2.5　パズル

東京大学情報基盤センター
田中 哲朗
ktanaka@tanaka.ecc.u-tokyo.ac.jp

■パズルの分類

ゲームの一般的な分類法に従うとパズルは以下の2種類に分類される.

- **一人完全情報ゲーム**
  フリーセル等のカードゲーム，倉庫番，ペントミノ，ルービックキューブ，15パズル
- **一人不完全情報ゲーム**
  カルキュレーション等のカードゲーム，マインスイーパ，マスターマインド（MOOなど），上海，テトリス等

ゲームの目的は，多くの場合はある初期状態（不完全情報の場合は，解答者には一部しか提示されていない）からルールに従ったプレイをして，ゴール状態に至ることにある．多くの場合は，評価値はゴールするか，ギブアップするかの二値であるが，ルービックキューブにおける秒数，マスターマインドにおける手数（プレイ数）など，多値の評価値を持つゲームもある.

計算機上のパズルのプレイは，多くの個人用計算機にインストールされているということもあって，一般的になってきている．そして，このことが新たなパズルファンを生み出す原因ともなっている．実際，パズル出題者としての計算機は，良質の乱数生成能力，ミスのないルール判定能力，記録集計能力において人間の比ではない.

たとえば，マインスイーパ（**図-1**）を計算機を使わずにプレイすることを考えてみる．初期配置として爆弾を配置することはカードでもできるが，隣接マスの爆弾数の計算を自動に行うことはカードではできない．したがって，あらかじめ計算済みのボードを多数用意するか，審判（あるいは出題者）に数えてもらうかのどちらが必要になる．どちらにしてもゲーム性を損ねてしまうことにつながる.

計算機で扱いにくいパズルとしては，ジグゾーパズル，知恵の輪等の物体の物理的な形状を利用したパズルが挙げられる．これらのパズルを計算機で扱うことが可能だとしても，問題になるのは，パズルのゲームの本質よりも，画像認識等の周辺技術と考えられるので，ここでは対象として取り上げないことにする.

また，パズル愛好家の間で人気の高いクロスワード，ナンクロ等のワードゲームも，利用可能な単語集合と，ヒントのあるゲームの場合は，ヒントと単語の対応を明確に定義するならば，一人完全情報ゲームと見なすことも可能だが，ワードゲームの面白さは，その前の制約を満たす単語を見つけ出すところにあるので，計算機に解かせる課題としては面白さに欠けると考えられる.

■問題の作成

計算機で出題するパズルの場合，初期配置を乱数で与える場合が多いが，完全情報ゲームの場合は倉庫番のように人間が作成した問題セット[☆1]を用いる場合もある.

乱数で初期配置を与える場合は，最善のプレイ（不完全情報ゲームの場合は，完全情報化して，本来得られない情報も与えたうえで）でも解けない初期配置を除くことも考えられる[☆2]．このためには，以下のような方法が考えられる.

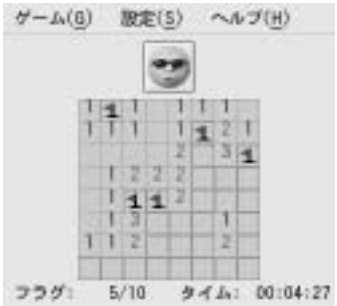


図-1　マインスイーパ

☆1 文献1）で計算機による問題の作成の研究もある.
☆2 マスターマインドのようにルールに従って生成された初期配置はすべて解けるゲームもある.

1．ゴール局面から出発して，最終手から逆向きにランダムに逆向きの手を生成していく．この作業を適当な回数繰り返した結果，得られる局面を初期局面とする．
2．乱数で初期局面を生成して，簡単なソルバーで解いて，解を見つけることができた問題のみを問題として提示する．このソルバーはすべての解のある初期局面を解ける必要はない．解のある初期局面を解けなければ，次の初期局面を乱数生成するだけである．

どちらの方法でも，作成された問題がランダムではなく「癖」が出てしまう可能性はある．1の方は，逆向きの手の生成の回数を超えるような正解手順を持つ問題は作成できないし，そうでなくても複数の経路で正解に至る（やさしい）問題は，唯一の経路でしか正解に至ることのない問題よりも高頻度で生成されることになる．

2の場合も，ソルバーの能力が低い場合はやさしい問題ばかりが生成されるという問題点がある．ある程度能力が高くても，解ける問題に偏りがあると，特定のテクニックを使った解法を必要とする問題がまったく生成されないということもあり得る．

■完全情報パズルの解法

パズルは離散最適化問題としてとらえることができるので，一般的な最適化アルゴリズムを適用することによって，解くことができる場合が多い．パズルを解くのによく用いられる探索アルゴリズムをいくつか挙げる．

全探索

完全情報ゲームで規模の小さい問題は，全探索ですべての解を求めることができる．このジャンルに属する例は多いので，多くは挙げない．

探索は多くの場合，木として表現される．単純に全探索を実現するには，深さ優先探索が多くの場合，用いられる．別の経路を経て同じ局面に至ることが頻繁にあり，再計算のコストが多い場合は局面表（transposition table）を持ち，一度計算した結果を再利用する方法が一般的である．また，ループがあるパズルでは局面表はループを検出するためにも用いられる．

初期配置の数が少ないパズルに関しては，解けた状態からスタートして後退解析（retrograde analysis）により，可能な解すべてのテーブルを作成しておく方法もある．

幅優先探索と深さ優先探索

解ける場合に，解に至る手順が一意であるパズルよりも，途中で合流があり，複数の解があるパズルの方が多い．このようなパズルで，解を1つ見つけることを目的とする場合は，探索順序が重要になってくる．

深さ優先探索（depth first search）　ある経路で探索して手詰りに陥った際に，まだ探索していない枝分かれのうちの，最も深い枝を先に探索するというもの．プログラミングは容易である．また，局面表を使わなければ，深さ分のメモリを消費するだけで済む．

幅優先探索（bredth first search）　まだ探索していない枝分かれのうちの，最も浅い枝を先に探索するというもの．この順序で探索を行うと，自然に最も浅い解を見つけることができる．

実装では，探索していない局面を深さ順に優先度キューに入れて，最も浅い局面を取って探索する方法が自然だが，局面を表現するデータ構造が大きい場合は，メモリ効率の点で劣る．

深さを閾値にして，深さ優先探索を行い，閾値を繰り返し大きくしていくことで，幅優先探索と同じ順序での探索を行うことも可能である．

ゲームの性質により，どちらが向いているかが決まる．深さdまでは幅優先探索，それより深い部分は深さ優先探索に切り替えるといった組合せも可能である．

最良優先探索

ある局面が解にどれだけ近いかを近似するヒューリスティック関数を用いて，未探索の局面のうち，最も解に近そうな局面を探索するのが最良優先探索である．中でも，A* はよく用いられる．

ある局面からゴールへの距離を見積もる関数 h(X) が，真の距離以下の値を返すことが保証されているとする．

$$\mathrm{cost}(X) = \mathrm{distance}(\mathrm{root}, X) + h(X)$$

としたときに，未探索ノードのうち，cost(X) が最小となるノードから探索していくと，ゴールを1つ見つけると，それが最短経路であることがいえる．距離を問題とせずに解を1つだけ見つければよいという場合も，distance(root, X) = 0 として A* で探索すると，h(X) が良い評価関数である場合は，早く解にたどり着くことが期待される．

実装方法はいろいろ考えられる．探索していない局面を深さ順に優先度キューに入れて，最も浅い局面を取って探索する方法が自然だが，局面を表現するデータ構造が大きい場合は，メモリ効率の点で劣る．

値を閾値にして，深さ優先探索を行い，閾値を繰り返し大きくしていくことによって，A* と同じ順序での探索を行うことも可能である．これを IDA*（Iterative Deepning A*）[2] と呼ぶ．

乱数を用いた探索

解に至る経路が複数あり，かつその数が十分である場合，行き止まりになるまで，乱数によって手を生成して選んでいく方法も有力である．たとえば，

- ある程度以上の手数にならないと解がなく，それを超えるとかなりの確率で解がある．幅優先探索ではそれ以下の手数のノードをすべて探索する必要があるが，メモリや時間の関係でできない．
- ある程度以上の手数でも解のあるなしに関して，局所性があり，ある枝を選ぶとその下には多くのノードがあるのに，解がまったくなく，別の枝は解ばかりということがある．この場合，深さ優先探索で，最初に解のない枝を選んでしまうと，その下のノードすべてをチェックしてからでないと，解にたどり着かない．
- 局面を評価する良いヒューリスティック関数がない．そのため，最良優先探索が行えない．

という条件では，この方法が有力になる．この方法は，プログラムとしても簡単になるというメリットがある．この方法の代表的なものとしては，Iterative Sampling[3]がある．

なお，この方法も深さ優先探索や幅優先探索と組み合わせることは可能である．

■不完全情報パズルの解法

不完全情報パズルの場合は，人間のトッププレイヤのレベルに達しないことも多い．よく用いられる探索アルゴリズムをいくつか挙げる．

探索による完全解

不完全情報ゲームであっても，隠されている情報の量が少ない場合には，その情報の事前確率を仮定した上で，全探索により，最適な戦略を決定することができる場合がある．

この手法によって解かれた例としては，マスターマインドがある．4 桁 6 色のマスターマインドの最適な（質問の平均回数を最小にする）戦略は，1994 年に求められた[4]．マスターマインドは**図 -2** のようなボードを利用するゲームである．出題者が指定した色の配列を当てる．

- 6 色のカラー・ピンを使った 4 桁の色の配列を当てる
- 同じ色を 2 回以上（リピートと呼ぶ）使った組合せも許す．

図 -2　マスターマインドのゲームボード

- 解答者は正解と思われる色の配列を提示する．出題者は，色と桁の両方が一致している数だけ黒い判定ピンを差す，桁が違うが同じ色があるものについては白い判定ピンを差す．
- 解答者はなるべく少ない回数で正解することを目標とする．

一般的な 4 桁 6 色のマスターマインドでは，初期状態が $6^4 = 1{,}296$ 通りしかなく，対称性を考慮するとさらに減らすことができる．また，一手進めるごとに可能な状態が減っていき，最善の戦略での平均手数が非常に短い．そのため，全探索が可能になっている．

マスターマインドの類似ゲーム MOO（正解と質問中にリピートを許さない）は初期状態が ${}_{10}P_4 = 5{,}040$ 通りだが，1996 年に同様の方法で解かれた．

評価関数の利用

ある手を選択した後の局面を評価する良い関数があれば，可能な手をすべて生成した上で局面の評価値を最大にする手を残せばよい．この関数としては，人間が設定する場合と，学習や統計的データを元に自動的に決定する方法と 2 種類考えられる．

これがうまくいった例としてカルキュレーションがある．カルキュレーション（calculation）☆3 は以下のようなルールのトランプ（カード）の 1 人遊びゲームで

☆3「カリキュレーション」，あるいは「計算」という名前で呼ばれることもある．

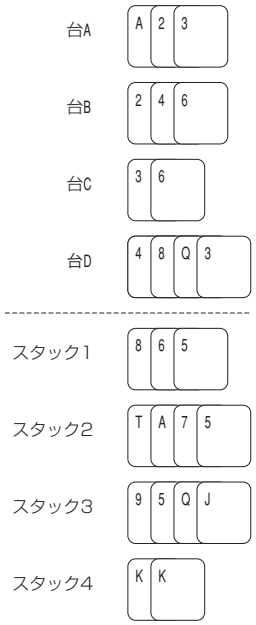


図 -3　ゲーム中の様子

ある．ゲーム中の様子を**図 -3** に示す．

目標　4 つの台に次のような列を左から右に順に置いておくこと（10 は T と記述する）．

台 A：A, 2, 3, 4, 5, 6, 7, 8, 9, T, J, Q, K

台 B：2, 4, 6, 8, T, Q, A, 3, 5, 7, 9, J, K

台 C：3, 6, 9, Q, 2, 5, 8, J, A, 4, 7, T, K

台 D：4, 8, Q, 3, 7, J, 2, 6, T, A, 5, 9, K

各台は数字が等間隔（間隔は台ごとに 1, 2, 3, 4 となっている）に並んでいて（13 を超えると 13 の剰余をとる），すべての台で最後に K がくる．数字だけが問題で，カードのスーツは問わない．それぞれの台で，左から何番目に現れるかを以下では順位と呼ぶことにする．カード 2 の台 A における順位は 2，台 B における順位は 1，台 C における順位は 5，台 D における順位は 7 となる．

スタック　作業用に先入れ後出し操作が許されるスタック[☆4]を用いる．使用してよいスタック数は 3 または 4 とするのが一般的である．

開始　52 枚のカードをよく切って，裏返しにしたままで山に置く．以下，山に含まれるカードの集合を山札と呼ぶ．台，およびスタックの初期状態は空になっている．

山から引く　山から 1 枚引いて手に持つ．次のいずれかの操作を行う．

台に出す　4 つの台のうちのどれかに出すことができるときは，出してもよい．

スタックに置く　台に出さなかったときは，いずれかのスタックに表向きに置く．

これを済ませたら，次にはカードを移すことができる．

カードを移す　スタックの先頭のカードのうちに，台に出すことのできるものがあれば，それを移してもよい．これは，何回でも行えるし，まったく行わなくてもよい．そしてまた山からカードを引く．

終了の判定　台の 4 つの列が完成し，表面にキングが 4 枚並べば成功．山が空なら，スタックから台にカードを移せるだけ移す．どうしても移せず，スタックにカードが残り台が完成しなければ失敗となる．

カルキュレーションに関しては，花澤が提案した人手で作成した評価関数によって，4 スタックで 93% と人間のエキスパートに迫る成功率が実現されていたが[5]，筆者は部分ゲームの解析結果を元に作成した評価関数で 4 スタックで，約 99% の成功率を達成した[6]．

乱数による可能局面の生成

可能局面の数が多く，列挙するのが困難なパズルの場合は，乱数で現在の状態と矛盾のない局面を複数個作成して，そのセットに対して最適な行動を探索によって求める方法が考えられる[7]．これは random sampling と呼ばれる手法で，プランニングの世界では一般的だが，パズルを解くのに使ったという報告例は多くない．

この方法は，生成する局面数が多いほど精度が上がるが，その代わりに探索のコストが高くなるという欠点がある．また，可能な探索深さでは，ゴールにも手詰まり局面にもたどり着かない場合は，評価関数と組み合わせないと行動を決定することができない．

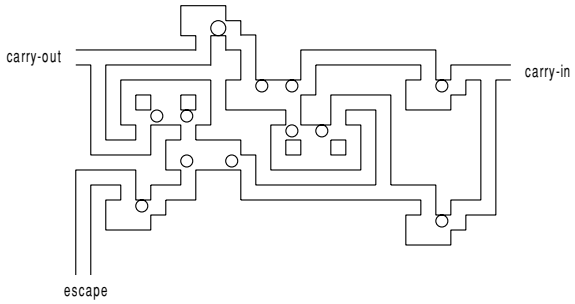
■計算量

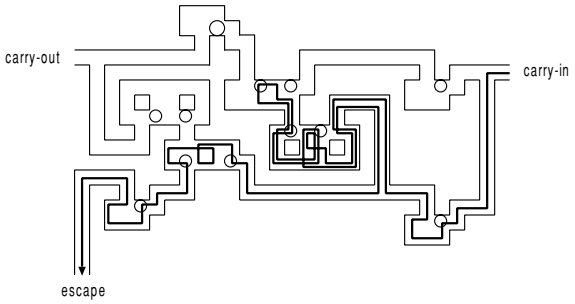
パズルゲームの大部分は，NP-hard であることが知られている．以下にいくつかの例を挙げる．

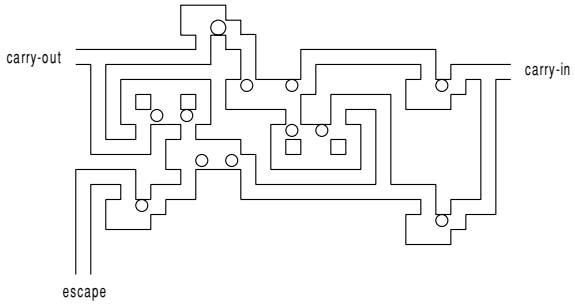
・倉庫番

実は，一般の倉庫番パズルは PSPACE 完全問題であることが知られている[8]．PSPACE 完全とは問題のサイズに対して多項式オーダのメモリを使用して解ける問題の中で最も難しい問題のクラスに属するということである．NP 完全問題も PSPACE 問題に含まれるので，PSPACE 完全問題は NP 完全問題よりも難しいと考え

☆4 場あるいは屑とも呼ばれる．







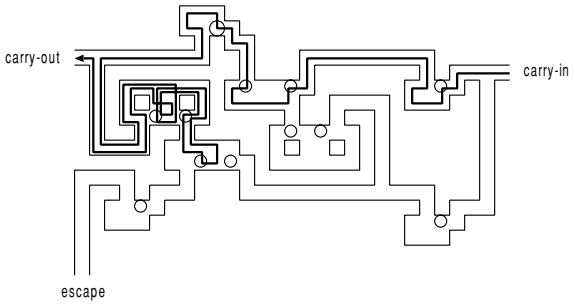


図 -4　1 ビットカウンタに対応する倉庫番パズル

図 -5　1 ビットカウンタ間の遷移

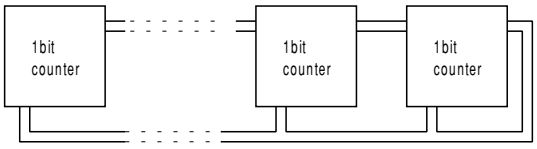


図 -6　n ビットカウンタに対応する倉庫番パズル

られている☆5.

倉庫番パズルが難しいことを示す問題の例を見てみる．**図 -4** は文献 8）で紹介されている One Way Device, Reverser, Pass-Reset などの部品を組み合わせて作成した 1 ビットカウンタである．上が 0 を表す状態で，下が 1 を表す状態となる．状態 0 からは**図 -5** の上のパスに従って carry-in から escape までを抜けることで状態 1 に遷移することができる．それ以外のパスを通ると残された部分が手詰まりのまま残ってしまう．また状態 1 からは図 -5 の下のパスに従って carry-in から carry-out に抜けると状態 0 に遷移することができる．

この 1 ビットカウンタを**図 -6** のように N 個つなげたものを考える．初期配置はすべての桁が 0 であり人が一番右のカウンタの carry-in にいるものとする．ゴールはすべての桁が 1 の状態とする．1 周して出発地点に戻ってくるごとに 2 進数としてみなしたときに 1 増えることになるので，ゴールに至るには 2^N-1 周回らなければいけないことが分かる．つまり問題のサイズに対して最短の正解手順が指数オーダになっている☆6.

・テトリス

一般のテトリスは幅 10 となっているが，ボードサイズを大きくした上で，落ちてくるブロックのシーケンスをあらかじめ与えて完全情報化することを考える．この上で，いくつかの決定問題（たとえば「M

☆5 P ≠ NP が未証明なのと同様に NP ≠ PSPACE もまだ証明されていないので，「より難しい」と言い切ることはできない．

☆6 正解手順が指数オーダになること自体は，PSPACE 完全問題であることの証明にはなっていないが，関係はある．

line 消すことが可能か」）がボードのサイズに関して NP-hard であることが証明されている.

これらの証明は，数学的にはともかく，ゲームプログラミングにおいては問題が NP-hard であることは，多くの場合前提になっている．また，逆に NP-hard であることの証明をされても，実際のゲームとは掛け離れた仮定の上で行われている場合も多い．たとえば，テトリスでは幅がゲームバランスのためか，幅が 10 以外のボードでは行われることは少ない.

■まとめ

完全情報のパズルでも人間のレベルに達しないものとしては倉庫番パズルがあるが，少数にとどまっていて，完全情報のパズルの多くは，全探索や最良優先探索により，人間のエキスパートを超えるレベルで解くことができるようになってきた. メモリとディスク容量の増加（および，ビット単価の減少）により，そのうちのいくつかは，個人の PC 上でも完全解を求めることが可能になってきている.

不完全情報のパズルに関しては, 探索ベースだけでは，展開するノードが多くなりすぎてしまうため，パズル特有の性質を人間が与えたり，学習によって獲得させるなど，研究の余地は残っている．ただし，むしろ，最適化アルゴリズムの研究の際に，評価の題材として扱われることの方が多いかもしれない.

参考文献
1）村瀬芳生，松原　仁，平賀　譲：「倉庫番」の問題の自動作成，第 38 回プログラミングシンポジウム資料集（1997）.
2）Korf, R. E.: Depth-first Iterative-deepening: An Optimal Admissible Tree Search, Art. Intell., Vol.27, pp.97-109（1985）.
3）Harvey, W. D. and Ginsberg, M. L.: Limited Discrepancy Search, Proceedings of IJICAI, Vol.1, pp.607-615（1995）.
4）Koyama, K and Lay, T. W.: An Optimal Mastermind Strategy, J. Recreational Mathematics, Vol.25, No.4, pp.251-256（1994）.
5）花澤正純：カリキュレーション（計算），bit 別冊「ゲームプログラミング」，共立出版社，pp.109, V117（1997）.
6）田中哲朗：部分ゲームの解析結果を用いたカルキュレーションの戦略，情報処理学会論文誌，Vol.43, No.10, pp.3064, V3073（Oct. 2002）.
7）Eckhardt, R.: Stan Ulam, John von Neumann, and the Monte Carlo Method, Los Alamos Science, Special Issue（15）, pp.131-137（1987）.
8）Culberson, J.: Sokoban is PSPACEcomplete, Technical Report 97-02, Department of Computing Science, University of Alberta, http://www.cs.ualberta.ca/~joe/Preprints/Sokoban（1997）.
（平成 15 年 6 月 26 日受付）