数独パズルを解くSASプログラム

知平　菜美子
兵庫県立大学・経済学部 4 回生

周防 節雄
兵庫県立大学・学術情報館

Sudoku Solving System Coded in SAS

Namiko Chihira
Faculty of Economics,
the University of Hyogo

Setsuo Suoh
Systems Information Centre,
the University of Hyogo

要旨

数独パズルを解く SAS プログラム（Sudoku Solving System: SSS）を開発した。数独パズルの局面を SAS データセットに表現しようとすると、9×9 のマトリックスの形状を採用するのが一般的である。つまり、9 変数、9 オブザベーションの SAS データセットとなる。パズルのルール上、横 1 列、縦 1 列、3×3 の 9 個のブロックのそれぞれに、1〜9 の数字を当てはめていくことになるが、SAS では原則的に横一列、つまりオブザベーションごとにしか計算処理できないので、本来 SAS で解くには不向きな問題である。本論文では、初期のマトリックス状の SAS データセットからパズルを解くのに必要な複数の SAS データセットを作成し､必要な処理を施した上で、それらのデータセットの情報を統合することで、パズルを解き進めていく手法を採用した。局面が進展するにつれて、同じ解法プログラムを繰り返し実行する必要があり、%include を使うことによってそれを自動的に行うことを可能にした。SAS 言語には副プログラムの概念はないので、マクロ言語を実質的にその代用品として活用した。今回開発したプログラムは、ほとんどの数独パズルを解くことに成功している。

キーワード : データハンドリング、SAS マクロ言語、%include 文、再帰的プログラミング

１．はじめに

我々は数独パズルを解く SAS プログラム（Sudoku Solving System: SSS）を開発した。このパズルのルール上、横 1 列、縦 1 列、3×3 の 9 個のブロックのそれぞれに、1〜9 の数字を当てはめていく作業が必要になるが、SAS では原則的に横一列、つまりオブザベーションごとにしか計算処理できないので、本来 SAS で解くには大変不向きな問題である。更に、SAS にはサブルーティンの概念がない上に、プログラム全体を繰り返し処理することも普通にはできない言語構造である。これらの難点を克服するために、SAS マクロ言語を最大限に活用して、一種の再帰的プログラミング技法を採用した結果、極めて興味深い構造の SAS プログラムを作成することができた。

２．数独の簡単な説明

数独(日本のパズル制作会社ニコリの登録商標)とは、日本では「ナンバープレイス（ナンプレ)」とも呼ばれているペンシルパズルである。数独問題は図 2.1 に例示する。世界的にも有名なパズルであり、その問題は様々な雑誌や新聞などにも掲載されている。また、世界パズル連盟の主催する、パズルの国際大会である世界パズル選手権にも毎年出題されている。このパズルは 9×9 の空いているマス目に 1～9 の数字を入れていく。基本ルールは図 2.2 で示している「行」「列」「箱」のそれぞれに 1～9 の全ての数字を入れていかなければならず、グループ内で数字が重複してはいけない。「行」は横列、「列」は縦列、「箱」（box）は 3×3 で囲まれた 9 個のマス目であり、全部で 9 個ずつある。

			8		1			
		6		3		9		
	7						3	
6		7				5		4
			2		6			
2				7				8
			4		5			
3		8				4		2
	1		3		9		8	

図 2.1 数独問題

出典：(株式会社ビデオ出版、SUPER ナンプレポータブル vol.1)

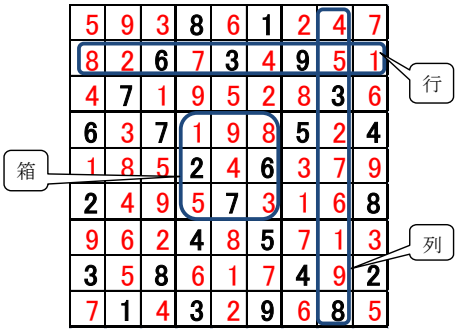


図 2.2 数独解答

３．SAS で数独を解く上での問題点

SAS で数独を解く際の問題点は、SAS では原則的には同一オブザベーション内での演算しか行うことができないという点である。数独という問題の性質上、行・列・箱という 3 つのグループについてそれぞれ演算を行い、解き進めていかなくてはならない。SSS では、問題を解き始める前にオリジナルのデータセットから行・列・箱ごとに容易に処理が可能となるデータセットを作成し、これらのグループの処理結果を合成して、各マス目の候補ナンバーを見つける方針を採用した。

４．SAS プログラムの解説

今回の SSS は次の 4 つのプログラムからなる。

①sudoku_macro.sas

②puzzle_make_original_data.sas

③puzzle_solve_final.sas

④reconstruct.sas

これらのプログラムの詳細は 4.3 で述べる。

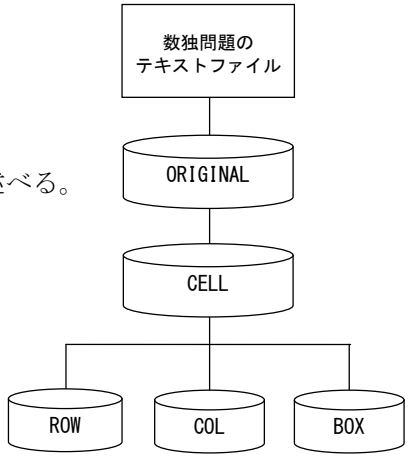


図 4.1 SAS データセット作成手順

```
000801000
006030900
070000030
607000504
000206000
200070008
000405000
308000402
010309080
```

図 4.2 テキストファイル

4.1　データ表現方法

まず、SAS で数独を解くためには、図 4.1 に示す様に、問題を入力し必要な SAS データセットを作成する。

はじめに、9 行 9 列の初期局面をテキストファイルに入力する。例えば、先の図 2.1 の問題は図 4.2 に示すようにテキストエディタで作成する。「0」は空いているマス目を表す。このテキストファイルを SAS データセット(ORIGINAL：9 変数・9 オブザベーション)に変換する(図 4.3)。空いたマス目に数字が確定されるたびに ORIGINAL に追加され、次なる処理に引き渡される。この ORIGINAL から、81 個の全てのマス目の位置情報（行ナンバー、列ナンバー、箱ナンバー）と数字の値を変数とするデータセット CELL(81 オブザベーション)を作成する(図 4.4 中央)。各マス目の箱（box）番号は図 4.4（左端）に示す。この CELL を使って、同一の行・列・箱に属するセルを同じオブザベーションにまとめて、それぞれ 3 つのデータセット ROW、COL、BOX に保存する(図 4.4 右)。これら図 4.4 にある 4 つのデータセットは SAS プログラム reconstruct.sas により作成される。このプログラムは sudoku_macro.sas のプログラムの中で定義されている SAS マクロのうち、all_filter と verify の中で%include で組み込まれており、ORIGINAL が更新されるたびに、自動的にこの 4 つのデータセットも更新される。

OBS	a1	a2	a3	a4	a5	a6	a7	a8	a9
1	0	0	0	8	0	1	0	0	0
2	0	0	6	0	3	0	9	0	0
3	0	7	0	0	0	0	0	3	0
4	6	0	7	0	0	0	5	0	4
5	0	0	0	2	0	6	0	0	0
6	2	0	0	0	7	0	0	0	8
7	0	0	0	4	0	5	0	0	0
8	3	0	8	0	0	0	4	0	2
9	0	1	0	3	0	9	0	8	0

図 4.3 データセット ORIGINAL

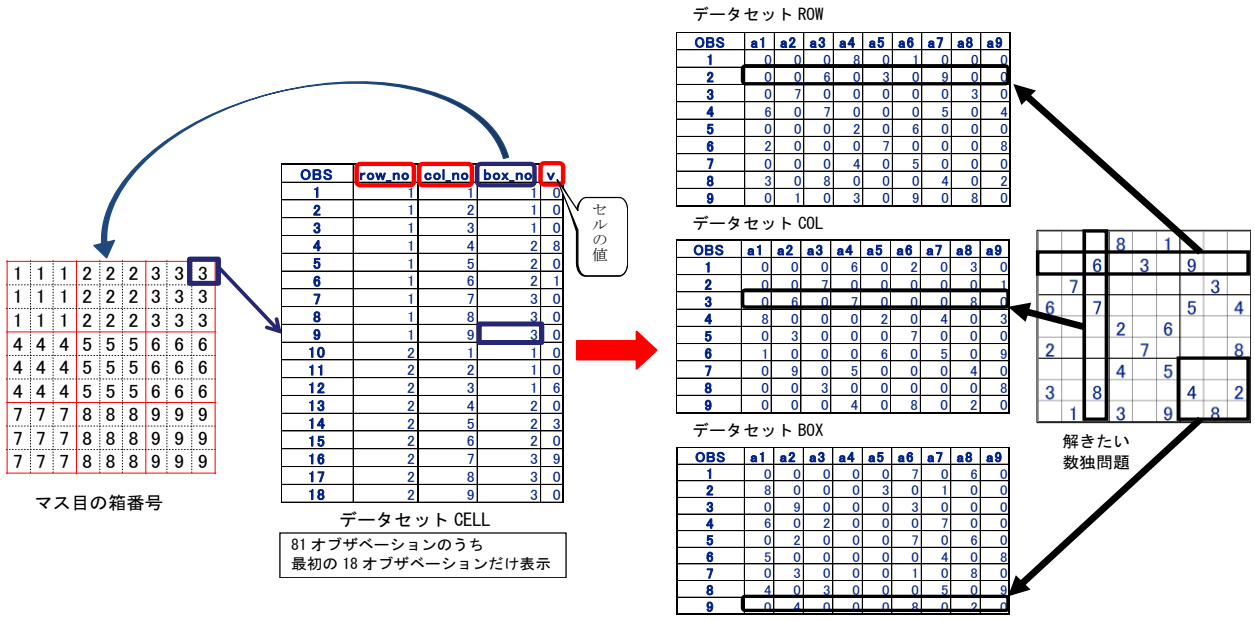


図 4.4 作成されたデータセット

4.2　フィルタ

SSS では始めにマス目毎に対応する数値変数 filter を作成し、それを基に正解の数字の特定作業が行われる。つまり filter とは思考のキーとなる最も重要な情報であると言える。具体的には、filter は空いているマス目が所属する行・列・箱のグループのいずれかで既に使用されている数字を使って最大 9 ケタの整数で表現したものであり、そこに含まれている数字はそのマス目では使えない。filter は $[x_1 x_2 \cdots x_9]$と表され、$x_i = 0$または i となる。例えば filter の値が「123406780」の場合、5 と 9 が使用できる、つまり候補ナンバーになる。

4.3　プログラムの構造

SSS を構成する 4 つの SAS プログラムの関係は図 4.5 に示す通りである。

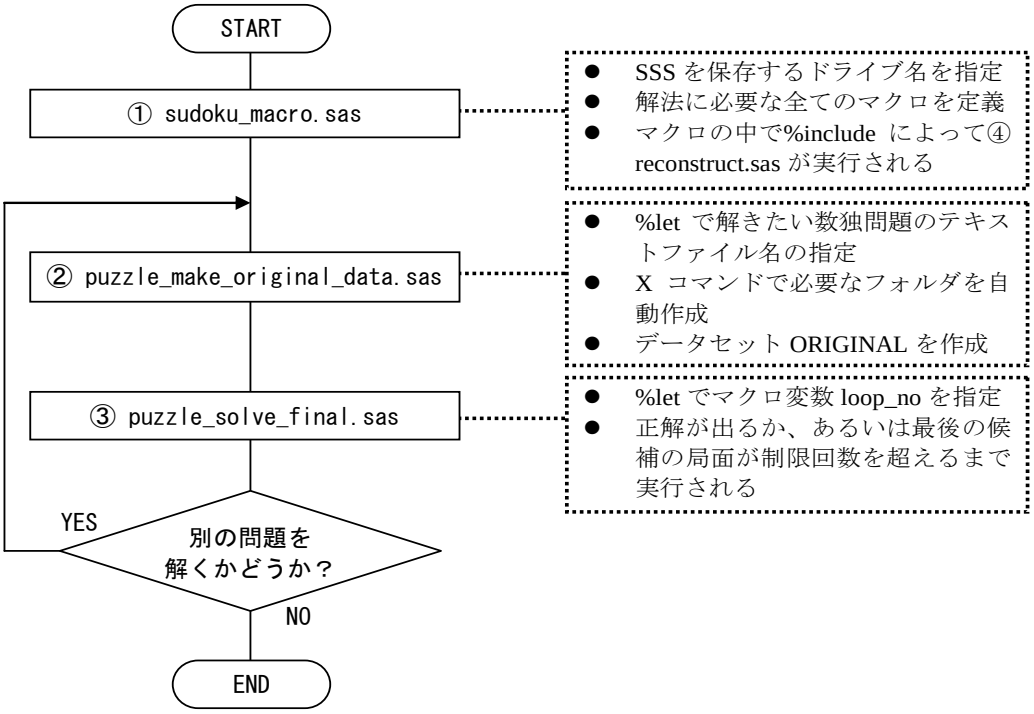


図 4.5　プログラム実行手順

① sudoku_macro.sas

ここには数独の解法に必要なアルゴリズムが役割別に SAS のマクロ言語で記述されており、29 個のマクロを定義している。利用者はこのプログラムの冒頭で%let によって SSS を構成する 4 つの SAS プログラムを保存するドライブ名を指定しなければならない。このプログラムは SAS セッションの始めに一度実行するだけでよい。詳細は周防・知平(2011)を参照されたい。これらのマクロの実行は③puzzle_solve_final.sas において行われる。

② puzzle_make_original_data.sas（付録 1 参照）

このプログラムでは解きたい数独問題をあらかじめテキストエディタで作成したテキストファイルを読み込んで、データセット ORIGINAL を作成する。そのためにプログラム冒頭で%let によって解きたい数独問題のテキストファイル名を指定する（拡張子は指定しなくてよい）。また、X コマンドで、DOS 命令が実行され、①sudoku_macro.sas で指定したドライブ（例えば C ドライブ）に、次の 2 つのフォルダが自動的に新規作成される。

(1) アウトプット画面に出力される情報のうち、ODS 機能を使って html 形式で保存する情報を格納するフォルダ[C:¥sudoku¥output_window]

(2) ログ画面の全情報(log.txt)、及び(1)以外のアウトプット画面の情報(output.txt)を保存するためのフォルダ[C:¥sudoku¥log_window]

一般的に SAS プログラムを実行すると、ログ情報が LOG 画面に、処理結果が OUTPUT 画面に表示される。SSS の実行をこの通常の方法で行うと、%include 文で何度も同じプログラムを書き加えていくので、LOG 情報の量が表示限度を軽く超えてしまい、あふれの警告情報が頻繁に出現して、実行の中断が起きる。しかし LOG 情報はプログラム開発中は貴重なディバッグ用情報として不可欠である。そのため、proc printto を使っ

て上の(2)のフォルダの中に外部のテキストファイル log.txt として自動的に出力・保存している。

OUTPUT 画面の情報も、プログラム開発中は動作の確認のため必要となる。その情報の中には、数独を解いている際のポイントとなるいくつかの局面だけを別個保存して、解法の際の進行状況を容易に確認できるようにしたい情報もある。また、SSS のエンドユーザの立場では、途中結果は重要ではなく、最後の結果の解答だけを見たい場合もある。従って、OUTPUT 画面の情報は 3 種類の異なる形で保存している。解法の際の局面の進行状況を確認できる情報は、上の(1)に保存している。最終結果の解答だけは、OUTPUT 画面(図 4.6)に直接表示している。その他の情報は(2)のフォルダに output.txt として保存した。上の(1)で html 形式で保存する際は、マクロ変数 round、which、new_which の値をファイル名に使い、実行中のどのタイミングで作成された OUTPUT 画面情報であるかが分かるようにした。例えば「round5_4_3.xls」(実際は html ファイル)はマクロ変数 round=5、which=4、new_which=3 の際に出力した局面を表している。この局面はラウンド 5 を実行中に現れた局面であることが分かる。which と new_which は作戦③と作戦④においてそれぞれ候補局面を設定するために使用している変数で、新しく局面を設定すると 1 増えるので、実際はその数字から 1 少ない値の時に出現したことを意味する。つまり、which=3(作戦③の第 3 番目の局面)から更に作戦④に進んだ時の new_which=2(作戦④の第 2 番目の局面)を表示している。

③ puzzle_solve_final.sas（付録 2 参照）

このプログラムの冒頭の%let でマクロ変数 loop_no をユーザーが指定する。敢えて指定しない場合のデフォルト値は 15 である。このマクロ変数 loop_no の値は各局面においてこのプログラムの実行を繰り返す回数の上限値を意味している。この値が小さすぎると、正解に近づいているにもかかわらず、回数制限のために途中で打ち切られる可能性がある。実際に数独を SSS に解かせた実験結果からデフォルト値 15 あたりが妥当と思われる。

このプログラムの冒頭では、%eval 関数によってマクロ変数 new_round と round の値も自動的に 1 ずつ増やされる。マクロ変数 new_round はこのプログラムの実行の開始から終了までにこのプログラムが繰り返し実行された総回数であり、マクロ変数 round は各局面においてこのプログラムを繰り返し実行した回数を表している。4.5 節で述べる作戦③や作戦④で探索の対象となる複数の候補局面（正解の局面かもしれないし、不正解の局面かもしれない）をそれぞれデータセット ORIGINAL としてセットする度にマクロ変数 round はリセットされる。

このプログラムは%include によって自動的に繰り返し実行されるので、各局面において無限ループに陥らないように実行を繰り返す回数をマクロ変数 round で数えておき、その値が loop_no の値を超えるとそれ以上の探索は中止して、次の候補の局面の探索に移る。従ってこのプログラムの実行が始まると、正解が出るか、あるいは最後の候補の局面が制限回数を超えるまで%include で自分自身のプログラムを組み込むことにより、自動的に繰り返し実行される。

④ reconstruct.sas（付録 3 参照）

このプログラムは、引き渡されたデータセット ORIGINAL から 4 つのデータセット CELL、ROW、COL、BOX を作成する。このプログラムは①sudoku_macro.sas の中の SAS マクロ all_filter と verify の中で%include により自動的に組み込まれて実行される。

4.4 プログラムの利用法

SSS を利用する時の手順を以下に示す。

① sudoku_macro.sas の冒頭で%let でプログラムを保存するドライブ名を指定後、サブミットする。

② puzzle_make_original_data.sas の冒頭で%let で、数独問題のテキストファイル名（拡張子不要）を指定し、サブミットする。（データセット ORIGINAL が作成される。）

③ puzzle_solve_final.sas の冒頭で%let によりマクロ変数 loop_no を指定後、サブミットする。

loop_no は指定しなければデフォルト値 15 が設定される。

上記の①②③を行った後、正解が得られれば、アウトプット画面に正解の最終局面が表示される（図 4.6 左)。正解が得られなかった場合は、途中結果として、作戦③で複数の候補局面を作る直前の局面、つまり正解であると確定している数字のみが入った局面が表示される（図 4.6 右)。

引き続き別の数独問題を解く場合は②と③を繰り返す。

正解が得られました

OBS	a1	a2	a3	a4	a5	a6	a7	a8	a9
1	5	9	3	8	6	1	2	4	7
2	8	2	6	7	3	4	9	5	1
3	4	7	1	9	5	2	8	3	6
4	6	3	7	1	9	8	5	2	4
5	1	8	5	2	4	6	3	7	9
6	2	4	9	5	7	3	1	6	8
7	9	6	2	4	8	5	7	1	3
8	3	5	8	6	1	7	4	9	2
9	7	1	4	3	2	9	6	8	5

図 2.1 の問題の実行結果のアウトプット画面

ここまでしか出来ませんでした

OBS	a1	a2	a3	a4	a5	a6	a7	a8	a9
1	0	0	5	3	0	0	0	0	0
2	8	0	0	0	5	0	0	2	0
3	0	7	0	0	1	0	5	0	0
4	4	0	0	0	0	5	3	0	0
5	0	1	0	0	7	3	0	0	6
6	0	0	3	2	0	0	0	8	0
7	0	6	0	5	0	0	0	0	9
8	0	0	4	0	0	0	0	3	0
9	0	0	0	0	0	9	7	0	0

図 5.2 の問題の実行結果のアウトプット画面

図 4.6　実行結果のアウトプット画面

4.5　4 つの作戦の解説

現在までに開発した SSS では 4 種類の作戦を採用している。実行する作戦は図 4.7 に示すように、作戦①から順次適用していき、その結果、新しいマス目に数字を確定することができなかった場合には次の作戦に移る。そこで新たに確定できる数字を見つけた場合は次の作戦に移らず、データセット ORIGINAL をアップデートした後、再び作戦①に戻って実行するように設計した。なぜならば、新たに確定できたマス目が 1 つでも増えることによって、その情報が他の行・列・箱に波及効果を与え、フィルタの更新が行われ、その結果として新たなマス目に数字が確定できる可能性が高くなるからである。

まず作戦①を実行する前に、局面探索のキーとなる変数 filter を作成しておく。

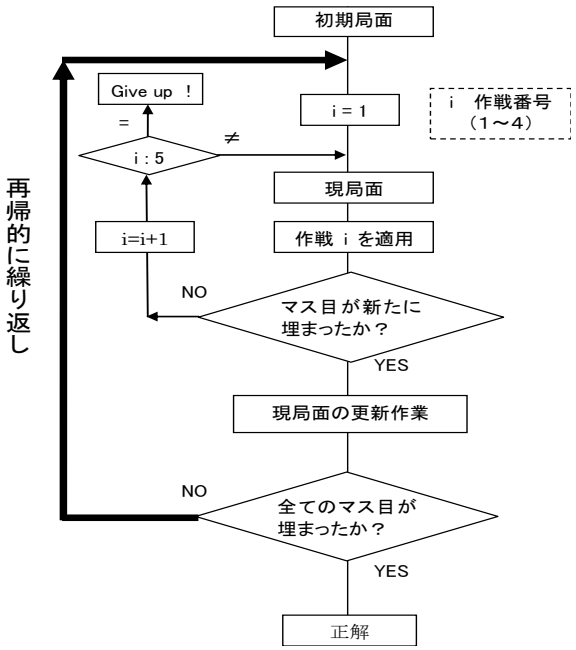


図 4.7　4 つの作戦の流れ

作戦①

この作戦①では数字の特定の方法は 2 つある。

1 つ目は、空のマス目に候補ナンバーが 1 個しか残っていない場合である。この場合は当然その数字が確定する。図 4.8 の(1)のマス目の filter は[123456089]であり、7 以外の数字はこのマス目が所属する行・列・箱のいずれかで使用されている。つまり候補ナンバーが 1 つしかない。従って、この時点でこのマス目の数字

は 7 に確定する。

2 つ目は、1 つのマス目に複数の候補ナンバーがある場合でも、行・列・箱のグループごとに見た時に或る数字がそのマス目だけに候補ナンバーとして残っている場合にも、その数字は確定になる。図 4.8 の(2)のマス目では候補ナンバーは複数あるが、この(2)のマス目のある行を見てみると、(2)のマス目以外の空いたマス目の filter には全て 2 が入っていることが分かる。つまりこの行において、2 を入れることができるのは(2)のマス目だけだということになる。従ってこのマス目の数字は 2 で確定する。

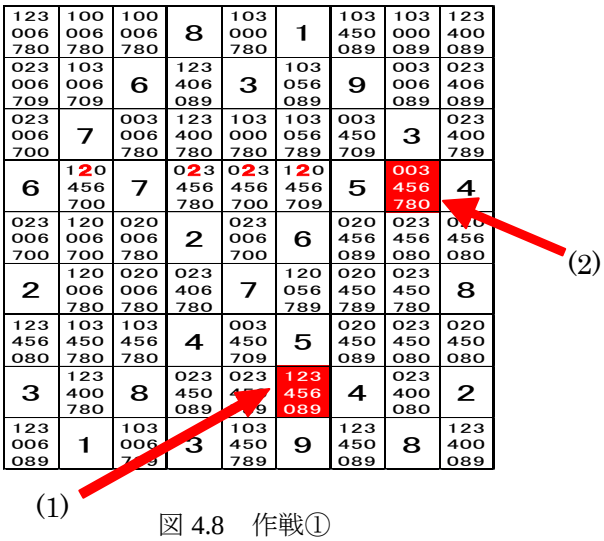


図 4.8　作戦①

この 2 種類の方法により新たに数字が見つかった場合は、空のマス目にその数字を追加した後、ORIGINAL を更新し、プログラム puzzle_solve_final.sas の先頭に戻る。この方法で新しい数字が見つからなくなるまで繰り返した後、次の作戦②へ移る。

作戦②

この作戦は、作戦①で新たな数字が確定できなくなった場合に、filter 情報だけのアップデートを試みる。まず候補ナンバーが 2 つしかないマス目を探す。そしてそのマス目が所属する行・列・箱のグループの中で同じ組み合わせの数字のマス目、つまり同じ filter 情報を持つマス目を探す。同じ数字の組み合わせのマス目が 2 つあった場合はそのグループの中ではもうその数字は他のマス目では使えない。そこでそのグループ内の他マス目の filter にさらにその 2 つの数字を追加し、filter をアップデートする。例えば、図 4.9 で黒いマス目 2 つは同じ行にあり、かつ候補ナンバーが共に 1 と 6 である。つまりこの 2 つのマス目のどちらかに 1 が入り、もう 1 つには 6 が入ることは確定している。従って、この行の他のマス目では 1 と 6 が候補になることはないので、同一行の他のマス目の filter には 1 と 6 を追加することができる（図 4.9 右）。

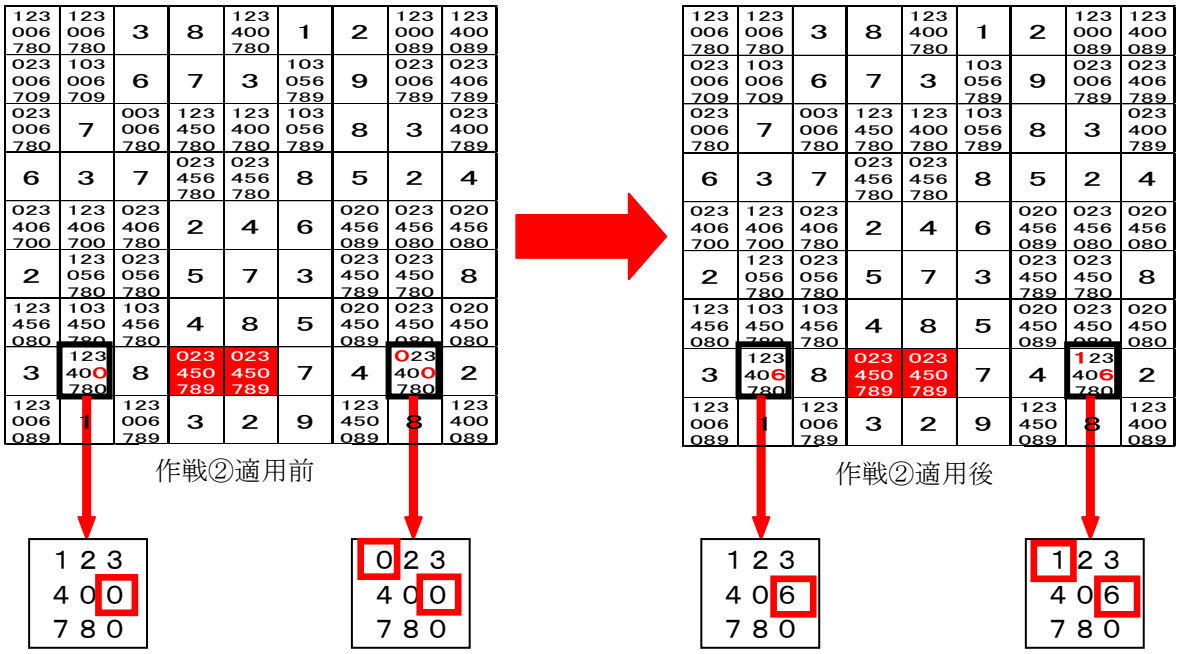


図 4.9　作戦②適用の前と後のフィルタ情報の変化

この確認を行・列・箱の全てに対して行った後、新しくなった filter を使って作戦①を再実行する。その結果、新しい数字が見つかればその数字を追加した後にまた作戦①に戻り、見つからなければ作戦②に進む。作戦②の実行にも関わらず、新しい数字が見つからなければ次の作戦③に移る。

作戦③

作戦②を実行した後でも新たな数字が見つからなかった場合には、正解が必ず含まれる複数の候補局面を作り、それらを一つずつチェックしていく。候補を作るために、現在の局面から候補ナンバーが 2 個あるマス目を行・列・箱のグループとは関わりなく任意の 2 個を取り出す。その取り出した候補ナンバーの全ての組み合わせ、つまり 4 通りの組み合わせの候補局面を作成する。この 4 通りの局面のいずれか 1 つは必ず正解なので、それぞれの候補局面に逐次、作戦①と作戦②を実行する。図 4.10 の場合、黒色のマス目を 2 つ選び、この 2 つのマス目の候補の全ての組み合わせ、つまり(1)のマス目には 1 か 7、(2)のマス目には 3 か 5 の数字を入れた 4 つの候補局面を作成し実行する。

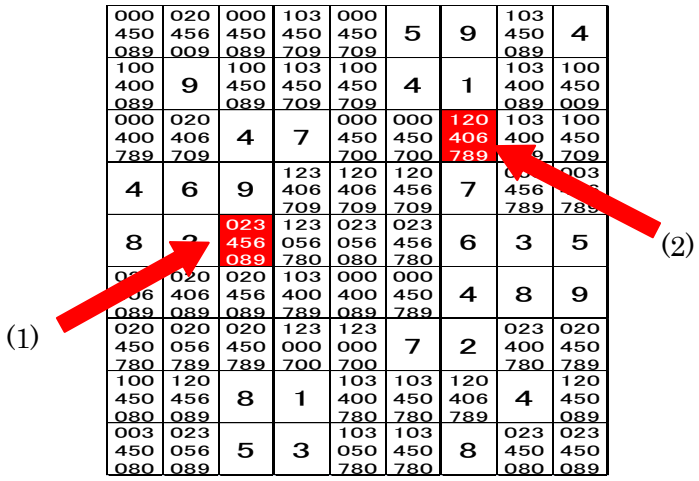


図 4.10　作戦③

その際、途中で誤答が判明するか、局面に進展がなくなった場合はそこで中止して、残りの局面を順次試していく。4 つの局面全てを試しても、正解に達しない場合は次の作戦④に移る。

作戦④

作戦③で試みた 4 つの局面それぞれに対して、更に候補ナンバーが 2 個ある任意のマス目を取り出し、新しく候補の局面を作成する。作戦④では候補ナンバーが 2 個あるマス目が 1 つしかない場合も想定し、マス目を 1 つだけ使用する場合と、2 つ使用する場合のどちらかを実行する。作戦③で作成した 4 通りの組み合わせの局面を 1 つ取り出し、候補ナンバーが 2 個のマス目を列挙する。その様なマス目が 2 つ以上ある場合には、それから任意にマス目を 2 つ取り出し候補ナンバーの組み合わせで 4 つの新しい局面を作る。マス目が 1 つしかない場合はそのマス目のみ使用し 2 つの新しい局面を作る。新しい局面を作成したらそれぞれの局面について作戦①から実行する。これを作戦③の 4 通りの局面すべてについてそれぞれ試す。従って、作戦④では最高 16 個の新しい局面を作成し、試すことができる。

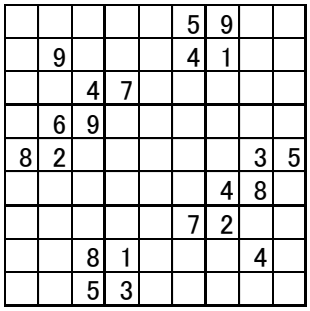
5．プログラムの性能とバージョンアップの可能性

数独の市販本やインターネット上に難問として紹介されている問題をいくつか試した結果、現時点では 1 つを除き正解を出すことに成功しているので、SSS はほとんどの数独の問題について解くことができるといっても過言ではない。解けた難問の 2 例を図 5.1 に示す。（解答は付録 4 を参照）

現プログラムで解くことのできない唯一の難問を図 5.2 に示す。この問題が掲載されているサイトによると「科学と応用数学の博士号を持つフィンランド人の環境科学者 Arto Inkala 博士が、解が一つだけ存在し「当

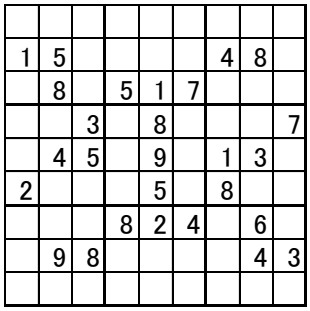
てずっぽう」ではなく「論理」のみですべてのマスを埋めることができる「正しい数独」の中で限りなく難しい、「世界一難しい数独」を作り出すことに成功した」という記述がある。

この問題を SSS に解かせてみると、作戦②まで適用して図 5.3 にあるシャドウのかかった 2 つのマス目の数字、5 と 3 を見つけた後、SSS は作戦③作戦④を適用したが、局面の進展はそれ以上得られなかった。



(難問 1)

出典：
(http://robohei.jugem.cc/?eid=1689)



(難問 2)

出典：
(http://kaka8.blog.so-net.ne.jp/archive/20110416)

図 5.1　解けた難問例

時間的制約を考慮しなければ、「しらみつぶし法」で全ての組み合わせについてチェックしていけば正解は必ず得られるが、feasible でない。つまり、組み合わせの爆発が起こり、計算時間がかかりすぎて正解にたどりつけない。

図 5.3 は、数字が確定していないマス目について、9 桁の数字から成る filter 情報を書き出している。filter の中にある 0 に対応する場所にある数字が候補ナンバーである。従って、そのマス目の 0 の個数が候補ナンバーの個数になり、その個数を全て掛け合わせた数字がこの局面から出現しうる総局面の数である。

実際に、図 5.3 からのその総局面数を計算してみると、2.3003×10^{31} 通りになる。これは、仮に 1 秒間に 1 億個の局面をチェックできたとしても 1000 兆年以上かかる計算になり、実行不可能である。

現在のバージョンでは作戦③と④において候補ナンバーが 2 つに限定されている 4 つのマス目を使用して 16 個の候補の局面を作成した後、それぞれを初期局面として作戦①から再開している。それにもかかわらず最後までは解けないので、今後の計画としては、さらに使用するマス目を 1 つ増やして 32 個の局面を作成して、作戦①から実行を再開できるようにプログラムのバージョンアップをするつもりである。この程度の局面数の増加は実行時間の点で実行可能範囲である。

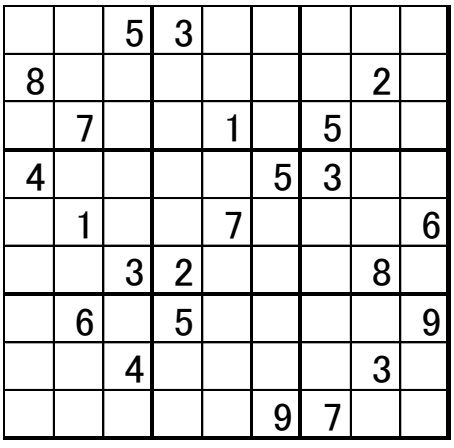


図 5.2　解けなかった難問
出典：
(http://gigazine.net/news/20100822_hardest_sudoku/)

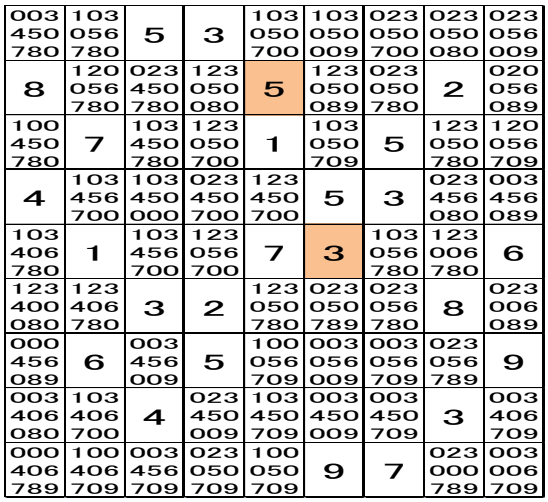


図 5.3　フィルタ情報の表示

参考文献

周防節雄・知平菜美子(2011)　SAS マクロ言語を使った数独パズルを解くプログラムの構造と制御方法、本論文集収録

付録１．puzzle_make_original_data.sas

```
/* puzzle_make_original_data.sas */

*★以下のマクロ変数を設定してください。;
%let Q=Q113; *数独問題の dat ファイル名の指定(拡張子は不要);

*以下の設定は変更不要;
%let round=0; *各局面ごとに繰り返した回数;
%let which=1; *作戦③で４つの候補局面（original_1～original_4）の設定;
%let new_which=0; *作戦④で 2 つの候補局面（original1・original2）の設定;
%let new_round=0; *puzzle_solve_final.sas を実行した回数;
%let select_sw=0; *作戦④で候補が２つか４つかの判定;

options noxwait;
x "cd &drive:¥";
x "md sudoku"; *&drive にフォルダ sudoku を作成;
x "cd sudoku";
x "md output_window"; *&drive:¥sudoku の下にフォルダ output_window 作成;
x "md log_window";      *&drive:¥sudoku の下にフォルダ log_window 作成;
x "exit";

%let log=&drive:¥sudoku¥log_window;
%let output=&drive:¥sudoku¥output_window;

options nocenter nodate nonumber ls=90;
filename in1 "&problem_folder¥&Q..dat";

data original;
      drop i;
      infile in1;
      array a {9} $ 1;
      do i=1 to 9; input @i a{i} @@; end;
      input; /* Release current data line.*/
run;

ods listing close;
ods html file="&output¥&Q..xls";

proc print data=original; title "data=original"; run;

ods html close;
ods listing;
```

付録２．puzzle_solve_final.sas

```
/* puzzle_solve_final.sas */
  *Before this program is run,
    "puzzle_make_original_data.sas" must be run.;
options mprint SPOOL;

%let loop_no=15; *各局面の繰り返し回数の上限を指定してください;

*以下は設定不要;
%let star=*; /* blank=Valid ODS, "*"=Stop ODS;*/
%let round=%eval(&round+1);
%let wrong=0; /*0=Not Wrong Answer,1=Wrong Answer*/
%let new_round=%eval(&new_round+1);


data original_before;
*this dataset will be compared with "new" original at the bottom;
      set original;
run;

ods listing close;
ods html
file="&output¥ROUND&round._&which._&new_which..xls";
proc print data=original;
title "ROUND &round:&new_round. data=original";
run;
ods html close;
ods listing;

PROC PRINTTO
log="&log¥logROUND&round._&which._&new_which..txt" NEW; RUN;
PROC PRINTTO
print="&log¥outputROUND&round._&which._&new_which..txt" NEW; RUN;

%all; *解法エンジン;
```

付録３．reconstruct.sas

```
/* reconstruct.sas */

data cell;
	keep	v row_no col_no box_no;
	array a {9} $ 1;
	*array row_no {9};
	*array col_no {9};
	*array box_no {9};

	set original;
	do i=1 to 9; row_no=_n_; col_no=i;
			box_no=MAX(INT((_n_-1)/3),0)+INT(
(i-1)/3)+2*INT((_n_-1)/3)+1;
			v=a{i};
			output;;
	end;
run;

proc print data=cell; title "data=cell"; run;

*-----------------------------------------;
proc sort data=cell; by row_no; run;

data row;
	keep a1-a9;
	array a {9} $ 1;
	retain a1-a9;

	set cell; by row_no;
	a{col_no}=v;
	if last.row_no then output;
run;

options nocenter ps=100;

proc print data=row; title "data=row"; run;

*-----------------------------------------;
proc sort data=cell; by col_no; run;

data col;
	keep a1-a9;
	array a {9} $ 1;
	retain a1-a9;

	set cell; by col_no;
	a{row_no}=v;
	if last.col_no then output;
run;

proc print data=col; title "data=col"; run;

*-----------------------------------------;

proc sort data=cell; by box_no; run;

data box;
	keep a1-a9;
	array a {9} $ 1;
	retain a1-a9 i;

	set cell; by box_no;
	if first.box_no then i=1;
	a{i}=v; i+1;
	if last.box_no then output;
run;

proc print data=box; title "data=box"; run;
```

付録４．図 5.1 の難問の解答

7	8	3	2	1	5	9	6	4
5	9	2	8	6	4	1	7	3
6	1	4	7	9	3	5	2	8
4	6	9	5	3	8	7	1	2
8	2	7	9	4	1	6	3	5
3	5	1	6	7	2	4	8	9
9	3	6	4	8	7	2	5	1
2	7	8	1	5	9	3	4	6
1	4	5	3	2	6	8	9	7

(難問 1)

3	6	2	9	4	8	5	7	1
1	5	7	2	6	3	4	8	9
4	8	9	5	1	7	3	2	6
9	1	3	4	8	2	6	5	7
8	4	5	7	9	6	1	3	2
2	7	6	3	5	1	8	9	4
7	3	1	8	2	4	9	6	5
6	9	8	1	7	5	2	4	3
5	2	4	6	3	9	7	1	8

(難問 2)