動的候補手拡大を用いたDf-pnアルゴリズム

美添一樹†

証明数探索は証明数/反証数の大小を基準として探索順序を決定する。証明数が小さいことは、探索すべき節点が少ないことを意味し、多くの問題では実際に良い手であることが多い。見込みの高い手から探索することは探索アルゴリズムの性能を考慮するうえで非常に重要な要素である。しかし、見込みの無い候補手が多数存在する場合、証明数/反証数の大小が基準として相応しくないことがある。そのような場合に、証明数/反証数の計算に一部の候補手の証明数/反証数のみを合計する手法を適用することを提案する。囲碁の19路盤における捕獲問題を対象に実験を行った結果、候補手をヒューリスティックに限定した場合にはこの手法は効果が無かったが、全合法手を対象に探索を行った場合には本手法は大きな効果を発揮することを示した。結果として、全合法手を対象に探索を行うことにより結果の正しさを保証しながら、探索速度は候補手をヒューリスティックに限定した場合の平均して3倍程度に抑えることができた。

Df-pn Algorithm with Dynamic Widening

KAZUKI YOSHIZOE†

Proof number search expands nodes with smaller (dis-)proof numbers. Nodes having (dis-)proof numbers are expected to need less number of nodes to be (dis-)proved, and in many cases these are in fact promising nodes. In other words, (dis-)proof number is a good measure for move ordering. However, when there are many number of unpromising child nodes, (dis-)proof numbers are not always a suitable measure for move ordering, because unpromising nodes temporarily increases (dis-)proof numbers. In such cases, we propose to use only part of the child nodes (in stead of all child nodes) upon calculating (dis-)proof numbers. Experiment was done for capturing problems of Go on 19x19 boards. The result shows that this method was not effective if combined with heuristically limited candidate move generator, but was highly effective if all legal moves were generated.

As a result, we could guarantee the correctness by generating all legal moves, while maintaining the speed performance approximately 3 times slower compared to searching based on heuristic move generation.

1. はじめに

AND/OR木探索には、証明数を用いたアプローチが有効であることが知られている。Df-pn[7]は証明数と反証数を用いた深さ優先探索アルゴリズムである。

Df-pnの研究は近年さかんに行われている。証明数の計算方法に工夫をすることにより、サイクルのあるグラフ上の探索を行う研究として、長井らによる親節点をたどることにより合流を検知する研究[8]、サイクルやGHI問題に対応した岸本らの研究[3),4)]等がある。

上記とは異なるアプローチとして岡部によって、経路分枝数を用いた探索アルゴリズムが提案されている[12)]。これは単純なアルゴリズムではあるが、合流が多い難解な詰将棋に対して、Df-pnと同等以上の性能を発揮している。

本論文では、証明数/反証数の計算について動的候補手拡大(Dynamic Widening)を導入し、Df-pnの挙動にどのような影響を与えるのか実験を行った。候補手拡大(Widening)とは、徐々に探索対象の候補手を増やすことにより、全体としての探索を高速化する手法であり、詰将棋に置いて合駒の探索を遅らせる、あるいは囲碁の捕獲探索に置いて候補手の脅威度を徐々に増大させる[2)]などの例がある。

通常は、OR節点の反証数は全ての子節点の反証数の合計として、反対にAND節点の証明数は全ての子節点の証明数の合計として、定義される。本論文で提案する手法は、反証数の計算をする際に、証明数/反証数の小さい子節点のみ選び出し、それらのみを合計する方式である。

† 中央大学 研究開発機構, Chuo University, Research and Development Center

囲碁の19路盤における捕獲探索を対象に実験を行った結果、候補手をヒューリスティックに限定した場合にはこの手法は効果が無かったが、全合法手を対象に探索を行った場合には、本手法は大きな効果を発揮することを示した。

2章では関連研究について述べる。3章で提案手法について解説し、4章で実験結果の分析を行う。最後にまとめと今後の課題を示す。

2. 関連研究：証明数探索の概要

二人プレイヤーゲームのゲーム木はAND/OR木であると考えられる。ここで、証明が先手(OR節点に対応する)の勝ちを示すものとし、反証が後手(AND節点に対応する)の勝ちを示すものとする。Allisは探索中のAND/OR木において、証明/反証の難しさを示す尺度として証明数及び反証数が有効なことを示した[1]。ある節点nの証明数$pn(n)$はnを証明するために証明しなければならない葉節点数の下限として定義される。逆に、節点nの反証数$dn(n)$は、nを反証するために反証しなければならない葉節点数の下限として定義される。既に証明された節点nの証明数/反証数は$pn(n)=0$、$dn(n)=\infty$、逆に既に反証された節点nの証明数/反証数は$pn(n)=\infty$、$dn(n)=0$と定義される。未展開の節点には、通常$pn(n)=dn(n)=1$と割当てられる。ある内部節点nの子節点を$n_1,\cdots,n_k$とすると、OR節点の証明数/反証数は以下の通り。

$$pn(n)=\min_{i=1,\cdots,k} pn(n_i), \qquad dn(n)=\sum_{i=1}^{k} dn(n_i).$$

AND節点の証明数/反証数は以下の通り。

$$pn(n)=\sum_{i=1}^{k} pn(n_i), \qquad dn(n)=\min_{i=1,\cdots,k} dn(n_i).$$

図1に以上の計算方法を図示する。

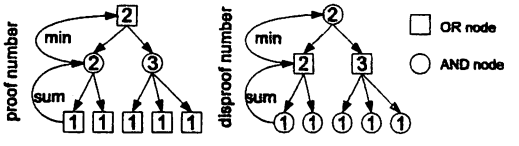

図1 証明数/反証数の計算

証明数探索(Proof-number search :PNS)は最良優先探索の1種である。各節点の証明数/反証数を計算する。根節点から順に、各OR節点では証明数が最小の子節点、各AND節点では反証数が最小の子節点を辿り、たどり着いた葉節点を展開する。展開の結果影響を受ける節点について証明数/反証数を更新する。以上の手順を、根節点が証明又は反証されるまで繰り返す。

Df-pn(Depth-first proof-number)探索[7]は証明数探索(PNS)を深さ優先探索に変更したものである。証明数探索と比較して内部節点の再展開が行われる頻度を軽減し、比較的小さな遷移表を用いて実行可能であるためメモリ効率も良い。Recursive Best-First Search[6]と似た方式で証明数/反証数の閾値を段階的に増加させることにより深さ優先探索として動作させている。

Df-pnは詰将棋[7]や詰碁[5]、チェッカーの解明[9]などで大きな成功を収めた。

3. 提案手法

提案する手法について述べる。

3.1 解決したい課題

前述のように証明数探索は、証明数/反証数が小さい節点から探索を行う。節点が展開された直後は、証明数/反証数は候補手の数に比例するが、探索中に大量の候補手が生じる局面で、証明数/反証数が一時的に増大することにより、探索順序が悪影響を受けることがある。

詰将棋などでは、候補手が少ない節点から探索することが良い場合が多いため、証明数を用いたアルゴリズムは適していると言える。しかし例えば囲碁の捕獲探索では、理論的に候補手を限定することは困難である。その結果、大量の候補手を生成したがそのほとんどが簡単に反証されてしまうような場合は、候補手生成直後の証明数の差に着目することが正しいかどうか疑問である。

たとえばOR節点で、先手の候補手が大量にある場合、反証数は増大する。特に、ほとんどの候補手が見込みが無いと思われる場合(囲碁の他、詰将棋で合駒がある局面など)、展開された直後は単純に候補手の数の違いが証明数/反証数の差になるため、節点の有望さの基準としては証明数/反証数は相応しくない。

そのような無駄な手が明らかに事前に分かる場合、それらの候補手の生成を遅らせることによって上述の問題を回避する工夫が行われている。たとえば詰将棋において、合駒の候補手は非常に多いが、ほとんどが無駄な手である可能性が高い。そのため、多くの詰将棋プログラムでは、合駒以外の手を先に全て探索した後で合駒の手を探索するという手法が用いられている。(そもそも合駒の手を生成しないため、当然、証明数/反証数の計算にも含まれない。)詰碁においても、パスはほとんどの場合無駄な手であるため、パス以外の手の探索が終わるまではパスは探索しないという手法がある。

3.2 動的候補手拡大(Dynamic Widening)

通常の証明数探索では、OR節点の反証数は子節点の

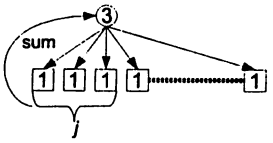

図2 Dynamic Widening

反証数の合計と定義される。

ここで提案する手法は、図2のように、本来全ての子節点の証明数を合計するところ、証明数が小さい方から j 個の子節点を選択し、選択した子節点の反証数を合計する手法である。

ある内部節点 n の子節点を $n_1, \cdots, n_k$ とすると、OR 節点では証明数の小さい順番 (AND 節点では反証数の小さい順) に子節点をソートしておき、最初の j 個だけを計算に用いる。つまり、OR 節点の証明数 / 反証数は以下の通り。

$$pn(n) = \min_{i=1,\cdots,j} pn(n_i), \qquad dn(n) = \sum_{i=1}^{j} dn(n_i).$$

AND 節点の証明数 / 反証数は以下の通り。

$$pn(n) = \sum_{i=1}^{j} pn(n_i), \qquad dn(n) = \min_{i=1,\cdots,j} dn(n_i).$$

選択する子節点の数 j をどのように設定すべきか、いくつか手法が考えられるが、本論文では以下の手法を実装し、実験を行った。

- 定数で固定 (top$n$)
- 子節点中の全体の上位 $1/n$ 個までの子節点 (rate$n$)

それぞれ、対象とする問題の性質によって利害得失があると考えられる。

特に、候補手の証明数 / 反証数の初期値を調整する手法 (Df-pn+)[7] と組み合わせることにより、最初は見込みのある手についてだけ証明数 / 反証数を合計することになり、その後徐々に見込みの薄い手も計算に用いるようになる。このように候補手を徐々に増やす手法が widening と呼ばれるが[2]、提案する手法は動的に widening を行うため、動的候補手拡大 (Dynamic Widening) と呼ぶことにした。

反証 / 証明された手は証明数 / 反証数が無限大となるのでソートによって後ろに回る。なので探索順序は変わるが、結果が正しいことは保証される。毎回ソートするのでその分時間が余計にかかるが、上位 j 位までをソートすればよく、また、2回目以降は順序が激しく入れ替わることは無いのでソートにかかる時間は少ないと期待される。

4. 実験結果及び解析

4.1 実験条件

問題は囲碁の捕獲探索を用いた。全て、19路盤の問題である。囲碁の捕獲探索では、詰将棋などと違い候補手を理論的に限定することが難しいという性質がある。

問題セットは Thomsen により web 上で配布されている問題集[11] を用いた。この問題集は、複数の連のどちらかを捕獲せよという問題を含んでいるので、単一の連の捕獲を目的とした問題のみ (全 434 問) を選んで実験対象とした。この問題集はコウが正解に関与しない問題のみが選ばれているため、今回の実験ではコウの有無は無関係である。

探索節点数は 200,000 節点を上限として実験を行った。また、捕獲探索対象の連のダメ数が 5 に達したら、捕獲をあきらめるように設定した。

4.2 候補手生成

候補手生成の方法を、全候補手を生成する方法 (ただし、石を取れる場合、及びアタリから逃げる場合には限定する) と、ヒューリスティックに候補手を限定する方法の二つを用意し、比較した。

ヒューリスティックの内容は、Thomsen による surrounding stones[10] のアイデアを用い、5-surrounding stones までの連のダメ、ダメを共有する味方の石、元ツギ、さらにそれらの一路周りなどを用いた。

図3と図4はそれぞれ、今回の実験で生成されたヒューリスティックな候補手と全合法手を示す。図中の a でマークされた連が捕獲探索の対象の連である。盤上に数値のある箇所が候補手として生成されている。数値の大きさは Df-pn+[7] で用いられる証明数 / 反証数の初期値に対応している。有望な候補手の証明数 / 反証数は小さく、見込みの薄い候補手の証明数 / 反証数は大きく設定することにより有望な手から先に探索を行う。実際の初期値は、図の数値を n とすると、OR 節点では証明数が $2^{(n-1)}$ 反証数が 1、AND 節点では証明数が 1 反証数が $2^{(n-1)}$ とした。

4.3 結　　果

解答能力の比較を表1に示す。

pass が正解数、miss が誤解答の数、exceed は 200,000 節点の探索では解答が得られなかったことを示す。解答数が最大なのはヒューリスティックな候補手を用いた場合、かつ候補手拡大を行わなかった場合の 283 問であるが、この場合は 9 問の誤解答を含んでいる。

探索中に生成された候補手数の平均を表2に示した。捕獲対象がアタリの状態にあるときは攻め側の候補手は

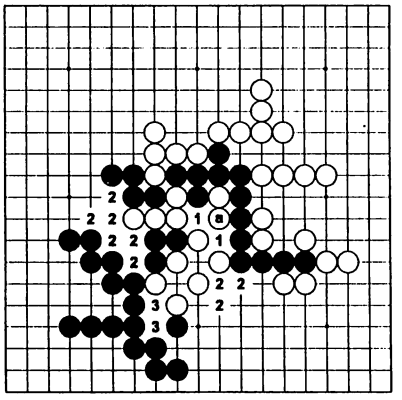

図3　ヒューリスティックな候補手の例

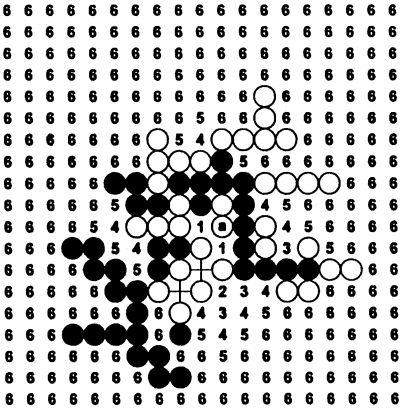

図4　全合法手の例

候補手		pass	miss	exceed
全合法手		31	0	403
全合法手	top20	108	0	326
全合法手	top10	138	0	296
全合法手	top5	139	0	295
全合法手	top2	8	0	426
全合法手	rate2	37	0	397
全合法手	rate5	50	0	384
全合法手	rate10	81	0	353
全合法手	rate20	124	0	310
全合法手	rate40	131	0	303
heuristic		283	9	140
heuristic	top20	253	9	172
heuristic	top10	205	8	221
heuristic	top10	166	9	259

表1　正解数

全候補手	308.4
heuristic	19.14

表2　平均候補手数

1個、受け側も多くの場合1個に限定されるため、平均の候補手数が減少している。

全候補手を生成した場合は、解答を得られた場合には全て正しい解答を返しているが、解答能力は最大で139問である。候補手拡大の方法については、top5とrate40であまり差が無い。rate40だと平均して10個を下回る子節点を合計し、top10やtop5とほぼ同等と思われる。実際に解答能力もtop5が139、rate40が131と大差ない。

また、上位jを選ぶ手法の中では、上位5位(top5)と上位10位(top10)が良いが、逆に上位2位(top2)は性能が急激に悪化している。

各問題ごとの速度比較を行った結果を図に示す。図は全て、探索中に節点が展開された回数を比較した結果である。

図5によれば、候補手をヒューリスティックに限定した場合のtop10は逆効果であることが分かる。左上にある点はtop10の方が優位な問題であり、右下は通常のままの方が良いことを示す。大部分の問題では遅くなっていることが分かる。右側に縦に並んでいる点は、探索を200,000節点で打ち切っているため、通常のままのDf-pn+では解けたが、top10では節点数制限を越えてしまった問題である。悪化する原因は、ヒューリスティックに候補手を限定した場合、生成された候補手の数自体が既に十分良い指標になっており、top10を用いることはそれを無視することになるためであると思われる。

次に図6は、全合法手を生成した場合の通常のDf-pn+とtop5の性能比較である。通常のままのDf-pn+ではほとんどの問題が節点数制限を超えてしまっている。この図より、top5は非常に大きな性能向上をもたらしていることが分かる。

図7はtop10とtop20の比較であるが、top10の方がほとんど常に良いもののいくつかの問題ではtop20の方が勝ることが分かる。図8ではその傾向がより顕著になり、少ない子節点を対象にするほど性能向上が不安定であるという傾向が見て取れる。

同様の傾向はratenでも現れる。図9が示すように、全合法手を生成した場合のrate40は大きな性能向上をもたらす。しかし、図10が示す、rate10とrate20の性能比較、及び図11が示すrate20とrate40の性能比較は、ともにばらつきが大きい。全体として、今回の問題セットについては、raten方式の方がtopn方式よりも性能の不安定さがやや大きかった。

以上より、今回実験を行った動的候補手拡大の方式の中ではtop5かrate40が性能が高いことが分かる。

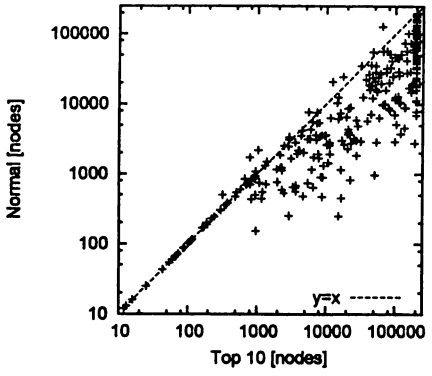

図 5　ヒューリスティックな候補手の場合の通常の Df-pn+ と top10

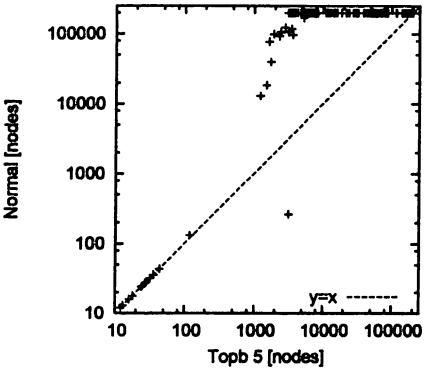

図 6　全合法手の場合の通常 Df-pn+ と top5

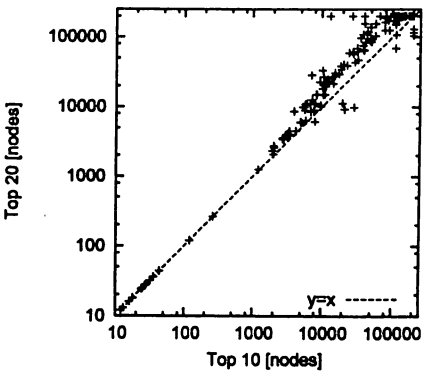

図 7　全合法手の場合の top20 と top10

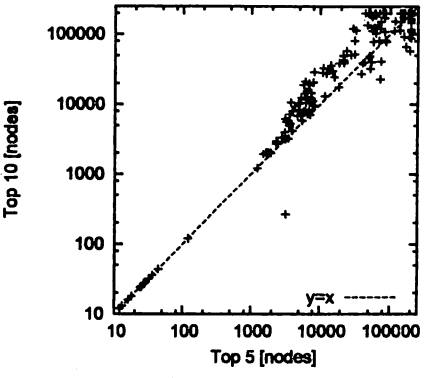

図 8　全合法手の場合の top10 と top5

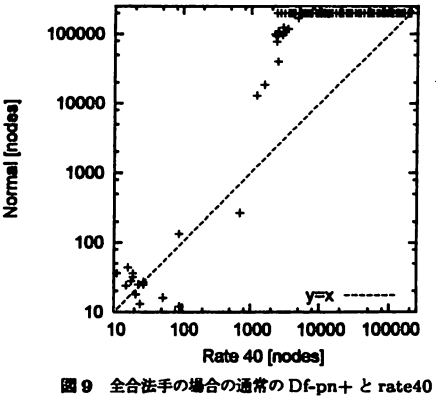

図 9　全合法手の場合の通常の Df-pn+ と rate40

最後に図 12において、一番解答能力の高かった、通常の Df-pn+ とヒューリスティックな候補手の組合せと、top5 と全候補手の組合せの性能を比較した。top5 でも節点数は平均して 3 倍程度劣っている。しかしながらヒューリスティックな候補手を用いた場合は誤答が 9 問存在した (正答は 283 問) のに対し、全候補手を生成する場合は、解の正しさが保証されている。

5. まとめと今後の課題

Df-pn 探索において、証明数 / 反証数の計算は通常は全ての候補手の合計を利用するが、OR 節点に置いては証明数が小さい方からいくつかの、AND 節点に置いては反証数が小さい方からいくつかの節点についてだけ合計する手法を実装した。

囲碁の捕獲探索においては、全合法手を生成した場合には平均 100 倍以上の速度向上が得られた。候補手をヒューリスティックに限定する手法と比較すると、節点数で測った性能は平均 3 倍程度劣っている。しかしながら、解の正しさが保証されているところは本手法の利点と言える。欠点としては、候補手拡大に用いる候補手数を小さくするにつれて性能が不安定になる性質がある。

以上より、本手法は、正しい解答を保証しながら速度向上を得たい場合、またはヒューリスティックに候補手を限定するのが難しい場合などに有効であると言える。

今後の課題としては、囲碁以外の様々問題に適用して性能を測定し、どのような性質のある問題に有効なのか

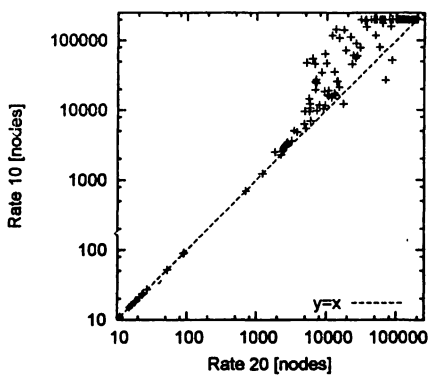

図 10 全合法手の場合の rate10 と rate20

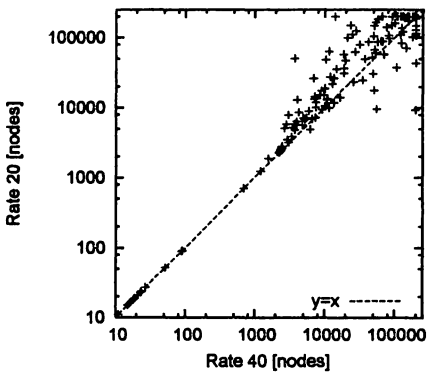

図 11 全合法手の場合の rate20 と rate40

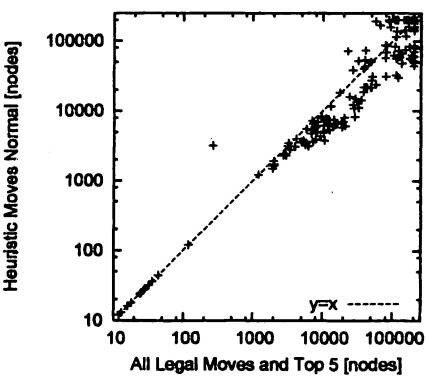

図 12 全合法手かつ top5 と
ヒューリスティックな候補手を用いた通常の Df-pn+

明らかにすることがあげられる。

参 考 文 献

1) L.V. Allis, M.vander Meulen, and H.J. vanden Herik. Proof-number search. *Artificial Intelligence*, 66(1):91–124, 1994.

2) Tristan Cazenave. Generalized widening. In RamonLópez deMántaras and Lorenza Saitta, editors, *ECAI*, pages 156–160. IOS Press, 2004.

3) A. Kishimoto and M. Müller. Df-pn in Go: An application to the one-eye problem. In *Advances in Computer Games 10*, pages 125–141. Kluwer Academic Publishers, 2003.

4) A. Kishimoto and M. Müller. A general solution to the graph history interaction problem. In *19th National Conference on Artificial Intelligence (AAAI'04)*, pages 644–649. AAAI Press, 2004.

5) A. Kishimoto and M. Müller. Search versus knowledge for solving life and death problems in Go. In *Twentieth National Conference on Artificial Intelligence (AAAI-05)*, pages 1374–1379. AAAI Press, 2005.

6) R.E. Korf. Linear-space best-first search. *Artificial Intelligence*, 62(1):41–78, 1993.

7) A. Nagai. *Df-pn Algorithm for Searching AND/OR Trees and Its Applications.* PhD thesis, Dept. of Information Science, University of Tokyo, Tokyo, 2002.

8) Ayumu Nagai and Hiroshi Imai. Application of df-pn algorithm to a program to solve tsume-shogi problems. *Transactions of Information Processing Society of Japan*, 43(6):1769–1777, June 2002.

9) Jonathan Schaeffer, Neil Burch, Yngvi Bjornsson, Akihiro Kishimoto, Martin Muller, Robert Lake, Paul Lu, and Steve Sutphen. Checkers is solved. *Science*, pages 1144079+, July 2007.

10) T. Thomsen. Lambda-search in game trees - with application to Go. *ICGA Journal*, 23(4):203–217, 2000.

11) Thomas Thomsen. Madlab website. http://www.t-t.dk/madlab/problems/index.html.

12) 岡部 文洋. 経路分枝数を用いた詰め将棋解図について. In *Game Programming Workshop 2005 (GPW05)*, volume 2005, pages 9–16, November 2005.