動的なマージンを用いる Futility Pruning

伊藤 裕, 橋本 剛, 橋本 隼一
北陸先端科学技術大学院大学 情報科学研究科

概　要

これまで futility pruning は, チェスでは有効であるが将棋では探索空間の広大さと選択的探索が主流な事もあり, 有効ではないと考えられていた枝刈り技術であった. しかし, 2006 年度世界コンピュータ将棋選手権以降, futility pruning を将棋に用いた Bonanza が好成績を収めてきた. この事を受けて, 将棋でも futility pruning が注目されることになった.
futility pruning は, 大雑把な評価値にマージンを持たせて, そのマージンが α β に入っていなかったら枝刈りをする技術である. これまで, この futility pruning で用いる適切なマージンには定数が用いられていた. これは, 局面状況によって適切なマージンが変化していた場合に, 対応できないという問題が発生する事になる. よって, 探索毎にマージンを動的に変化させる必要がある. 本稿では, futility pruning のマージンを定数で扱っている事による問題点を解析し, その改良案として, 探索毎に動的なマージンを決定するアルゴリズムの実装を行った. そして, 動的なマージンを用いた futility pruning が静的なマージンを用いたものに “175 勝 138 敗 1 分” という戦果をあげた.

A Dynamic Margin for Futility Pruning

Yutaka Ito, Tsuyoshi Hashimoto, Junichi Hashimoto
Japan Advanced Institute of Science and Technology

Abstract

Futility pruning was previously used in the domain of chess, but not for shogi: it is indeed believed that futility pruning is inappropriate for selective search, which is used a lot in this game. Also, the huge search space of shogi is another difficulty. However, the program Bonanza won the 16^{th} World Computer Shogi Championship using futility pruning. After this event, this pruning method got more attention.
Futility pruning is a forward pruning technique. It uses an evaluation function with an appropriate margin: if, for a given node, the difference between the evaluation and the beta value is greater than the margin, then the node is pruned. Until now, this margin was given a static value, which is an important problem. Also, finding the best margin is difficult. This report analyzes these problems and discusses our solution: to adapt the margin for every search of the game tree. And, dynamic margin for futility pruning matched adversary static margin. This result is ’175 wins, 138 losses, 1 drow’.

1　はじめに

チェスでは全幅的探索が主流であり futility pruning[5] が探索効率化で有効な技術であったが, 将棋では探索空間が広大なので有効でないと考えられてきた. だが, 2006 年の世界コンピュータ将棋選手権でチェス風の全幅的探索と futility pruning を使う Bonanza が優勝しこの常識が覆された.[7]

この Bonanza の成果により, 今後コンピュータ将棋では futility pruning が使用される事が多くなると考えられる. ところが, futility pruning は機能を拡張した extended futility pruning[4] 以来あまり研究されていない. そこで, 本稿では futility pruning に注目し枝刈りの際に用いるマージンに焦点を当て改良法を検討する.

2　関連研究

これまで無駄な探索を抑制するために提案されてきた前向き枝刈りの技術は, 末端まで探索せ

ずに最善手となる見込みの無さそうな手の探索を削減する方法である. しかし, 枝刈りせずに探索を行えば良い展開になる手が, 展開されずに刈られる可能性がある.

futility pruning は前向き枝刈りの内の 1 つであるが, その他にも様々なものがある. この前向き枝刈り技術では何か基準となる値を用いて枝刈りを行うのだが, 基準値となる値の扱い方でそれぞれ分類できる. あらかじめ定数を与えておく静的な分類として prob cut[3], multi cut[2] 等がある. 探索ノードや, 探索木から得られた情報から基準値を決定する動的な分類として, null move pruning[1], history pruning 等がある.

しかし, 更に分類していくと 3 つ目の分類が可能である. それは, 元々静的であった枝刈り方法を動的にする事により更に効果を上げるといった分類である. この動的でもあり, 静的でもあるという分類には “DS-cut” 等がある.
上記で登場した “DS-cut” の元となる SS-cut では, そっぽの指し手を判定し探索の際に読むべき重点箇所を判定していたが, 着手がそっぽかどうか判定する際の基準値を一定の値に固定すると不適切な場合が生じていた. そこで, 松原ら [8] は, そっぽカットで使用する閾値を探索中の統計を利用する事により動的に決定する様にした. その結果, これまでのそっぽカットでは不適切な判定だった局面で適切な判定が行える様になり, 局面状況に対するより柔軟な探索が出来る様になった.

問題点が解消されて更に適切な判断が出来る様になった方法がある.

3 futility pruning と問題点

3.1 futility pruning

一般的に, α β 探索では horizon node(残り深さ = 0) で評価関数により評価値を計算する. その値が β 以上の場合カットとなる. 実際の評価値 (eval_value) は, 式 (1) の様に駒割 (material_value) と駒割以外の評価値 (position_value) の和からなると考えられる. 通常駒位置による評価は, 駒割の評価に比べてコストが大きい. もし, 駒割以外の評価値の最大値 (max_postion_value) がわかっていれば, 少なくともカットが起こるか否かは駒割以外の評価値を求める事なく, この最大値のみで判定出来る.

futility pruning はこの考え方を利用した前向き枝刈り法である. ただし, 真の最大値は不明なので, その代わりにある程度大きな値 (margin) を用いる. 以下, margin については式 (2) を満たすものとして議論を進める.

- eval_value : 局面評価値
- material_value : 駒割
- position_value : 駒割以外の評価値

$$eval\_value = material\_value + position\_value \quad (1)$$

$$margin \geq |position\_value| \geq 0 \quad (2)$$

さて, frontier node では式 (3) に従って β カットが行われる. ここで, この判定をマージンを利用した判定に置き換える事を考える. 式 (1) 式 (2) より式 (4) が成立するため, 式 (3) が成立するための十分条件は式 (5) である.

この式 (5) は, β カットを駒割からマージンを引いた値によって判定出来る事を示している.

$$eval\_value \geq \beta \quad (3)$$

$$eval\_value \geq material\_value - margin \quad (4)$$

$$material\_value - margin \geq \beta \quad (5)$$

futility pruning では, この判定を frontier node(残り深さ = 1) で行っている. この手法の利点は, 軽い処理で枝刈り判定することで探索コストを低減出来ることである. また, extended futility pruning[4] と呼ばれる方法では, 残り深さ 3 以下のノードに対しても, futility pruning と同様な枝刈り判定を行っている.

両者共に, 使用するマージンは残り深さの探索による評価値の変化分よりも大きい値を用いている.

3.2 問題点

futility pruning は, マージンが定数であるが, これは乱暴な方法に見える. それは, 局面によって適切なマージンが異なっていると考える方が自然であろう. マージンが大きすぎた場合, 本来枝刈りとなるべき探索が枝刈りにならず, 通常の探索に近い状態になり, 余分な処理が増え効率が悪くなる. また, マージンが小さすぎた場合, つまり式 (2) を満たさない場合は本来なら枝刈りをせずに読むべき探索が枝刈りされて危険である.

Bonanza では適切なマージンを, 90%安全に枝刈りが出来る値を定数として与え, 使用している. しかし, 局面の状況によっては適切なマージンの値が変化している可能性がある. その場合, 90%安全であると設定した値でも, 実際に細かく見てみると局面によっては安全では無くなっている. かといって, そういった変動を考慮すると値の決定が難しくなる.

4 成功率の調査

futility pruning における定数で与えられた適切なマージンが, 局面状況により変動している事を確認するための調査を行った. 本稿で言う適切なマージンとは, ある程度安全な枝刈りであり十分探索コストも削減できるマージンの事である.

今回, 枝刈り成功率を指標にそのマージンがどの程度安全かを判断する. 枝刈りの成功というのは, あるノードが枝刈りと判定され実際の探索においても β カットとなっているかどうかの事である. マージン n の成功率は, n の時から∞までの回数の合計で計算する. それは, マージン n を設定した場合 n 以上で設定した時の分も含んだ成功率になるからである. CountSuccess(n) をそのマージンにおける枝刈りが成功した回数, CountAll(n) をそのマージンにおいて枝刈りが発生した回数として, 次の式で定義する.

$$\frac{\sum_{n}^{\infty} CountSuccess(n)}{\sum_{n}^{\infty} CountAll(n)}$$

成功率の調査の際には, 任意に指定した残り深さで実際は枝刈りはせず, 枝刈りしたと想定しそのまま探索を続行する.

例として, マージン 800 の成功率を見る時は, 800 以上の時でどの程度成功しているかを見る. つまり, マージン 0 の場合は全ての回数の和から成功率を求めている.

4.1 基本設計

本稿のプログラムは Extended Futility Pruning を用いる. この時残り深さ 3 以下で枝刈りを行う. 実装環境としては, 全幅探索を用いた α β 法で, 深さ優先の反復深化を使用し, 末端以降は静止探索として取り合い探索を行う探索アルゴリズムを使用する. 局面評価には TACOS[6] の評価関数を使用した. また, 枝刈り判定時に利用する material_value を TACOS では評価関数で処理が軽い駒割, 駒位置による評価を (LightValue) とした. 参考までに, 枝刈り条件とカット条件を下記に示す.

(枝刈り条件)$LightValue$ − マージン $\geq \beta$

(β カット条件)$LightValue$ − 残りの評価 $\geq \beta$

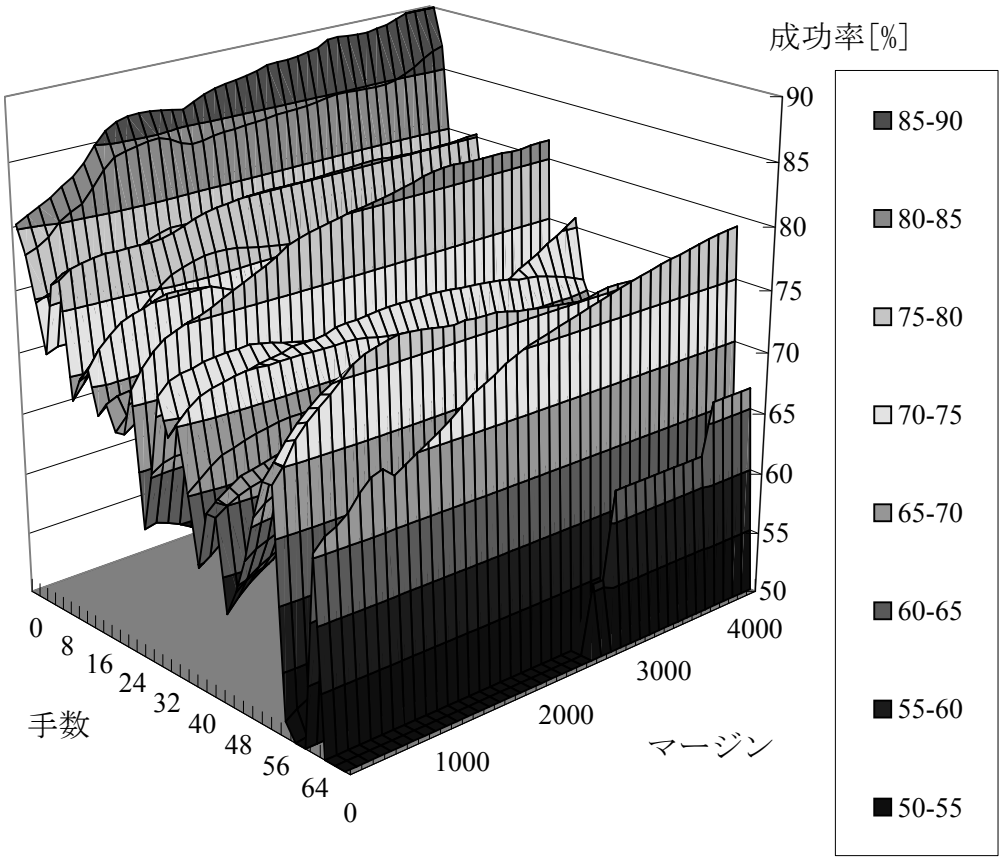

図 1: 手数, マージンに対する成功率の変化

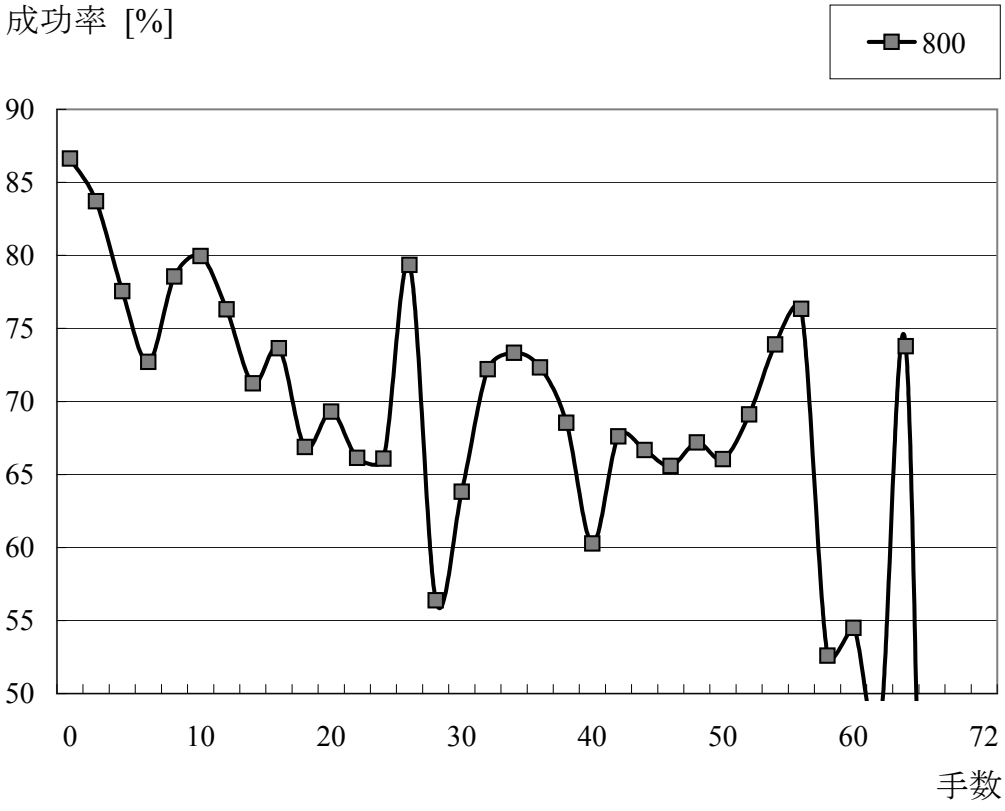

図 2: マージン 800 時の成功率の変化

4.2 成功率の手数による変動

以上の条件で, 初期局面から自己対戦を行い, 問題点で挙げた”局面の状況によっては適切なマージンの値が変化している” という問題を, それぞれのマージンでの成功率を見る事により確認していく. これは, 局面の進行状況を手数で考え, 成功率の変化を調べるためである. そこで, 実際に一手毎の探索時にマージンを 0〜5000 まで 50 毎に設定した時の, Extended Futility Pruning の成功率を調べた.

その結果, 初手から 51 手までの手数とマージンに対する成功率が図 1 の様になった. 図 1 より, どのマージンを使用しても 50〜60%程度しか成功率が無い場合と, 80%以上の成功率がある場合が存在する事がわかった. しかし, 中盤では高い成功率のマージンが存在していたとしても, 局面によっては成功率の低い場合が存在する. この事から, 局面の状況によっては最適なマージンが変化している事が確認できた.

また, 図 2 ではマージン 800 とした場合の局面変化による成功率の変化を調べた. これらの値は, 様々な局面から総合的に得られたデータから見て, 成功率が 75%となっている. これらの図からも, 手数が進むに従って成功率が低くなっているのが読み取れる.

5 動的なマージンを使う Extended Futility Pruning

5.1 動的なマージン

4 章の実験より, 局面の状況において適切なマージンは異なる事がわかった. 静的な値を用いると, 局面の状況変化によっては対応できないと言える. そこで, その問題を解決する方法として局面毎に適切なマージンを設定する必要があると考えた. しかし, 将棋の局面状況は大量に存在するので, 局面に適した静的な値を用いると局面に応じた量だけ定数が必要になるので現実的ではない. もし, マージンを進行度別に序盤, 中盤, 終盤とそれぞれ別の値を用意していたとしても, 図 1 から同じ進行度の中でも成功率が大きく異なる場合もあるので良い改良案とは言えない. そこで, これらを解決する方法として探索毎において適切なマージンを導出できれば, 細かく様々な局面に対応した値を使用できると考えた. そうする事により, 事前に統計を取り, 定数を大量に用意する必要も無くなる.

5.2 マージンの決定方法

動的なマージンの決め方として, 探索の途中までは適切なマージンを調べるために, 4 章と同じ方法で各マージンについて成功率のデータを収集する. この時集まった, 枝刈りをしたと仮定した回数 CountAll(n) の数が一定数以上になったら, マージンを決定する. これは, データ総数が少ない時に決めたマージンで安易に決めるのは危険と考えたからである. このデータ量は, 3 万, 5 万, 10 万の時でそれぞれ対戦させた結果, 5 万の場合での結果が一番良かったので, 5 万の場合を採用した. また, 枝刈りに使用するマージンは成功率が 75%以上の値とした. これは, 4 章の結果から 75%以上の値を用いようとすると, 使える状況が少なすぎると判断したからである.

5.3 動的なマージン決定例

実際に探索中に決定した例は図 3, 図 4 となった. この様に, 局面によっては同じ成功率のマージンでも, 値が変わる事がわかる.

また, 図 5 が初期局面から自己対戦させた時のある局面での, 決定に使用したデータの内容であり, この場合 75%の成功率があるマージンは 500 となる.

5.4 評価方法と結果

本稿のプログラムの評価として, 動的なマージンと静的なマージンで対戦を行い, 性能の評価を行った. その際, お互い 1 手 30 秒固定で探索を行った.

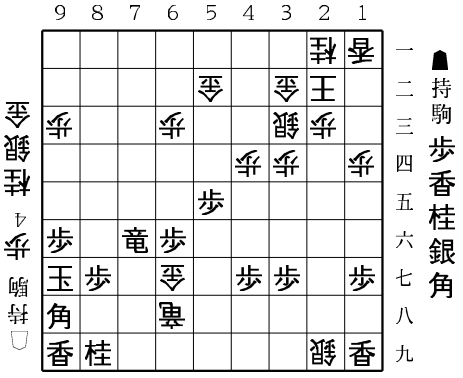

図 3: 動的マージン：500
指し手：△8 八銀打 (成功率 75%)

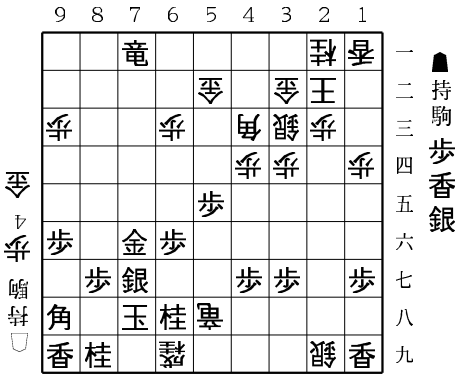

図 4: 動的マージン：300
指し手：△6 八成桂 (成功率 75%)

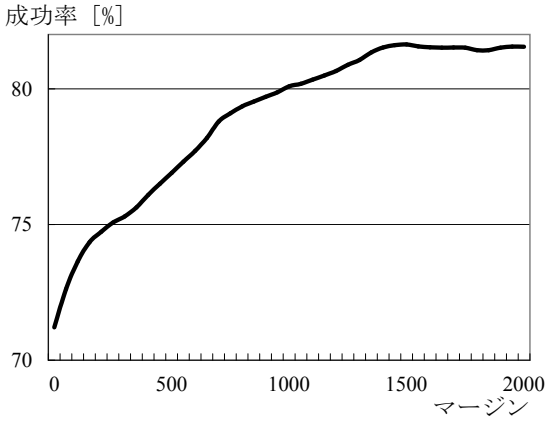

図 5: ある局面での収集したデータの内容

5.4.1 対戦相手について

対戦相手として, 静的なマージンの値 800 を用いたものを選んだ. これは, Bonanza で使用されているマージンの値を本稿のプログラムに適用した場合での値であり, 本稿で用いたプログラムではこのマージンでの成功率は 75%となった. この成功率は, 様々な局面を対戦した事により得られた成功率の合計から得られたものである. 対戦相手である静的な futility pruning を実装したプログラムと実装していないものを, 156 の局面から先後交代で対戦させた結果 “170 勝 139 敗 3 分” という結果が得られた.

5.4.2 結果と考察

性能評価として, 局面からの自己対戦を行った. まず, Extended Futility Pruning での最適 (成功率 75%) なマージンを動的に設定するプログラムを用意した. 75%に設定した理由として, 対戦相手の静的なマージンでの成功率と一致する様に, 動的に決める時も適切な成功率を 75%以上とした. この条件での対戦の結果, “175 勝 138 敗 1 分” となった. この結果より, 静的なマージンを使用した Extended Futility Pruning より, 動的にマージンを決定した方が, より強くなっている結果を示した. これにより, 動的にする事により静的なマージンでの枝刈りがより適切に行われる事が出来たと判断できる. また, この結果を受けて他の静的な前向き枝刈り方法も, 同様の方法でより良く出来るのではないかと考えられる.

6 今後の予定

今回, 残り深さ 3 での動的なマージンとなっているが, 残り深さ 2, 1 の場合も個別に動的なマージンを設定すると, 更なる精度の高い枝刈りになるのでは無いかと考えられる.

マージンの違いが, どの様な理由から発生するのか不明であるので, 解明していく.

現在の段階では, 自己対戦の結果のみでしか評価していないので他の評価もする.

選択的探索においての実装を検証していき, 有効性を確認する.

また, 今回の提案方法により導かれるマージンは, コンピュータ将棋における時間操作の指標となるのではないかと考えている. 例えば, マージンの値が低くなる局面は大量の時間を割いて考えても, 最善手が変化しない様な局面であり, 短い時間の探索の結果でも悪手が無いと考えられたとしたら, 逆に高く出る局面はそれだけ評価値が変化しやすいと考えられ, じっくり考えるべき局面として時間を多めにする必要がある, といった時間増減の判断基準に使えるのではないかと考えているので, 今後試していく.

参考文献

[1] D. F. Beal, “Experiments with the Null move”, Advances in Computer Chess 5, pp. 65-79, 1999.

[2] Y. Björnsson and Tony Marsland, “Multi-Cut α β-Pruning in Game-Tree Search”, Theoretical Computer Science, vol. 252(1-2), pp. 177-196, 2001.

[3] M. Buro, “ProbCut: An Effective Selective Extension of the Alpha-Beta Algorithm”, ICCA JOURNAL 18(2), pp. 71-76, 1995.

[4] E. A. Heinz, “Extended Futility Pruning”, ICGA JOURNAL 21(2), pp.75-83, 1998.

[5] J. Schaeffer, “Experiments in Search and knowledge”, Ph.D. Thesis, University of Waterloo, reprinted as Technical Report TR 86-12, 1986.

[6] 橋本 剛, “将棋プログラム Tacos のアルゴリズム”, アマトップクラスに迫る コンピュータ将棋の進歩５ (松原 仁編著), 共立出版, pp. 33-66, 2005.

[7] 保木 邦仁, “コンピュータ将棋における全幅探索と futility pruning の応用”, 情報処理学会誌, Vol. 47 No. 8, pp. 884-892, 2006.

[8] 松原 圭吾, “そっぽの指し手を排除する手法の提案”, 第 10 回 ゲームプログラミング ワークショップ, pp. 110-113, 2005.