経験的知識と類似局面を用いた証明数探索の効率化

島野 拓也[1,a)]　金子 知適[1,b)]

概要：本稿では, 詰碁の探索を効率化を目標とし, 証明数探索中での類似局面を用いたシミュレーション [1] の拡張した手法を提案する. 証明済みの部分木を P, それを利用して証明する局面の候補を Q と定義する. そして, その Q の対象を, P の兄弟ノードから探索木全体のノードに拡張する手法である. しかし, 対象範囲が広いため, シミュレーションの回数が膨大になってしまう可能性がある. そこで, 解手順, 深さ, 攻めの成功率などの経験的知識を活用して, Q の候補を絞る手法を導入し, その問題の解決を試みた. そして, ノード数の増減や, P と Q の木の不一致率などの指標を用いて, 様々な詰碁の問題で実験を行い, 提案手法の性能を評価した.

Enhancement of Proof Number Search by Using Empirical Knowledge and Similar Positions

Takuya Shimano[1,a)]　Tomoyuki Kaneko[1,b)]

Abstract: In this paper, we propose expansion methods of the simulation by similar positions in proof number search to improve the efficiency in solving Tsume-go problems. Let P and Q be the proven subtree and the options of proving positions by using the P. The method that expand the target of Q to nodes of whole the search tree from sibling nodes of the P. However, it is presumed that the number of times of simulation is enormous on account of broad in scope. We propose methods that reduce options of Q by using empirical knowledge of mating sequence, depth and success probability on account of solving it's problem. To show the effectiveness of the proposed methods, we conducted experiments on a lot of Tume-go problems by using indexes of the number of proven nodes and mismatch rate.

1. はじめに

近年, コンピュータによる詰碁ソルバーの研究が非常に進歩しており, 多くの詰碁の問題を高速に解くことが可能になってきた. 一方, 探索時間が非常に長くかかり, 解けないような難解な問題もあるため, 更なる効率化が必要と考える. また局所的な詰みに強くなれば, 実際の対局にも有利に働くと期待できる.

まず詰碁の研究背景として, 証明数や反証数という概念を用いた探索アルゴリズムや岸本らが導入した, 証明数, 反証数に加え, 類似局面を用いるシミュレーション [1] が成果を上げている. そして, UCT の RAVE [2] やチェスのヒストリーヒューリスティック [3] のような過去の探索で得た経験的知識を用いた手法も囲碁の研究では成果を上げていて, 今後, 詰碁の研究にも導入できる可能性が高い. このシミュレーションは, 類似局面では同じ解手順で詰む場合が多くあり, それを探索中に用いることで探索を高速化する狙いがある. 概要としては, 探索中に, あるノードが詰みと判明した場合, その兄弟ノードでシミュレーションを行う手法である. 詰碁では, 兄弟ノード以外にも類似局面が多数存在し, 同じような解手順になる場合があり, この手法の拡張が可能であることが考えられる. しかし, 類似局面を正確に定義できないため, 探索木の中から適当な類似局面を抽出し, シミュレーションを行うことは難しい問題がある.

本稿では, 詰碁の探索を効率化するため, シミュレーションの拡張に加え, 類似局面の抽出問題を過去の経験的知識

[1] 東京大学大学院総合文化研究科
Graduate School of Arts and Sciences, The University of Tokyo
[a)] shimano@graco.c.u-tokyo.ac.jp
[b)] kaneko@graco.c.u-tokyo.ac.jp

を用いて解決することを目標とする. 類似局面は石の形や解手順などの性質が似ている場合が多いため, ヒューリスティックな評価も似た性質を持つ可能性が高いと考えられる. そして, 精密な評価でなくとも局面を分類する基準ができれば, 類似局面を抽出するときの指標として十分役に立つと期待できる. シミュレーションを証明数探索の探索木全体に拡張する手法, それに組み込む P と Q の $depth$ の順序関係を用いた類似局面の抽出手法, そして, 攻めの成功率を評価に用いるヒューリスティック [5] を用いた手法を紹介する. そして, これらの手法の性能を調査するため, 詰碁の問題を用いて比較実験を行った.

2. 関連研究

2.1 証明数探索

証明数探索とは証明数, 反証数という概念を用いて探索をする手法の一つであり, 証明数の概念と証明数探索は Allis が導入した [4]. 証明数は, 各ノードにおいて, そのノードが詰みであることを証明するために, 反証数は, 不詰みであることを証明するために, 展開しなければならない子ノードの最小の数を意味する. ノード n の証明数, 反証数はそれぞれ $pn(n)$, $dn(n)$ と記し, 以下のように定義する [6].

- n が先端ノード
  - 詰み
    - $pn(n) = 0$
    - $dn(n) = \infty$
  - 不詰み
    - $pn(n) = \infty$
    - $dn(n) = 0$
  - 不明
    - $pn(n) = 1$
    - $dn(n) = 1$
- n が内部ノード
  - OR ノード
    - $pn(n) = min\{pn(n_i), n_i \in children(n)\}$
    - $dn(n) = \sum_{n_i \in children(n)} dn(n_i)$
  - AND ノード
    - $pn(n) = \sum_{n_i \in children(n)} pn(n_i)$
    - $dn(n) = min\{dn(n_i), n_i \in children(n)\}$

上記の定義を用いてノードを展開していき, ルートノードの証明数もしくは反証数の値が 0 となるまで最良優先探索を行う手法である. 証明数 (反証数) の値が小さいノードほど証明 (反証) しやすいと考えられ, それを優先的に探索しているので効率化が期待される. 詰碁の研究では, これを進化させた df-pn [7], df-pn$^+$ [8], λ df-pn [9] なども成果を上げている.

2.2 類似局面を用いたシミュレーション

類似局面を用いたシミュレーションの手法は河野 [10] の詰将棋の研究に導入された. priority と simulation の 2 つの概念を導入して探索を行う手法である. 詰将棋の局面と指し手に priority を半順序集合で与え, simulation は, その priority を用いて枝刈りを行う. priority の順序関係が $P > Q$ となるような類似局面 P と Q があったとき, P の後に Q の探索を行う. 仮に P の解手順に従い, Q を詰めることができるならば, Q の探索を削減することが可能である. 河野の詰将棋での priority は, 受け側の玉と合駒の距離と, 王手に対する玉の逃げる手としていた. この priority の定義はゲームによって変化し, simulation も priority に大きく依存するため, この手法を適用するゲームによって, 適当な priority を設定することが重要であると考えられる.

2.3 証明数探索における類似局面を用いたシミュレーション

河野 [10] のシミュレーションを岸本ら [1] が詰碁の証明数探索に応用した. 証明済みの局面 P の解手順を用いて, 未探索の局面 Q の探索を行う手法である. 探索中にあるノードが証明されたとき, その兄弟ノードも似た局面で, 証明されたノードと同じ解手順で証明できる可能性が高いと予想される. 例えば, 応手が死活に全く関係ない手の場合は解手順が変わることはなく, この手法が適用でき, 探索を大幅に省くことが可能になる. 図 1 のように, 局面 P の解手順を用いて, 局面 Q, Q', Q'' とそれぞれ試していき, シミュレーションが成功したものはその手順を Q の解とし, 解手順の不一致で失敗したものは, シミュレーションを行わなかったものとして, 通常の証明数探索を行う. シミュレーションが失敗してた場合でも, 図 2 のようにシミュレーションの効果で一部の部分木の pn, dn の値が更新されている可能性があり, 改めて証明数探索を行う場合に, 一から行うよりも探索すべき箇所が少なくなる場合がある.

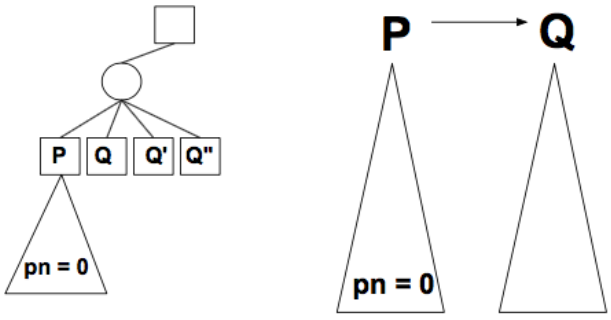

図 1 シミュレーション

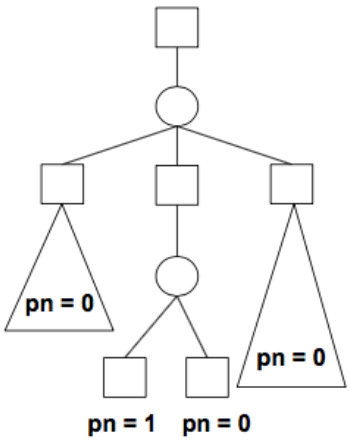

図 2 シミュレーションが失敗した場合

2.4 攻めの成功率を用いたヒューリスティック

攻めの成功率を用いたヒューリスティック [5] は, 証明数探索で証明数を計算するときに導入する手法の一つである. 探索中に各座標における着手回数と攻めが成功した回数を記録し, 次に探索を行うときに, 過去に攻めが成功した割合の高い候補手を優先的に選択する. 過去に攻めが成功している場合, 今回も同じ手で成功する可能性が高く, 有望な候補手であると予想される. 有望な手の可能性が高いノードを優先的に探索することによって, 探索の効率化が期待できる. 定義は以下に示す.

- 攻め側のノード n の成功率

$$SuccessProbability(n) = \frac{SuccessTimes(n)}{MoveTimes(n)}$$

- 攻め側のノード n の失敗率

$$FailureProbability(n) = 1 - SuccessProbability(n)$$

- $pn(n)$ の更新

$$pn(n) = pn(n) * FailureProbability(n)$$

上記の定義に従い証明数の更新を行う. 攻めが成功する確率が高いほど証明数の値は 0 に近づくので, 狙い通りの探索が期待される.

3. 提案手法

本稿の目的は, シミュレーションを拡張し, 汎用性を高めて, 詰碁の探索を効率化することである. 具体的には, シミュレーションに適した局面を探索木全体から抽出できるようにし, シミュレーションの適用範囲を証明数探索中の探索木全体に拡大する手法を提案する. 例えば, 図 3 のようなルート局面を考えたとき, P の黒の 5 手目からの手順と Q の黒の 3 手目からの手順は, 兄弟ではないノードではあるが, ほとんど同じ解手順で詰めることができる. この例のように, 受け側の応手が解手順にほとんど影響を与えない場合が詰碁の探索には存在するので, 探索木全体でのシミュレーションの効果は期待できる.

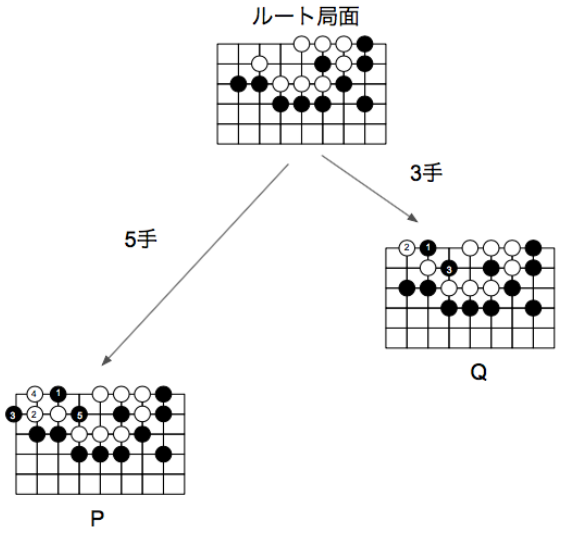

図 3 兄弟ではないノードでのシミュレーション

提案手法の名称と流れは, 以下のように定義する.

(1) AddMoves

(a) 探索木全体の証明数, 反証数の更新後, その探索木全体の中から $pn = 0$ となる部分木をシミュレーションに用いる P とする.

(b) 証明済みの部分木 P が存在する場合, それを利用して証明する候補となる局面 Q を以下の条件に従い, 探索木全体の中から全て選択する.

- Q が OR ノードである.
- Q の pn, dn の値が共に 0 でない.
- P のルートからターミナルまで $pn = 0$ となる $path$ を解手順と定義し, その解手順の $move$ が Q の局面で全て合法手である.
  これは, Q に選ばれる時点で P の解手順に従えないものは除外する目的である. 本当は, Q の子孫ノードも全て合法手か調べたほうがより良いが, 毎回各ノードに対して子孫ノードまで確認するのは非効率だと考えられ, 最小限の労力で判定するために, Q のみの局面とした.

(c) 証明済みの P を用いて Q に選ばれたノードを全て順番にシミュレーションする.

これらの条件のみで探索の効率化ができれば良いが, 問題によっては Q がほとんど絞られず, 膨大な回数のシミュレーションをしなければならない可能性があり, ここから先は更に Q の数を絞るための提案手法を紹介する.

(2) AddDepth

この手法は, AddMoves に P と Q の深さの順序関係を導入する.

- Q の選択対象を, $P.depth < Q.depth$ を満たすノードとする.
  この手法は, 解まで残りが少なそうなノードを優先的にシミュレーションすることにより, シミュレーションが失敗した場合のコスト削減を期待している. ただし, 早い段階でシミュレーションが成功する問題に関しては, この条件を付け加えることによって不利になってしまう場合も考えられる.
- 逆に, Q の選択対象を, $P.depth \geq Q.depth$ を満たすノードとする.
  この手法は, 先程とは逆で, 未展開のノードが多い浅い場所のノード (解を導くまでに探索しなければいけないノード数が多い) で, シミュレーションが成功すると削減効果が高いと考えられる. しかし, $P.depth$ が非常に深い場合, 木全体のほとんどが対象となってしまうため, AddMoves の選択範囲を絞れない可能性がある.

(3) AddSuccess
この手法は, AddMoves に攻めの成功率のヒューリスティック [5] を導入する.
(a) 既存研究と同様に, 探索中に各ノードに着手場所, 着手回数, 攻めが成功した回数を記録し, それらの情報をもとに AND ノード下で $pn = 0$ のときに木全体の各ノードの成功率の値を更新する.
(b) AddMoves で選ばれた Q の候補の中から, 成功率の高いノードを優先的にシミュレーションを行う.

4. 実験

4.1 実験設定

基本の詰碁 初段・1・2 級 [11] の問題とオリジナルの問題を用いて, 証明数探索と提案手法の比較を行った. 証明数探索と提案手法で同じ詰碁を解いた場合のノード数の違い, 証明された P と Q の解手順の不一致の割合 (不一致率), 不一致率とノード削減率をそれぞれ比較した. 探索に必要とするノード数は少ないほうが効率的であり, ノード数が削減ていた場合, それがシミュレーションの効果だと確かめるために不一致率を観察した. 不一致率が低い問題ほどシミュレーションの効果を期待できる. また, 不一致率とノード削減率にどの程度影響するのかを確かめるために, それらの関係性も調べた.

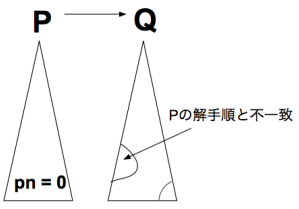

図 4 解手順の不一致

4.2 実験

4.2.1 任意の局面 P, Q を用いた比較

まずは, 提案手法に見込みがあるか検証するために, 詰碁の探索で出現するノードの中から, シミュレーションが適用可能な P と Q を任意に指定して, シミュレーションを行った. そして, 証明数探索のみで Q の探索を行ったときと, シミュレーションを併用した場合とで, 比較実験を行った. 証明数探索のみに比べシミュレーションを併用した場合の方が, 探索に必要としたノード数が少なく, P と Q の解手順の不一致率が低ければ, 提案手法に見込みがありそうと期待できる. 実験には 20 題の詰碁を使用し, 証明数探索とシミュレーション併用時のノード数の増減, 問題ごとの P と Q の不一致率, 不一致率とノード削減率の関係を検証した.

図 5 では, 横軸に証明数探索のノード数, 縦軸にシミュレーション併用時のノード数として, ノード数の増減を比較した. この結果, シミュレーションが適用可能な P, Q を選んだ場合, 証明数探索のみで探索を行うよりもシミュレーションと併用させた方が, ほとんどの問題でノード数が削減されている.

また, 図 6 では, 横軸に問題番号, 縦軸に不一致率として, 各問題の不一致率を検証した. この結果, ほとんどの問題の不一致率が 10%以下と低くなっており, ノード数削減の要因は, シミュレーションの効果である可能性が高いと考えられる. これらのことから, 本稿の提案手法は, 詰碁探索に効果的であると期待できる.

更に, 不一致率とノード削減率の相関を調べてみたが, 図 7 のように, 不一致率が 10%以下の場合はほとんど関係無いと考えられ, シミュレーションによるノード削減効果は, 不一致率よりも P と Q の局面の性質に大きく依存すると予想される.

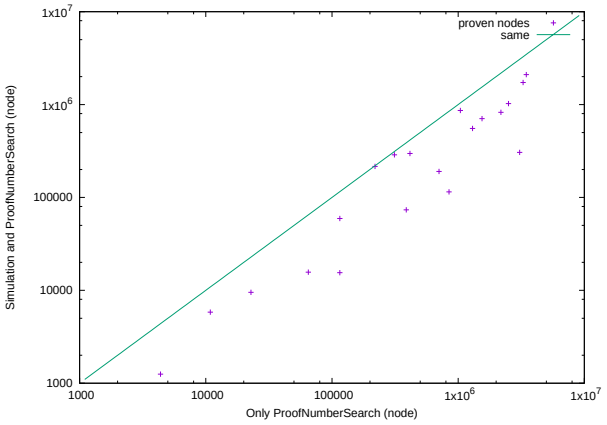

図 5 証明数探索とシミュレーション併用時のノード数の比較

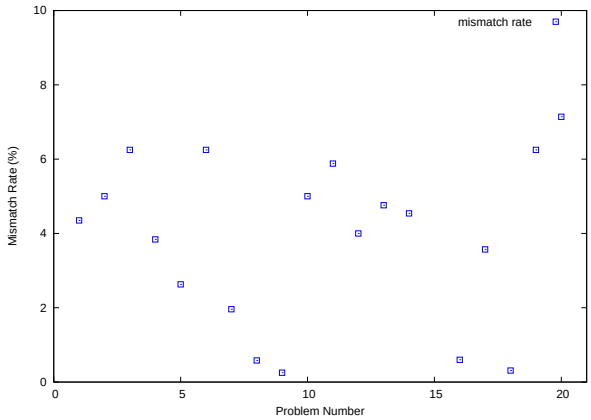

図 6 各問題の不一致率

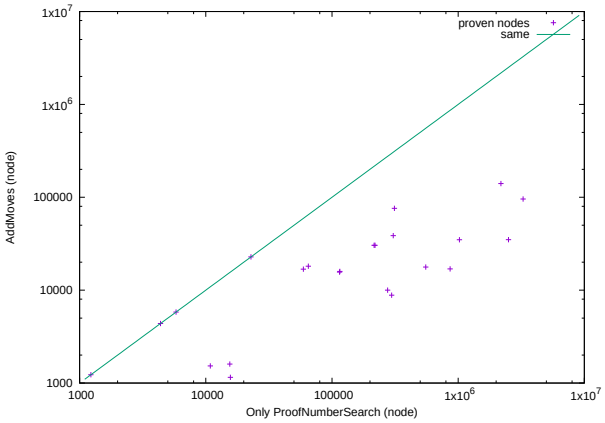

図 8 証明数探索と AddMoves のノード数の比較

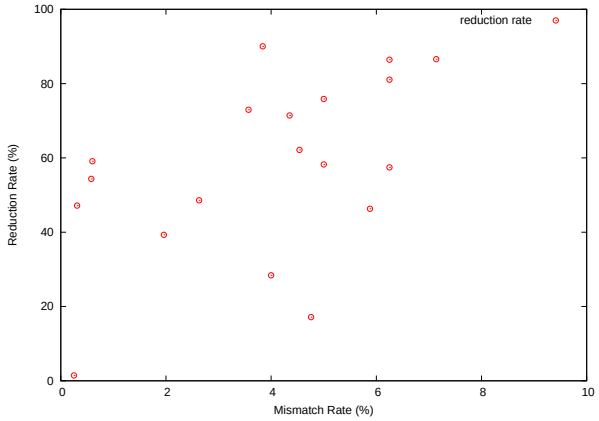

図 7 不一致率とノード削減率の関係

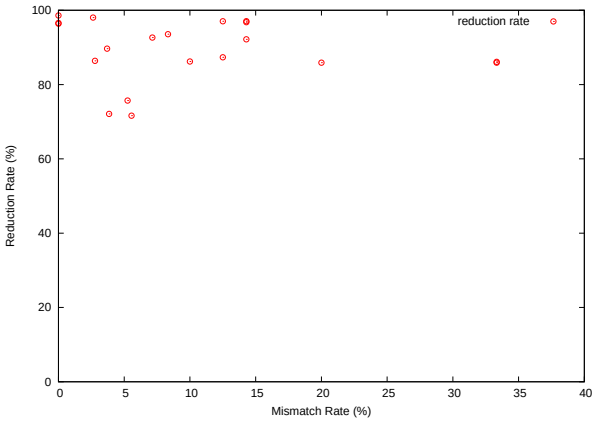

図 9 AddMoves における不一致率とノード削減率の関係

4.2.2 AddMoves の比較

ここでは, 詰碁の問題 40 題で, AddMoves の性能を評価する. また, ここから先は, 同じ詰碁 40 題の問題を使用する.

まずは, 図 8 のように, 横軸に証明数探索のノード数, 縦軸に AddMoves のノード数として, ノード数の増減を比較した. 結果は, 40 題中 20 題が削減に成功, 4 題が同一, 16 題が AddMoves の探索が終わらない, となった. テストした半分は効果があったが, 逆にシミュレーションの数が膨大で探索が終わらないものも 40%あったため, この手法は詰碁の問題の性質に大きく左右されてしまう欠点があると考えられる.

次に, ノード削減が成功した問題の不一致率とノード削減率の関係を検証した. 図 9 のように, この手法が成功した問題の不一致率は全体的に低く, ほぼ全ての問題に対して, ノード削減率が 70%以上の高い値となった. 探索が終わらない問題を解決することができれば, この手法は詰碁探索において, 有効ではないかと考えられる.

4.2.3 AddDepth の比較

ここでは, *depth* を利用した AddDepth の性能を評価する.

- $P.depth < Q.depth$ の場合 :
  AddMoves で探索失敗した問題のうち, 3 題は成功した. ノード削減率はそれぞれ, 約 83%, 約 93%, 約 94%, であったため, 証明数探索よりは効率的だと考えられる. しかし, 16 題中 3 題しか改善されていないかつ, AddMoves で解けた問題で AddDepth では解けない問題が複数発生した. 解答率が AddMoves 以下であり, AddMoves と AddDepth のノード数の増減も変化が無かったため, この手法はあまり有効ではないと考えられる.
- $P.depth \geq Q.depth$ の場合 :
  探索に成功した問題, ノード数は AddMoves と全く同じだった. 提案手法の章でも記載したが, 今回のテストケースではシミュレーション対象の $P.depth$ が非常に深く, Q の選択対象範囲がほとんど探索木全体であり, AddMoves から絞れていない可能性が考えられる. テストケースを増やせば結果は変わる可能性があるが, こちらもあまり見込みがあるとは考えにくい.

以上のことから, P と Q の *depth* の順序関係がシミュレー

ションに影響を与える可能性は今のところあまり高くないと予想される.

4.2.4 AddSuccess の比較

ここでは, 攻めの成功率を利用した AddSuccess の性能を評価する.

図 10 のように, 横軸に AddMoves のノード数, 縦軸に AddSuccess のノード数として, ノード数の増減を比較した. ノード数は, AddMoves に比べ, やや不利である. AddMoves で解けなかった問題は 16 題中, 3 題解決できた. 逆に AddSuccess のみで失敗してしまった問題もあったが, 解答率は AddMoves より若干高かった. 今回の実験では期待通りの結果が得られなかったが, 多少なりとも成功率がシミュレーションに影響を与えている可能性があることから, 上手く利用すれば性能向上に繋げる一つのヒューリスティックな評価になるのではないかと期待したい.

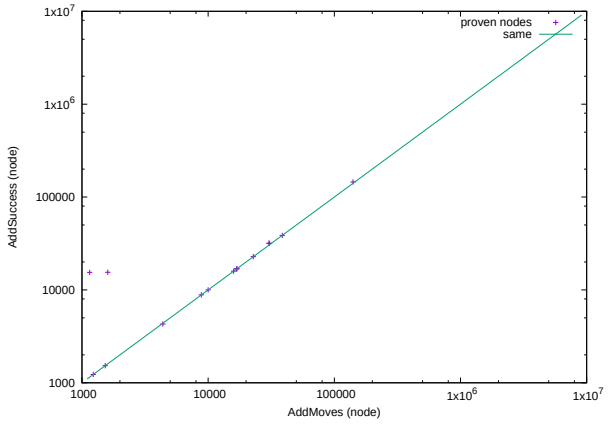

図 10 AddMoves と AddSuccess のノード数の比較

5. おわりに

今回の実験では, シミュレーションを証明数探索の木全体に拡張する手法は証明数探索のみに比べ, 多くの問題でノード数を 70%以上削減することができた. ただし, 40%の問題では、解ける問題が解けなくなるので, 改善の余地が残る. これらのことから, 今回のテストケースの半数では効果を上げているので, 失敗した問題の原因を解決し, 対処することができれば, シミュレーションの拡張自体は詰碁の探索において有効であると考えられる. しかし, P と Q の *depth* の順序関係や成功率を用いた手法は期待通りの結果を得ることができなかった. 今回用いた priority が詰碁のシミュレーションにおいて適切ではないこと, シミュレーションの順番による影響が少ないこと, テストケースで設定した詰碁の問題がシミュレーションに適していないことなどが原因として考えられる. 例えば, priority の件は, 指し手の成功率が高いノードとシミュレーションが適用できる可能性が高いノードは必ずしも近いとは限らない. シミュレーションは P の解手順に依存するが, 成功率は手順を問わないため, 不一致になりシミュレーション失敗になる場合も多いはずである. また, AddMoves にヒューリスティックを組み込む前後で, ノード数にほとんど変化が無いことから, シミュレーションをする順番はあまり影響しない可能性があると考えられる. さらに, 詰碁の問題に関しては, 今回の実験では, 特に条件を与えなかった. しかし, 創作詰碁の解手順は一通りであり, 死活に影響が無いようなダメなどが省かれている場合が多い. シミュレーションは応手の合法手に無駄が多いほど効果が高いが, このように無駄が少ない問題はシミュレーションの適用できる場面が減り, 効果が発揮しにくい可能性がある. よって, このような創作詰碁よりも, 詰みの手順が複数通りあり, 応手も無駄な手が多い実践的な死活問題で, 提案手法を試してみるとまた違う結果になる可能性が高い. 今後はこのシミュレーションの汎用性を高めるため, 問題とシミュレーションの相性や priority がシミュレーションに与える影響を調べながら, 今回上手く行かなかった, 適当な類似局面の抽出に尽力していきたい.

参考文献

[1] A. Kishimoto and M. Mueller. Df-pn in go: An application to the one-eye problem. In Advances in Computer Games 10, pp. 125-141. Kluwer Academic Publishers, 2003.

[2] 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.

[3] Jonathan Schaeffer. The History Heuristic and Alpha-Beta Search Enhancements in Practice. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 11, No. 11, pp. 12031212. ISSN 0162-8828 (1989).

[4] L.V. Allis, M. van der Meulen, H.J. van den Herik, Proof number search, Artificial Intelligence 66 (1) (1994) 91-124.

[5] 島野 拓也, 金子 知適: 証明数と経験的知識を用いた探索の効率化, 第 20 回ゲームプログラミングワーク ショップ, 107-112 (2015).

[6] 長井, 今井. df-pn アルゴリズムと詰将棋を解くプログラムへの応用. 情報処理学会論文誌, 43(6): 17691777, 2002.

[7] Nagai, A. and Imai, H.: Proof for the Equivalence Between Some Best-First Algorithms and Depth-First Algorithms for AND/OR Trees, KOREA-JAPAN Joint Workshop on Algorithms and Computation, pp. 163-170 (1999).

[8] A.Nagai and H.Imai. Application of df-pn$^+$ to othello endgames. In Game Programming Workshop in Japan '99, pp. 16-23, Oct. 1999.

[9] Yoshizoe,K., Kishimoto, A. and Mifller, M.: Lambda depth-first proof number search and its application to Go, 20th International Joint Conference on ArtificiaI Intelligence(IJCAI-07), pp. 2404-2409 (2007).

[10] Kawano, Y. (1996). Using similar positions to search game trees. In Nowakowski , R. J., editor, Games of No Chance, volume 29 of MSRI Publications, pages 193-202. CambridgeUniversity Press.

[11] 石田 芳夫. (2002). 囲碁 基本の詰碁 初段・1・2 級 (ポケット版・囲碁シリーズ). 成美堂出版.