数独の問題作成支援システムの設計と開発

前田一貴† 奥乃博‡
†京都大学 工学部 情報学科 ‡京都大学 大学院情報学研究科 知能情報学専攻

1 はじめに

ペンシルパズルの一種である「数独 (ナンバープレース)」はかつてから日本ではよく知られていたが，近年世界的なブームとなっており計算機科学や脳科学の面から様々な解析が行われている．数独の自動解法は，その問題規模 (9×9 の場合) とルールの単純さから比較的容易であり，種々の高速解法システムが公開されている．一方，数独の問題自動作成では，見た目の美しさと問題の難易度とのバランスが難しく，良質な問題の作成はパズル作家の手に委ねられている．最近になって非常に良質・高速な自動生成エンジンの開発が報告されている [1].

我々は，数独の難易度と問題構造との関連を解明することを中心課題としながら，作成途中の数独問題についての様々な情報を提供することで問題の作成支援をするシステムを開発している．ここで，提供する情報としては，各マスに入り得る候補数字や，解とヒントの依存関係，問題の難易度といった情報が挙げられる．本稿では，こうしたシステムの設計方針と開発について報告する．さらに，問題の自動生成への展開についても述べる．

2 数独の問題定義と計算機による解法

数独とは，9×9 に分割された正方形の各行，各列，3×3 に区切られた各ブロックの各マスに 1 ～ 9 の数が 1 つずつになるように埋めていくパズルである．最初に何も数が入っていない場合，上記を満たすような数の入れ方は無数に存在する．そこで，いくつかの数 (20 ～ 30 個程度) をヒントとして与え，解がただ 1 つに定まるようにする．このヒントの配置のことを問題と呼ぶ．図 1 はそのような問題の例である．数独のパズルとしての面白さは，このような単純なルールから論理的に導かれる様々な関係を駆使して，なんとかして解を捻り出すことにある．

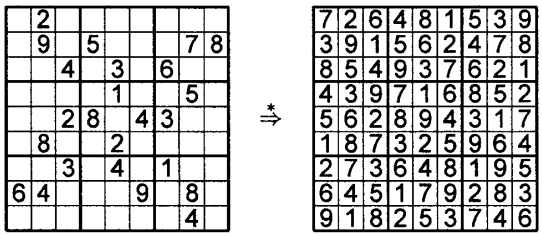

図 1 数独の問題例

数独問題の自動解法で，真っ先に思いつくのはバックトラック法 (深さ優先探索) であろう．実際にバックトラック法を用いれば，探索順を候補の少ないマスを優先するなど工夫することで，最近の計算機であればどのような問題でも一瞬で解くことが可能となる．さらに，Knuth の Dancing Links と呼ばれるデータ構造 [2] を応用することで，与えられた (複数解のものを含む) 問題の全解を効率よく列挙することができる．

一方，人手で解くために探索を効率化するための関係として，下記のようなものが知られている．([3] がよくまとまっている).

- Naked Singles: そのマスに入りうる数が 1 つしかないとき，その数が入る．
- Hidden Singles: ある行，列，ブロックである数の入りうるマスが 1 つしかないとき，そのマスに入る．
- Naked Pairs: ある行，列，ブロックで，ある 2 つの数のみが候補であるようなマスが 2 つしか存在しないとき，それ以外のマスにその 2 つの数は入らない．
- Hidden Pairs, X-Wing, XY-Chain, Coloring, 等.

直ちに入る数が決まる自明な上 2 つの関係以外は，候補を減らすための，人手で探すのは困難な関係であり，実際にそうした枝刈り法を使わなければ解けないような問題も数多く存在する．また，こうした知られている枝刈り法を総動員しても解けないような「超難問」の存在も知られており，これはバックトラック法で解く以外にない．どの枝刈り法が有効かを調べることにより，問題の難易度をある程度決定することができる．

3 問題作成支援システムの設計

数独問題の自動解法が容易であるのに対して，問題作成の難しさは次の 3 つの課題にある:

(1) 数独問題の解の一意性の保証,
(2) ヒント配置の美しさ,
(3) 難易度の制御.

一般にヒント配置は，中心について点対称な問題が好まれる傾向にある (図 1 もその例)．難易度制御は，そもそも難易度の定義が不可欠であり，これまでに難易度の定義に取り組んできたが，まだ決定的なものは得られていない．本稿では，候補数の多いマスの数を制御することで，間接的に難易度を制御する．

問題作成の基本手順は生成検証法 (generate-and-test) である．まず準備として，マスに入れる数の候補を 2 つに分類する:

- Dead: その数が入ると解なしになる．
- Active: その数が入るような解が存在する．

これを候補の死活と呼ぶ．この分類を用いると，2 章で述べた枝刈り法とは Dead 候補を探す方法であり，問題を作成するということは全ての空きマスの Active 候補がただ 1 つになるようにすることであると言える．

以上の考察を踏まえると，数独問題作成支援システムとして提供すべき機能は，以下の通りである．

(1) ヒントを配置するマスの指定・表示.
(2) 実際にヒントを配置し，それに対応して空きマスに入る数の候補を表示．ここではもちろん候補の死活も表示．これにより，「ヒントを配置していった結果，解なしの問題ができてしまった」という状況を防ぐとともに，どのマスが一意に定まり，どのマスがまだ決まっていないのか，という情報を随時得ることができる.
(3)「次にどのヒントを配置すべきか」を決める手助けとし

Design and Implementation of Sudoku Puzzle Constructing Support System: *Kazuki Maeda and Hiroshi G. Okuno (Kyoto University)*

て，ヒントとして数を配置すると候補の死活がどう変化するかを表示．

(4) 既に一意に定まった (Active 候補がただ 1 つの) マスは色を付けてわかりやすくする．

(3) を実現するために，各マスをさらに 9 分割してそれぞれに 1 ～ 9 の数を割り振り，各数をマウスでポイントすると，その数をヒントとして置いた結果どの候補が Dead になるのかを色を付けて表示する．そうすると，数をクリックするとそのマスにヒントとして配置したり外したりできるというのが自然であろう．(2)(3) で候補の死活求める問題は，2 章で述べた問題を解くアルゴリズムを使用する．

ある候補の死活を調べるためには，その候補をヒントとして加えた問題が解を持つかどうかを調べればよい．問題作成途中でこの操作をすると，もしその候補が Active ならば一般には大量の複数解を持つが，ここでは解を持つことがわかれば十分なので，解のうちの 1 つが得られた時点で打ち切る．また，こうして得られた解に含まれる他の空きマスの数についても Active であることがわかるので，もうその数については調べる必要がなくなる．また，Dead ならば解が存在しないので，可能性を全て調べれば必ず判定できる．これを全ての候補について繰り返せば，全候補の死活が得られることになる．

4 実装

以上に述べたような機能を持つシステムを，なるべく使いやすいインターフェースを目指して，実際に開発を行なった．開発言語は Java を用いた．

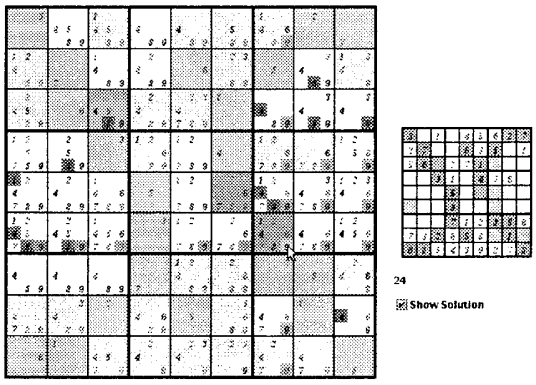

図 2 問題作成支援システムの使用例

図 2 は実装したプログラムの実行画面であり，(6, 7) にヒントとして 9 を置こうとしている場面である．ピンク色が候補から除かれる数を，赤色がヒントを置くことで Dead になる候補を表わす．候補のうち青色の数が Active 候補，灰色の数が Dead 候補，背景が黄色いマスはヒントが置かれているマス，薄い黄色のマスは入る数が一意に定まったマス (つまり全てのマスが黄色か薄い黄色になれば問題の完成となる)，緑色のマスは作成者が指定したヒントを置く予定のマスとなっている．また，右の小さな盤面は補助のために表示しているもので，黄色のマスがヒントを置くことで新たに一意に定まるマスを示している．これより，例の場面では (6, 7) に 9 をヒントとして加えることで，(6, 7) を含めて 6 個のマスが確定することがわかる．

本問題作成支援システムを実際に使用してみると，Dead の候補にヒントを置くことを防止することにより解なし問題ができることがなくなるため，あまり数独に詳しくない人が使ったとしても適当にヒントを置いていくだけで問題が作れることがわかった．面白いのは，このシステムではヒントを外すときの変化も見ることができるようになっているので，試しに既に完成した数独の問題を入れてみると，各ヒントと解の一種の依存関係を見ることができる．この機能を用いると，数独の問題構造の解析への応用が期待できる．

5 問題自動生成への応用

本問題作成支援システムで用いた死活判定を問題自動生成へ応用した．すなわち，ヒントを配置するマスを指定し，それらのマスを Active 候補がなるべく少なくなるものを優先して選んで順に埋めていくことを考えた．もし結果が複数解，つまり 2 つ以上 Active 候補が存在するマスが残った場合はやり直すようにすれば，何度か繰り返せば一意解の問題が得られることが期待できる．

しかし，実際にこの方法を試したところ，Core 2 Duo E6700 2.67GHz マシンにおいて 24 ヒント問題生成にパターンにもよるが 1 問平均 3～4 秒を要した．そこで，高速化のために以下のような改良を行った．

(1) 最初の半分程度は Active 候補を減らすことは考えず，ランダムに入れるヒントを選ぶ．この際，一切の死活判定は行わず，最低限の候補の計算のみを行う．

(2) 残りのマスで Active 候補を一気に減らす．

(1) で死活判定を行わなくても問題がうまく作れるのは，最初の 10 個程度のヒントを適当に埋める間に Dead 候補が出現する可能性は低いためである．これで，24 ヒントの問題を 1 問作るのに要する時間は平均 1～2 秒となり，同じパターンで比べると約 200% の高速化が見られた．ただ，1 問生成に要する試行回数を見ると改良後は 1.5 倍程度と増加傾向で，回数のばらつきも大きくなっており，1 度の試行で生成に成功する確率が低下していることがわかる．これは，Active 候補を減らす機会が減ったことによるものと考えられる．

本手法の問題点は，ヒント数を 24 未満にすると生成が急激に困難になることにある．また，25 ヒントならば多くのパターンで生成できたものの，24 ヒントでは生成できない，もしくは生成に時間を要するパターンも多いことが実行時間の測定実験中に顕在化した．これについては問題の所在を解明し，更なる改良を行う必要がある．

なお，本自動生成システムにはヒントを置く場所のパターンを自動生成する機能をつけた．これにより，ヒントの配置から自動で問題を生成することが可能である．

6 おわりに

本稿では，計算機による数独問題の作成支援について述べてきた．今後は先に述べた問題自動生成法の更なる改良，及び枝刈り法に基づく簡易的な難易度判定機能を洗練，発展させる，さらにこれまでにない視点からの数独の解析などが課題である．

なお，本稿で解説したシステムを下記 URI で公開する:

`http://winnie.kuis.kyoto-u.ac.jp/SUDOKU/`

謝辞: 本研究の一部は科学研究費補助金萌芽研究，及び中山隼雄科学技術文化財団の支援を受けた．

参考文献

[1] Puzzle Generator Japan, `http://puzzle.gr.jp/`

[2] Donald E. Knuth, "Dancing Links", `http://arxiv.org/pdf/cs/0011047`

[3] Tom Davis, "The Mathematics of Sudoku", `http://www.geometer.org/mathcircles/sudoku.pdf`