2009－GI－21 (16)
2009／3／9

最適解を求める2つの探索アルゴリズムのスーパーパズにおける性能比較について

鈴木裕俊[†1] 岸本章宏[†1]

A* アルゴリズムと IDA* アルゴリズムは、最適解を保証しながら、パズルを解くことができる探索アルゴリズムである。本論文では、これらのアルゴリズムをスーパーパズに適用し、性能比較を行う。他のパズルにおける先行研究では、IDA* の方が A* よりも優れたアルゴリズムであることが知られている。本論文の実験では、先行研究の結果とは異なり、スーパーパズの探索空間の性質のために、A* の方が IDA* よりも高速に解答できることを示す。

Performance Comparisons of Two Search Algorithms that Guarantee Optimal Solutions in Superpuzz

Hirotoshi Suzuki [†1] and Akihiro Kishimoto[†1]

The A* and IDA* algorithms are the search techniques that can solve puzzles, while always guaranteeing optimal solutions. This paper adapts these algorithms to Superpuzz and compares their performance. According to previous research on other puzzles, IDA* has been shown to be superior to A*. However, unlike what is obtained in previous research, experimental results presented in the paper show that A* solves problems more quickly than IDA*, because of a property of the search space of Superpuzz.

1. はじめに

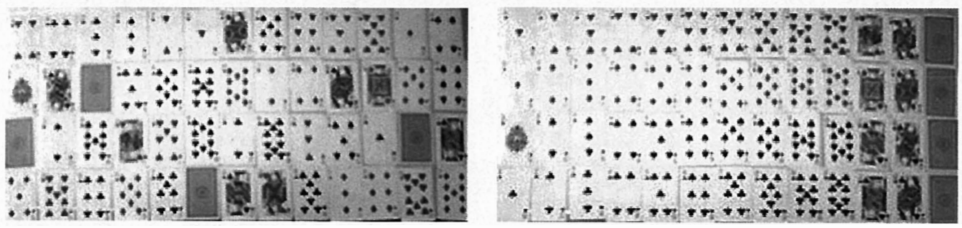
パズルの探索空間では、計算量の組み合わせ爆発が生じるため、パズルを効率良く解く探索アルゴリズムの開発は人工知能において代表的な研究テーマである。スーパーパズは、トランプを用いたパズルである。このパズルは、情報処理学会のプログラミング・シンポジウムにおいて、GPCC(Games and Puzzles Competition on Computers) の課題に選ばれたこともあり、計算機科学者にとっては有名な問題である。サイズが 4×13 のスーパーパズの問題の探索空間は、平均で $O(10^9)$、最大で $O(10^{10})$ 程度であると考えられている[12]。このため、高速にスーパーパズを解くアルゴリズムの開発は、トリビアルな問題ではなく、面白い研究課題である。

解の最適性 (最短手順の解を返すこと) を常に満たしつつ、効率良くスーパーパズを解くためには、探索空間の性質に合う探索アルゴリズムの選択が重要である。これまでのパズルを解くアルゴリズムでは、探索アルゴリズムのベースとして、Korf の IDA* アルゴリズム[5] を利用し、IDA* に様々な改良を施してきた[1),4),7)]。IDA* は、最良優先探索アルゴリズムである A* アルゴリズム[2] を深さ優先探索で実現したアルゴリズムであり、A* に比べて、様々な利点がある。IDA* では、深さ優先探索を利用するため、A* よりもメモリの使用量が少ない。さらに、IDA* の実装には、A* で利用するオープンリストが不要である。N をオープンリストにある節点数とすると、新規節点の挿入・削除に関するオープンリストの操作には、$O(\log N)$ の計算時間がかかるので、IDA* の単位時間あたりの展開節点数は、A* よりも大きい。IDA* の欠点は、反復深化法を利用するために、以前に展開した節点の再展開が起こることである。しかし、これまでのパズルでは、この再探索の割合は、探索空間が木であることを仮定すると全体の探索に比べて少ない[5] 上に、ハッシュ表を利用すれば、この再展開をさらに減らせる[7] ことが知られている。

一方、A* アルゴリズムの利点は、IDA* のような再探索が起こらないことが挙げられる。例えば、バイオインフォーマティックスのシークエンスアライメント問題[6),10)] やゲームにおける経路探索問題[9)] など、別経路から同一節点に至る場合が頻出する探索空間では、A* に基づく方法を利用するのが標準的である。

本研究では、効率良く最適解を求めるスーパーパズ・プログラムを開発するために、探索アルゴリズムの評

†1 公立はこだて未来大学システム情報科学部情報アーキテクチャ学科
Department of Media Architecture, School of Systems Information Science, Future University-Hakodate
Email: {m1205034, kishi}@fun.ac.jp



(a) 初期局面　　(b) 目標局面

図 1　スーパーパズ

価を行う。本論文の貢献は、次の通りである。

- マンハッタン距離に基づくヒューリスティック関数を利用した二つの探索アルゴリズム (A* と IDA*) を実装し、性能を比較した。その結果、これまでのパズルでの結果とは異なり、A* の方が IDA* よりも約 1.2 倍高速にスーパーパズの問題を解けた。問題を解くのに要した展開節点数に関しては、A* は IDA* の 27%であった。
- A* の展開節点数が、IDA* よりもはるかに少ない理由を詳しく解析した。その結果、IDA* の展開節点数に対する内部節点の再展開の割合は 72%であるのに対し、A* の全格納節点数に対するオープンリストに保持される節点数の割合は 34%であった。これらは、スーパーパズの探索空間では、別経路から同一節点に至る場合が頻出することを示唆している。

本論文の構成は次の通りである。第 2 章では、スーパーパズのルールとパズルの性質について説明する。第 3 章では、関連研究として、スーパーパズの先行研究と A* および IDA* アルゴリズムについて取り扱う。第 4 章では、本研究で実装したスーパーパズ・プログラムのヒューリスティック関数について説明する。第 5 章では、実験結果について議論し、第 6 章では、まとめと今後の展望について述べる。

2. スーパーパズのルールと性質

スーパーパズとは 1 組のトランプを用いた一人遊びである。スーパーパズでは、1 から 12 までのカードをランダムに 4 行 13 列に並べ、ゲームを開始する (図 1(a) を参照)。カードのない位置は「穴」と呼ばれ、次の 3 つの規則のいずれかに従って、1 枚のカードをこの穴に移動できる。

- 穴 H が左端 (1 列目) にあれば、任意のマークの 1 は H に移動できる。
- H の左隣のカードと同じマークで、数字の 1 つ大きなカードは H に移動できる。
- H の左隣のカードが 12 または穴の場合は、どのカードも H に移動できない。

スーパーパズの目的は、カードを穴に移動することを繰り返し、最終的に図 1(b) のように各マーク毎にカードの 1 から 12 を昇順に並べるか、またはどのような手順でも図 1(b) ような目標局面に至ることが出来ないことを示すことである。また、わずかなルール変更によって 4×13 よりも小さなサイズのゲームも定義できる。

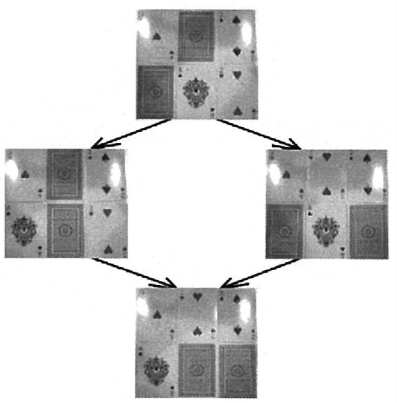


図 2　2×3 のスーパーパズの同一局面の生成例

スーパーパズの特徴として、別経路 (以下、経路のことを**手順**とも呼ぶ) によって同一節点 (同様に、以下節点のことを**局面**とも呼ぶ) に至ることがある。例えば、2×3 のスーパーパズの同一局面の生成の例を図 2 に示す。この例では、ハートの 2 とスペードの 1 のどちらを先に動かしても、最終的には同じ局面に到達する。

3. 関連研究

本章では 3.1 節ではスーパーパズの先行研究について述べ、3.2 節では本研究に関連する探索アルゴリズムについて説明する。

3.1 スーパーパズ・プログラムの研究

スーパーパズを解くプログラムでは、前節で説明した複数手順による同一局面が存在する性質のために新しい局面 P を生成したときには、P は以前に生成した局面であるかどうかを判定する必要がある。この同

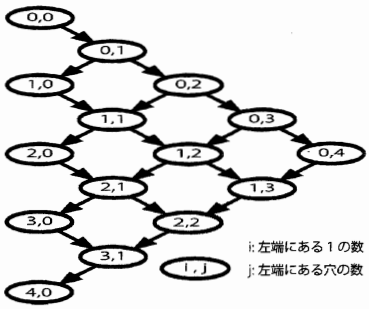


図 3 スーパーパズの状態遷移図 (文献[13] より)

一局面の判定がなければ、P 以下の探索が 2 回以上行われることがある。この無駄な探索は、性能低下につながる。新谷は、基数探索[8] を利用して、同一局面の判定を高速に行い、4×6 のスーパーパズで高性能な手法を開発した[12]。

文献[13),14)] では、数字が 1 のカードを 1 列目に移動した後には、このカードは 2 列目以降に移動できないという性質を利用し、探索空間の依存性を判定した。この手法では、ある局面の (i, j) 要素成分を (左端のカードの 1 の数, 左端の穴の数) と定義する。左端にあるカードの 1 と穴の和の最大数は 4 であるので i と j は $0 \leq i+j \leq 4$ を満たす整数である。図 3 に (i, j) 要素成分の状態遷移の依存性を示す。この状態遷移の性質を利用すれば、スーパーパズの探索に必要なメモリの使用量を減らすことが出来る。例えば、探索する必要のある局面が (1,0) と (0,2) のみであると仮定する。この場合には、(1,0) や (0,2) から (0,1) に遷移することはないので、メモリ上から (0,1) 要素成分に属する局面を削除できる。新谷はこの手法を用いれば、2GB 程度のメモリで 13 列のスーパーパズが解けると理論的に予想した。

3.2 最適解を保証する探索アルゴリズム

前節で説明した新谷の研究では、スーパーパズを解いたときに解の最適性を保証できず、最短手順ではない解を返すことがある。

この節では、最適解を必ず保証する A* と IDA* という二つのアルゴリズムについて取り扱う。任意の節点 n と m に対して、n の子節点 m に移動する場合のコスト $c(n, m)$ を 1 と定義すれば、両アルゴリズムは、スーパーパズにおいて常に最短手順を返せる。

解の最適性を保証するために、両アルゴリズムに共通する概念に、楽観的なヒューリスティック関数 $h(n)$ がある。$h(n)$ は、ヒューリスティックなどを利用して局面 n を評価することで、n から目標局面 o に至るまでにかかる合計コストを推定する関数である。$h(n)$ が楽観的であるとは、どのような局面 n に対しても $h(n)$ は o までの実際の合計コストよりも大きく見積もることがないと定義する。$h(n)$ が楽観的であることを保証しつつ、探索アルゴリズムの性能を改善するような $h(n)$ を開発する研究は、重要なテーマの一つである[1),4)]。

3.2.1 A* アルゴリズム

$g(n)$ を初期局面 s から局面 n に至るまでにかかった合計コストの最小値とすると、s から n を経由して目標局面に至る合計コストの推測値 $f(n)$(f-値と呼ぶ) を次のように定義する:

$$f(n) = g(n) + h(n).$$

A* は、オープンリストとクローズドリストを利用して節点の展開を行う最良優先探索である[2)]。オープンリストは、未展開節点 n のすべてを f-値の小さな順に並べるデータ構造であり、クローズドリストは、すでに展開した節点すべてを保持するデータ構造である。クローズドリストは、新たに生成された節点がすでに展開済みであるかの判定と、A* が解を見つけたときの解答手順の計算に利用する。

A* の代表的な実装手法では、最初に初期節点 s をオープンリストに格納した後、目標節点を見つけるか、または目標節点が存在しないことを証明するまで、次の 1 から 4 の操作を繰り返す。ただし、$cl(m)$ をクローズドリストに保存された m の f-値とする。

(1) オープンリストから f-値が最小の節点 n を取り出す。オープンリストが空のときには、解は存在しない。

(2) n が目標節点であれば、最小コスト $f(n)$ と s から n に至る手順を返す。

(3) n を展開し、n の子節点 m すべてを生成する。m の状況によって、次の処理を行う。

  (a) m がクロードリストにあり、$cl(m) \leq f(m)$ ならば、m はオープンリストに格納しない。

  (b) m がクローズリストにあり、$cl(m) > f(m)$ ならば、$cl(m) = f(m)$ に更新し、m をオープンリストに入れ直す。

  (c) それ以外のときには、m をオープンリストに入れる。

(4) n をクローズドリストに格納する。

A* の f-値は、初期局面から目標局面までにかかる合計コストよりも大きく見積もることがない。さらに、A* は、f-値の小さな順番に節点を展開する。このため、A* が目標局面をまだ見つけていない場合、最適解の合計コストは、必ずオープンリストの先頭にある節点の f-値以上であることが証明できる。この性質より、A* が目標状態を見つけたときには、必ず最適解である。

A* では、探索中のすべての局面を記憶するので、莫大な節点数を持つ探索空間で解を求めるときは、メモリの使用量が大きくなる。また、A* のオープンリストでは、f-値が最小の節点を取り出す必要があり、順位付きキューなどのデータ構造を用いるのが通常であ

る。オープンリストに格納された節点数を N とすると、オープンリストへの新たな節点の格納および f-値を取り出して、オープンリストの構造を維持するのにかかる計算量は $O(\log N)$ である。

3.2.2 IDA* アルゴリズム

IDA* は、A* と同様に最適解を保証する手法であるが、反復深化法に基づく深さ優先探索である[5]。この深さ優先探索の性質のため、現在探索中の経路上にある節点のみをメモリに保持しておけば、解を求められるので、IDA* は A* よりもメモリの使用量が遥かに少ない。さらに、IDA* では、A* で用いられているオープンリストが不要であるので、A* よりも高速に節点を展開できる。

IDA* では、通常の反復深化とは異なり、f-値を閾値にして深さ優先探索を区切る。初期局面を s とすると、IDA* は最初の閾値を $f(s)$ に設定する。もし、この閾値 $f(s)$ で、目標局面 g を発見できれば、g は最適解である。閾値 $f(s)$ で g を発見できなければ、IDA* は閾値をより大きなものに設定し、s から再探索を行う。節点 n を前回の反復時に f-値が閾値を超えた節点の中で f-値が最小の節点とすると、IDA* は再探索の閾値を $f(n)$ に設定する。IDA* は、g を見つけるか、または g が存在しないことを証明するまで、各反復でこの閾値の再設定と s からの再探索を繰り返す。

IDA* のような反復深化法は、初期節点からの探索を繰り返すので、以前の反復で展開済みの節点を再展開してしまう。さらに、スーパーパズのような探索空間では、複数の手順で同一局面 n に至ることがある。別手順で再び n に至るときには、IDA* が現在の反復での閾値を超えるまで n をすでに探索済みであることが多いので、新しい節点を展開できない頻度が増加する。このようにして、節点の再展開の頻度が増加すれば、IDA* の性能の低下につながる。

高性能な IDA* はハッシュ表を用いて、このような再展開の頻度を減らすのが通常である[7]。IDA* のハッシュ表では、節点 n をハッシュ・キーにして、n 以下に目標節点があることの証明に必要な閾値を、対応するハッシュ表の要素に保持する。また、n の探索を行う前には、ハッシュ表を参照し、n 以下を展開する閾値と比較する。もし、ハッシュ表に保存された値がこの閾値よりも大きければ、n 以下の探索を省略できる。

4. 本研究で利用したスーパーパズ・プログラムのヒューリスティック関数

本論文では、A* と IDA* の性能を比較し、スーパーパズの探索空間の性質について調べることが目的であるが、解の最適性を常に保証するためには、楽観的なヒューリスティック関数 $h(n)$ を開発する必要がある。本論文では、15 パズルなどでよく利用されているマンハッタン距離に基づく $h(n)$ を作成した。

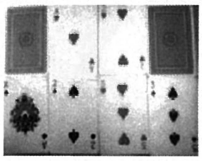


図 4 マンハッタン距離計算の例 (サイズ 2)

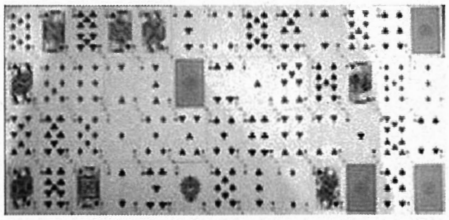


図 5 問題例 (H1)

スーパーパズにおけるマンハッタン距離 $h(n)$ は、次のようにして定義する。M_1, M_2, M_3, M_4 をそれぞれスペード、クラブ、ハート、ダイヤとし、R_{i_1,i_2,i_3,i_4} を j 行目にマーク M_{i_j} のカードを 1 から 12 まで並べ、目標局面を達成したいと仮定する。$D(i_1,i_2,i_3,i_4)$ を j 行 k 列目 $(1 \leq j \leq 4, 1 \leq k \leq 12)$ にマーク M_{i_j} のカード k 以外のカードが置かれている枚数と定義する。R_{i_1,i_2,i_3,i_4} の可能性は、マークの数の順列だけ $(4\times3\times2\times1 = 24$ 通り) 存在するが、$D(i_1,i_2,i_3,i_4)$ が最小のものを $h(n)$ と定義する。

図 4 にマンハッタン距離の例を示す。この図において、1 列目にハート、2 列目にスペードを並べる場合には、ハートの A、2、および 3 とスペードの 3 が対応する位置にないので、$D(i_1,i_2) = 4$ である。一方、1 列目にスペード、2 列目にハートを並べる場合には、ハートの A および 2 とスペードの A、2、および 3 が対応する位置にないので $D(i_1,i_2) = 5$ である。1 列目にハートを並べる方が、目標を達成しやすそうに見えるので、この例では $h(n) = 4$ となる。

スーパーパズでは、k 列にカード k を置くには必ず穴を経由する必要があるので、マンハッタン距離は楽観的なヒューリスティック関数である。また、n の子節点のマンハッタン距離の計算は、n のマンハッタン距離からの差分計算で行えるので、高速である。

5. 実験結果

前節で説明したように、A* と IDA* を用いたスーパーパズ・プログラムを実装し、Intel CPU Core2Duo 2.66GHz、メモリは 2GB の PC 上で、両方のプログラムに Jonathan Schaeffer によるスーパーパズ問題集を解かせることで、性能比較を行った.

Schaeffer の問題は、3 つの難易度 (Easy, Normal,

Hard) から構成されるが、Easy は簡単であったため、Normal と Hard のみを利用した。図 5 は、Hard カテゴリーに属する問題例である。問題数は、Normal が 3 題、Hard が 9 題の計 12 題である。両アルゴリズムに各問題を解かせる制限時間は、360 秒または A* が 2GB のメモリを使い切るまでとし、解答時間や展開節点数を計測した。IDA* のハッシュ表の実装には、Zobrist の方法[11] を利用した。

表 1 と図 6 に A* と IDA* の性能を比較した結果を示す。図 6 は、IDA* と A* の実行時間を両アルゴリズムが解けた問題について描画したものである。一つの点は、各問題における両アルゴリズムの解答時間を表し、$y = x$ の線より下にある点は、A* の方が IDA* よりも高速にその問題を解答したことを表す。H3 と H4 は、どちらのアルゴリズムにも解けなかった問題である (表 1 を参照)。

この図と表より、A* と IDA* の解答能力は同等であるが、A* の方が高速なアルゴリズムであると言える。A* が問題を解くのに要した展開節点数に関しては、平均で IDA* の 27%であるが、単位時間に展開できる節点数は、IDA* の方がはるかに高速である。その結果、A* の解答速度は、IDA* の 1.2 倍程度である。本論文で利用した A* の実装では、オープンリストの操作に関するオーバーヘッドが全体の計算の 60%である。さらに、新たな子局面を生成するときには、A* では局面のコピーを行う必要があるのに対し、IDA* では現局面を更新して子局面を生成し、探索後に元に戻すだけでよい。このため、IDA* では A* よりも高速な実装が実現できる。

表 2 は、両アルゴリズムが解いた問題について、A* と IDA* の性質を詳しく解析したものである。この表において、A* では、オープンリストに保持している節点数が最大のときの、オープンリストおよびクローズドリストに格納された節点数を示している。一方、IDA* では、新規節点を展開した回数 (新規展開数) と以前に展開したことのある節点を再展開した回数 (再展開数) を示した。さらに、両アルゴリズムが探索した空間の平均分岐因子についても計算した。

本論文で利用したスーパーパズの問題の平均分岐因子は 2.6 から 4.0 程度である。もし、探索空間が木の場合には、A* のクローズドリストに格納された節点数に対するオープンリストに格納された節点数の割合は、平均分岐因子に近い値のはずである。IDA* の再展開数に対する新規展開数の割合についても、同様のことが言える。しかし、実際には A* のオープンリストに格納された節点数は、クローズドリスト上にある節点数よりもはるかに少ない。A* がメモリに格納している全節点数に対するオープンリスト上にある節点数の割合は、15%から 53%の間に分布している。さらに、IDA* の展開節点数に対する再展開数の割合は、66%から 87%であり、IDA* の性能低下の原因である。

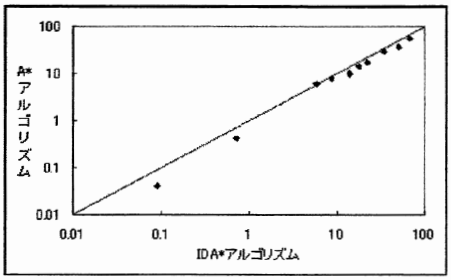


図 6 A* と IDA* の解答時間の比較

これらの数値より、スーパーパズの探索空間では、複数の経路から同一節点に到達する場合が非常に多いことが推測できる。

6. ま と め

本研究では、スーパーパズの最適解を求めるために A* と IDA* を利用したプログラムを実装し、性能の比較を行った。実験結果では、スーパーパズの探索空間の性質のために、A* の方が IDA* よりも優れた探索アルゴリズムであるという結論を得た。

今後の課題には、マンハッタン距離よりも優れたヒューリスティック関数を利用した場合の性能比較がある。ヒューリスティック関数の改良として考えられる手法には、アブストラクションを利用した方法[3] がある。より正確なヒューリスティックを導入すれば、通常は探索アルゴリズムの展開節点数の削減につながるが、展開節点数の削減効果は、展開節点数の多いアルゴリズムの方が大きい傾向がある。その結果として、IDA* の方が A* よりも性能が良くなる可能性がある。

参 考 文 献

1) J. Culberson and J. Schaeffer. Pattern databases. *Computational Intelligence*, Vol.14, No.3, pp. 318–334, 1998.
2) P.E. Hart, N.J. Nilsson, and B.Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, Vol.4, No.2, pp. 100–107, 1968.
3) R.C. Holte, M.B. Perez, R.M. Zimmer, and A. J. MacDonald. Hierarchical A*: Searching abstraction hierarchies efficiently. In *13th National Conference on Artificial Intelligence (AAAI)*, Vol.1, pp. 530–535, 1996.
4) A.Junghanns. *Pushing the Limits: New Developments in Single-Agent Search.* PhD thesis, Department of Computing Science, University of Alberta, 1999.
5) R.E. Korf. Depth-first iterative-deepening: An

表 1　A* と IDA* の性能比較

	A*		IDA*	
	展開節点数	解答時間 (秒)	展開節点数	解答時間 (秒)
N1	743,786	6.11	2,076,304	5.81
N2	3,805,946	29.99	13,241,013	34.58
N3	2,152,617	17.24	8,361,695	22.01
H1	4,916,012	36.92	19,763,094	50.88
H2	6,871,894	56.85	23,143,611	67.24
H3	—	—	87,215,552	360.00
H4	—	—	86,306,984	360.00
H5	88,495	0.42	375,963	0.72
H6	1,028,313	7.83	3,557,923	8.75
H7	1,366,444	9.84	5,803,211	13.96
H8	4,235	0.04	29,718	0.09
H9	1,920,919	14.66	7,078,527	17.86

表 2　A* の格納節点数と IDA* の展開節点数の詳細

	A* の格納節点数		IDA* の展開節点数		平均分岐因子	
	オープンリスト	クローズドリスト	新規展開数	再展開数	A*	IDA*
N1	832,767	743,784	692,830	1,383,474	4.02	4.03
N2	2,255,933	3,805,939	3,649,550	9,591,463	3.35	3.41
N3	1,172,262	2,014,574	2,422,718	5,938,977	3.50	3.69
H1	1,183,926	4,916,005	4,979,526	14,783,568	3.02	3.26
H2	3,909,666	6,871,885	6,889,517	16,254,094	3.40	3.56
H5	15,015	84,789	94,593	281,370	2.58	2.81
H6	466,566	1,006,626	999,117	2,558,806	3.28	3.46
H7	577,798	1,366,442	1,400,832	4,402,379	3.32	3.41
H8	970	2,910	3,835	2,5883	3.82	4.17
H9	1,133,250	1,920,913	2,009,914	5,068,613	3.37	3.48

optimal admissible tree search. *Artificial Intelligence*, Vol.27, No.1, pp. 97–109, 1985.

6) R.E. Korf and W.Zhang. Divide-and-conquer frontier search applied to optimal sequence alignment. In *17th National Conference on Artificial Intelligence (AAAI)*, pp. 910–916, 2000.

7) T.A. Marsland and A.Reinefeld. Enchanced iterative-deepening search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol.16, No.7, pp. 701–710, 1993.

8) R. セジウィック. アルゴリズム C 第 2 巻. 近代科学社, 1996.

9) P.Yap. Grid-based path-finding. In *Advances in Artificial Intelligence. 15th Conference of the Canadian Society for Computational Studies of Intelligence, AI 2002*, Vol. 2238 of *Lecture Notes in Artificial Intelligence (LNAI)*, pp. 44–55. Springer, 2002.

10) T.Yoshizumi, T.Miura, and T.Ishida. A* with partial expansion for large branching factor problems. In *17th National Conference on Artificial Intelligence (AAAI)*, pp. 923–929, 2000.

11) A.L. Zobrist. A new hashing method with applications for game playing. Technical report, Department of Computer Science, University of Wisconsin, Madison, 1970. Reprinted in *International Computer Chess Association Journal*, 13(2):169-173, 1990.

12) 新谷敏朗. スーパーパズを解くプログラム. 第 5 回ゲーム・プログラミングワークショップ, pp. 84–91, 1999.

13) 新谷敏朗. スーパーパズの状態遷移に関する考察. 情報処理学会研究報告, 第 3 巻, pp. pp.41–48, 2000.

14) 新谷敏朗, 三宅洋行. スーパーパズの解の探索に関する考察. 第 7 回ゲーム・プログラミングワークショップ, pp. 171–178, 2002.