、時間制限やメモリ制限等でゲーム木の探索を適当な深さ打ち切り、木の末端節点である局面の優劣を判断する評価値を与える。評価値は、木を探索したプレイヤーが有利な局面であるほど、高い数値を持たせるのが一般的である。古くから用いられてきたゲーム木探索手法は、この評価値を与える**評価関数**を基にした手法である。

評価関数とは、局面を入力する事で、局面の優劣を数値化して評価値として出力する関数である。一般的に評価関数は、局面からその基準となる評価要素や特徴を抽出し、それぞれの評価要素に対応した重みを与え、その線形和によって表されることが多い。例えば、将棋ならば「駒の価値、駒取りの損得、駒の動きやすさ、成金、玉の危険度」、オセロでは「隅や辺における重要なパターン、裏返す枚数」などの評価要素が存在する。この評価関数を基に、木探索を行う手法を **minimax 木探索** という。この手法について説明をする。

minimax 木探索は、まず木探索を適当な深さで打ち切る。そして末端節点である局面に対して、評価関数によって評価値が与えられる。先手番のプレイヤー (木を探索したプレイヤー) が有利な局面では評価値が大きく、後手番のプレイヤーが有利な局面では評価値が小さいとする。

先手番は評価値を可能な限り大きな手を選択しようとし、後手番は逆に小さくしようとする。先手番の局面の節点を Max 節点、後手番の局面の節点を Min 節点と呼ぶ (図 2.2)。末端の評価値を元に、双方のプレイヤーが最善手を選択していくことで、その木における最善手が決定される。

図 2.2 を用いて説明する。まず末端節点 (Mini 節点) の評価値が評価関数によって得られているとする (図 2.2-1)。その評価値を元にその親節点 (Max 節点) は、自分にとっていい局面、つまり評価値が高い手を選択しようとする為、子節点 (Mini 節点) の中から評価値が最も高い手を選ぶ (図 2.2-2)。同様に次の親節点 (Mini 節点) は、自分にとって悪い局面、つまり評価値が低い手を選択しようとする為、子節点 (Max 節点) の中から評価値が最も低い手を選ぶ (図 2.2-3)。最後にルート節点である Max 節点 (木を探索したプレイヤー) は、自分にとっていい局面を選択する為、子節点 (Mini 節点) の中から評価値が最も高い手を選ぶ (図 2.2-4)。以上より、この場合では最善手で評価値 55 の局面 A に結びつく手

¹チェッカーでは、双方が最善手をプレイと引き分けになることが 2007 年に証明された [21]。現在の技術の限界はチェッカーであると言える

を、木を探索したプレイヤーは着手として選択する。

また Minimax 木探索をする際に、数値の大小関係に注意をすると、探索を省ける部分があることが分かる。Minimax 木探索において、必要最小限の節点だけを探索する改善手法が $\alpha\beta$ 探索である [4]。図 2.2-1 において末端節点の局面に評価値を与えていく際、局面 C の評価値 65 を得た時に局面 D の評価値は要らなくて済む事がわかる。何故なら、局面 D の評価値が 65 より小さい場合は、親節点 (Max 節点) に選ばれず、65 より大きい場合は、親節点に選ばれたとしても、その次の親節点 (Mini 節点) に兄弟節点である評価値 55 の節点と比較され、必ず選択されない為である。

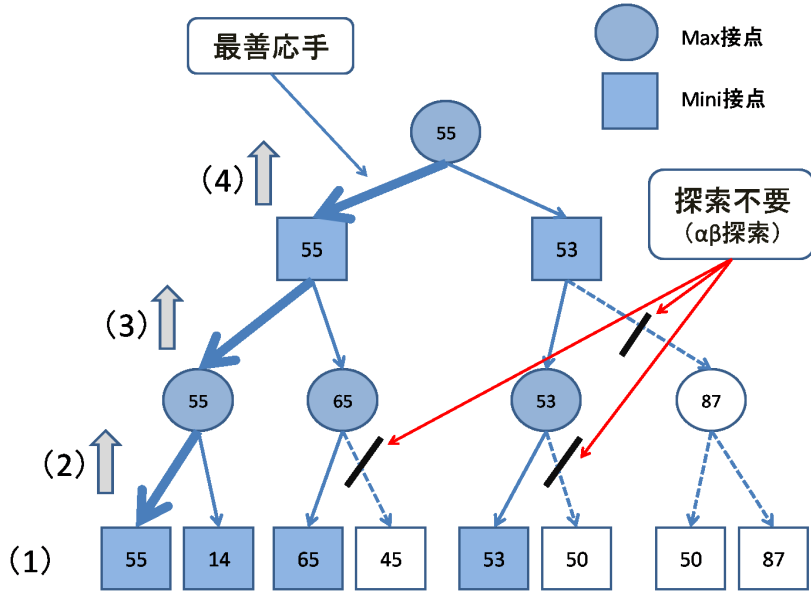


図 2.2: MiniMax 木探索

将棋では現在、この $\alpha\beta$ 探索に基づく木探索手法が主流となっている。こうした $\alpha\beta$ 探索に基づく将棋プログラムの中には、現在プロ 4 段のレベルに達しているプログラムも存在する [10]。

評価関数ベースの minimax 木探索を用いた代表的な囲碁プログラムに **GunGo**[20] があげられる。GnuGo は、オープンソースの囲碁プログラムである。GnuGo は多数の開発者と長年の労力によって、評価関数を 5 万行にも及ぶパターンデータベースで実現し、局面の評価を行った。しかし、精度の高い評価関数の実現は困難であり、棋力としてはアマチュアの 2,3 級程度であった。

何故ならチェスや将棋と異なり、囲碁では各石の価値は平等である。また互いの石が占拠した領域の広さを競うゲームである為、領域の広さを評価要素とすればよいのだが、実際に互いの領域が確定してくるのはゲームの最終段階における為、それ以前に領域の広さを求める事は難しい。

序盤には定石²が存在する。しかし局所的な最善応手のパターンであるため、全局的な状況から注意深く選択をしないと定石自体が悪手になるケースも多々ある。

中盤の評価はさらに難しいと言われている。互いの領域の広さも明確でなければ、定石のような決まった進行も存在しない。人間のプレイヤーでも「厚い/薄い」「形が良い/悪い」

²お互いが最善と考えられる手を行った場合の一連の手のこと。囲碁では序盤でよく見られる。

「石が軽い/重い」などの囲碁用語を用いて局面を説明し、優劣を総合的に評価する。しかしこれらの用語はアマチュア有段者以上でなければ意味を理解する事自体難しいとされている。

以上からまとめると、囲碁は将棋等の他のゲームに比べて、

1. 探索空間が大きすぎる。
2. 盤面を評価要素で表す事が極端に難しい為、精度の高い評価関数の作成が非常に困難である。

という理由から、強いプログラムの作成が難しいとされていた。

2.1.2 モンテカルロ囲碁

前項で言及したように、囲碁では従来の minimax 木探索では、精度の高い評価関数の作成が困難であるために強い囲碁プログラムの実現が困難であった。そこで登場したのが、モンテカルロ法を用いたモンテカルロ囲碁である。モンテカルロ囲碁では、局面の評価に評価関数を必要としない。この手法の基本的なアイデアは囲碁は途中局面の評価は難しいが、終局した局面の評価なら簡単であるという点にある。評価関数による途中局面の評価は難しいが、終局した局面であれば、後は領域を数えあげるだけで評価は分かる。つまり、評価したい局面 A から終局面 B を想定し、その B の評価を A の評価とする方法である。この方法ならば評価関数は作成しなくてもよい事になる。

モンテカルロ囲碁では、モンテカルロ法に基づいたゲームのシミュレーション (プレイアウトと呼ぶ) によってある局面の評価を行う。プレイアウトとは、ある次手の合法手に対し、ゲームの終局までランダムに合法手を選び進むことである。ある合法手 i を選択してプレイアウトを行った回数を s_i 、プレイアウトによって得られた報酬の和を X_i とする。報酬としては、プレイアウトの結果が勝ちであれば 1 を、負けであれば 0 を与える。この時勝率 \bar{X}_i の式は

$$\bar{X}_i = \frac{X_i}{s_i} \quad (2.1)$$

によって与えられる。モンテカルロ囲碁ではこの勝率 \bar{X}_i を合法手 i の評価値として利用する。

例えば図 2.3 を用いてモンテカルロ囲碁について具体的に説明する。

1. 現局面の合法手の中から、ある手を選択する
2. その手からランダムに終局まで合法手を選び続ける (プレイアウト)
3. 終局面から勝敗を判断し、その結果を報酬としてを 1 で選んだ手の報酬に与える。
4. 1 から 3 を、全合法手に対して何度も行い、一番勝率が高い手を次手とする。

この手法では、評価関数を用いていない為、新しいゲームや複雑な評価が必要なゲームにおいて作成者がそのゲームに精通していなくても、ルール等を知ってさえいれば、ある程度のプログラムが作成出来る為有効とされている。

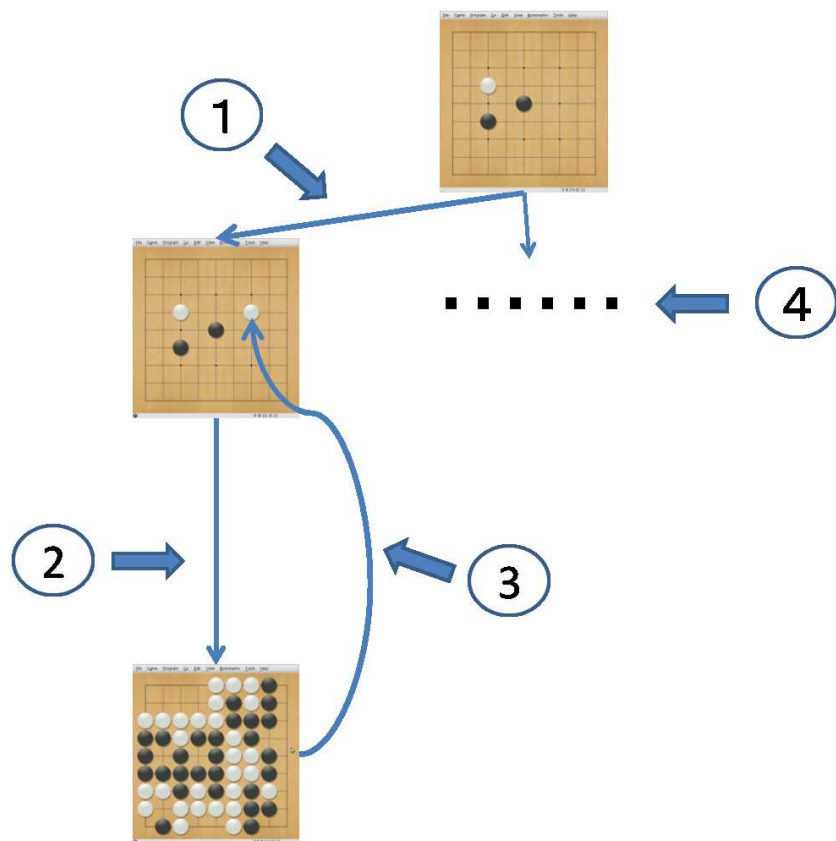


図 2.3: プレイアウト

2.1.3 モンテカルロ木探索

モンテカルロ囲碁では、ルートの各子節点局面の評価をプレイアウトで得る事で、着手を選出していた。しかしこの手法では例えば、長い一連の正解応手を発見することが難しい。プレイアウトが乱数ベースで進められる為、プレイアウト中に相手の明らかにミスな手を選んでしまっている可能性がある。こうした場合、このプレイアウトを行った局面の評価は上がってしまうが、実際には相手のありえないミスの上に成り立っている評価という事になる。よって長い一連の正解応手を発見したい局面であると(例えば囲碁ではシチョウ、死活などがそれに当たる)、プレイアウト中に正しい手がお互いに選択される事が難しい為、発見できない事が多々ある。

モンテカルロ囲碁において単純なプレイアウトを用いてプレイアウトの回数を増加させていくことで、その程度で棋力が頭打ちになるのかを調べた論文も発表されている [23]。

こうしたモンテカルロ囲碁に対し、木探索の要素を付け加えることで、前述の問題を改善した手法がモンテカルロ木探索である。モンテカルロ木探索とは、まずはモンテカルロ囲碁と同様にルートの各子節点局面についてプレイアウトを行い、局面の評価を得る。この時に、より良い評価が返ってきた局面をより有望な局面とみなし、よりプレイアウト数を多くその局面で行うようにする。

次に、各局面においてプレイアウトが行われた回数を記録しておき、その回数がある値を越えた場合には、その局面(局面 A とする)の合法手から一手を選択することで(この時の局面を局面 B とする)、局面 A 節点の下に局面 B 節点を生成する。そしてその局面 B についてプレイアウトを行う。つまり、プレイアウトを繰り返していくごとに探索木の中で有望とされた箇所が成長する事になる。制限時間の間、こうした木探索を繰り返して木を成長させ、より有望な局面(探索が最も行われたルートの子節点)となる手を選出する。これがモンテカルロ木探索の概要になる。

2.1.4 モンテカルロ木探索の理論的背景

本節では、モンテカルロ木探索の代表的なアルゴリズムである **UCT** について述べる。

多腕バンディット問題

多腕バンディット問題とは、多くの腕を持つスロットマシンをプレイするギャンブラーに基づく機械学習の問題である。スロットマシンをプレイすると、各スロットマシンはそれぞれ異なる確率分布に基づいて報酬を返す。ギャンブラーは繰り返し各スロットマシンをプレイし、その報酬の合計を最大化することを目的とする。ギャンブラーは最初から各スロットマシンに関する知識を一切もたない。多腕バンディット問題は、このような状況で毎回のプレイによって得た知識によって、現段階で最善のマシンを選び報酬を最大化することと、他のマシンをプレイしてそのスロットマシンに関する知識を増やす事とのバランスを取る問題であり、強化学習では収穫と探検のジレンマとして知られている。

UCB1 アルゴリズム

多腕バンディット問題に対する最善の戦略は知られているが [16]、計算量、消費メモリがともに膨大に大きくなるため、現実の問題には適さない場合が多い。そこで Auer らに

よって発表された UCB1 アルゴリズム [19] という戦略は、計算量を小さく保ちつつ、高い報酬が得ることが出来る手法である。この UCB1 を用いる事で、常に最善のマシンを選択した場合の報酬と実際に得た報酬の差 (の期待値) がある値以下に抑えられる事が証明されている。この戦略は、各スロットマシンについて、**UCB1 値**を計算し、その式が最大になるマシンにコインを投入するというものである。

$$UCB(i) = \bar{X}_i + c\sqrt{\frac{2\log(n)}{n_i}} \quad (2.2)$$

式 2.2 に示したものが UCB1 値である。 \bar{X}_j は j 番目のマシンの報酬のその時点での期待値、 n_j は j 番目のマシンにそれまでに投入したコインの数。 n はそれまでに全てのマシンに投入したコイン数の合計を示す。この式は、(期待値)+(バイアス) という構成になっている。バイアスの部分は、そのマシンに投入されたコインの数が少ないほど、分子の全コイン投入数に比べ分母の値が小さくなるため、大きな値となる。 c はあるマシンの探索されやすさに、そのマシンの探索回数による影響の大きさを与える定数である。 c が大きいほど、バイアスの係数が増える為、投入されたコイン数が少ないマシンに対してより投入しようとする。 $c = 2$ が用いられる場合が多い。

UCB の考え方は、

- 期待値の高い所に多くのコインを投入する
- コインが少ない場合は、単に運悪く実際より期待値が低い可能性があるので、その分を考慮して優遇する

ということである。

UCT(UCB applied to Trees)

L.Koscis らが提案した UCB1 値を用いた木探索アルゴリズムが **UCT(UCB applied to Trees)** である [15]。本節では囲碁における UCT アルゴリズムについて説明する。

囲碁における UCT アルゴリズムは図 2.4 のように大きく分けて木探索部とプレイアウト部から構成される。UCT ではプレイアウトを行う合法手の選択を、UCB1 値で行い有望な局面に対してより多くのプレイアウト回数を行うアルゴリズムである。ルート節点の局面から順に UCB1 値の最も高い子節点を辿っていき、末端の節点まで到達した時に、その局面でプレイアウトを行い、その局面の勝率を評価値として得る。そしてその評価値を辿ってきた各節点に対し伝えて、各節点での UCB1 値を更新する。以上を制限時間内で繰り返し、最終的にルート節点の子節点で最も UCB1 値が高い局面につながる手を選出する。

具体的に図 2.5 で説明する。

1. UCB1 値による局面選択

ルート節点 (最初の局面) から子節点へ木を UCB1 値に基づき辿っていく (UCB1 値が最も高いものを選択する)。木の末端節点に辿りつくまで辿り続ける。

2. 局面の展開

辿りついた末端節点について、もしある値以上のプレイアウト数が行われていた場合、末端節点の局面から合法手を一つ選出する。そして末端の子節点として節点を展開 (作成) する。

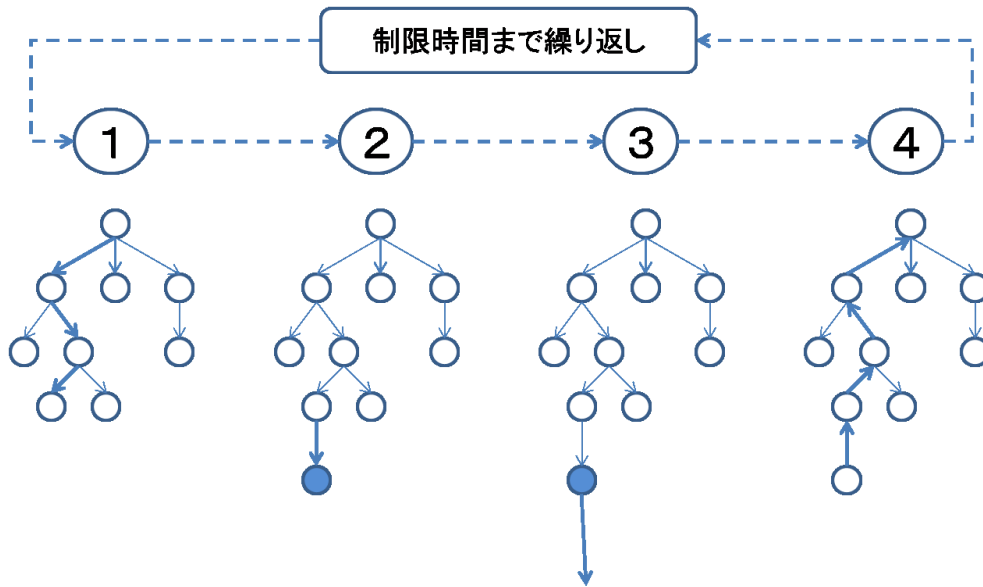


図 2.5: モンテカルロ木探索

2.2 モンテカルロ木探索の改善

本節では UCT モンテカルロ木探索の改善手法のうち、代表的なものについてのみ説明する。

2.2.1 囲碁の知識を用いたプレイアウトの改善

MoGo や CrazyStone[3, 9] などの囲碁プログラムでは、囲碁における UCT モンテカルロ木探索において、プレイアウトの際に囲碁特有の知識パターンを用いて精度を向上させた。プレイアウトにおいて完全なランダムで合法手を選び進めると、意味のない手順で進んでしまうケースが発生してしまう。そうしたケースを省き、より囲碁らしく手をプレイアウト時に進める為に、プレイアウト時に知識・パターンを用いて合法手を生成する。

囲碁プログラム MoGo では、違法手の確認・ 3×3 のパターンを用いている [9]。この 3×3 パターンは、囲碁の知識から「ハネ・キリ」などの囲碁では「手筋」と呼ばれる形を局所的な応手のパターンとして抽出しプレイアウト時に参考に使っている。この手法をプレイアウト時に用いたプログラムと、純粋なランダムプレイアウトのプログラムとを比較した所、大幅な勝率上昇が確認された。

2.2.2 RAVE

囲碁における、UCT-RAVE(Rapid Action Value Estimate)[8]について説明する。RAVE とは、ある着手 i を選択した後の局面の評価値を求める際に、その着手 i からプレイアウトして得た結果を利用するだけでなく、以前にある別のプレイアウトにおいて、もしその着手 i が現れていたのなら、そのプレイアウトの結果も着手 i を選択した後の局面の評価値に利用する手法である。しかし通常は、その着手を選択した後の局面の評価値は、その

着手が選ばれる状態に依存する。よって、この手法はプレイアウト回数が多い時の指標としては用いる事が出来る。これより、RAVEによって得られた評価値 \bar{X}_{RAVE} と通常のUCTによって得られた評価値 \bar{X}_{uct} との加重平均をUCT-RAVEにおける評価値 \bar{X}_{value} とする。

$$\bar{X}_{value} = \beta \bar{X}_{RAVE} + (1 - \beta) \bar{X}_{uct} \quad (2.3)$$

$$\beta = \sqrt{\frac{k}{3s + k}} \quad (2.4)$$

ここで、 s はその着手が実際に選択された回数であり、 k は重みが等しくなるために必要な探索回数である。この重みはその着手の探索回数が多くなるほど少なくなり、探索回数が多くなれば実際の評価値を重視する。

この手法はゲームの知識は用いていないが、そのゲームの性質に大きく依存する。囲碁のように、「手順が多少前後してもその手の重要度が変わらない」というような場合においてのみ有効である。

2.3 モンテカルロ木探索の並列化

本節ではモンテカルロ木探索の並列化についての関連研究を説明する。モンテカルロ木探索をさらに改良する方法として、より大きな探索木の構築と、プレイアウト数の増加が挙げられる。探索木を改善すれば、より有望な局面でプレイアウトを行える。また、プレイアウト数の増加によって、局面の優劣の評価がより正確になる。

モンテカルロ木探索の並列化を行うことで、単位時間あたりのプレイアウト数と探索木の大きさを増やせるので、囲碁プログラムの性能向上につながる。さらに、近年は、シングルコア当たりのCPU速度の向上率は以前よりも低い為、ハードウェアの改善による恩恵を受けるためには、並列化は必要不可欠である。以下、並列化のオーバーヘッドと、様々な並列化の関連研究について述べる。

2.3.1 モンテカルロ木探索のオーバーヘッド

一般に、木探索の並列化には、逐次探索では起こらない、次の3つのオーバーヘッドが生じる。

- 1. 探索オーバーヘッド
探索オーバーヘッドは、逐次探索が行っていなかった部分を並列探索が行う無駄な探索である。並列化によって、無駄な探索がかえって減る場合も存在するが、この場合には、通常は逐次探索自体に改良の余地があることを示している。
- 2. 同期オーバーヘッド
同期オーバーヘッドは、あるプロセス(またはスレッド)の計算が終了するのを他のプロセスが待つ際に生じるアイドル時間である。例えば、同期オーバーヘッドには、共有メモリ環境で情報を共有する際に生じるロックによる排他制御がある。

- 3. 通信オーバーヘッド

通信オーバーヘッドは、他のプロセスと情報の送受信を行う際に生じる通信遅延である。

これらの3つのオーバーヘッドは、並列探索の低下の原因になるだけでなく、相互依存の関係にある。つまり、並列探索がよい性能を発揮するためには、これらのオーバーヘッドの組み合わせがなるべく少ない手法の開発が必要である。

モンテカルロ木探索の並列化では、構築する探索木をより大きくすることと、プレイアウト数を増やすことが重要である。しかし、並列化によるオーバーヘッドの問題だけでなく、並列モンテカルロ木探索では、探索木の大きさの増加とプレイアウト数の増加のどちらに、より重点を置くべきかが現状では明確ではない。

2.3.2 Leaf 並列化

Leaf 並列化について説明する (図 2.6)。Leaf 並列化 [1, 2] は、一つのプロセスが探索木の管理を行い、残りのプロセスが葉節点でプレイアウトを行う手法である。代表のプロセス (マスター) が、UCT 木探索を行い木の葉節点でプレイアウトを行う際に、残りのプロセス (スレーブ) から $p1 \sim p4$ に、葉節点の局面情報を伝える。情報もらった $p1 \sim p4$ はその局面からプレイアウトを行い、結果をマスターに伝える。マスターはスレーブらの情報をまとめた評価値で探索木の更新を行う。

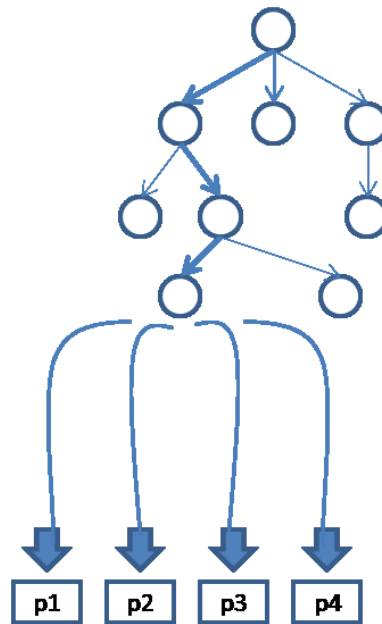


図 2.6: Leaf 並列化のイメージ図

従来の Leaf 並列化には二つ問題がある。一つ目は、各プロセスのプレイアウト結果を毎回集計する際に、一番プレイアウトが遅かったプロセスのプレイアウト時間まで残りのプロセスに無駄な待ち時間が生じてしまう。二つ目は、情報が共有されていない為、本来なら無駄なプレイアウトであるとあるプロセスが分かった際に、他のプロセスのプレイアウト

トは、既に無駄なプレイアウトになってしまっている可能性が出てくる事である。例えば、16 プロセス中 10 プロセスが負けた場合その手は悪手である事が予想されその手のプレイアウトを打ち切りたいが、無駄であろうプレイアウトを残り 6 プロセスは続けてしまう。このため、Leaf 並列化はプレイアウト数は増加出来るが性能が悪い事が報告されている。

加藤らは、Leaf 並列化の同期オーバーヘッドを減らす方法を提案している [13] が、探索木の情報が更新されない間は、同一の葉節点のプレイアウトを何度も行ってしまう可能性がある。

2.3.3 Tree 並列化

Tree 並列化について説明する (図 2.7)。Tree 並列化は、モンテカルロ木探索が構築する探索木を各プロセス (またはスレッド) で共有し、木の展開、プレイアウト、および UCB 値の更新を行う [9]。Tree 並列化は、より大きな探索木を作成できるので、最も自然な並列化法の一つである。CPU コア間でメモリを共有できる環境では Tree 並列化は多くの強い囲碁プログラムで利用されている [6, 7]。

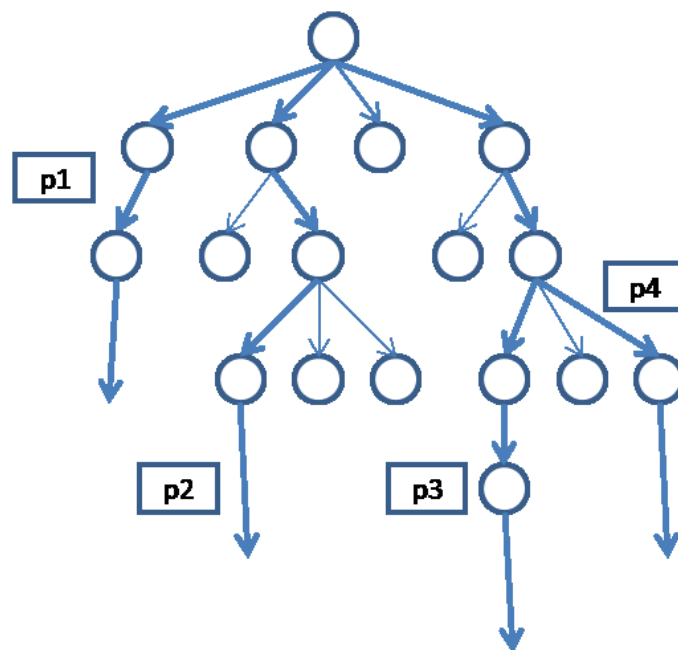


図 2.7: Tree 並列化のイメージ図

Tree 並列化では、プレイアウトは各スレッドが独立に行えるが、木の展開や UCB 値の更新の際に、他のスレッドと共有する木にアクセスする必要がある。木の安全な共有には、ロックなどの排他制御によるスレッド間での同期が必要である。このため、例えば複数のスレッドが同一の節点の情報を更新しようとする場合には、排他制御のために同期オーバーヘッドが生じ、性能低下につながる。木を共有するスレッド数が多くなれば、同期オーバーヘッドの増加はより深刻な問題になると考えられる。

Chaslot らは共有メモリ上において、共有木を大域的にロックする手法と、局所的にロックする手法を比較した [2]。大域的にロックする手法は共有木全体を各プロセス間でロック

し、一度に一つのプロセスしか探索木にアクセスを許さない手法である。一つのプロセスが共有木に探索結果を書き込む間、他のプロセスは別々のプレイアウトを行うようにしてある。

局所的にロックする手法は複数のプロセスが共有木にアクセスすることを可能にした。複数のプロセスが共有木にアクセスを行うと、同じ節点のデータにアクセスする際に衝突が起きる。これを局所的にロックを行い、一つの節点には一つのプロセスしかアクセス出来ないようにした。

その結果、局所的なロックの手法がより高い勝率をあげていた。大域的なロックは同期オーバーヘッドが強すぎる為、4スレッド程度までしか効果が出ないとされている。局所的なロックでは、プレイアウト数は大域的なロックの二倍に上昇しより効果を上げた。

Enzenberger らは、共有メモリ環境での Tree 並列化において、C++の volatile 機能の特徴と IA-32 や Intel-64 CPU のハードウェアの性質を利用し、探索木アクセス時の排他制御処理を取り除いた [6]。この手法では、木の共有の際の排他制御の処理が不要であるので、同期オーバーヘッドを取り除けるが、

UCB 値の更新の際に、更新すべき情報が時折失われることが理論上は生じる。しかし、この情報の喪失は、現実にはほとんど起こらず、囲碁プログラムの強さにはほとんど影響しないと結論付けている。報告された実験結果としては、排他制御ありのプログラムではスケラビリティが9路盤では2,19路盤では3スレッド程度までしか出ないが、排他制御なしのマルチスレッドでは9路19路共に8スレッドまでスケラビリティが確認されている。

ネットワークで繋がれたPCなど、メモリが分散する環境では、同期オーバーヘッドに加え、PC間の通信も必要であるので、木共有のオーバーヘッドはさらに深刻である。Gelly らは、各プロセッサが構築した探索木の重要な部分のみを定期的にブロードキャストし、共有することによって、同期・通信オーバーヘッドを減らしている [7]。

Tree 並列化の主要な最適化手法の一つに、Chaslot らが提案した探索時の仮想的ロス (Virtual Loss) が挙げられる [2]。Virtual Loss とは共有木探索時に、互いに異なるプロセス (スレッド) が、同じ探索をなるべくしない為の工夫である。あるプロセスが木を辿っていく際に、その辿られた節点の評価値を仮想的に下げる事で、他のプロセスが同じ局面を選択降下しにくくする手法である。これにより、探索オーバーヘッドを減らす事に成功している。Chaslot らの実験では、Virtual Loss を使用することで性能向上が見られた。

2.3.4 Root 並列化

Root 並列化について説明する (図 2.8)。

Root 並列化 [1] は一般的に探索木を各プロセス (スレッド) が共有しない手法である。各プロセス (p1~p4) は、異なる乱数シードを用いて、独自に探索木を作成する。Root 並列化が次の一手を選択する際には、一つのプロセスが、各プロセス上にあるルート局面における各着手の情報を集計し、その集計情報に基づいて最も有効な手を選択する。

Root 並列化の集計方法は、訪問回数の多い着手を重要であるとみなす総和制が一般的である。

Cazenave らや Chaslot らが用いた総和制とは [1, 2]、独立に探索を行った各プロセスの探索木情報を一つの代表プロセスが集計し、その平均値によって、手を選出する方法である。Cazaneve らは各プロセスから集めた各合法手の勝率の平均値によって、手を選出す

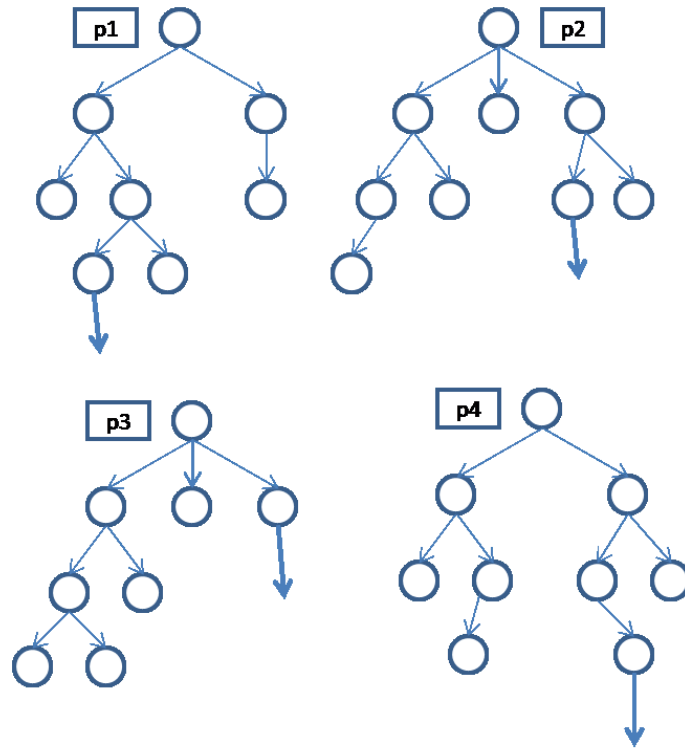


図 2.8: Root 並列化のイメージ図

る。Chaslot らは勝率ではなく、各合法手の探索中における訪問回数の平均値が一番大きい手を選出する³。

現在のUCTモンテカルロ木探索においては、逐次においても訪問回数によって手を選出する手法が主流となっている。

具体的に図 2.9 で説明する。プロセス 1~4 がそれぞれ探索を行ったとする。探索が終わり、木の情報から、各プロセスは A,B,C という 3 つの手をそれぞれ上位に選んでいる。そしてそれぞれの手がどれだけ探索されたか訪問回数になる。訪問回数の和をとり、最も大きい訪問回数を持つ手が次手として選出される。この例の場合、B の手が訪問回数が一番大きい為、選出される。

Root 並列化では、探索木を共有しないので、大きな探索木を構築できるとは限らない。つまり、複数プロセスが同じ探索木を構築してしまうことによる探索オーバーヘッドの増大が問題として考えられる。しかし、乱数を利用して行うプレイアウトの結果が異なる場合が存在するので、各プロセスが互いに異なる性質の木を構築できる。さらに、ルート局面を各プロセスにブロードキャストする探索開始時と着手の情報を集計する探索終了時のみ通信・同期オーバーヘッドが生じるので、探索木を共有する方法に比べて、並列探索によるオーバーヘッドはるかに少ない。このため、Root 並列化では、他の手法よりも単位時間あたりに行えるプレイアウト数が大きい。

また、一つのプロセスでは一局中に何回か悪手が出てしまうものである。しかしそうした悪手を複数のプロセスで情報を集計する事で回避している。

Root 並列化は他の手法よりも実装が比較的に簡単なため汎用性が高い。その為、様々

³訪問回数と同じ手が存在した場合には、勝率によって選択する

Proc1	訪問回数	Proc2	訪問回数	Proc3	訪問回数	Proc4	訪問回数
A	800	C	900	A	600	B	800
B	600	B	500	B	500	C	700
C	100	A	50	C	100	A	50

集計	訪問回数	投票数
A	1500	2
B	2300	1
C	1800	1

図 2.9: Root 並列化における集計方法の具体例

なプログラムで実用可能であり、少なからず性能を向上する事が出来る。

Cazenave らは、ルートの子節点のみを、他プロセス間で同期をとった Root 並列化を提案した [1]。制限時間中に何度か、一つのマスタープロセスが他のルートの子節点情報を集め、その情報の平均をとり、他のプロセスに投げる手法である。この手法で、従来の Root 並列化と比較し 4~8 プロセスまで性能が上がることを示した。しかし、16 プロセスでは性能の低下が見られた。これは同期オーバーヘッドの為と思われる。

また、加藤らは囲碁プログラム Zen を用いて、Zen のクラスタ Root 並列化を行った [14]。これにより、従来の逐次の Zen よりも性能があがる事を示している。

2.3.5 先行研究における各並列化手法の比較

Cazenave らの研究では [1]。Single-Run(Root) 並列化、At-the-leaves(Leaf) 並列化、Multiple-Runs(ルートの子節点のみ共有) 並列化について実験を行った。各手法ごとに、10000 回のプレイアウト制限を設け、1 スレッドから 16 スレッドまでの効果を調査した。結果では、At-the-leaves 並列化が一番性能が悪く、Multiple-Runs 並列化が性能が高かった。しかし、16 スレッドの場合、Multiple-Runs 並列化の性能は落ちており、逆に Single-Run 並列化は 16 スレッドまで性能が上昇している結果となっていた (図 2.2)。

表 2.2: 10000 シミュレーションにおける、Single-Run 並列化と Multiple-Runs 並列化の GnuGo3.6 との対戦勝率比較 (T.Cazenave et al. 2006)

並列化手法	1CPU	2CPUs	4CPUs	8CPUs	16CPUs
Single-Run(Root) 並列化	45.0 %	62.0 %	61.5 %	65.5 %	66.5 %
Multiple-Runs 並列化	49.0 %	58.5 %	72.0 %	72.0 %	68.0 %

Chaslot らの 16CPU コアの共有メモリ環境での実験でも、同様に Leaf 並列化の効果は他に比べて劣るとされている。Root 並列化は、Tree 並列化よりも有効であり、さらにコ

ア数以上の性能を出すこともあった [2].

表 2.3: 13 路盤・一手 1 秒での, Root 並列化, Tree 並列化の GnuGo3.7.10 との対戦勝率比較 (G.Chaslot et al. 2008)

並列化手法	1CPU	2CPUs	4CPUs	16CPUs
Leaf 並列化	26.7 %	26.8 %	32.0 %	36.5 %
Root 並列化	26.7 %	38.0 %	46.8 %	56.5 %
Tree 並列化	26.7 %	33.8 %	40.2 %	49.9 %

表 2.4: 9 路盤 Root 並列化, Tree 並列化の GnuGo3.7.10 との対戦勝率比較 (G.Chaslot et al. 2008)

一手あたりの秒数 (s)	Root 並列化	Tree 並列化
0.25	60.2 %	63.9 %
2.50	78.7 %	79.3 %
10.0	87.2 %	89.2 %

13 路盤での実験では, Root 並列化はスーパーリニアな結果で Tree 並列化を上回っている (図 2.3). 9 路盤での結果では Root 並列化と Tree 並列化はほぼ同等の結果となっている (図 2.4). この理由を, 文献 [2] は, 13 路盤においては, Root 並列化の作る木が Tree 並列化に比べて浅く, 探索が局所解から脱出しやすいためであり, 9 路盤ではもともと探索木が浅い為, 13 路盤のような差が出にくいと推測しているが, 具体的な根拠は示されていない.

以上の結果よりまず, Leaf 並列化はやや他の並列化に劣ると言える. しかし, それぞれの実験では, リソースは 16 スレッドまでしか使用されていない. さらに, 実験に使用しているプログラムも強いプログラムとは言えず, 勝率の計測も, 時間は 1 秒程度, 盤のサイズは 9, 13 路盤でのみの実験である. 実際の囲碁では, 19 路盤が主流であり, 時間も一手あたり 1 秒から 10 秒までは使用されている. 以上より, Root 並列化と Tree 並列化の性能差については, 未だ明確に示された論文は少ないと言える.

第3章 本研究で実装した Root 並列化

本節では、本研究で実装した合議制による Root 並列化について説明する。3.1 節で今回実装に用いた Fuego について、3.2 節で従来の Root 並列化手法の総和制の欠点について、3.3 節で今回我々が Root 並列化手法に提案する合議制について、3.4 節では総和制と合議制の述べる。

3.1 Fuego (対象囲碁プログラム)

本節では本研究で扱った対象囲碁プログラム fuego について説明する。Fuego[5] は、アルバータ大学を中心に開発されているオープンソースの囲碁プログラムである。



図 3.1: fuego

2009 年 5 月に行われた Computer Olympiad では、9 路盤で 1 位、19 路盤で 2 位という成績を残しており、現在最も強いプログラムの一つである [18]¹。Fuego のモジュール関係については図のようになっている。

- GtpEngine
ゲームとは独立したモジュールであり、GTP (Go Text Protocol) の実装がメインである。

¹使用言語 C++

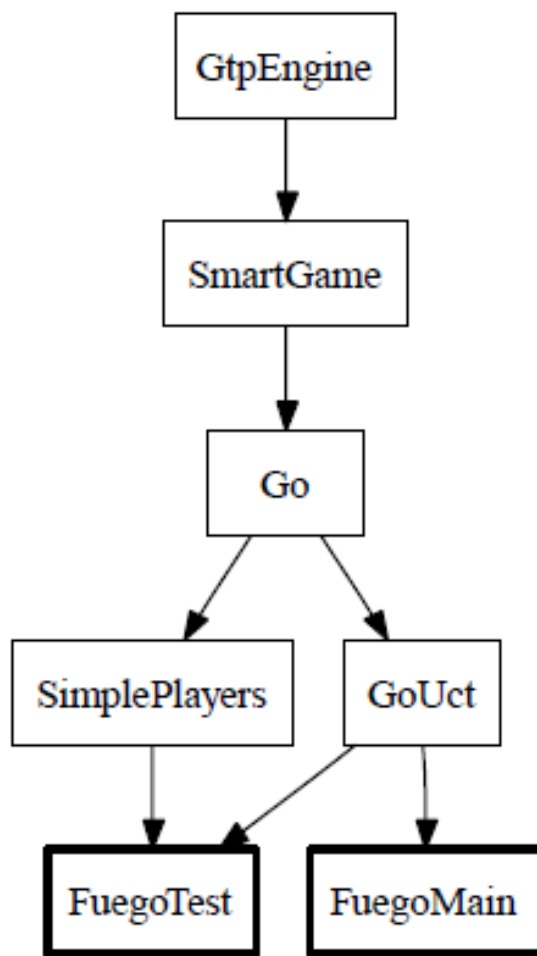


図 3.2: fuego のモジュール図 文献 [5] 参照

- SmartGame
様々なゲームに対応する為のモジュール。ゲームの探索の基本部分の実装がメインである。例えば、Minimax 木探索やモンテカルロ木探索の基本部分等がある。
- Go
ゲームの中でも囲碁に特化した機能を提供する。囲碁盤や定石、ルール等が挙げられる。
- GoUct
囲碁の中でも、UCT 探索を担う機能を提供する。

Fuego では、RAVE を利用した UCT 探索が用いられている。さらに、MoGo と同様に囲碁の知識に基づく様々なパターン [9] を利用して、プレイアウトの性能を改良している。

また、Fuego はスレッドを利用した共有メモリ環境での並列化 [6] と、MPI を利用した分散メモリ環境での並列化に対応している²。共有メモリ環境での Tree 並列化においては、Virtual Loss の最適化も取り入れている。

3.2 総和制の欠点

従来の Root 並列化手法である総和制において、いくつかの欠点がある。まずモンテカルロ木探索は、一位の手の情報が他の順位の手に比べ信頼度が高いという性質を持っている。総和制では二位以下の情報も同等に扱っている為、例えばそこそこに探索される手が総和を取った際に、抽出されてしまう可能性が出てくる。図 2.9 において、B は 3 つのプロセスで 2 位の評価を得ているが、1 位の手の方が信頼度が高く、2 位以下の手は悪手を含んでいるかもしれない。しかし総和ではこの場合、B の手が選出されてしまう。

特に 19 路盤では、各プロセスが一位に選出する手は合法手が多いため分散しがちであるが、どのプロセスもある一定数は探索してしまう手というものが共通して存在する事がある。このような場合に、そうした手が選出されてしまいがちである。

3.3 合議制

本節では今回実装したコンピュータ囲碁における合議制について述べる。まず 3.3.1 でコンピュータ将棋における合議制について、次に 3.3.2 で今回提案するコンピュータ囲碁における合議制について、最後にその実装方法について説明する。

3.3.1 コンピュータ将棋における合議制

コンピュータ将棋で提案された合議制 [25] について説明する。小幡、伊藤らによって提案されたコンピュータ将棋における合議制とは、まず思考アルゴリズムの違う複数の将棋プログラムを用意する (これをシステムとする)。この用意した各将棋プログラムに同一局面を与え、得られた次手について多数決をさせて最も多くの票を集めた手をシステム全体としての次手に選出する手法である。

²分散メモリ環境での並列化法については、ソースが公開されていないため、技術的な詳細は不明である。

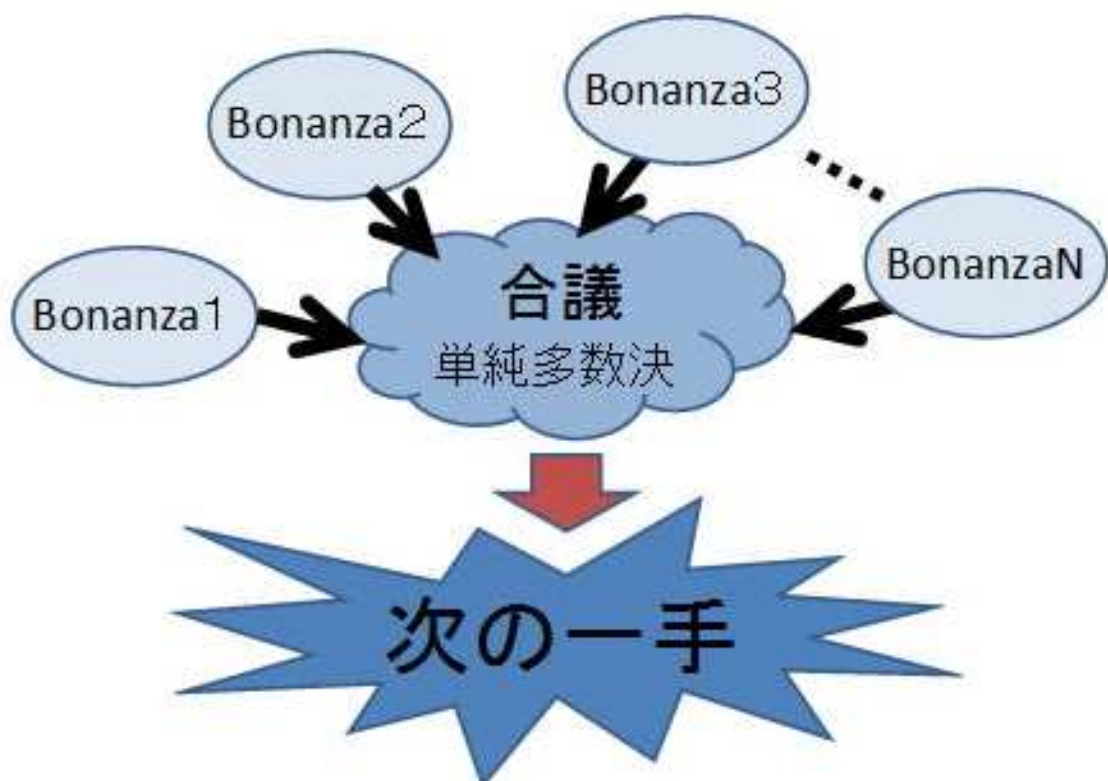


図 3.3: コンピュータ将棋における合議制 (Bonanza を利用)

将棋プログラムは評価関数ベースの木探索であるので、思考アルゴリズムの違う複数のプログラムを用意するために

1. 評価関数に乱数を与える事で複数のプログラムを用意する
2. 違う将棋プログラムを複数用意する

という手法をとり、合議制の評価を行った。杉山, 小幡, 伊藤らは、将棋プログラム Bonanza[?] を用いて 1 を実現し、評価を行った [26]。また、2 を実現するために、Bonanza 以外の二つの同等に強い将棋プログラムを用いて、合議システムを構成し評価を行った [24]。

実験の結果、自己対戦では、合議を行ったシステムの方が優位に勝ち越す事が知られている。また、他のプログラムとの対戦勝率も、従来のプログラムより合議を行う方がより勝ち越した結果が得られている。

しかし、なぜ合議によって強さが向上するかの論理的な説明は現在も難しいとされている。

3.3.2 コンピュータ囲碁における合議制の提案

今回我々がモンテカルロ木探索の Root 並列化手法に提案した、コンピュータ囲碁における合議制について説明する (図 3.4)。

コンピュータ囲碁における合議制は、コンピュータ将棋で有効とされた合議制に基づく。コンピュータ将棋では評価関数に乱数を加え、違う思考アルゴリズムのプログラムを用意して行った合議制であるが、モンテカルロ木探索の囲碁プログラムの場合、そうした必要がない。

この手法では、従来の Root 並列化と同様に、各プロセスは、互いに異なる乱数シードで、独立にモンテカルロ木探索を行う。次の一手の決定時には、各プロセスは自身が最善と判断する着手を 1 つだけ候補手として選択する。最も多くのプロセスに選ばれた手を次手として選出する (単純多数決)。

具体的に図 2.9 で説明する。図 2.9 の場合、総和制では B の着手を選出した。しかし合議制では投票数に注目する。この場合、A の手が最も投票を集めた為、A の手が次の着手となる。

また、我々の実装でも、各プロセスが選択する候補手は、そのプロセス内の探索木で、訪問回数が最大の手とした。さらに、多数決により選択できる着手が 2 つ以上ある場合には、全プロセスの訪問回数の合計が最大である着手を選択した。

3.3.3 実装方法

今回の Root 並列化の実装について説明する (図 3.5)。Fuego0.3.2 のソースに変更を加え、各プロセスが MPI で通信を行うようにした。通信の流れは以下ようになる。

1. マスタープロセス (p1) に送られた相手 (pE) の手、もしくは初期局面を、MPI 通信で各スレーブプロセスに送る。
2. 各プロセスはその局面から探索を開始し、それぞれ制限時間で木を作成する。そしてその木情報 (ルートの合法手の情報) を p1 に送る。
3. p1 は情報を集計し、総和制もしくは合議制の手法で次手を選出する。その手を各プロセスに送信し、各プロセスも盤面を更新する。
4. p1 は次手を pE に GoGui-twogtp コマンドに

よって引き渡す。5. pE が思考し出した答えを p1 が GoGui-twogtp コマンドによってもらい、1. へ戻る。

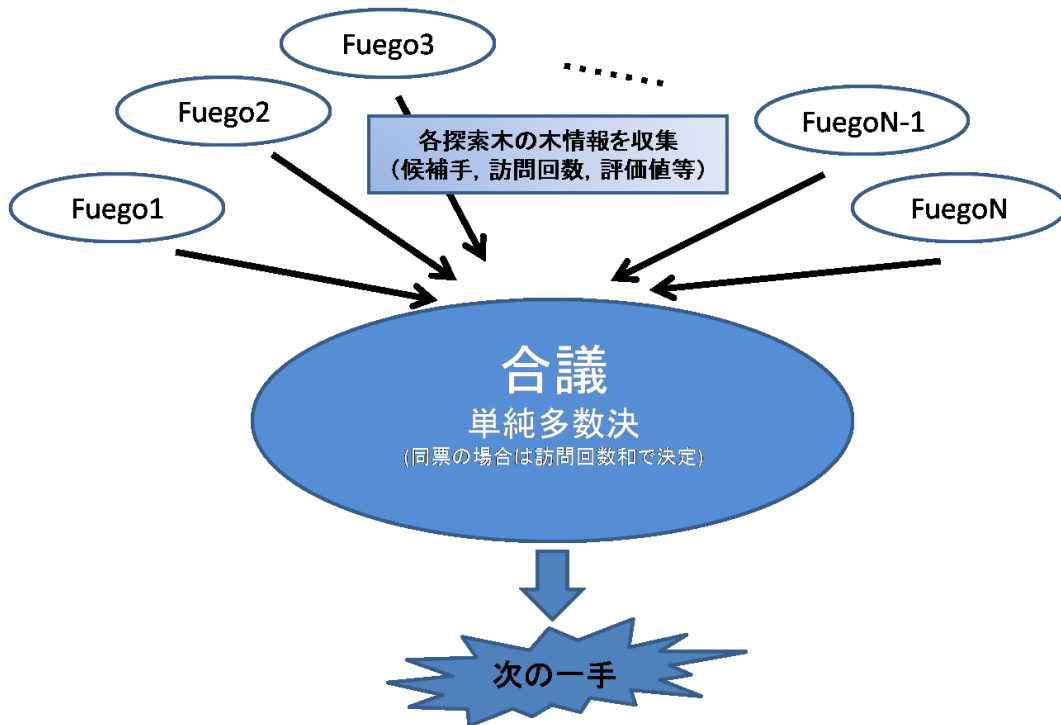


図 3.4: 合議制のイメージ図

具体的な集計方法を図 2.9 で説明する。総和制の例では、この場合 B の手が選出された。しかし合議制の場合では 4 プロセス中、一位として A が 2 票、B が 1 票、C が一票の票を集めた為、A の手が次手となる。

また、今回の Root 並列化では、定石は使用しなかった。これは、使用によって全てのプロセスの着手が同じになってしまうためである。さらに、先読みについても今回はオプションによって外した。

3.4 総和制と合議制の差

総和制による Root 並列化は、平均化した着手を選択することである。図 2.9 の B のような、どのプロセスでも大抵探索されてしまうような平均的な手が一位とされる場合が多い。プログラムのアルゴリズムによるが、どのプロセスでも大抵探索される手が、好手の場合ならよいが、パターンマッチング等の影響で必ず探索されてしまう悪手の場合もある。

合議制では、総和制よりも、各プロセスが最善手と判断する手を、次善手以降であると判断した手よりもはるかに高く評価する。何故なら、モンテカルロ木探索では一位の手がより他の手より信頼度が高い性質を持っている為である。つまり平均的に探索された手であろう 2 位以下の手を重視しない。

その一方で、意見が複数に分かれた場合には、少数にとっては最善手であるが、他のプロセスの多くが二位以下の悪手とみなす着手を選択してしまうことがある。

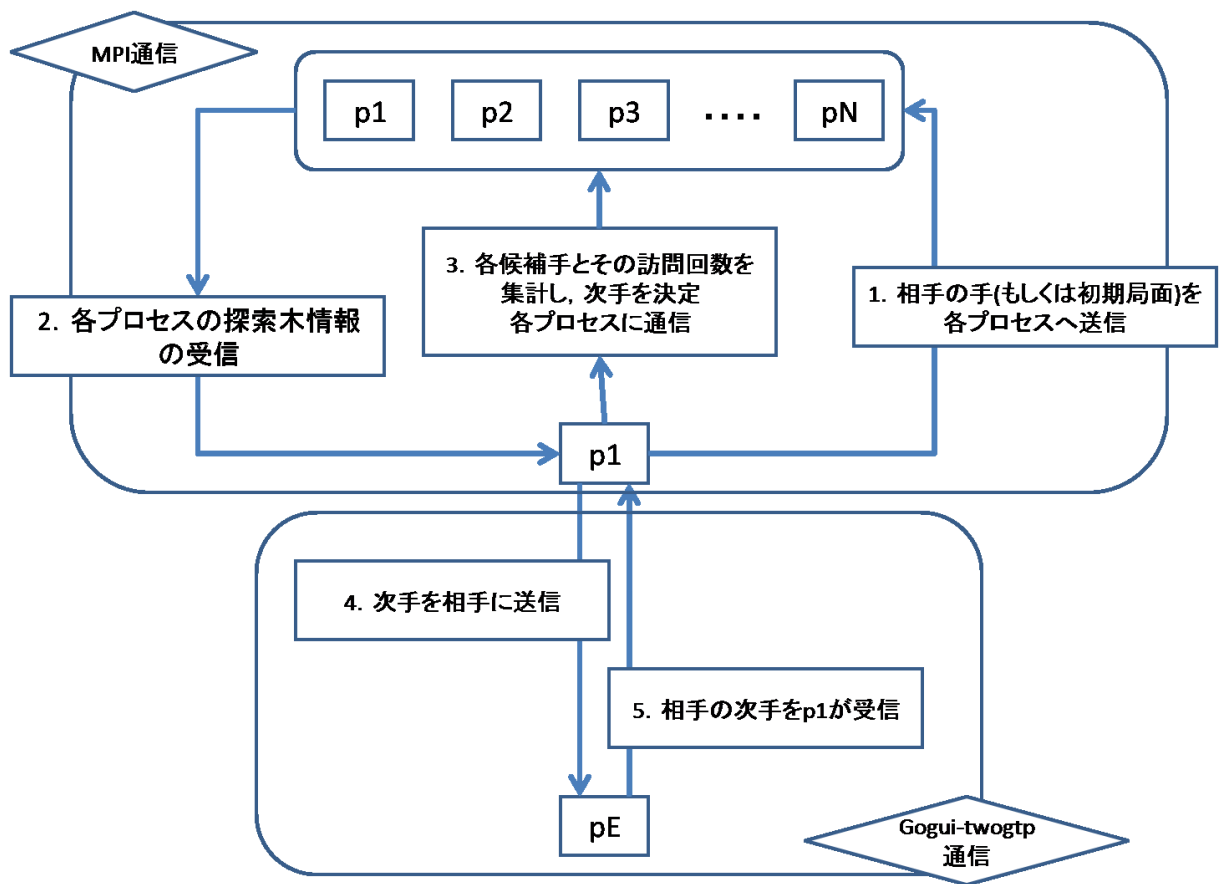


図 3.5: 合議システム

第4章 実験と評価

本章では、まず今回行った実験の環境・条件について4.1, 4.2節で述べ、4.3節で実験の構成を説明し、以降で行った実験の結果・評価を述べる。

4.1 実験環境

本研究の実験は以下の環境で行った。

- 使用プログラム Fuego version 0.3.2
- PC クラスタ ネットワークで接続された8ノード×8コア
- メモリ 各ノードごとに共有メモリ 16GB
- CPU 各CPUはXeon E5410 2.33GHz × 2
- MPI MPICH2¹
- 使用言語 C++
- 対戦用プログラム Mogo release3

4.2 実験条件

全実験中の対戦実験は、9, 19路盤(コミ六目半, 中国ルール)での1手10秒で、9路では200局、19路では100局という条件で行った。Fuegoには序盤用の定石ファイルが存在しているが、実験では定石ファイルを利用しなかった。また、各対戦では異なる乱数の初期値を設定した。このようにして、全局の対戦棋譜の中に似た対局が現れないようにした。

Root 並列化を用いる実験では、前章で述べたようにMPI通信で実装したFuegoを用いる。通常はPCクラスタの限界である64プロセスまで使用したRoot 並列化であるが、4.9節では、思考を各64プロセスが8回繰り返す事で、仮想的に512プロセスのRoot 並列化を実現した。

また、Root 並列化の比較対象となるTree 並列化としては、LockFree マルチスレッドを用いたFuegoを使用した。このマルチスレッドは共有メモリで有効で、16スレッドまでは台数効果があるとされている[6]。今回は使用するPCクラスタの制限により、8スレッド共有のTree 並列化までを実現し、比較実験を行った。

4.8節、4.9節における実験では、見本となるプログラムを用意し、同一の棋譜を読ませて着手を比較した。着手の性質の際に用いた F_{tree} とは、Tree 並列化8スレッド共有

¹<http://www.mcs.anl.gov/research/projects/mpich2/>

Fuego に一手 30 秒思考させたプログラムである。また一致率の際に用いた F80s とは、逐次 Fuego に一手 80 秒思考させたプログラムである。

4.3 実験の概要

本章の実験の構成は以下のようになる。

- 対戦実験による Root 並列化の有効性
- Root 並列化と Tree 並列化との比較
- Root 並列化と Tree 並列化の融合
- 総和制と合議制の比較
- 総和制と合議制の着手の性質
- 一致率による Root 並列化の評価

4.4 対戦実験による Root 並列化の有効性

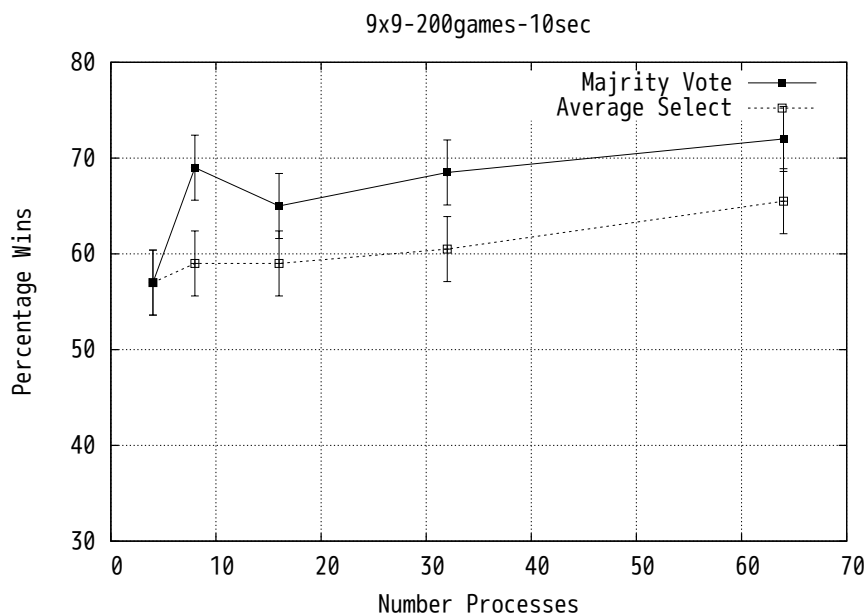


図 4.1: 9 路盤:Root 並列化 (総和制・合議制 4~64 コア) vs 逐次 Fuego

図 4.1 から図 4.5 に 9, 19 路盤における様々な CPU コア数による両 Root 並列化法 (総和制と合議制) の対戦勝率を示す。まず, 図 4.1 と図 4.4 が逐次 Fuego と両手法の対戦勝率グラフである。逐次 Fuego, 両 Root 並列化手法の Fuego のパラメータは最適で同じものとした。

総和制と合議制の両手法とも, コア数の増加に伴い勝率が向上している。さらに合議制では, 従来の Root 並列化法である総和制よりもこの実験では高い勝率が得られた。特に 19 路盤では優位に総和制よりも合議制は勝ち越している。

囲碁プログラムには, 各プログラムの実装方法により, 選択する着手に個性があるのが通常である。このため, 自己対戦による実験だけでなく, 他のプログラムとの対戦を行うことは重要である。

そこで, Root 並列化の対戦相手として逐次版の MoGo を利用し, 合議制と総和制の勝率を調べた結果を図 4.3,4.5 に示す。この実験でも, 総和制と合議制の両方で, CPU コア数を増加させれば, 勝率が上昇することが確認できた。しかし, Fuego との自己対戦で得られた合議制の優位性は, 9 路盤においては, MoGo との対戦結果では, 特に見受けられなかった。さらに, 総和制では, CPU コア数を 32 から 64 に増加させても勝率はほとんど変化しなかった。

19 路盤においては, 合議制の優位性が確認できた。しかしやはり CPU コア数を 32 から 64 に増加させても, 勝率の変化は少なかった。このため, この実験結果からは 64 コア以上利用する Root 並列化の勝率は, あまり上昇しない可能性もあると言える。

Chaslot らの実験では, RAVE が実装されておらず, これが Root 並列化のコア数の増加による勝率上昇率に影響を与えている可能性がある。これが Chaslot らの Root 並列化 (総和制) の勝率の上昇率が高い可能性の一つであると考えられる。

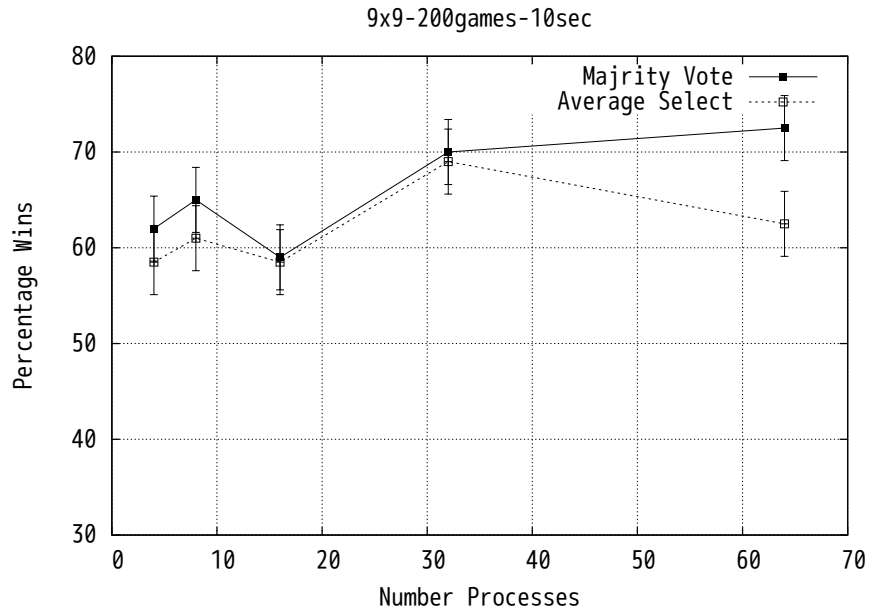


図 4.2: 9路盤:Root 並列化 (総和制・合議制 4~64 コア RAVE なし) vs 逐次 Fuego(RAVE なし)

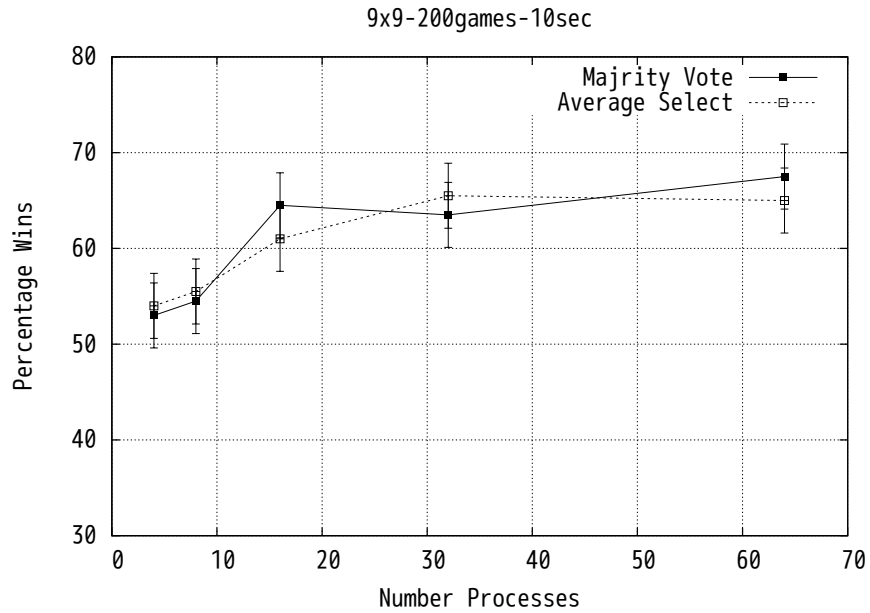


図 4.3: 9路盤:Root 並列化 (総和制・合議制 4~64 コア) vs 逐次 Mogo

図 4.2 に、逐次および Root 並列化された Fuego から RAVE オプションを外したときの 9 路盤における自己対戦の勝率を示す。この表より、RAVE なしの場合にも自己対戦でも、合議制の方が総和制よりも勝率が高い傾向があることが分かる。しかし、Chaslot らの実験のような Root 並列化の効果は見られなかった。これは RAVE なしの状態も、Fuego がある程度の強さを持っているプログラムであるからと推測される。

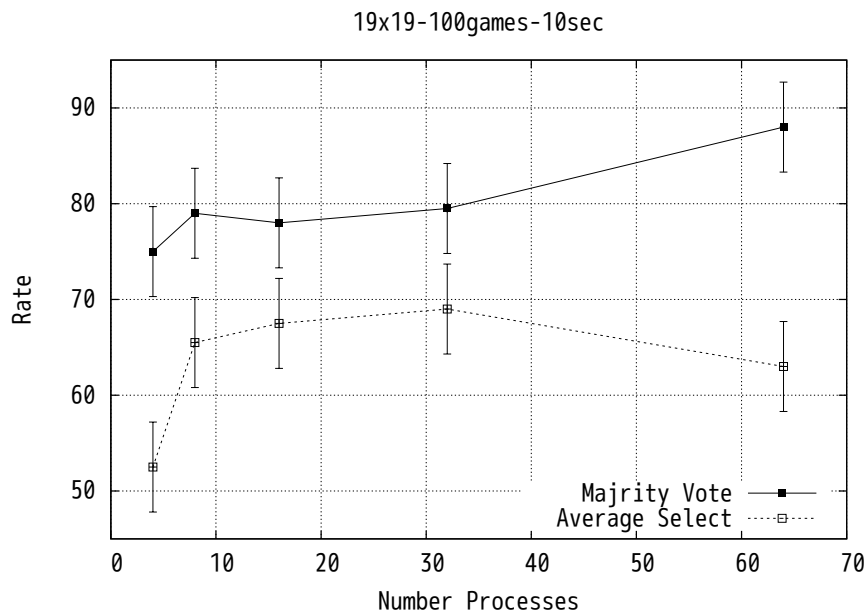


図 4.4: 19 路盤:Root 並列化 (総和制・合議制 4~64 コア) vs 逐次 Fuego

4.5 Root 並列化と Tree 並列化との比較

Root 並列化の探索木非共有によるデメリットを調べるために、Fuego に元から実装されているロックなし Tree 並列化と Root 並列化の中で最も効果があるとわかった合議制 Root 並列化 (64 プロセス使用) との対戦を行い、性能比較を行った。Tree 並列化では、Virtual Loss の利用など、最適な実装を利用した。

図 4.6, 4.7 は 9, 19 路盤における、Tree 並列化での利用コア数を 1 から 8 に変化させた際の合議制 64 コア Root 並列化の勝率である。Chaslot の実験とは異なり、64 コア Root 並列化の性能は、9 路では 6 コアの Tree 並列化に負け越しており、4 コアと同等となった。更に 19 路では、4 コアにも優位に負け越している。これより、リソースの使い方としては Tree 並列化の方がより効率がよいと言える。Enzenberger らの Fuego での実験では、ロックなし Tree 並列化の効果は、7 コアまでであり、8 コアでは、7 コアよりも弱くなっていた [6]。同様に、我々の結果も 9 路盤においては、8 コアに対する合議制の勝率は上昇している²。

²Martin Müller 氏によれば、16CPU コアから構成される共有メモリマシンでの Fuego の自己対戦では、引き続き勝率の向上が見受けられている。このため、8 コアにおける合議制の勝率上昇の理由は、単に自己対戦数が少ないための可能性もある。

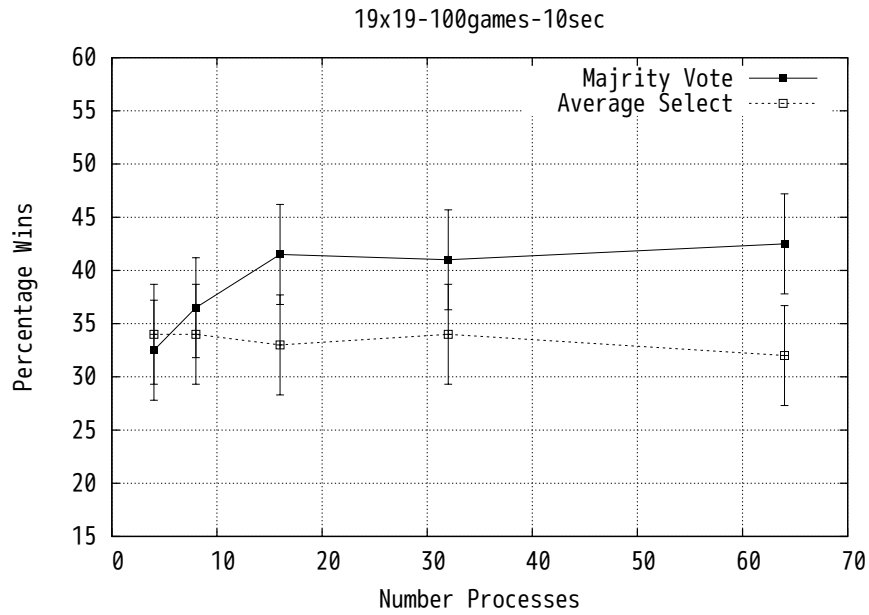


図 4.5: 19 路盤:Root 並列化 (総和制・合議制 4~64 コア) vs 逐次 Mogo

ロックなし Tree 並列化では, UCB1 値 (2 章参照) 更新時に更新すべき情報が消えることがある. 文献 [6] では, この情報喪失の頻度は少ないとあるが, 多数コアを利用する際には, 情報喪失の頻度が上昇する可能性も考えられる. コア数の増加に伴う情報喪失の頻度と性能への影響の調査は, 今後の課題の一つである.

4.6 Root 並列化と Tree 並列化の融合

Root 並列化と Tree 並列化の欠点を補う自然な方法として, ノード内部のコアではロックなし Tree 並列化を行い, ノード間では Root 並列化を行う方法が考えられる. 8 コアの Tree 並列化を行う Fuego を F_{tree} として, F_{tree} を 8 つ利用した Root 並列化 (以後, **8 × 8 Root 並列化** と呼ぶ) の効果を調べた. 表 4.6, 4.6 は, 19,9 路盤における 8 × 8 Root 並列化の合議制と総和制について対戦相手をそれぞれ F_{tree} と MoGo にしたときの勝率である.

F_{tree} に対し, 9 路盤において, 図 4.6 では逐次版 Fuego を 64 個利用する合議制の勝率は 49.5% であったが, 64 という同じ CPU コア数を使用する合議制の 8 × 8 Root 並列化では 67.0% (表 4.6 を参照) まで上昇した. 同様に, 図 4.3 では, 64 コアでの合議制の MoGo への勝率が 67.5% あったのに対し, 8 × 8 Root 並列化では 77% であった.

また, 19 路盤においても同様な勝率向上の結果が見られる. これらの事実より, 8 × 8 Root 並列化は Fuego の強さ向上に貢献していると言える.

表 4.6 において, 9 路盤における 8 × 8 Root 並列化では, 総和制の勝率の方が合議制よりも約 2% 高く, 合議制との差はほとんど見られなかった. 19 路盤の表 4.6 においては逆に, 合議制の方が 3% ほど高い勝率をあげていた.

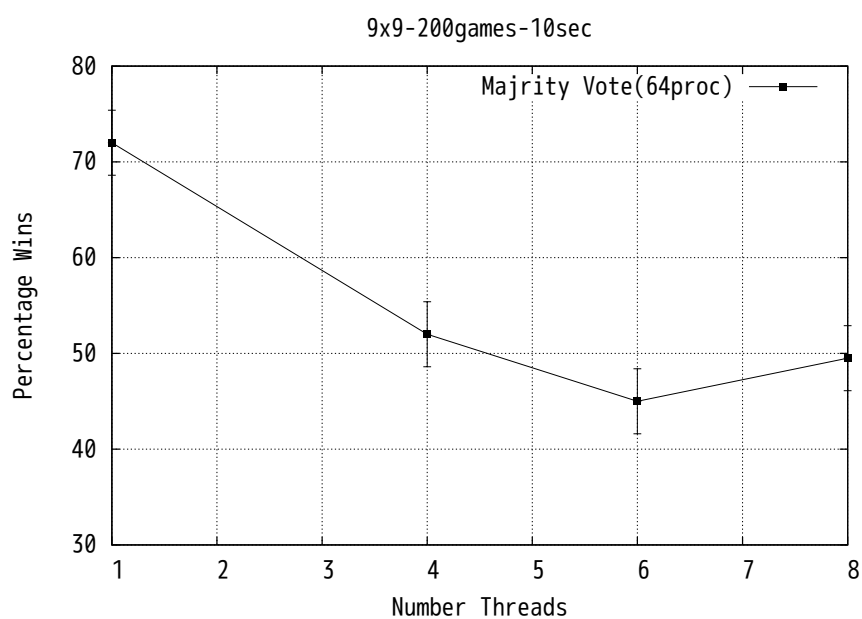


図 4.6: 9 路盤:Root 並列化 (合議制 64 コア)vsTree 並列化 (1~8 スレッド)

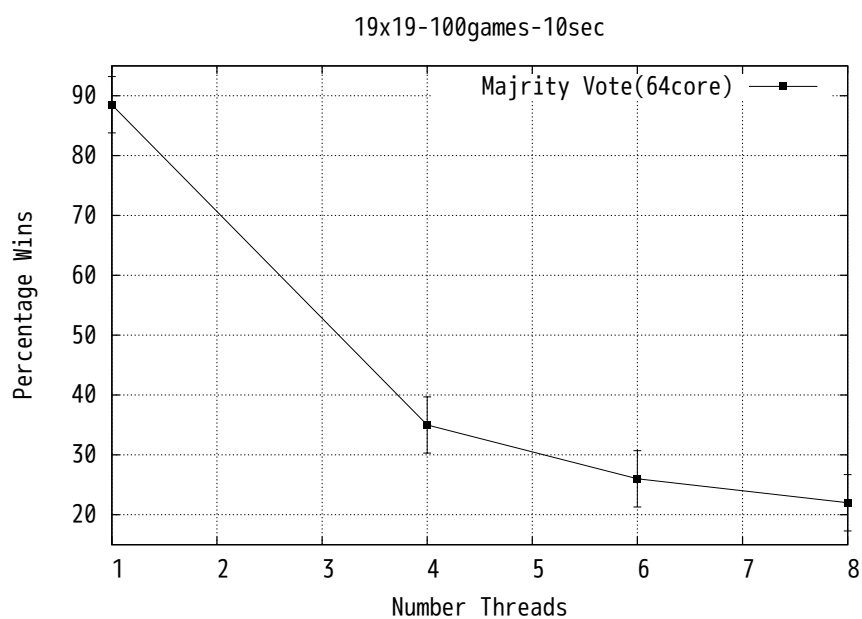


図 4.7: 19 路盤:Root 並列化 (合議制 64 コア)vsTree 並列化 (1~8 スレッド)

表 4.1: 9 路盤 合議制・総和制の勝率比較 (%) (8 × 8 Root 並列化)

8 × 8 Root 並列化	vs F_{tree}	vs Mogo
合議制	67.0	77.0
総和制	68.5	79.0

表 4.2: 19 路盤 合議制・総和制の勝率比較 (%) (8 × 8 Root 並列化)

8 × 8 Root 並列化	vs F_{tree}	vs Mogo
合議制	76.0 %	62.5 %
総和制	73.0 %	59.0 %

4.7 総和制と合議制の比較

総和制と合議制の Root 並列化プログラム同士を対戦させた結果が表 4.7, 4.7 である。対象は 8 × 8 Root 並列化の場合と、64 コア Root 並列化の場合である。9 路盤においては、8 × 8 Root 並列化では 3 % 合議制が負け越した。しかし、それ以外の場合では合議制が優位に勝ち越している。特に 19 路盤では 64 コア Root 並列化における合議制が 7 割も総和制に勝ち越している。

表 4.3: 9 路盤 8 × 8 Root 並列化 & 64 コア Root 並列化における合議制 vs 総和制の勝率比較 (%)

並列化手法	合議制の勝率
8 × 8 Root 並列化	47.0 %
64 コア Root 並列化	56.5 %

表 4.4: 19 路盤 8 × 8 Root 並列化 & 64 コア Root 並列化における合議制 vs 総和制の勝率比較 (%)

並列化手法	合議制の勝率
8 × 8 Root 並列化	59.0 %
64 コア Root 並列化	70.0 %

4.8 総和制と合議制の着手の性質

総和制と合議制による囲碁プログラムの振る舞いを調べるため、各手法が選択する着手の性質を調べた。

Root 並列化では、各プロセスから返ってきた各手の訪問回数を集計し、次の一手を選択するだけであるので、合議制と総和制の選択する着手が一致するかどうかは簡単に確認できる。この性質を利用して、9, 19 路盤において、逐次 Fuego を 64 個立ち上げた合議制 Root 並列化 (64 コア Root 並列化) と逐次 Fuego との 200 局の対戦棋譜に出現する Root 並列化の手番の局面で、合議制と総和制の着手の一致率を調べた。同様に、8×8 Root 並列化では、合議制 8×8 Root 並列化と F_{tree} (4.6 節参照) との 200 局の対戦棋譜を利用した。

その結果、9 路盤においては、64 コア Root 並列化と 8×8 Root 並列化の両方では、合議制と総和制の次の一手の一致率は、およそ 97% であった (表 4.5, 4.6 を参照)。そして 19 路盤における次の一手の一致率は、およそ 92% から 95% 程度であった (表 4.7, 4.8 を参照)。着手が一致しなかった残り数%の局面では、合議制・総和制とも各プロセスの最善手が一意に定まらず、少なくとも二極分化した。

着手が一致しなかった局面について、合議制と総和制の選出する着手の性質をより詳しく調べるため、長い時間制限 (30 秒) で着手選択を行う F_{tree} を利用した。この実験では、 F_{tree} を一番強いプログラムと仮定し、 F_{tree} にこれらの局面を探索させた。そして、 F_{tree} が出力した探索木の情報と、合議制・総和制の選出した着手と各プロセスの集計情報を比較し、詳しく分析した。

その結果が表 4.5 から 4.8 になる。 F_{tree} との一致数とは、 F_{tree} が探索し選出した手と、各手法の手が一致した数である。平均順位差とは、 F_{tree} の着手と一致した、((各手法の選出手の順位)-1) の平均である。以下のような式で表す事ができる。

$$\text{平均順位差} = \frac{\sum_{\text{全局面}} (F_{tree} \text{ と着手が一致した手の順位} - 1)}{\text{全局面数}} \quad (4.1)$$

例えばある局面において、 F_{tree} が A という着手を選出したとする。合議制 Root 並列化の集計結果において A が 1 位、総和制 Root 並列化の集計結果において A が 3 位だとすると、合議制はこの局面での順位差は 0、総和制は 2 ということになる。

9 路盤では両棋譜ともに F_{tree} との一致数が倍近く合議制の方が高い。また平均順位も同程度に合議制が高い。19 路盤では、64 コアでの棋譜がより顕著に差が表れている。およそ 4 倍近く一致数が多く、平均順位差も 0.5 離れている。対戦勝率の結果からも、64 コア Root 並列化の方がより総和制に比べて合議制の効果が高いと言える。

これらより、合議制の方が総和制よりも良い手を選んでいく傾向があると推測できる。しかし、9 路盤での MoGo との対戦や 8×8 並列化の自己対戦の結果では、必ずしも合議制の方が総和制よりも効果があるとも言えない。これは、一局における不一致局面数に関係すると思われる。9 路盤では合議制と総和制の着手が不一致する局面数は、およそ平均して 1,2 局面程度しかない。このため、合議制がより良い手を選出していたとしても、一局の勝敗に影響を与えられるかは疑問である。しかし 19 路盤では、特に 64 コア Root 並列化では一局に 10 局面から 20 局面程度存在する。このため、図 4.5 のように良い勝率を挙げているのではないかと思われる。

また 9 路盤の 64 コア Root 並列化において、合議制と総和制の着手が不一致であった局面で、合議制が F_{tree} の着手と一致した局面と総和制が F_{tree} と一致した局面をそれぞれ図 4.8 と図 4.9 に示す。

図 4.8 では、合議制が A7、総和制が A4 を選出した (F_{tree} の着手は合議制と同じであった)。この場合、A7 が好手であり、黒の大石を殺している。A4 につながれば、黒に A7 につながれてしまい、攻め合いに負けるので、上辺の白 5 子が取られてしまう。

図 4.8 における、総和制と合議制での上位 5 位までの着手に関する情報を表 4.9 に示す。A4 は各 CPU コアで平均的に高い訪問回数を得たために、総和制では選択されているのに対し、合議制では 64 コア中 9 コアしか A4 を選択していない。一方、総和制は A7 を 3 番目に有力な手であると考えているのに対し、合議制では 28 コアが A7 を最も有望であると考えている。この例は、実際には悪手であるが、平均的に訪問回数が高いために、最善であるとみなしてしまう総和制の弊害を合議制が防いでいると言える。

図 4.9 では、合議制が D1、総和制が E8 を選出した。この場合は F_{tree} の着手は総和制と同じであった。この場合、D1 は単に当たりをするだけの手であるうえに、コウ材を一つ消費しているので損な手である。一方、E8 は、ヨセとして大きい場所である。

図 4.9 の各着手の集計情報は、表 4.10 である。総和制では E8 の訪問回数が最も多いのに対し、合議制が E8 を選んだ CPU コアの数 は 16 と D1 の 22 よりも小さい。

表 4.5: 9 路盤 64 コア 200 試合の棋譜 (着手の不一致数 3.4% (187/5500 局面))

Root 並列化手法	F_{tree} との一致数	平均順位差
合議制	74 手	1.52 位
総和制	37 手	1.83 位

表 4.6: 9 路盤 8×8 200 試合の棋譜 (着手の不一致数 2.3% (159/6759 局面))

Root 並列化手法	F_{tree} との一致数	平均順位差
合議制	62 手	1.54 位
総和制	36 手	1.75 位

表 4.7: 19 路盤 64 コア 200 試合の棋譜 (着手の不一致数 8.7% (2277/26064 局面))

Root 並列化手法	F_{tree} との一致数	平均順位差
合議制	593 手	2.92 位
総和制	137 手	3.40 位

表 4.8: 19 路盤 8×8 200 試合の棋譜 (着手の不一致数 4.9% (1314/27032 局面))

Root 並列化手法	F_{tree} との一致数	平均順位差
合議制	328 手	2.65 位
総和制	189 手	2.85 位

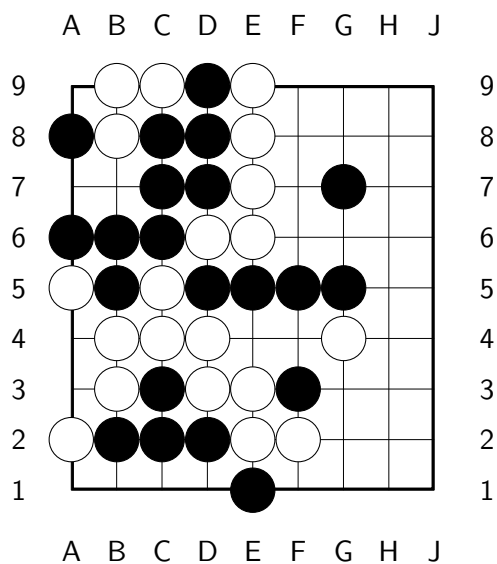


図 4.8: 合議制が F_{tree} の着手と一致した局面 (白番)

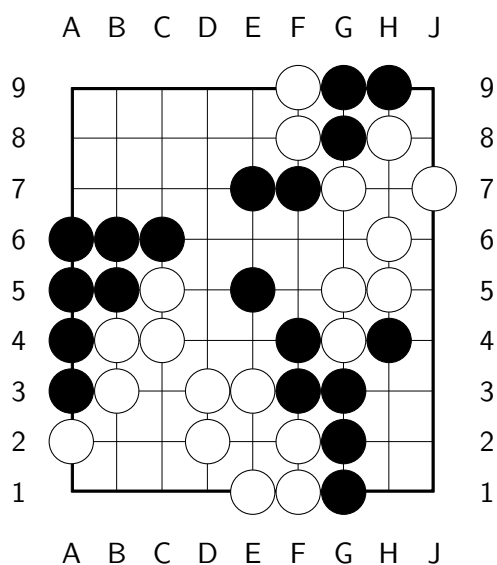


図 4.9: 総和制が F_{tree} の着手と一致した局面 (黒番)

表 4.9: 図 4.8 での総和制と合議制の各着手の情報 (上位 5 位まで)

総和制	候補手	平均訪問回数	合議制	候補手	平均訪問回数	投票数
1 位	A4	29,416	1 位	A7	25,917	28
2 位	H5	25,984	2 位	H4	20,460	14
3 位	A7	25,917	3 位	H5	25,984	13
4 位	H4	20,460	4 位	A4	29,416	9
5 位	H6	2,618	5 位	H6	2,618	0

表 4.10: 図 4.9 での総和制と合議制の各着手の情報 (上位 5 位まで)

総和制	候補手	平均訪問回数	合議制	候補手	平均訪問回数	投票数
1 位	E8	24,637	1 位	D1	20,910	22
2 位	B2	23,276	2 位	B2	23,276	21
3 位	D1	20,910	3 位	E8	24,637	16
4 位	E9	10,894	4 位	E9	10,894	5
5 位	B1	7,535	5 位	B1	7,535	0

4.9 一致率による Root 並列化の有効性

本節では、Root 並列化がどの程度まで台数効果が期待できるかを一致率を用いて実験・評価した。

本実験ではまず一手 80 秒という長い時間制限で思考する、最適なパラメータの逐次 Fuego プログラム (以下、F80s とよぶ) を用意した。そして F80s を用いて 9, 19 路盤ともに 400 局の棋譜を用意した (対戦相手は逐次 Fuego, 逐次 Mogo(共に一手 10 秒) とした)。F80s が思考した全局面 (9 路が 10591 局面, 19 路が 46193 局面) を、あるプログラム A に思考させるとする。その時の、プログラム A の一致率は以下のように定義した。

$$\text{一致率} = \frac{\text{F80s と A の着手が一致した局面数}}{\text{全局面数}} \quad (4.2)$$

4.9.1 一致率比較実験

以下の 4 つのプログラムに対し、F80s が思考した全局面をそれぞれ以下の条件で思考させた。

- 1. 逐次 Fuego(一手 1 秒 ~64 秒)
- 2. 総和制 Root 並列化 Fuego(一手 1 秒, 4 コア ~64 コア)
- 3. 合議制 Root 並列化 Fuego(一手 1 秒, 4 コア ~64 コア)
- 4. Tree 並列化 Fuego(一手 1 秒, 2 スレッド ~8 スレッド)

この一致率の変化をグラフにまとめたものが図 4.10,4.11 になる。まず Root 並列化については 9, 19 路盤ともに、一致率が頭打ちになっている事がわかる。32 コアから 64 コアにかけては、一致率の伸びは見られない。逆に Tree 並列化では、4 スレッドの時点で 64 コア Root 並列化の一致率を上回っている。対戦実験の結果同様に、Root 並列化の効果は Tree 並列化と同程度の差が見られた。

また、総和制と合議制については、1%程度合議制の方が一致率が高い結果となった。前節で行った着手の性質の実験とは、思考した秒数に差があった。1秒思考の方では、各プロセスが作成する探索木情報にバラエティが10秒に比べて出ないため選出する手が似てくる傾向があると推測される。

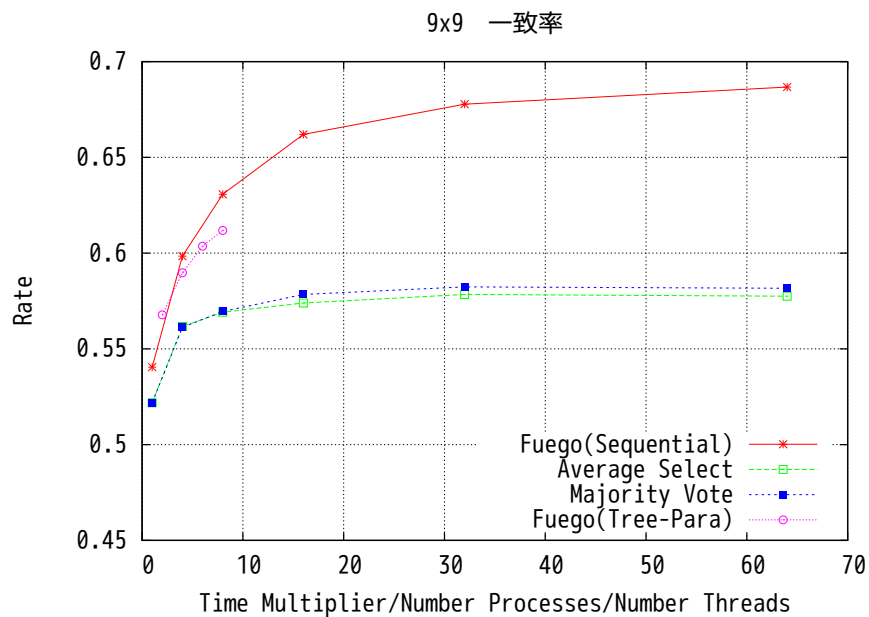


図 4.10: 9 路盤:F80s との着手一致率: 逐次 Fuego,Root 並列化 (合議制・総和制),Tree 並列化

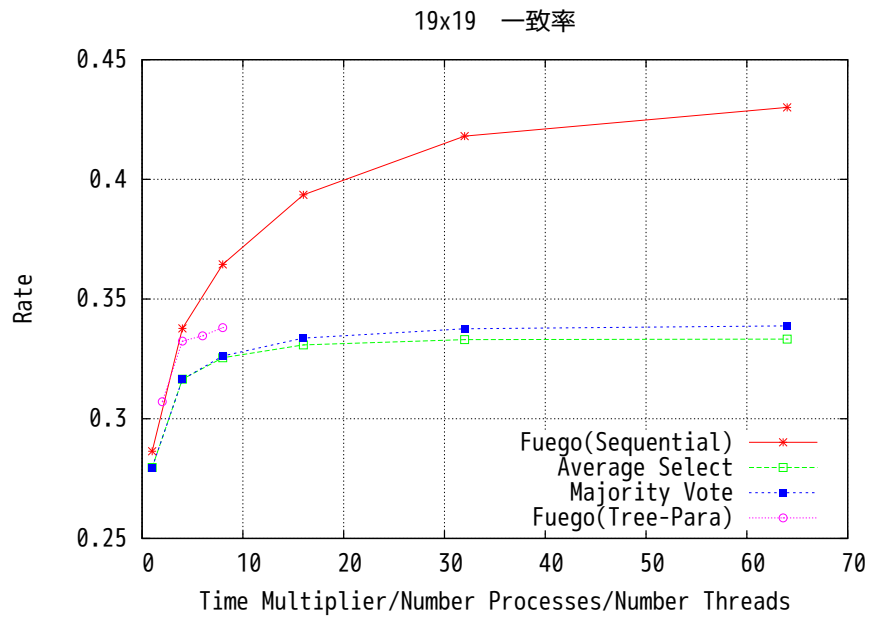


図 4.11: 19 路盤:F80s との着手一致率: 逐次 Fuego,Root 並列化 (合議制・総和制),Tree 並列化

4.9.2 より多くのプロセスを用いた Root 並列化の一致率

次に 19 路盤における合議制の Root 並列化について、512 プロセスまで使用し、一致率の変化具合を調べた (図 4.13)。わずかな一致率の上昇はあったが、ほぼ頭打ちであると言える。図 4.11 も考慮すると、1 秒での探索では Root 並列化はリソース 32 コア程度で頭打ちが見られてしまう事がわかる。

4.9.3 最善的 Root 並列化の一致率

Root 並列化における一番の課題は、あるプロセスが見つけた良手を抽出する事が難しい点にある。この実験では 19 路盤において、Root 並列化における各プロセスの中で、どれか一つのプロセスが F80s の着手と同じ手を選出した場合に一致とみなす、**最善的一致率**を計測した (図 4.13)。1 プロセスでは一致率は 0.28 程度に対し、64 プロセスの時点の一致率が 0.64 程度まで上昇した。さらに 512 プロセスでは 0.76 程度まで上昇した。

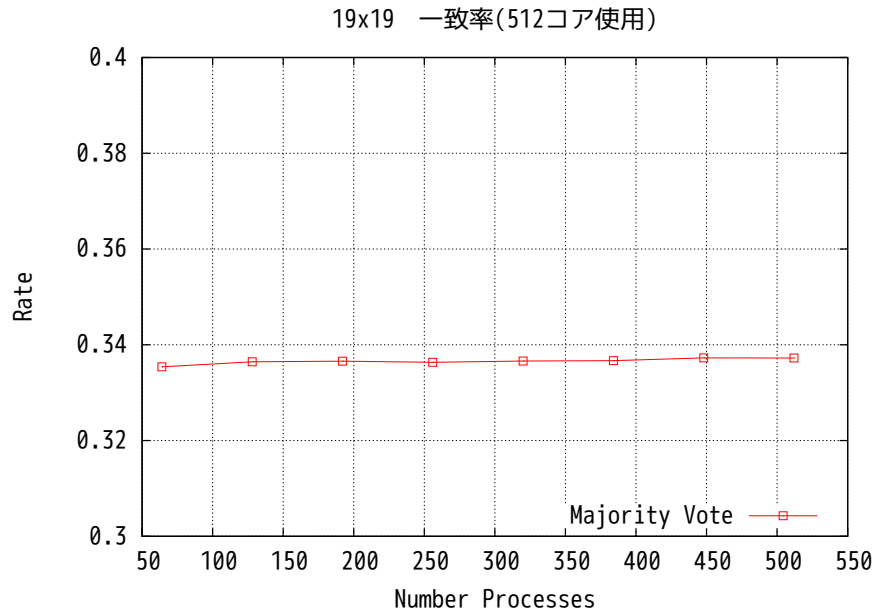


図 4.12: 19 路盤:F80s との着手一致率: Root 並列化 (合議制 64-512 コア)

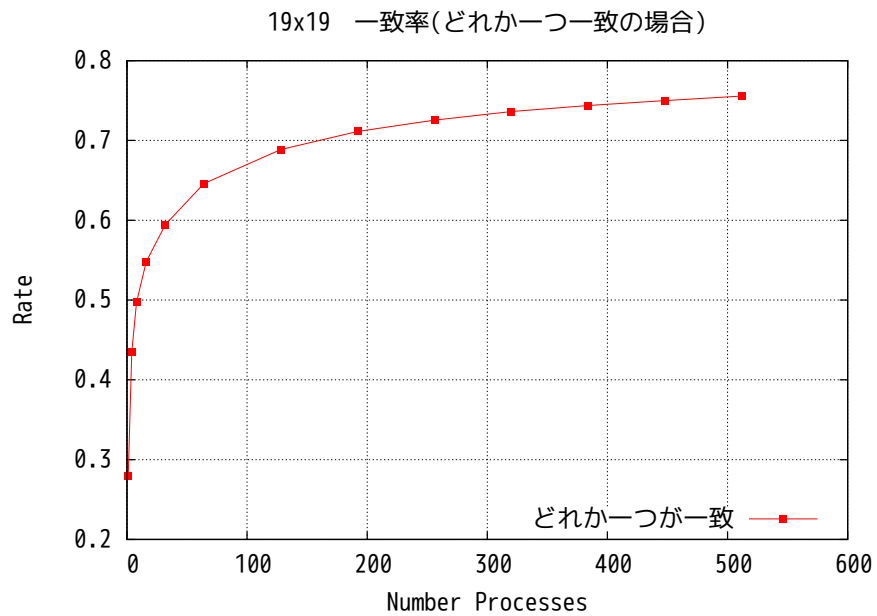


図 4.13: 19 路盤:F80s との最善的一致率: Root 並列化 (合議制 1-512 コア)

4.9.4 一位度別一致率

実際のモンテカルロ木探索では、局所的なパターンなどが多く用いられている (2章参照)。それにより、ある局面ではどのプロセスも同じ手を選出するという事態がしばしば起こる。そういう局面では Root 並列化の効果が少ないと言える。またそうした局面は並列化の手法によらず決まった手を選出してしまっているの、どの並列化手法でも一定の一致率を挙げていると思われる。こうした局面を分類することで、より並列化手法の効果が見えてくると推測される。

そこで我々は局面を分類する基準として一位度を定義した。一位度とは、その局面において全プロセス中何個のプロセスがその手を選出して一位になったかを数値的に表したもので、式は

$$\text{一位度} = \frac{\text{各プロセスから一番多く選出された着手の投票数}}{\text{全プロセス数}} \quad (4.3)$$

となる。この値によって、F80s の棋譜における全局面を分類した。また今回この全プロセス数については、512 プロセスを利用し各局面ごとに一位度を求めた。

まず表 4.11 に一位度の割合を示す。9 路盤では盤面が小さく合法手も平均的に少ない為、やはり一位度が高くなりやすい局面が多いと言える。19 路盤では、合法手が多く探索空間も大きい為、一位度が低い局面が多いと言える。そして 9, 19 路盤ともに、一位度別に各並列化手法の一致率をまとめてグラフにしたものが、図 4.14~4.23 となる。

9, 19 路盤ともに、一位度が大きい場合 ($0.6 \leq \text{一位度}$) には、Root 並列化は一致率の向上は見られない事がわかる (図 4.16~4.18, 図 4.21~4.23 参照)。特に $0.8 \leq \text{一位度}$ では、早い段階 (およそ 8 コア) で頭打ちとなり一致数が一方向に向上していかない。これより、一位度が高い局面では、Root 並列化は台数による性能向上は望めないことがわかる。

また 9, 19 路盤で、一位度が小さい場合 ($\text{一位度} < 0.4$) の振る舞いは異なっている。9 路盤では特に $\text{一位度} \leq 0.2$ では一位度が下がっているが、19 路盤では逆に上昇している (図 4.14~4.15, 図 4.19~4.20 参照)。

それ以外の場合である、一位度が 0.5 の近辺では、緩やかに台数効果をあげているグラフと言える。

また総和制と合議制の一致率の差は、 $0.4 \leq \text{一位度}$ では、あまり大きな一致率の差が見られないが、逆に $\text{一位度} < 0.4$ となるような局面では、合議制の方が良い一致率をあげている。着手の傾向が揃い出すと両手法の差は出てこないが、そうでない場合にはやはり合議制の方が総和制を上回る結果を出している。

表 4.11: 全局面における一位度の分類具合

一位度	9 路盤	19 路盤
0.0-0.2	0.0418 %	0.1266 %
0.2-0.4	0.0795 %	0.2542 %
0.4-0.6	0.2253 %	0.2715 %
0.6-0.8	0.2047 %	0.1832 %
0.8-1.0	0.4483 %	0.1642 %

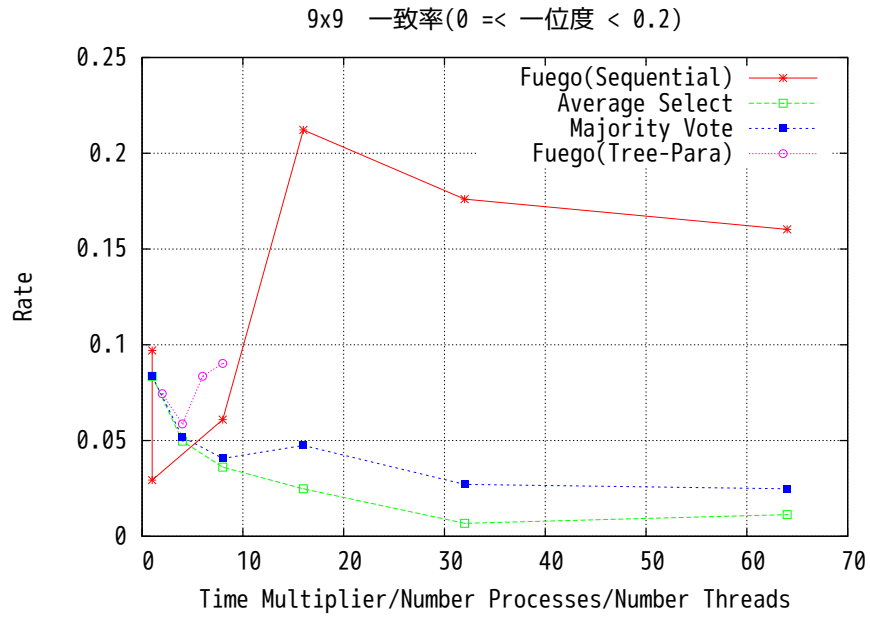


図 4.14: 9 路盤 ($0 \leq \text{一位度} < 0.2$) の局面における一致率

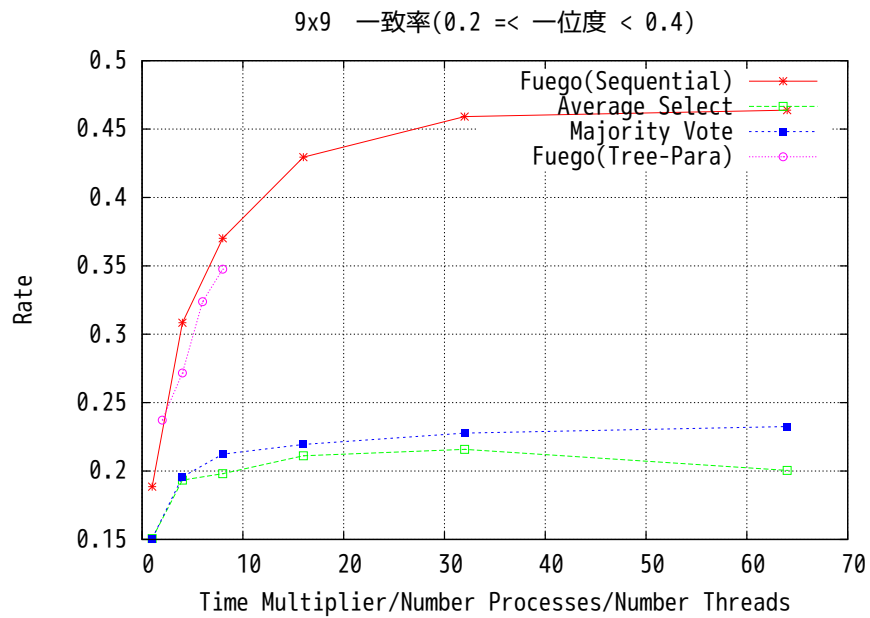


図 4.15: 9 路盤 ($0.2 \leq \text{一位度} < 0.4$) の局面における一致率

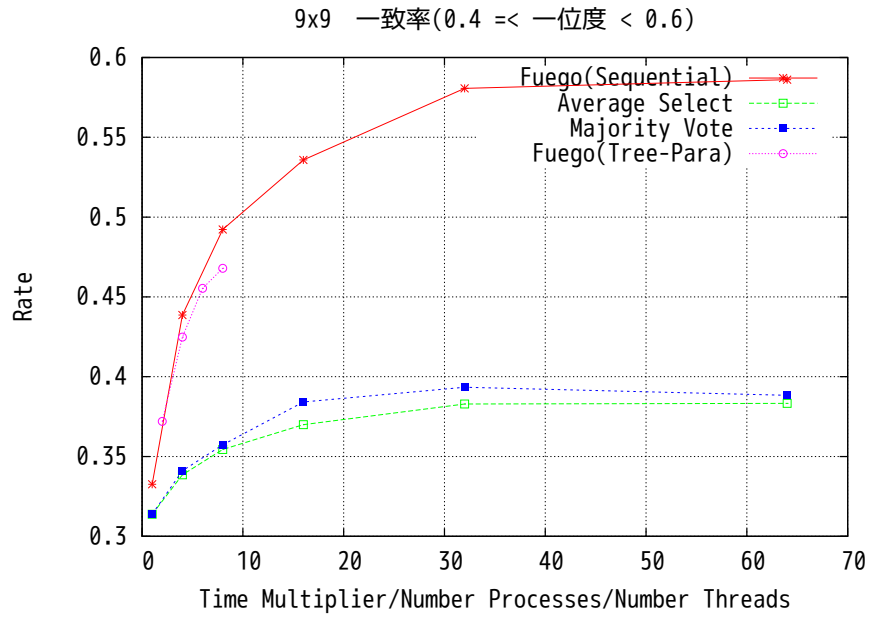


図 4.16: 9 路盤 ($0.4 \leq \text{一位度} < 0.6$) の局面における一致率

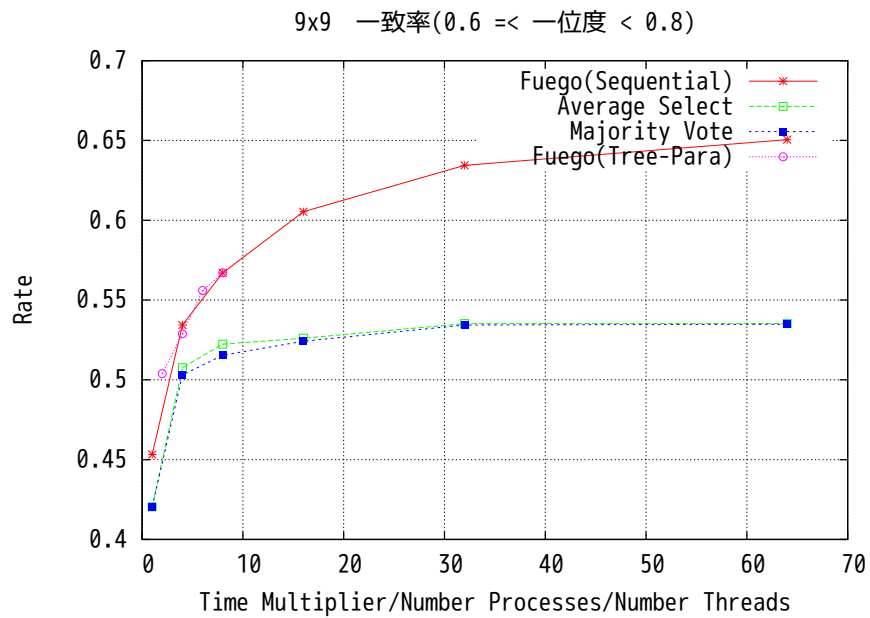


図 4.17: 9 路盤 ($0.6 \leq \text{一位度} < 0.8$) の局面における一致率

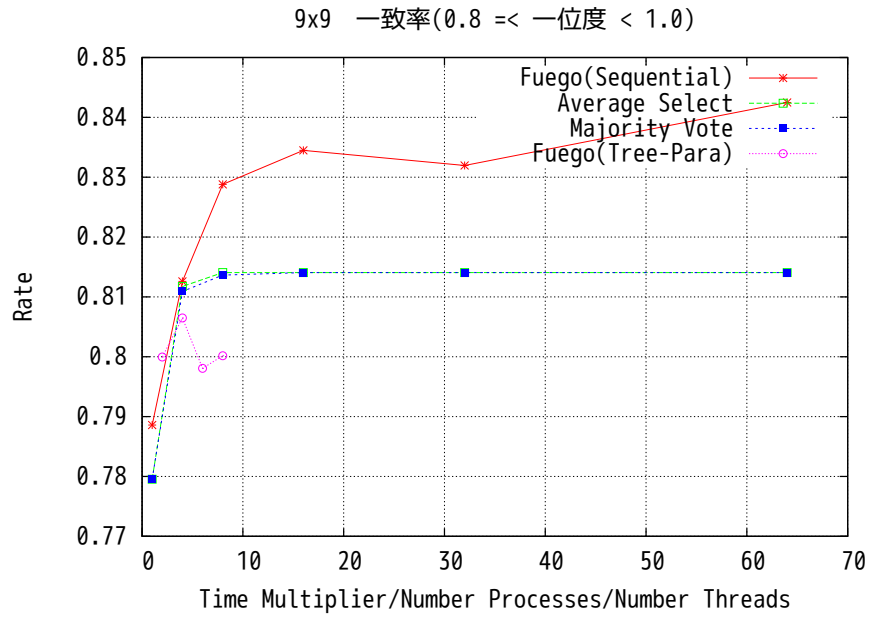


図 4.18: 9 路盤 ($0.8 \leq \text{一位度} < 1.0$) の局面における一致率

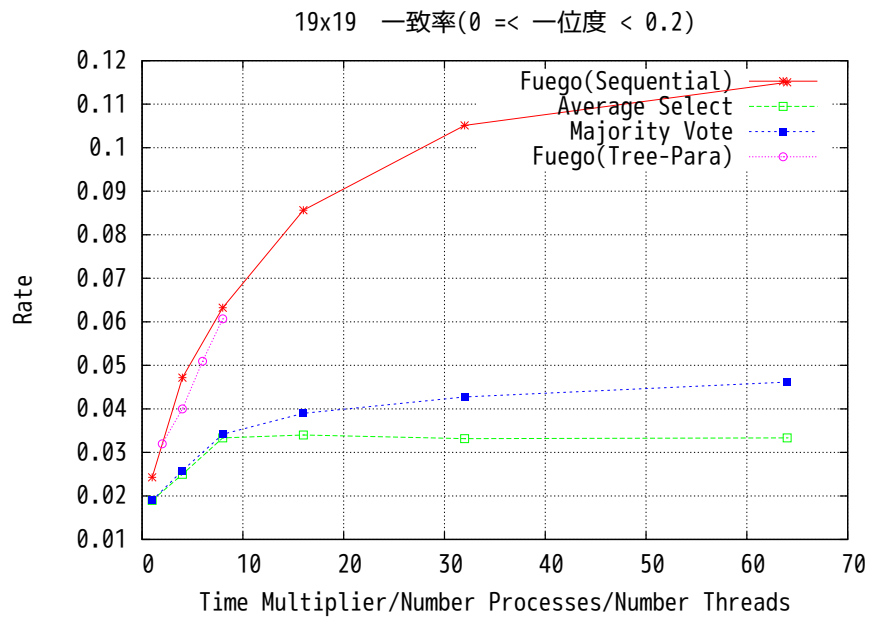


図 4.19: 19 路盤 ($0 \leq \text{一位度} < 0.2$) の局面における一致率

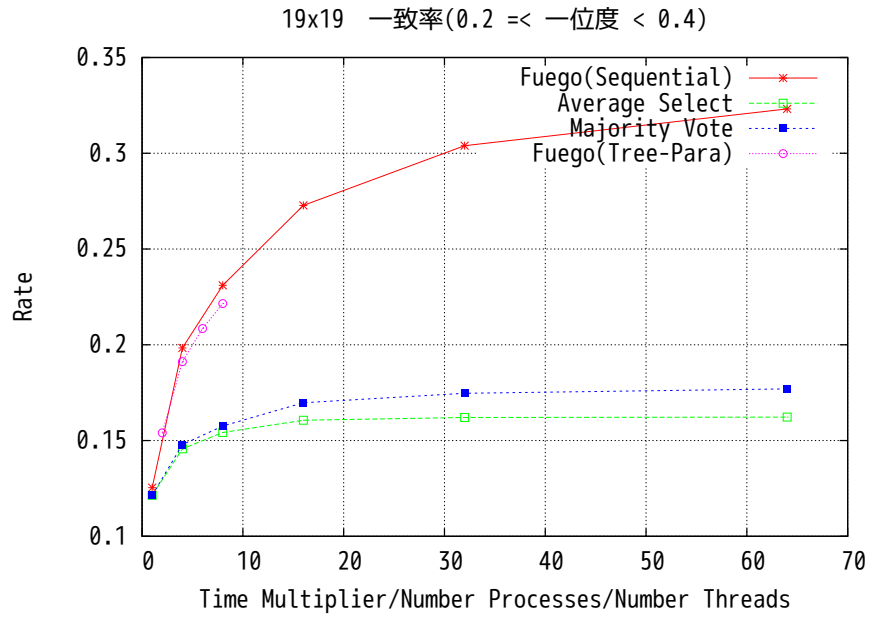


図 4.20: 19 路盤 (0.2 ≦ 一位度 < 0.4) の局面における一致率

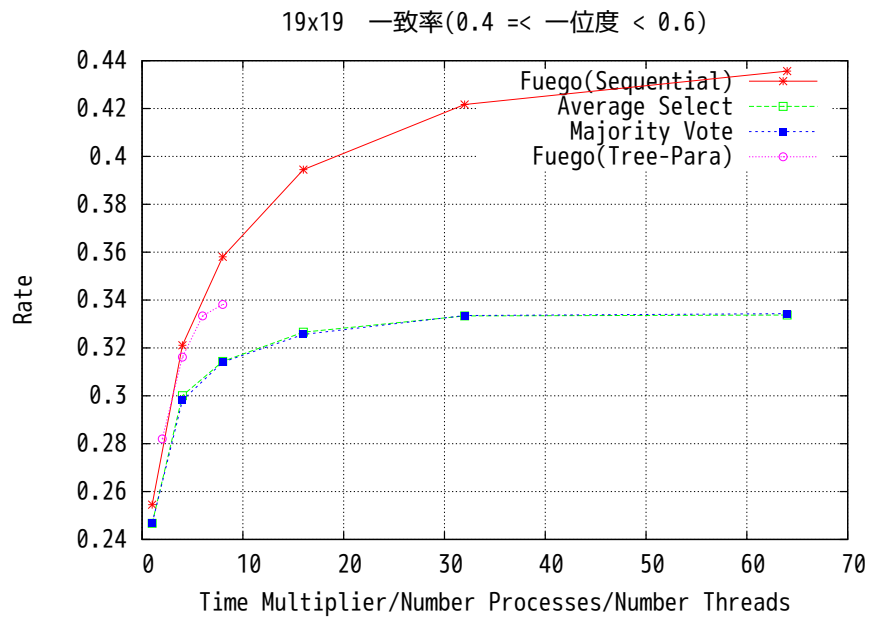


図 4.21: 19 路盤 (0.4 ≦ 一位度 < 0.6) の局面における一致率

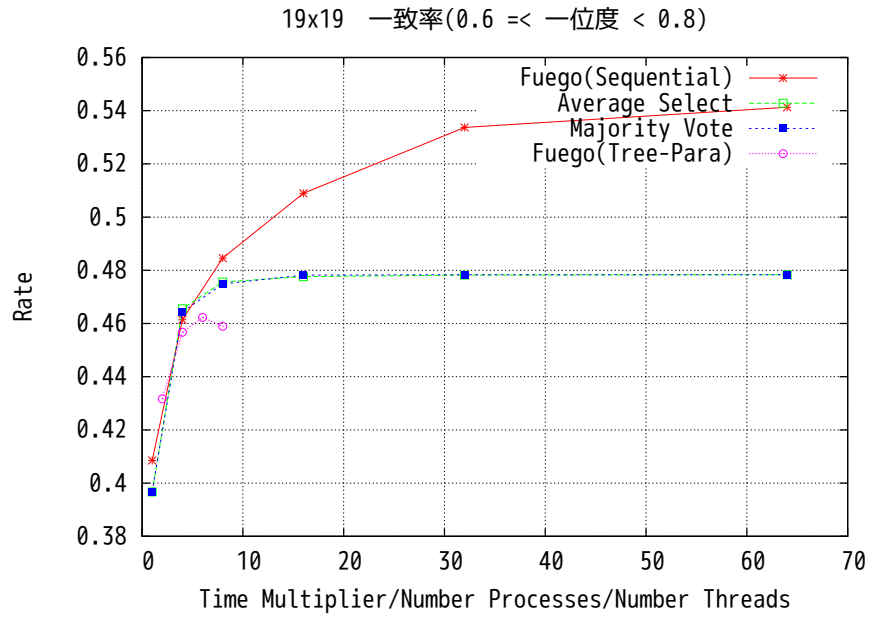


図 4.22: 19 路盤 (0.6 ≦ 一位度 < 0.8) の局面における一致率

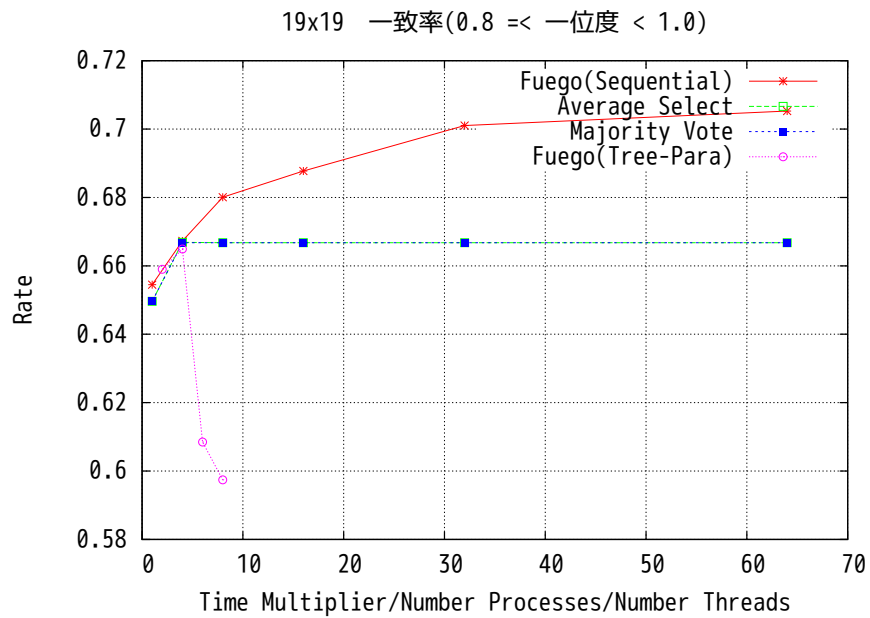


図 4.23: 19 路盤 (0.8 ≦ 一位度 < 1.0) の局面における一致率

4.9.5 局面段階数別一致率

一局の囲碁において、どの局面で Root 並列化が効果を挙げるかを調べるために、局面段階数を用いて全局面を分類した。局面段階数とはその局面が一局のうち、どの段階まで進行しているかを表す。ある試合のある局面 A について、その局面が n 手目で、試合の長さが L としたとき、

$$\text{局面段階数}(A) = \left\lfloor \frac{x}{L} \right\rfloor \quad (4.4)$$

例えば一局 100 手の長さがある試合において、ある局面が 30 手目だとした時、その局面段階数の計算は $\lfloor 30/100 \rfloor$ より、「局面段階数 = 0.3」と言える。

9, 19 路盤の各全局面について、局面段階数別に分けた一致率のグラフが図 4.24, 4.25 になる。9 路盤では、 $K < 0.2$ が最も一致率が低く、次に $0.8 \leq K < 1.0$ が低い。それ以外の局面数では同等の一致率を示している。19 路盤では、 $K < 0.2$ の場合が特に一致率が低い。19 路盤では序盤の合法手が非常に多い為である。

さらに付録図に、Tree 並列化、合議制・総和制 Root 並列化それぞれの局面段階数別のグラフを載せておく。これより、局面ごとにどの手法が有効であるかが、考察できる。(しかし今回の実験は 1 秒における一致率であるため、実際の試合においてこの考察を用いる事は難しい)

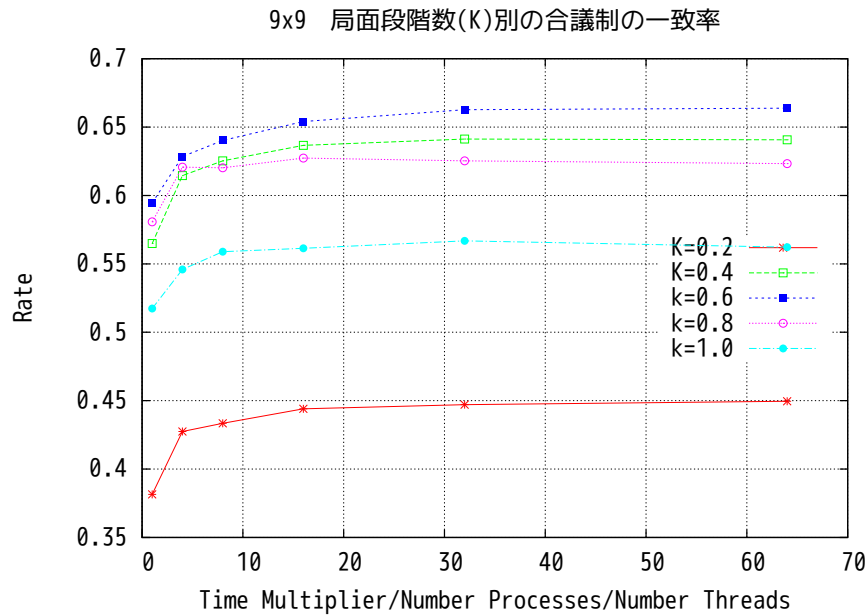


図 4.24: 9 路盤 局面段階数別の合議制一致率

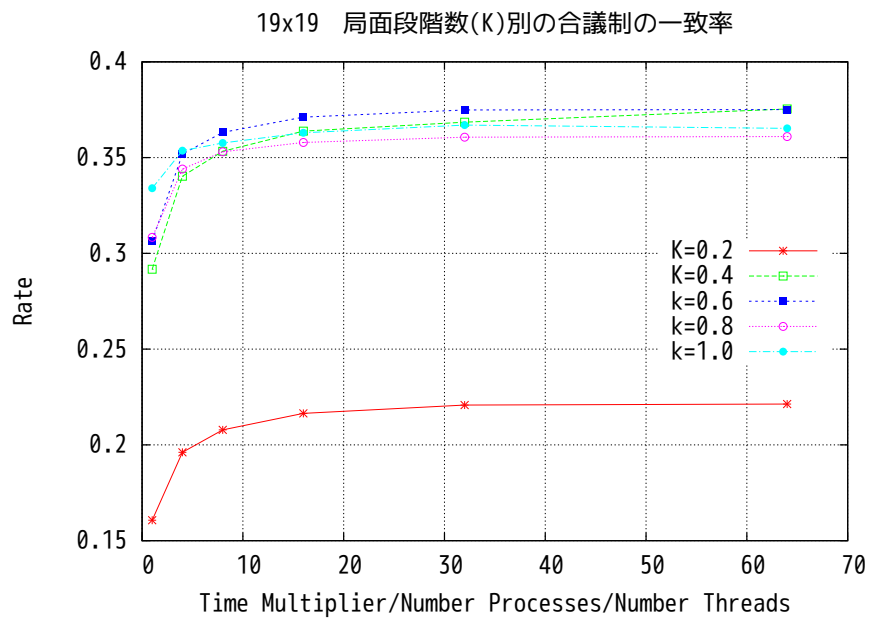


図 4.25: 19 路盤 局面段階数別の合議制一致率

第5章 結論

5.1 まとめ

本研究では、強い囲碁プログラム Fuego を用いてモンテカルロ木探索並列化について以下の3点を調査した。

Tree 並列化と Root 並列化の効果差の再調査

2章で説明した Chaslot らの先行研究における比較実験と比べ、我々の結果では、Tree 並列化の性能が大きく上回る事が確認された。対戦実験においては、9路盤では64コア Root 並列化は Tree 並列化の4スレッドと同等程度、19路盤では4スレッドに大いに負け越すという結果となった。実践に近い条件下では、以上のような差が計測された。

しかし、一致率の実験では、9路盤において Tree 並列化4スレッドの一致率と64コア Root 並列化の一致率が同程度であるのに対し、19路盤での一致率は Tree 並列化8スレッドと64コア Root 並列化が同程度であった。

また今回一致率の実験では、着手の偏りを局面ごとに分ける為に一位度を定義した。一位度とは、(4.9.4)にあるようにその局面について、合議制 Root 並列化によって一位と選出された手が全体の何割の投票数を得て選出されたかを表す数値である。一位度別の一致率について、一位度が高い局面では、Tree 並列化の一致度が Root 並列化に対して急激に下がる事も確認された。これは、対戦実験が思考時間10秒であるのに対し、一致率の実験では1秒という制限があるために生じた差だと思われる。

また、Fuego における Root 並列化と Tree 並列化を融合した並列化についても提案し、実験と評価を行った。今回の実験では、64コアのリソースの使い方として、8スレッド Tree 並列化を1つのプログラムとして、そのプログラムを8つ Root 並列化を行った。(これを8×8並列化と呼ぶ) その結果、同じ64コアのリソースの使い方でも、64コア Root 並列化に比べて、8×8並列化の方が大幅に性能が高いことが判明した。またもとの8スレッド Tree 並列化に対しても8×8並列化の方が性能が高く、更に強いプログラムにおいても Root 並列化の効果が示された(4.6節参照)。

総和制と合議制の比較

コンピュータ将棋で有望とされた合議制を、コンピュータ囲碁に取り入れ新しく提案し、実験・評価を行った。その結果、従来の Root 並列化手法である総和制に対し、優位であることが判明した。対戦実験では、9路盤では同程度の勝率であったが、19路盤では優位に高い勝率を得ている(4.4節, 4.7節参照)。さらに64コア Root 並列化、8×8並列化における両手法の対戦結果においても、合議制が勝ち越していた(4.6節参照)。

さらに両手法の着手の分析を、 F_{tree} を用いて行った。 F_{tree} とは、8 コアの Tree 並列化を行った Fuego である。 F_{tree} を一手 30 秒思考させたプログラムをより強いプログラムであると想定し、両手法と F_{tree} に複数の同一局面を思考させ、着手の差を分析した。その結果、合議制の方がより F_{tree} の着手に近い手を選出している事が判明した。

また、一致率の実験においても、ほぼ全ての実験で合議制の方が総和制の一致率を上回っていた。以上から、合議制の方が有力な Root 並列化手法と言える。

大規模な数の CPU コアを用いた場合の Root 並列化の効果の予測

Root 並列化の効果の予測について幾つかの実験で評価を行った。まず、対戦実験では逐次 Fuego, Mogo を対戦相手として、Root 並列化は 4 コアから 64 コアまで勝率の上昇が見られた (4.4 節参照)。

しかし、一致率の実験においては、64 コアで頭打ちと思われる兆候が見られた。512 プロセスまで利用した一致度のグラフでは、32 プロセス以降、一致率の大きな変化は見られず、上昇は微々たるものであった。

さらに、Root 並列化が最適な集計手法によって最善手を抽出できたと仮定した場合には (4.9.3 節参照)、64 プロセスまではニアに一致率が上昇するが、512 プロセスでは限界が見られた。

また、一位度別一致率の実験では、一位度が高いような局面では、台数を 64 コアまで増やしても、一致率を上昇させる事ができなかった。これは Root 並列化は一つ一つのプロセスの探索が浅い為、深い探索でないと見つけられない手を抽出できない事を意味している。こうした局面では Root 並列化では、一向に一致率を改善することが出来ないと予想される。

逆に、一位度が低い局面では、19 路盤においては台数効果が見られた。およそ逐次 1 プロセスの 4 倍近い一致率を 64 コアで達成した (図 4.19)。着手がプロセスによって割れるような場合に、合議制 Root 並列化による多数決の効果が表れた。しかし 9 路盤においては、一致率が下がる傾向があった。この原因についてはまだ解明されていない。

最後に、局面段階数で局面を分類した結果、局面段階数 < 0.2 の場合は一致率が著しく低かった。9 路盤では $0.8 \leq$ 局面段階数の場合でも他に比べ一致率が低かった。19 路盤では、局面の段階で一致率が変化はしないことがわかった。序盤ではやはり合法手が多いため、Root 並列化のような探索が浅い手法ではより深く思考したプログラムとは一致しにくい。

5.2 今後の課題

より正確な一致率の実験

今回実験を行った一致率による台数効果の実験では、思考時間を1秒とした。しかし、Fuegoのような強いプログラムにおいては、ある程度の思考時間が無いと本来の強さを発揮できない場合が多い。思考時間を10秒とした一致率の実験を行う事で、より正確な実践での台数効果が計測できると思われる。

効率のよいRoot 並列化

節4.9.3にあるように、Root 並列化では各プロセスの着手から最善である手を抽出できる事が好ましい。最善の場合には、図4.13のように一致率が向上する事がわかった。Root 並列化手法でこうした着手を抽出できる方法が存在すれば、Root 並列化の効果が更に向上する事が予想される。

探索のバラエティ化

Root 並列化のデメリットとして、探索オーバーヘッドが大きい。異なる乱数シードを基にモンテカルロ木探索を開始してはいるが、同じような探索に偏りがちである。

各プロセスに探索の傾向をもたせ、局面ごとに各プロセスの重みを適切につける事で着手の精度の向上が見込まれる。

謝辞

本研究を進めるにあたり，大変多くの方にお世話になりました。

岸本章宏助教授には研究の進め方や，論文の書き方，日頃から多くの助言を頂きました。研究に対する姿勢や，考え方まで様々な面で多くの知識を伝授して頂きました。また囲碁という自分の好きな分野に，僅かながらでもたずさわる事が出来たのも，先生のおかげでした。

佐々政孝教授には学部のことからご指導頂き，いつも暖かい目で見守って下さりました。論文作成やプレゼンテーションなど要所での確なご指導，ご教授を頂きました。

そして佐々研究室の皆様には公私にわたり多くの助言を頂きました。太田君や岩橋さんを始め，先輩後輩の皆様には本当にお世話になりました。更に同期である数理・計算科学専攻の皆様にも日々励ましてもらい，感謝の意で一杯です。

ここに心より感謝の意を表します。

付録図

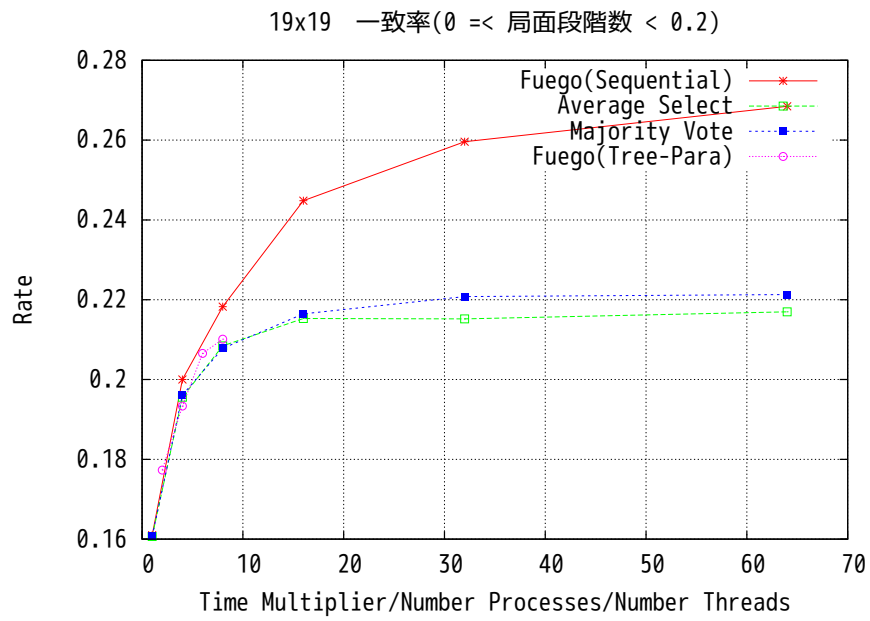


図 5.1: 19 路盤 局面段階数別一致率

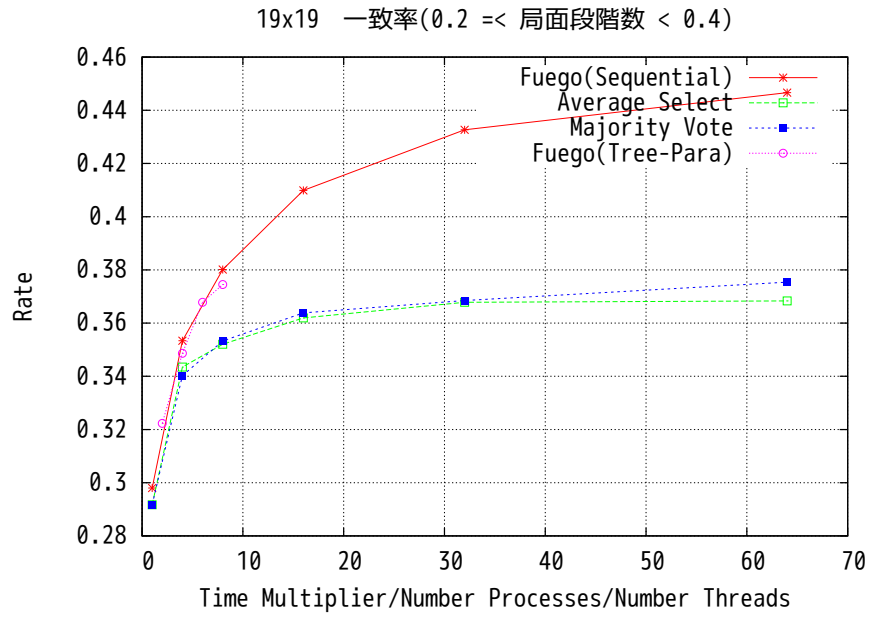


图 5.2: 19 路盤 局面段階数別一致率

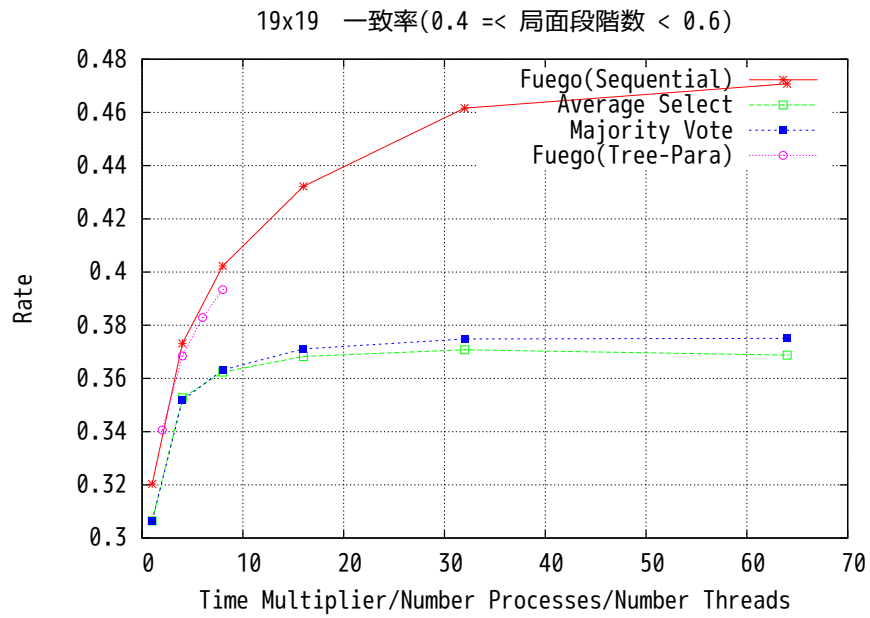


图 5.3: 19 路盤 局面段階数別一致率

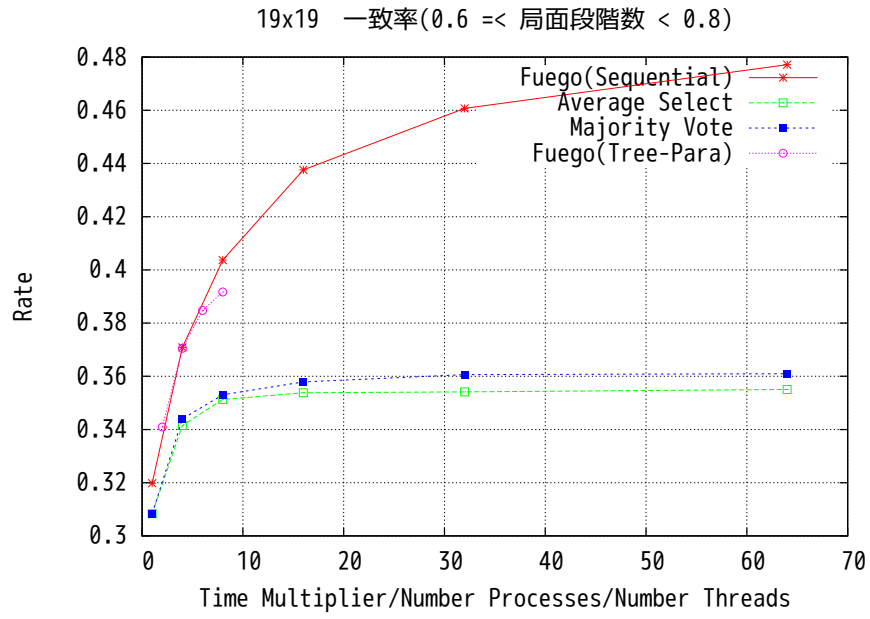


图 5.4: 19 路盤 局面段階数別一致率

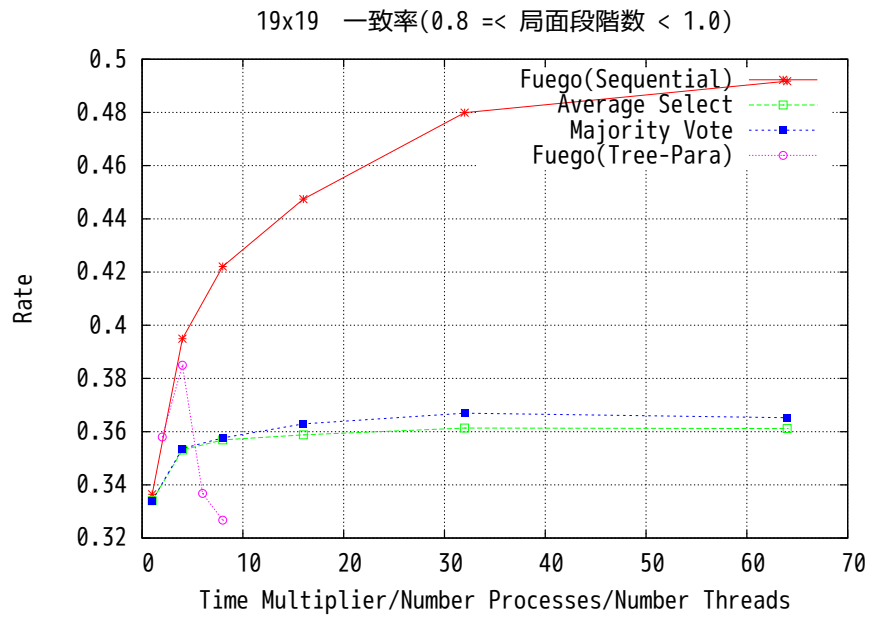


图 5.5: 19 路盤 局面段階数別一致率

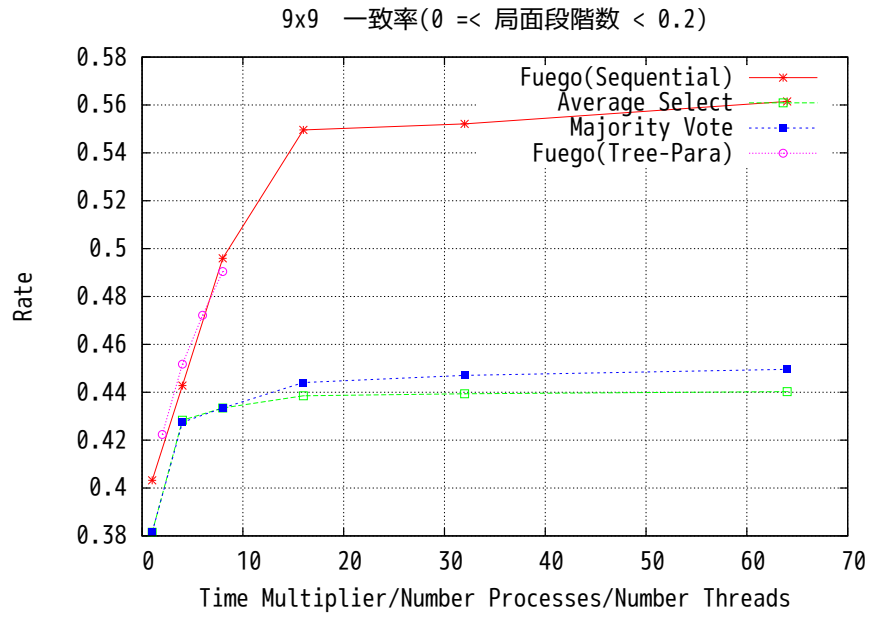


図 5.6: 9 路盤 局面段階数別一致率

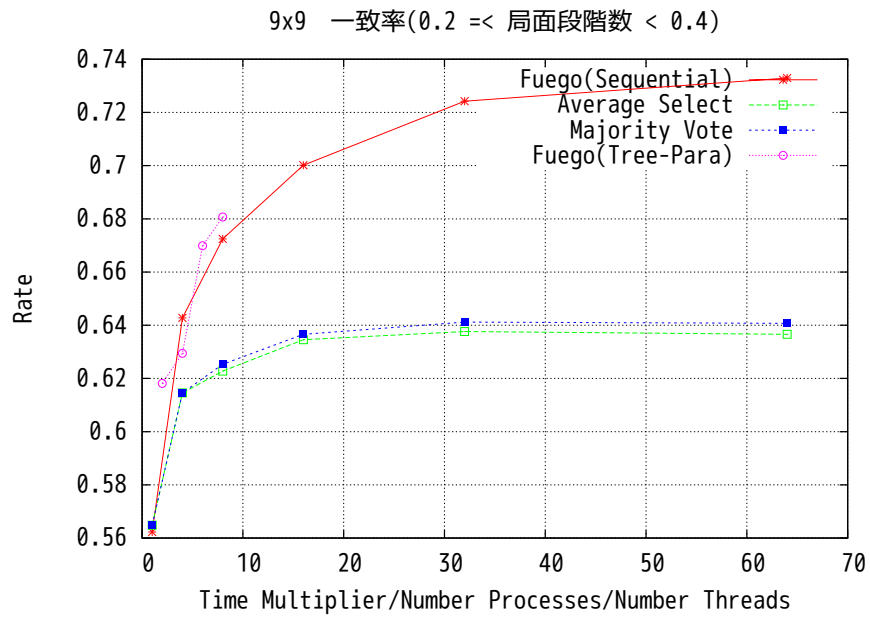


図 5.7: 9 路盤 局面段階数別一致率

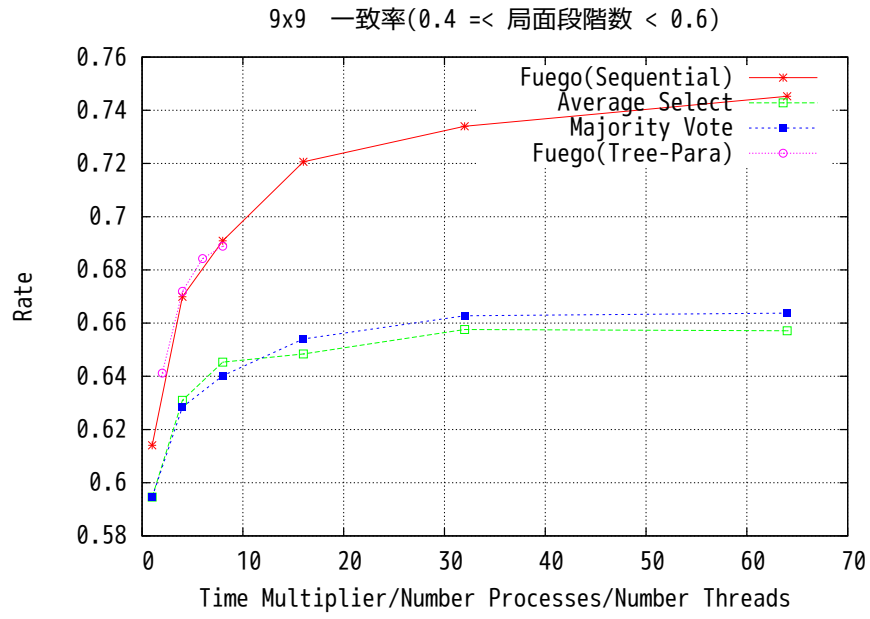


図 5.8: 9 路盤 局面段階数別一致率

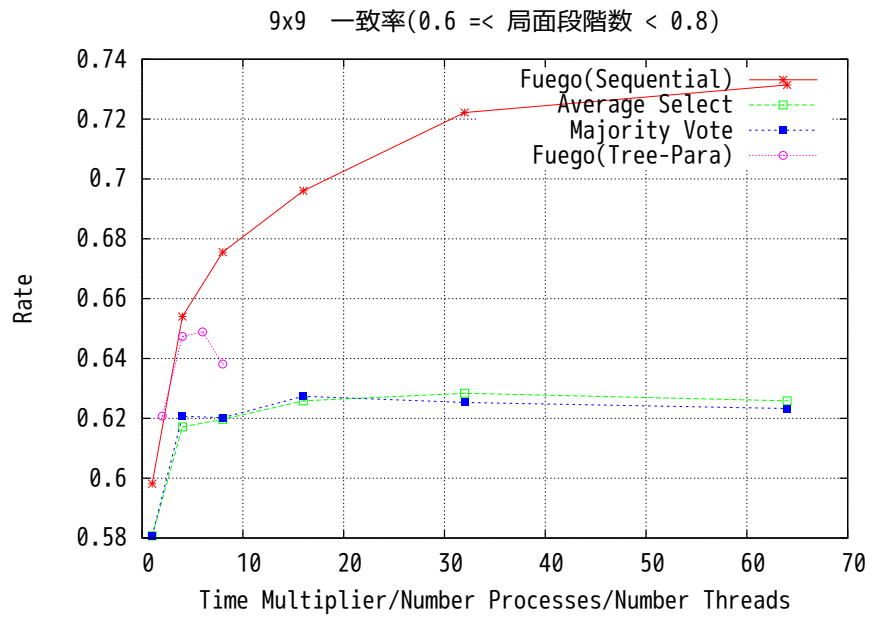


図 5.9: 9 路盤 局面段階数別一致率

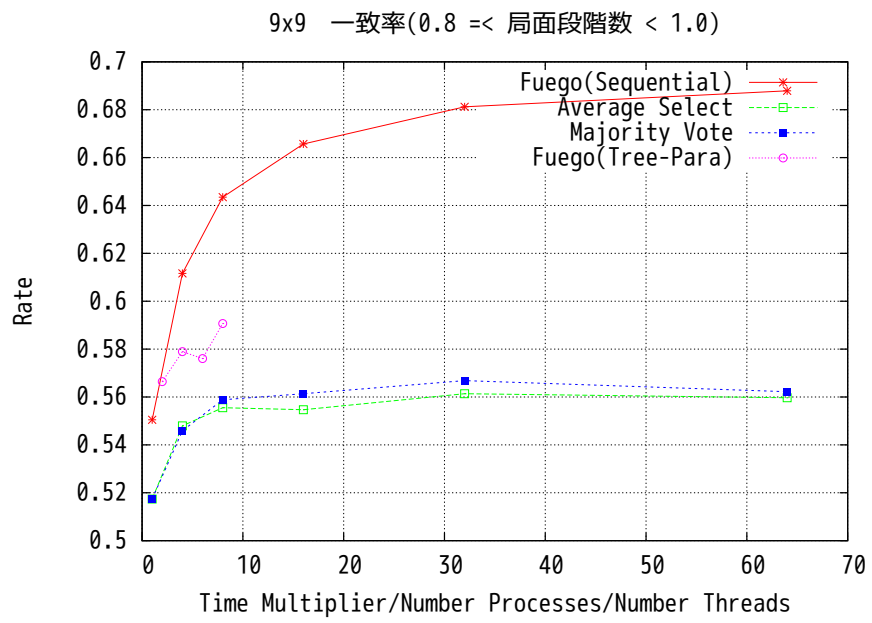


图 5.10: 9路盤 局面段階数別一致率

関連図書

- [1] T. Cazenave and N. Jouandeau. On the parallelization of UCT. In *Proceedings of the Computer Games Workshop*, pp. 93–101, 2007.
- [2] G. M. J-B. Chaslot, M. H.M Winands, and H. J. van den Herik. Parallel Monte-Carlo tree search. In *Proceedings of the 6th International Conference on Computer and Games*, Vol. 5131 of *Lecture Notes in Computer Science*, pp. 60–71, 2008.
- [3] R. Coulom. Computing Elo ratings of move patterns in the game of Go. *ICGA Journal*, Vol. 30, No. 4, pp. 198–208, 2007.
- [4] D.E.Knuth and R.W.Moore. An analysis of alpha-beta pruning. In *Artificial Intelligence*, pp. 293–326, 1975.
- [5] M. Enzenberger and M. Müller. Fuego - an open-source framework for board games and Go engine based on Monte-Carlo tree search. TR 09-08, University of Alberta, 2009.
- [6] M. Enzenberger and M. Müller. A lock-free multithreaded Monte-Carlo tree search algorithm. In *Advances in Computer Games 12*, 2009.
- [7] S. Gelly, J. B. Hoock, A. Rimmel, O. Teytaud, and Y. Kalemkarian. The parallelization of Monte-Carlo planning. In *Proceedings of the 5th International Conference on Informatics in Control, Automation, and Robotics*, pp. 198–203, 2008.
- [8] S. Gelly and D. Silver. Combining online and offline knowledge in UCT. In *Proceedings of the 24th International Conference on Machine Learning*, pp. 273–280, 2009.
- [9] S. Gelly, Y. Wang, Remi Munos, and O. Teytaud. Modification of UCT with patterns in Monte-Carlo Go. Technical Report 6062, INRIA, 2006.
- [10] GPSshogi. <http://gps.tanaka.ecc.u-tokyo.ac.jp/gpsshogi/>. 2009.
- [11] J.Schaeffer and A.Plaat. Kasparov versus deep blue: The re-match. In *International Computer Chess Association Journal*, pp. 95–101, 1997.
- [12] J.Scheaffer. One jump ahead: Challenging human supremacy in checkers. In *Springer-Verlag*, 1997.
- [13] H. Kato and I. Takeuchi. Parallel Monte-Carlo tree search with simulation servers. In *Proceedings of the 13th Game Programming Workshop*, pp. 31–38, 2008.

- [14] H. Kato and I. Takeuchi. Zen のクラスタ並列化. In *Proceedings of the 14th Game Programming Workshop*, pp. 22–26, 2008.
- [15] Kocsis.L and Szepesvari.C. Bandit based monte-carlo planning. In *17th European Conference on Machine Learning(ECML 2006)*, pp. 282–293, 2006.
- [16] Lai.T.L and Robbins.H. Asymptotically efficient adaptive allocation rules. In *Advances in Applied Mathematics*, pp. 4–22, 1985.
- [17] M.Buro. The othello match of the year: Takeshi murakami vs logistello. In *International Computer Chess Association Journal*, pp. 189–192, 1997.
- [18] M. Müller. Fuego at the Computer Olympiad in Pamplona 2009: a tournament report. TR 09-09, University of Alberta, 2009.
- [19] P.Auer and N.Cesa-Bianchi.P.Fischer. Finite-time analysis of the multiarmed bandit problem machine learning. pp. 235–256, 2002.
- [20] GNU Go GNU Project. <http://www.gnu.org/software/gnugo/>. 2009.
- [21] Jonathan Schaeffer, Neil Burch, Yngvi Björnsson, Akihiro Kishimoto, Martin Müller, Robert Lake, Paul Lu, and Steve Sutphen. Checker is solved. In *Science 14 September 2007*, pp. 1518–1522, 2007.
- [22] S.Gelly.Y.Wang, R.Mumos, and O.Teytaud. Modification of uct with patterns in monte-carlo go. In *RR-6062-INRIA*, pp. 1–19, 2006.
- [23] Yoshimoto.H, Yoshizoe.K, Kaneko.T, Kishimoto.A, and Taura.K. Monte carlo go has a way to go. In *Twenty-First National Conference on Artificial Intelligence(AAAI-06)*, pp. 1070–1075, 2006.
- [24] 小幡拓弥, 杉山卓弥, 保木邦仁, 伊藤毅志. 将棋における合議アルゴリズム: 既存プログラムを組み合わせて強いプレイヤーを作れるか? In *Proceedings of the 14th Game Programming Workshop*, pp. 51–58, 2009.
- [25] 小幡拓弥, 埜雅織, 伊藤毅志. 思考ゲームによる合議アルゴリズム - 単純多数決の有効性について. 第 22 回ゲーム情報学研究会, 2009.
- [26] 杉山卓弥, 小幡拓弥, 斎藤博昭, 保木邦仁, 伊藤毅志. 将棋における合議アルゴリズム-局面評価値に基づいた指し手の選択. In *Proceedings of the 14th Game Programming Workshop*, pp. 59–65, 2008.