Paper 225-2012

# *Sudoku-Solving System by SAS®*

Setsuo Suoh, the University of Hyogo, Kobe, Hyogo Prefecture, Japan

## ABSTRACT

We have developed the Sudoku-Solving System by SAS. The SAS data set is supposed to be one of the most inconvenient data set to attack Sudoku puzzles because of its structure. Our system has four crucial factors: (1) how to depict the values of 81 boxes in a SAS data set environment; (2) how to attack Sudoku puzzles efficiently and smartly without using brute force; (3) the recursive technique employed for tree searching during the attacking process, using the %INCLUDE statement; (4) an option for recording all of the attacking process or history of a tree search. It consists of three SAS programs, one of which has 21 SAS macro definitions, and succeed in solving the hardest Sudoku puzzles available on the Internet within a reasonable time.

## INTRODUCTION

We have picked up Sudoku puzzles and succeeded in developing our solving system by SAS called “Sudoku Solving System: SSS). Every Sudoku puzzle has a unique solution that can be obtained logically. The objective of the game is to enter numbers into the blank cells within 9x9 cells, so that each row, column and 3x3 box contains every number from 1 to 9. Figure 1.1 shows an example of an initial situation, and Figure 1.2 shows its solution. It is one of the hardest Sudoku puzzles available on the Internet.

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

Figure 1.1. Sudoku Puzzle ‘Q999’

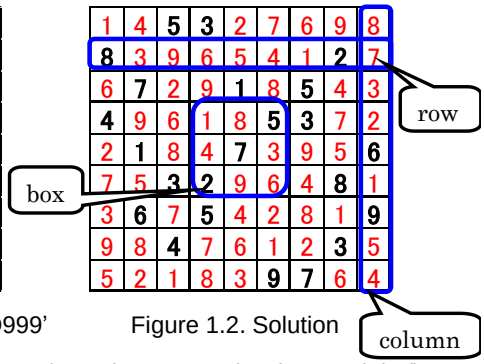


Figure 1.2. Solution

(Data Source: http://gigazine.net/news/20100822_hardest_sudoku/)

Since the SAS data set is ready only for “observation” wise operation, it is supposed to be one of the most inconvenient data set types to attack Sudoku puzzles. To overcome this problem, we designed a SAS data set to contain all the information on the entire board situation, from which information of row, column or box could be easily gained in need. In the present paper we explain our system focusing on three crucial factors: (1) how to depict the values of 81 boxes in a SAS data set environment; (2) how to attack Sudoku puzzles efficiently and smartly without using brute force; (3) the recursive technique employed for tree searching during the attacking process, using the %INCLUDE statement.

Almost all game playing programs including Sudoku would face a combinatorial explosion, if the brute force searching were adopted during the tree search. Our system focuses on only several blank cells to avoid this problem.

## HISTORY OF DEVELOPING SSS

In the early stage of developing SSS, it consisted of only one SAS program (ver.0) that included two basic Sudoku attacking strategies; Operation (1) and Operation (2), both of which will be explained in the later section. Although it successfully solved beginners’ level puzzles, it failed to solve middle class level ones. After having found what else more to be needed, we added more new operations one by one.

During this process we encountered a very serious, fundamental problem that our SAS program was not be able to be compiled, probably because of failure of resolving macros. As discussed later, our system employs the recursive technique to repeat executing the same SAS program that includes many macro definitions. Since we use the %INCLUDE statement for recursion, the same macros were defined again and again in spite of no necessity. We guessed that the SAS system would not allows users to define the same macros over a limit of a certain times, which we thought is fair enough.

By this time the program grew sizable and also complicated, we decided to split it into three parts; (i) Set initial data sets and macro variables, (ii) Define all macros and (iii) Sudoku solving engine. In this way only the solving engine was recursively used. As a result, we overcame the compiling problem. After this, we kept on adding more sophisticated Sudoku Operations (3) to (6), to upgrade it to ver.1 and ver1.5, ending up with SSS (ver.2) that successfully solves all Sudoku puzzles it encountered on the Internet. The difference between ver.1 and ver.1.5 is that during the attacking process, the initial board situation is either fixed, or rotated 90 degrees clockwise with a limit of three times until a solution can be reached. During the solving process, SSS selects only a few blank cells to fix

their numbers according to Operations (3) to (6). In the case that there are many such blank cells, it selects the “first” few blanks, which means that the selection depends on their accidental locations of board situation. In the worst case, only after the third rotation one Sudoku puzzle was solved. All the performing results of different versions are shown in the final section.

The version 2 SSS has become a huge system with 29 macros, while ver.1 had 39 macros. This is the result of keep adding extra operations every time our system failed in solving new Sudoku puzzles. Some of the later added operations looked like each other with a slightly different way of usage. By this time we learned much about Sudoku tactics through different versions of SSS, and decided to make it more compact to develop ver.3 SSS so that it would be much easier for further upgrading it in the future.

## STRUCTURE OF THE LATEST VERSION OF SSS

In the present section we focus on the latest version of SSS: ver.3, and explain how it is constructed. It consist of three programs; (1) Sudoku_macro.sas, (2) puzzle_make_original_data.sas, and (3) puzzle_solve_final.sas

### REPRESENTATION OF SUDOKU BOARD SITUATION IN SAS DATA SETS

Figure 2 shows how the initial board situation is represented in SAS data sets.

(1) Given a Sudoku puzzle.

(2) Input the numbers by a text editor and save it as a text file, denoting “0” as a blank cell.

(3) Make a SAS data set ORIGINAL out of the text file.

(4) Out of ORIGINAL make a SAS data set CELL with 81 observations, each of which represents a row number, column number and a box number together with its numerical value. The box number can be calculated by the following expression:

```
BOX_NO=max(int((ROW_NO-
1)/3),0)+int((COL_NO-
1)/3)+2*int((ROW_NO-1)/3)+1
```

(5) Out of CELL make a SAS data set ROW that is coincidentally identical to ORIGINAL.

(6) Out of CELL make a SAS data set COL. Each observation represents cell information belonging to the same column.

(7) Out of CELL make a SAS data set BOX. Each observation represents cell information belonging to the same box.

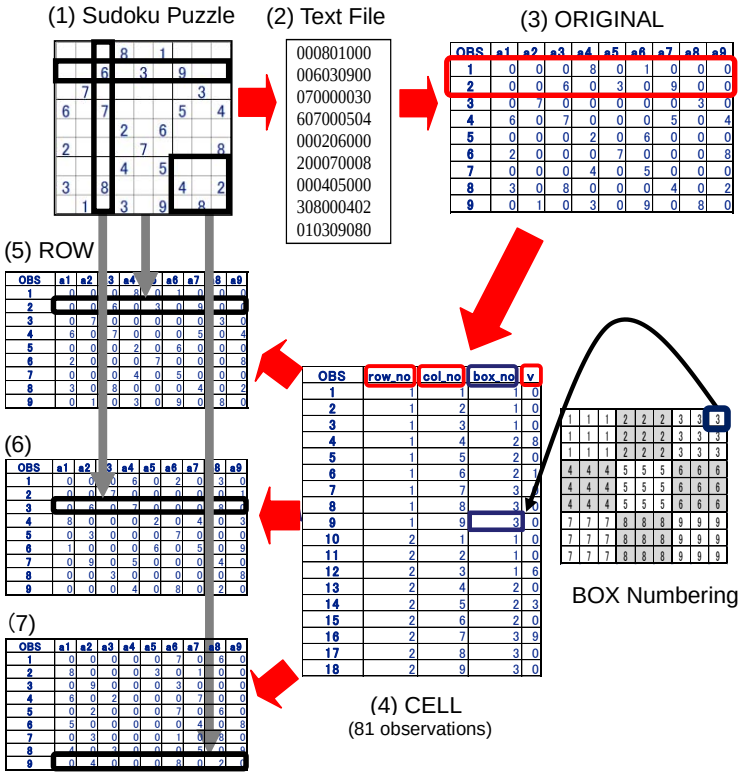


Figure 2. How to Represent Sudoku Puzzles in SAS data sets

These four data sets, CELL, ROW, COL and BOX are most crucial data sets in SSS, and they are always updated automatically by running a macro ‘RECONSTRUCT’. Figure 3 shows hierarchical structure of them.

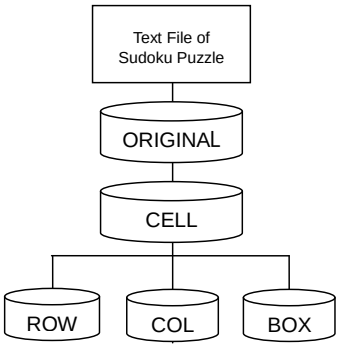


Figure 3. How to Create SAS Data sets

## STRUCTURE OF SAS PROGRAMS

SSS (ver.3) consists of three SAS programs, sudoku_macro.sas, puzzle_make_original_data.sas and puzzle_solve_final.sas. Users of SSS must run them in this order.

(1) SUDOKU_MACRO.SAS

In this program there are twenty one macros defined. On top of this program users are asked to specify fundamental information to macro variables such as a drive name where the SSS system is saved. For more details refer to Appendix II.

Figure 4 shows the structure of all of the SAS macros. The macro ALL is a core of Sudoku solving engine, and it directly invokes seven macros from ALL_FILTER to OUTPUT. They also further invoke other macros, and so on. Those macros in bald type have the %include statement in which 'puzzle_solve_final.sas' is executed recursively.

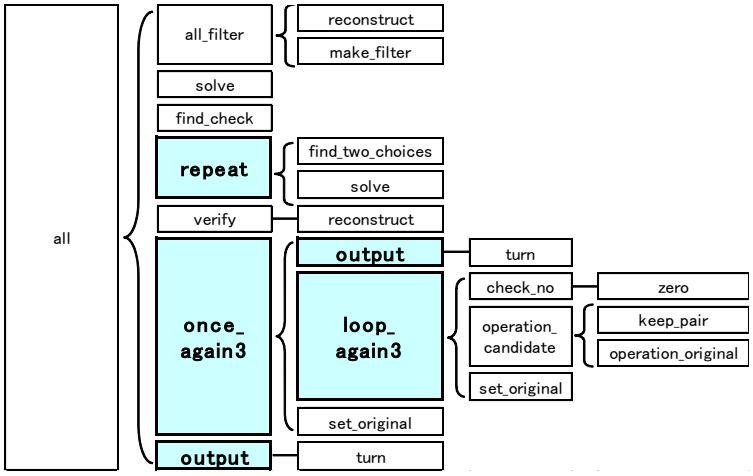


**Figure 4. the Structure of SAS Macros of SSS**

(2) PUZZLE_MAKE_ORIGINAL_DATA.SAS

In the beginning of this program users are asked to specify a text file name of Sudoku puzzle to a macro variable Q. Other macro variables are automatically specified by our system. There are some X commands by which new folders are automatically created and also old output and log information are deleted. The specified text file of Sudoku puzzle is converted to SAS data set INITIAL, out of which four SAS data sets ORIGINAL_1 to ORIGINAL_4 are created by rotating 90 degrees clockwise respectively. They will be used as SAS data set ORIGINAL for later stages. The SAS code of this program is as follows.

```
/* puzzle_make_original_data.sas */
* Specify a file name of Sudoku puzzle to the following macro variable Q;
%let Q=Q113; *Sudoku Puzzle saved as a DAT file (No extensions necessary);

* Don't change the following macro variables definitions;
%let round=0;     * No. of repetitions of executing 'puzzle_solve_final.sas'
                    for each candidate board situation;
%let new_round=0; * No. of times of executing 'puzzle_solve_final.sas' recursively;
%let time_sw=0;   * It will be set to 1 as soon as 'starttime' is set on;
%let rotate=1;    * Every time the initial board is rotated 90 degrees clockwise,
                    varies from 2 to 4;
%let rotate_sw=0; * Set to 1 every time each of four initial board situations is set
                    as SAS data set ORIGINAL.
                    Reset to 0 when no solutions were found after New Operation (3);
%let op3_no=0;    * Eventually total No. of candidate board situations created
                    in the beginning of New Operation (3) is set;
%let set3_no=0;   * During New Operation (3) each candidate board situation is given
                    a number from 1 to whatever the total number of them as a part of
                    its SAS data set name;
options noxwait;
x "cd &drive:\&folder";
x "md output_window";
x "cd output_window";
x "del *.*";
x "cd ..";
x "md log_window";
x "cd log_window";
x "del *.*";
x "exit";
%let log=&drive:\&folder\log_window;
%let output=&drive:\&folder\output_window;
options nocenter nodate nonumber ls=90;
filename in1 "&problem_folder\&Q..dat";
```

```
data initial;
    drop i;
    infile in1;
    length x $ 1;
    do i=1 to 9; row=_n_; col=i; input @i x $ @@; output; end;
    input;
run;

proc print; title "initial"; run;

%rotate(1) %rotate(2) %rotate(3) %rotate(4)
```

(3) PUZZLE_SOLVE_FINAL.SAS

This program is a sort of main program. At the end of this program SAS macro ALL is invoked. It further invokes all the rest of other macros either directly or indirectly, as shown in Figure 4. It is executed recursively by invoking four macros REPEAT, ONCE_AGAIN3, OUTPUT and LOOP_AGAIN3 denoted by bald type in Figure 4. The SAS code of this program is as follows.

```
/* puzzle_solve_final.sas */ /* ver.3 */
 *Before this program is run, "puzzle_make_original_data.sas" must be run.;

options &non.notes &non.source;

*No Need to Specify Below;
 %let round=%eval(&round+1);
 %let new_round=%eval(&new_round+1);
 %let found=0; * 1 is set when a new number was found;
 %let wrong=0; * 1 is set in case an incompatible situation occurs;

data _null_;
    if &time_sw=0 then do;
                       starttime=DATETIME(); put starttime= time.;
                            *presents SAS time;
                       call symput("starttime",starttime);
                            *sets running starting time in SAS time;
                       end;
run;

%let time_sw=1; * 1 is set after setting 'starttime'to 1;

%set_rotate; *By rotating the original situation 90 degrees clockwise,
              four'original' board situations have already been created.
              Each one of them is now set to SAS dataset ORIGINAL;

data original_before;
    *Save ORIGINAL by a different dataset name for later comparison;
    set original;
run;

&star. ods listing close;
&star. ods html file=
"&output\&rotate._ROUND&round._Operation(3)ORIGINAL&set3_no..xls";
&star. proc print data=original;
title "ROUND &round:&new_round. data=original"; run;
&star. ods html close;
&star. ods listing;

&star. PROC PRINTTO log=
"&log\log&rotate._ROUND&round._Operation(3)ORIGINAL&set3_no..txt" NEW; RUN;
&star. PROC PRINTTO print=
"&log\output&rotate._ROUND&round._Operation(3)ORIGINAL&set3_no..txt" NEW; RUN;

%all; *Solving Engine;
```

## RECURSIVE TECHNIQUE

Calculation of factorial is often used to explain the recursive technique. The recursive definition of n! is as follows.

$$f(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot f(n-1) & else \end{cases}$$

We wrote a set of two SAS programs to calculate 4! recursively. The first program 'factorial_set_initial.sas' defines SAS macro FACTORIAL in which the second program 'factorial.sas' is executed recursively by using the %include statement. Note that the program must be terminated by the %abort statement when the value of macro variable N becomes 2. Although the ERROR message appears in the SAS log, it is not a malfunction of the program, but a normal termination.

```
/* factorial_set_initial.sas */
options mprint nocenter;
%let N=4; *Calculate N! ;
data factorial;
 f=&N; run;proc print;
 title "Initial value N=&N"; run;
%macro factorial;
%let N=%eval(&N-1);
data factorial; set factorial; f=&N*f; run;
proc print; title "N=&N"; run;
%if &N=2 %then %abort;
%include "C:\SAS_Forum\2012\recursive\factorial.sas";
%mend;
```

```
/* factorial.sas */
 %factorial;
```

```
18   /* factorial.sas */
19   %factorial;
MPRINT(FACTORIAL):   data factorial;
MPRINT(FACTORIAL):   set factorial;
MPRINT(FACTORIAL):   f=3*f;
MPRINT(FACTORIAL):   run;

MPRINT(FACTORIAL):   proc print;
MPRINT(FACTORIAL):   title "N=3";
MPRINT(FACTORIAL):   run;

MPRINT(FACTORIAL):   data factorial;
MPRINT(FACTORIAL):   set factorial;
MPRINT(FACTORIAL):   f=2*f;
MPRINT(FACTORIAL):   run;

MPRINT(FACTORIAL):   proc print;
MPRINT(FACTORIAL):   title "N=2";
MPRINT(FACTORIAL):   run;

ERROR: Execution terminated by an %ABORT statement.
```

SAS LOG1. After Running 'factorial.sas'

```
Initial value N=4
OBS     f
 1      4
```

```
N=3
OBS      f
 1      12

N=2
OBS      f
 1      24
```

OUTPUT1. After Running two Programs

The recursive technique similar to the above example was employed in SSS, in other words, embedding the % include statement makes it possible to repeat executing the same SAS program when becoming necessary. As a result, the SAS system keeps on 'copying' and inserting the respective SAS code on the spot as shown in the right. When a final solution was found, the SAS code remained untouched must be discarded by using the %abort statement.

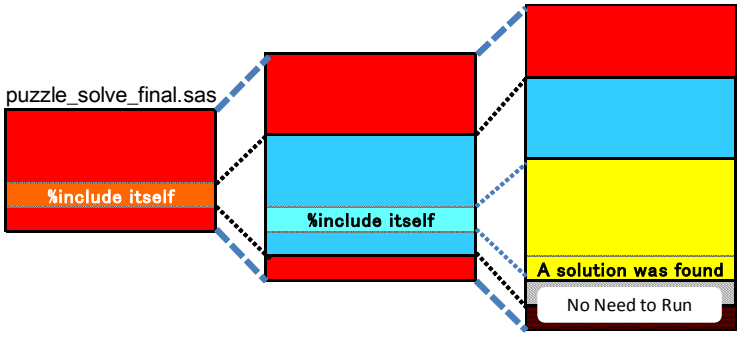


**Figure 5. Result of %include Statement**

## STRATEGIES EMPLOYED BY THE LATEST VERSION OF SSS

The Sudoku attacking strategies of the latest model of SSS consists of three operations; Operations (1) to (3). The first two operations are very fundamental, and the result gained from them is always correct and determined. The third operation is an auxiliary method that is used when the final solution cannot be reached by the first two operations. It is a sort of searching procedure for seeking a final solution out of multiple number of candidates. Obviously, beginners' level Sudoku puzzles tend to be solved only by the first two operations, and experts' level

puzzles could not be solved without Operation (3).

## FILTER

A new concept of the 'filter' for all 81 cells was introduced to represent cell situation. It is defined as follows;

filter = $[x_1 x_2 \cdots x_9]$ where $x_i = 0$ or $i$

For instance, when the filter of a blank cell is [123406780], then 5 or 9 may be applicable for the respective cell. The value of filter can be obtained by cross-referencing row group, column group and box group of each cell. This concept was found very useful for Sudoku attacking strategy.

## OPERATION (1)

There are two cases in which a new number can be found. The first case is very simple in which only one number is available for a certain blank cell. It can be found easily by checking its filter information. For example, take a look at the cell (1) in Figure 6. Its filter is [123456089] that indicates that the cell must be 7.

The second case is not as simple as the first one, but not so complicated as imagined. From the filter of cell (2) the cell's value may be 1, 2 or 9. If further examined in the same row, however, '2' is not allowed in all the rest of four blank cells, which means that cell(2) must be 2.

Accordingly, SSS updates the board situation by adding newly found two numbers in cell (1) and (2), and go back to Operation (1) recursively until no more new numbers can be found. After this, SSS goes to Operation (2).

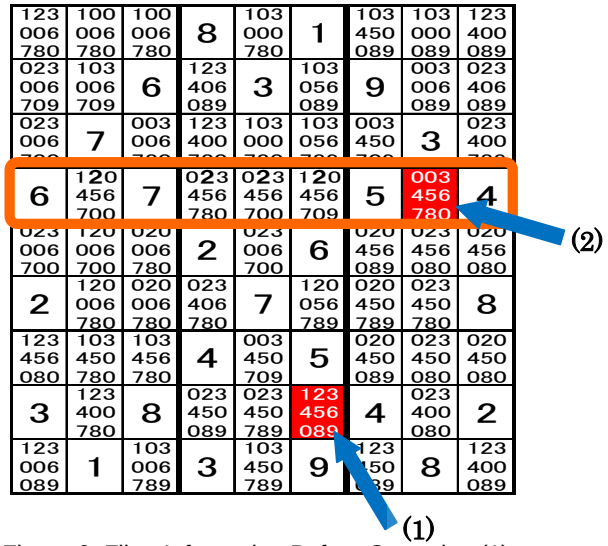


Figure 6. Filter Information Before Operation (1)

## OPERATION (2)

First of all, SSS tries to update filter information, and look for two blank cells with the same combination of only two numbers as candidates among the same group of row, column or box. If there are such two blank cells, the two numbers cannot be used any more by other cells in the same group, which means that the filter information can be updated. Figure 7.1 shows such an example. There are two dark cells with a filter [023450789]. The number of these two cells must be either 1 or 6, which means that the other two blank cells indicated by downward arrows in the same row can be neither 1 nor 6. Thus the filter information of these cells can be updated as shown in Figure 7.2. After updating the filter information, SSS goes back to Operation (1). After going to and from Operations (1) and (2), and no more new numbers cannot be found, SSS goes to Operation (3)

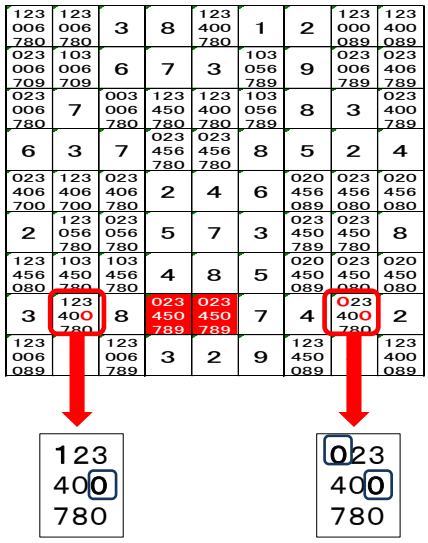


Figure 7.1. Before Operation (2)

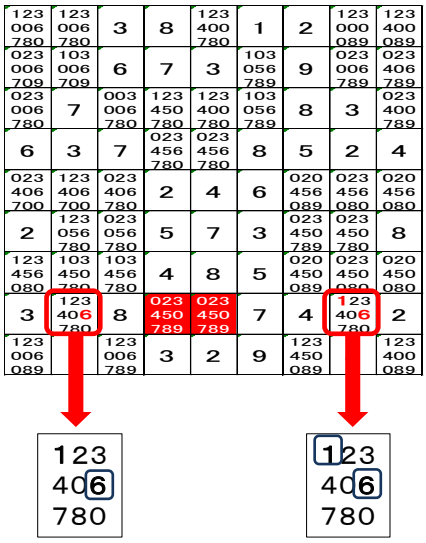


Figure 7.2. After Operation (2)

## NEW OPERATION (3)

As mentioned in HISTORY OF DEVELOPING SSS, many sophisticated operations have been added one by one in the earlier versions of SSS. From that experience we became confident that after Operations (1) and (2) it would be much better to select a reasonable number of candidate board situations and to concentrate on solving them as if they were initial board situations. The correct solution must be included among them.

When we started to developing earlier version SSS, we presupposed that there always exist a few blank cells with two numbers as candidates, and realised that it was not true in the case of expert level Sudoku puzzles and that there are almost always some blank cells available with 'three' numbers as candidates. Without them we would have resorted to the brute force to attack Sudoku, which we hate to do. We, therefore, adopted the New Operation (3) in which SSS tries to first look for a certain number (default value is 6) of blank cells with 'two' numbers as candidates. In case that the number of such cells is smaller than the default value, more blank cells with 'three' candidate numbers are searched to supplement the lack of blank cells to be attacked. The number of total candidate board situations is, therefore, between 64 ($=2^6$) and 729 ($=3^6$). Usually this number is around 300 because of mixed combination of two and three. In most cases final solution can be found before all these candidate board situations are tried.

Along with the creation of the New Operation (3), SAS macros defined in 'sudoku_macro.sas' were refined, ending up with decreasing the number of SAS macros from 29 to 21.

## COLLABORATION OF OPERATIONS (1) TO (3)

The collaboration of Operations (1) to (3) is shown in Figure 8.

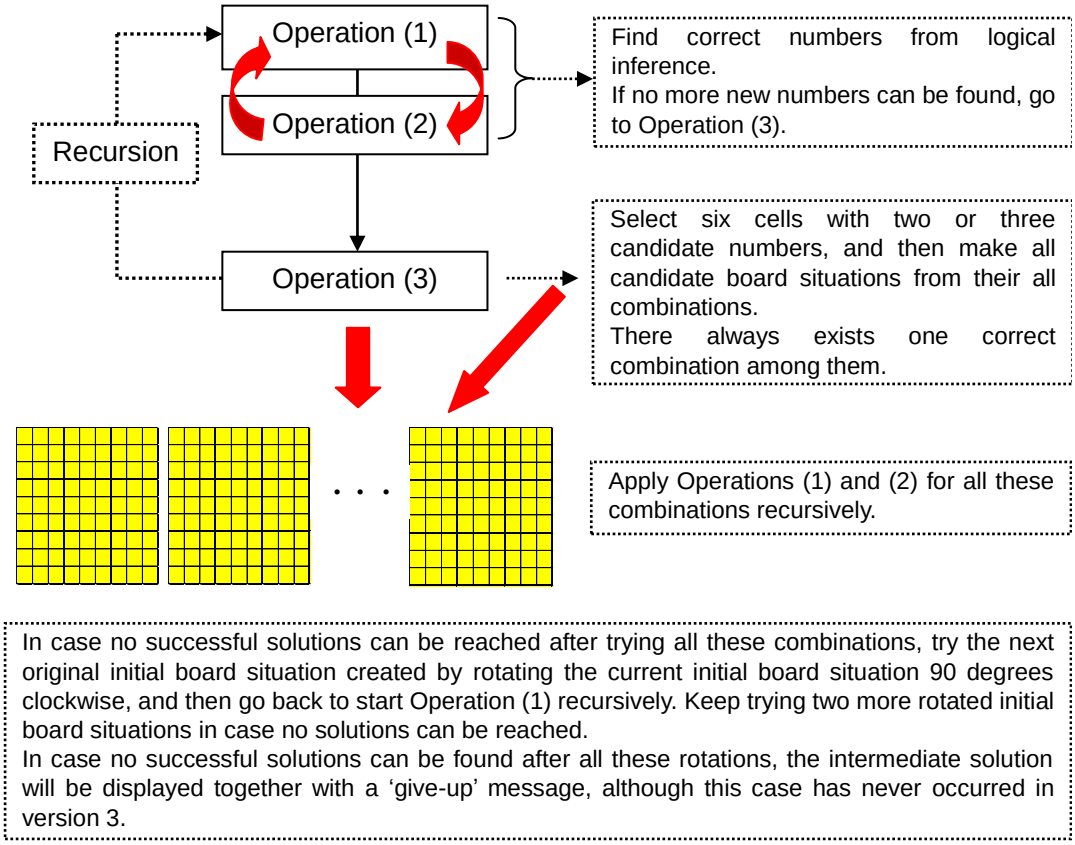


Figure 8. Collaboration of Operations (1), (2) and (3)

## PERFORMANCE OF SSS (VER.3) AND RESULTS

We have been developing the Sudoku solving system, SSS for nearly one year. Overcoming some serious problems, both of versions 2 and 3 of SSS succeeded in solving all the hardest Sudoku puzzles available on the Internet. Table 1 shows the performances of different versions of SSS. 'N' denotes the failure of solving. All these Sudoku puzzles,

a1 to a10 and Q999 are supposed to be the hardest Sudoku puzzles available on the Internet. The first ten Sudoku puzzles, a1 to a10 are shown in Appendix I, and 'Q999' is shown in Figure 1.1.

SSS produces substantial amount of information in both SAS LOG and OUTPUT windows, and we would easily see a warning message that these windows are full, without any consideration. The maximal number of lines displayed in each window is approximately 99,999. Most of OUTPUT window information of SSS is saved in an external file specified by the user. It is a record of how SSS solved Sudoku puzzles, while SAS LOG information is able to either be saved in an external file specified by the user or discarded. If it is discarded, substantial amount of run time can be saved. The run time of ver.1.5 In Table 1 includes time for storing SAS LOG information. The number in the column denoted by an asterisk is the number of times of execution of solving engine, SAS program, 'puzzle_solve_final.sas'.

| Version | ver.1 | | | ver.1.5 | | | ver.2 | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Opera-tions Employe | 1, 2, 3, 4, 5 | | | 1, 2, 3, 4, 5 + Rotation | | | 1, 2, 3, 6, 5 + Rotation | | | |
| Sudoku Puzzles | Result | * | Run Time ( No LOG ) | Result | * | Run Time ( With LOG ) | Result | * | Run Time ( No LOG ) | No. of Rotations |
| a1 | N | 314 | 12min12sec | Y | 488 | 34min57sec | Y | 715 | 1 h 3min33sec | 1 |
| a2 | Y | 204 | 8min46sec | Y | 204 | 12min12sec | Y | 376 | 21min59sec | 0 |
| a3 | N | 79 | 2min49sec | N | 332 | 24min11sec | Y | 300 | 19min46sec | 1 |
| a4 | N | 233 | 8min33sec | Y | 914 | 1 h 21min2sec | Y | 1602 | 2 h 4min4sec | 3 |
| a5 | Y | 187 | 7min21sec | Y | 187 | 13min6sec | Y | 140 | 7min49sec | 0 |
| a6 | N | 240 | 9min54sec | Y | 428 | 34min42sec | Y | 811 | 1 h 10min52sec | 1 |
| a7 | N | 88 | 3min14sec | N | 368 | 30min59sec | Y | 29 | 55sec | 0 |
| a8 | Y | 156 | 6min2sec | Y | 156 | 10min33sec | Y | 64 | 2min34sec | 0 |
| a9 | Y | 83 | 3min1sec | Y | 84 | 5min16sec | Y | 82 | 3min24sec | 0 |
| a10 | N | 266 | 10min23sec | Y | 747 | 59min21sec | Y | 988 | 1 h 7min44sec | 2 |
| Q999 | Y | 123 | 4min46sec | Y | 123 | 4min46sec | Y | 261 | 13min49sec | 0 |

| Version | ver.3 | | | | ver.3 | | | |
|---|---|---|---|---|---|---|---|---|
| Opera-tions Employed | 1, 2, New 3 (select_no=6) + Rotation | | | | 1, 2, New 3 (select_no=7) + Rotation | | | |
| Sudoku Puzzles | Result | * | Run Time ( No LOG ) | No. of Rotations | Result | * | Run Time ( No LOG ) | No. of Rotations |
| a1 | Y | 532 | 24min24sec | 1 | Y | 1984 | 1 h 43min25sec | 1 |
| a2 | Y | 372 | 13min38sec | 0 | Y | 277 | 12min52sec | 0 |
| a3 | Y | 180 | 5min20sec | 0 | Y | 146 | 6min43sec | 0 |
| a4 | Y | 519 | 18min2sec | 1 | Y | 1543 | 1 h 42min32sec | 1 |
| a5 | Y | 107 | 2min11sec | 0 | Y | 460 | 10min13sec | 0 |
| a6 | Y | 283 | 9min36sec | 0 | Y | 957 | 44min14sec | 0 |
| a7 | Y | 445 | 14min14sec | 0 | Y | 1505 | 1 h 20min15sec | 0 |
| a8 | Y | 27 | 1min9sec | 0 | Y | 971 | 1 h 40sec | 0 |
| a9 | Y | 104 | 3min36sec | 0 | Y | 728 | 34min12sec | 0 |
| a10 | Y | 1268 | 58min1sec | 2 | Y | 3318 | 4 h 19min7sec | 2 |
| Q999 | Y | 1248 | 1 h 4min6sec | 2 | Y | 888 | 26min16sec | 0 |

**Table 1. Performances of Different Versions of SSS**

It is obvious that the more the rotations occur, the more the run time required. Adjustment of default value of the number of candidates' cells for version 3 affected the run time it took before finding a solution. It took SSS (ver.3) with default value of 6 much less time than that with default value of 7 except for the hardest Sudoku puzzle, Q999.

OUTPUT 2 shows the result after Sudoku puzzle 'Q999' was solved successfully.

This kind of programs usually needs the tree searching procedure, but we do not have one. Instead, we use the recursive technique. In the early stage of developing SSS, we were not confident that it would work, but our confidence has become stronger as our system has been developed to solve Sudoku smartly and perfectly.

In Appendix II all the SAS macro definitions in SAS program 'sudoku_macro.sas' are shown. The source code of the whole three SAS programs that consist of SSS (ver.3) can be downloaded from the following URL, so that anyone could use our system.

```
Problem # Q999 was successfully solved.
No. of 90-Degree Clockwise Rotations: 0
No. of Repetitions of Running 'puzzle_solve_final.sas': 888 times
Run Time:          1493 Seconds

OBS    a1    a2    a3    a4    a5    a6    a7    a8    a9

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

**OUTPUT 2. Result after Sudoku Puzzle 'Q999' Was Solved**

(http:/mighty.gk.u-hyogo.ac.jp/confidential/sudoku.zip)

## REFERENCES

T Barron, D.W. (1968), *Recursive Techniques in Programming*, Elsevier.

## ACKNOWLEDGMENTS

After the author started developing SSS, Ms Namiko Chihira, 4th year undergraduate student joined my project. Without her dedicate cooperation it could not have been done successfully. We adopted the recursive technique for attacking Sudoku puzzles. I learned it when I was an undergraduate student nearly forty years ago. I happened to find a book written by Barron(1968) in the office of my then supervisor, Prof. M. Sawamura, otherwise I would not have learned it.

## CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Name: Setsuo Suoh
Enterprise: the University of Hyogo
Address: 8-2-1 Gakuen-nishimachi, Nishi-ku
City, State ZIP: Kobe City, Japan, 651-2197
E-mail: suoh@gk.u-hyogo.ac.jp
Skype name: charliesuoh

<Paper title>, continued

# Appendix I. Hardest Sudoku Puzzles Available on the Internet

Sudoku a1

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

Solution a1

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

Sudoku a2

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

Solution a2

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

Sudoku a3

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

Solution a3

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

Sudoku a4

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

Solution a4

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

Sudoku a5

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

Solution a5

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

Sudoku a6

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

Solution a6

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

Sudoku a7

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

Solution a7

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

Sudoku a8

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

Solution a8

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

Sudoku a9

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

Solution a9

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

Sudoku a10

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

Solution a10

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

(Source: http://www.aisudoku.com/en/AIsudoku_Top10s1_en.pdf)

## Appendix II. SAS Code of 'sudoku_macro.sas'

```
/*sudoku_macro.sas*/
%let drive  =C;      /* Drive Name where SSS is saved */
%let folder =sudoku; /* Folder name where the result of execution of SSS will be saved
*/
%let program=sudoku; /* Folder name where three SAS programs of SSS are saved */
%let non=no; /* (1) No LOG information is displayed ="no"
                (2) LOG information is displayed    =" " */
%let star=*; /* (1) Only the final page of OUTPUT window is displayed ="*"
                (2) All information of both LOG and OUTPUT windows are saved in
external filess =" " */
%let select_no=6; /* The No. of cells selected for Operation (3) */

*options mtrace macrogen;

* First of all, please save the following SAS programs in '&drive:\&folder'. ;
*(1) sudoku_macro.sas;
*(2) puzzle_make_original_data.sas;
*(3) puzzle_solve_final.sas;

* The users do not need to specify the macro variables below;
*(1) Full path of SSS engine, 'puzzle_solve_final.sas';
  %let run_solve="&drive:\&program\puzzle_solve_final.sas";
*(2) Folder name where text files of Sudoku puzzles are saved. ;
  %let problem_folder=&drive:\&folder\problem;
*=========================================================================;
%macro rotate(rotate_no);
%* Creates four SAS data sets, ORIGINALx1 to ORIGINALx4
   by rotating the initial Sudoku puzzle 90 degrees clockwise three times.;

proc sort; by row; run;
data originalx&rotate_no;
    array a {9} $ 1;
    retain a1-a9;
    set; by row;
    do i=1 to 9; a{col}=x; end;
    if last.row;
run;

ods listing close;
ods html file="&output\&Q.matrix&rotate_no..xls";

proc print; title"matrix&rotate_no"; run;

ods html close;
ods listing;

data initial;
    keep  x new_row new_col;
    rename new_row=row new_col=col;
    set initial;
    new_row=col;
    new_col=10-row;
run;

proc print; title"new matrix&rotate_no"; run;

%mend rotate;
*=========================================================================;
%macro set_rotate; %* Sets 90 degrees rotated board situation to ORIGINAL;
%if &rotate_sw=0 %then %do;
%let round=0; %* How many times 'puzzle_solve_final.sas' is recursively executed
                 for each candidate board situation during Operation (3).;

%let op3_no=0;  %* The No. of candidate board situations created
                   in the beginning of Operation (3);
%let set3_no=0; %* The Identification number of candidate board situation
                   to currently be checked during the execution of Operation (3);

data original;
```

```
    set originalx&rotate;
run;

%let rotate_sw=1;
                        %end;
%mend;
*=============================================================================;
%macro make_filter(dsname); %* Make 'filter' for each row, column and box. ;
data phase1;
    drop i;
    array a  {9} $ 1;
    array x  {9};
    array sw {9} $ 1;
    set &dsname;
    do i=1 to 9; if a{i}=0 then x{i}=123456789;
                           else do; if sw{a{i}}="Y" then call symput("wrong","1");
                                                   else do; sw{a{i}}="Y";
                                                            x{i}=a{i}*10**(9-a{i});
                                                        end;
                                end;
    end;
run;

&star. proc print; title "ROUND &round: data=phase1(&dsname)"; run;

data phase2;
    drop i;
    array x {9};
    set;
    filter=0;
    do i=1 to 9; if x{i} NE 123456789 then filter+x{i}; end;
run;

&star. proc print; title "ROUND &round: data=phase2(&dsname)"; run;

data &dsname._filter;
    keep &dsname._no &dsname._filter;
    set;
    &dsname._no=_n_;
    &dsname._filter=filter;
run;

&star. proc print; title "ROUND &round: data=&dsname._filter"; run;

%mend make_filter;
*=============================================================================;
%macro all_filter; %* Synthesize three filters of row, column and box.;
%reconstruct;

%make_filter(row);
%make_filter(col);
%make_filter(box);

proc sort data=cell; by row_no; run;
data all_filter; merge cell row_filter; by row_no; run;

proc sort data=all_filter; by col_no; run;
data all_filter; merge all_filter col_filter; by col_no; run;

proc sort data=all_filter; by box_no; run;
data all_filter; merge all_filter box_filter; by box_no; run;

proc sort data=all_filter; by row_no; run;
&star. proc print; title "ROUND &round: data=all_filter"; run;

data filter;
    drop i xrow_filter xcol_filter xbox_filter;
    length filter      $ 9;
    length xrow_filter $ 9;
    length xcol_filter $ 9;
    length xbox_filter $ 9;
```

```
    set all_filter;
    xrow_filter=row_filter;
    xcol_filter=col_filter;
    xbox_filter=box_filter;

    do i=1 to 9; if substr(xrow_filter,i,1)=i OR
                    substr(xcol_filter,i,1)=i OR
                    substr(xbox_filter,i,1)=i then substr(filter,i,1)=i;
                                                   else substr(filter,i,1)=0;
    end;
    if v NE 0 then filter=" ";
run;

&star. proc print; title "data=filter"; run;

data find_cell;
    drop i;
    set filter;
    zero_cnt=0;
    if filter=" " then return;
    do i=1 to 9; if substr(filter,i,1)=0 then zero_cnt+1; end;
run;

&star. proc print; title "ROUND &round: data=find_cell"; run;

data candidate;* Find candidate numbers for each cell by applying 'filter';
    set find_cell;
    if filter=" " then return;
    candidate=123456789-filter;
run;

&star. proc print; title "ROUND &round: data=candidate"; run;
%mend all_filter;
*=====================================================================;
%macro solve(row_col_box_no); %* Operation (1);
data cell;
    keep v row_no col_no box_no found;
    set candidate;
    if zero_cnt NE 1 then return;
    do i=1 to 9; if substr(filter,i,1)=0 then do; v=i; found=1; end; end;
run;

&star. proc print; title "ROUND &round: data=cell(2)"; run;

proc sort data=cell; by row_no; run;

data original;
    keep a1-a9;
    retain a1-a9;
    array a {9} $ 1;
    set cell; by row_no;
    if first.row_no then i=0;
    i+1;
    a{i}=v;
    if last.row_no;
run;

&star. proc print; title "ROUND &round: data=original(2) Easy Update from
&row_col_box_no"; run;

%let skip_sw=0;
proc sort data=candidate; by &row_col_box_no; run;

data select1;
    keep &row_col_box_no all_candidate cnt_no1-cnt_no9;
    length all_candidate $ 90;
    retain all_candidate;

    length xcandidate $ 9;
    array cnt_no {9};

    set candidate; by &row_col_box_no;
```

```
    xcandidate=candidate;
    if first.&row_col_box_no then all_candidate=" ";
    all_candidate=trim(left(xcandidate || all_candidate));
    if last.&row_col_box_no then do; do i=1 to 9;
                                cnt_no{i}=0;
                                do k=1 to 81;
                                  if substr(all_candidate,k,1)=i then cnt_no{i}+1;
                                end;
                            end;
                            output;
                        end;
run;

&star. proc print; title "ROUND &round: data=select1"; run;

data select2;
    keep &row_col_box_no cnt_no1-cnt_no9;
    array cnt_no {9};
    set select1;
    do i=1 to 9;
        if cnt_no{i}=1 then do; output; call symput("skip_sw",1); return; end;
    end;
run;

&star. proc print; title "ROUND &round: data=select2"; run;
%if &skip_sw=1 %then %do;
data select3;
    keep row_no col_no box_no candidate cnt_no1-cnt_no9;
    merge select2(in=in1) candidate; by &row_col_box_no;
    if in1;
run;

&star. proc print; title "ROUND &round: data=select3"; run;

data select4;
    drop xcandidate i;
    array cnt_no {9};
    length xcandidate $ 9;
    set select3;
    do i=1 to 9;
        if cnt_no{i}=1 then do; xcandidate=candidate;
                                  if substr(xcandidate,i,1)=i then
                                    do; v=i; output; end;

                            end;
    end;
run;

&star. proc print; title "ROUND &round: data=select4 (New found numbers)"; run;

proc sort data=select4; by row_no; run;

data found_nos;
    keep new1-new9 row_no mactch_key;
    array new {9} $ 1;
    retain new1-new9;

    set select4; by row_no;
    mactch_key=row_no;
    do i=1 to 9; if i=row_no then new{col_no}=v; end;
    if last.row_no then output;
    do i=1 to 9; new{col_no}=" "; end;
run;

&star. proc print; title "ROUND &round: data=found_nos (Only new found numbers)"; run;

data originalx;
    set original;
    mactch_key=_n_;
run;

&star. proc print; title "ROUND &round: data=originalx"; run;
```

```
data original;
    keep a1-a9;
    array a   {9} $ 1;
    array new {9} $ 1;

    merge originalx found_nos; by mactch_key;

    do i=1 to 9; if new{i} NE . then a{i}=new{i}; end;
run;

                    %end;

&star. proc print;
title "ROUND &round: data=original(updated) from &row_col_box_no"; run;
%mend solve;
*=====================================================================;
%macro find_two_choices(row_col_box_no); %* Operation (2): Update 'filter';
data xcandidate;
    set candidate;
    if zero_cnt=2;
run;

proc sort data=xcandidate; by &row_col_box_no zero_cnt candidate; run;
&star. proc print data=xcandidate;
title "ROUND &round: Cells with only two candidates: &row_col_box_no";
run;

data two_candidates;
    keep &row_col_box_no candidate;
    set xcandidate; by &row_col_box_no candidate;
    if first.&row_col_box_no=1 then delete;
    if first.candidate=0;
run;

&star. proc print data=two_candidates;
title "ROUND &round: data=two_candidates: &row_col_box_no";
run;

data two_candidates(drop=buffer candidate);
    set two_candidates;
    length buffer $ 9;
    buffer=candidate;
    candidate_copy=candidate;
    candidate1=indexc(buffer,"123456789");
    if candidate1 NE . then buffer=translate(buffer,"0",candidate1);
    candidate2=indexc(buffer,"123456789");
run;

&star. proc print data=two_candidates;
title "ROUND &round: data=two candidates: &row_col_box_no"; run;

proc sort data=candidate; by &row_col_box_no; run;

data merge_candidate(drop=candidate_copy);
    length vvv 8;
    merge candidate two_candidates; by &row_col_box_no;
    if candidate EQ candidate_copy then dont_change="NO";
    if zero_cnt=1 then do; vvv=right(translate(candidate," ","0")); v=vvv; end;
run;

&star. proc print data=merge_candidate;
title "ROUND &round: data=merge_candidate: &row_col_box_no";
run;

data choose_candidate(drop=xxcandidate);
    set merge_candidate;
    length xxcandidate $ 9;
    if candidate1 EQ . OR dont_change="NO" then return;

    xxcandidate=candidate;
    if candidate=. then return;
```

```
    substr(xxcandidate,candidate1,1)="0"; candidate=translate(xxcandidate,"0"," ");
    substr(xxcandidate,candidate2,1)="0"; candidate=translate(xxcandidate,"0"," ");
run;

&star. proc print data=choose_candidate;
title "ROUND &round: data=choose_candidate: &row_col_box_no";
run;

data candidate;
    keep filter row_no col_no box_no v row_filter col_filter box_filter zero_cnt
         candidate;
    set choose_candidate;
    if v NE 0 OR dont_change="NO" OR candidate1=. then return;

    substr(filter,candidate1,1)=candidate1;
    substr(filter,candidate2,1)=candidate2;

    zero_cnt=0;
    do i=1 to 9; if substr(filter,i,1)=0 then zero_cnt+1; end;
run;

&star. proc print data=candidate;
title "ROUND &round: data=candidate updated from two candidates: &row_col_box_no";
run;
%mend find_two_choices;
*============================================================================;
%macro find_check;
%* Set SAS macro variable 'found' to 1 if a new number has been found in this round.;
%* Set SAS macro variable 'not_yet' to the No. of cells unsolved.;
proc compare data=original_before compare=original out=result noprint; run;

&star. proc print data=result; &star. var a1-a9;
title "ROUND &round: data=result: X <- Newly found"; run;

data found;
    keep found_sw;
    set result end=final;
    array a {9} $;
    retain found_sw 0;
    do i=1 to 9; if a{i} NE "." then found_sw=1; end;
    if final then do; call symput("found",found_sw); output; end;
run;

&star. proc print data=found; title "ROUND &round: If new numbers were found,
found_sw=1 "; run;

data _null_;
    set original;
    array a {9} $ 1;
    do i=1 to 9; if a{i}="0" then not_yet+1; end;
    call symput("not_yet",not_yet);
run;

%mend find_check;
*============================================================================;
%macro repeat;
%* Run 'puzzle_solve_final.sas' recursively when new correct numbers are found;
%* When no new numbers were found, execute Operation (2).;
%if &found=1
     %then %include &run_solve;
     %else %do; %find_two_choices(row_no);
                %find_two_choices(col_no);
                %find_two_choices(box_no);

                data final;
                    set candidate;
                    if filter=" " then return;
                    candidate=123456789-filter;
                run;

                &star. proc print data=final;
                title"ROUND &round: data=final";
```

```
                run;

                %solve(row_no);
                %solve(col_no);
                %solve(box_no);
            %end;
%mend repeat;
*============================================================================;
%macro verify; %* Judge if the board situation becomes illegal or not,
                  after adding candidate numbers.;
%reconstruct;

data verify;
    length all $ 9;
    array a {9} $ 1;
    array cnt_no {9};
    retain wrong 0;

    set row col box;
    do i=1 to 9; substr(all,i,1)=a{i}; end;
    do i=1 to 9;
        cnt_no{i}=0;
        do j=1 to 9; if substr(all,j,1)=i then cnt_no{i}+1; end;
        do k=1 to 9; if cnt_no{k}>=2 then wrong=1; end;
    end;
    call symput("wrong",wrong);
    if wrong=1 then stop;
run;

&star. proc print data=verify; title"verify!"; run;
%mend verify;
*============================================================================;
%macro once_again3; %* Control of Operation (3);
%if &not_yet=0 %then %output;
               %else
 %do; %if &found=1 %then %include &run_solve;
                  %else
                    %do; %if &set3_no=0 %then %loop_again3;
                                        %else
                             %do; %if &set3_no=&op3_no
                                    %then;
                                    %else
                                %do; %set_original(3,&set3_no,set3_no);
                                     %include &run_solve;
                                %end;
                            %end;
                    %end;
 %end;
%mend once_again3;
*============================================================================;
%macro loop_again3; %* Create new candidate board situations in Operation (3);
data original_operation2;
    set original;
run;
%check_no(&select_no);
%operation_candidate(3,&zero_all,&number,&op3_no,op3_no,&select_no);
%set_original(3,&set3_no,set3_no);
%include &run_solve;
%mend loop_again3;
*============================================================================;
%macro check_no(n1);
%* preliminary process for creating candidate board situations in Operation (3).
%* Set the No. of cells with only two candidate numbers to macro variable 'number'.;
%* Set the No. of candidate board situations created in Operation (3) to macro
variable 'zero_all'.;

data x2candidate;
    set final;
    if zero_cnt=2 then do; cnt+1; output; end;
    %global number;
    call symput ("number",cnt);
run;
```

```
%zero(2);

data x3candidate;
    set final;
    if zero_cnt=3;
run;

%zero(3);

data xxxcandidate;
    set x2candidate x3candidate;
    if _N_<=&n1.;
run;

&star. proc print; title"xxxcandidate"; run;

data zero_candidate;
    array a{&n1.};
    retain a1-a&n1.;
    set xxxcandidate end=finish;
    a{_N_}=zero_cnt;
    if finish=1;
run;

&star. proc print; title"zero_candidate"; run;

data zero_candidate;
    array a{&n1.};
    set zero_candidate;
    all_cnt=1;
    do i=1 to &n1.; all_cnt=all_cnt*a{i}; end;
%global zero_all;
call symput ("zero_all",all_cnt);
run;

%mend check_no;
*=============================================================================;
%macro zero(no); %* Get candidate numbers from a selected cell.;

data x&no.candidate;
    set x&no.candidate;
    length buffer $ 9;
    buffer=candidate;
    %do i=1 %to &no.;
     candidate&i=indexc(buffer,"123456789");
     if candidate&i NE . then buffer=translate(buffer,"0",candidate&i);
    %end;
run;

%mend zero;
*=============================================================================;
%macro operation_candidate(n1,n2,n3,n4,n5,n6);
%* Make all candidate board situations from the combination of
   all possible candidate numbers for selected cells;
data two_candidates;
    keep row_no col_no xx1-xx3;
    set xxxcandidate;
    %do i=1 %to 3;
    xx&i=candidate&i;
    %end;
run;

&star. proc print;title"data=two_candidates";run;

data two_candidates;
array v{&n2.};
set two_candidates;
do x=1 to &n6.;
  if xx3=.
     then do; repeat=2**(_N_-1);
              do i=1 to &n2.; v{i}=xx2; end;
```

```
              do i=1 to &n2. by repeat*2;
               do j=i to i+repeat-1; v{j}=xx1; end;
              end;
          end;
     else do; repeat=3**(_N_-1-&n3.);
              do i=1 to &n2.; v{i}=xx3; end;
              do i=1 to &n2. by repeat*3;
               do j=i to i+repeat-1; v{j}=xx1; end;
               do j=i+repeat to i+repeat*2-1; v{j}=xx2; end;
              end;
          end;
end;
run;

&star. proc print; run;

%do i=1 %to &n2;
        %keep_pair(&i);
%end;

&star. proc print;title"keep_data";run;

%let xxxcnt=&n4;

%do i=1 %to &n2;
    %let xxxcnt=%eval(&xxxcnt+1);
    %operation_original(&n1,&i,&xxxcnt);
%end;
%let &n5=&xxxcnt;

%mend operation_candidate;
*============================================================================;
%macro keep_pair(no);
%* Make SAS data sets, each of which represents candidate numbers for selected cells.;
data keep_pair&no;
    keep row_no col_no v&no;
    set two_candidates;
run;
%mend keep_pair;
*============================================================================;
%macro operation_original(x1,x2,x3);
%* Make all candidate board situations by merging SAS data sets 'keep_pair&x2'
   and 'final';
proc sort data=keep_pair&x2; by row_no col_no; run;
proc sort data=final; by row_no col_no; run;

data original_&x1._&x3.;
    keep row_no col_no v;
    merge final keep_pair&x2; by row_no col_no;
    if v&x2 NE . then v=v&x2;
run;

data original_&x1._&x3.;
    keep a1-a9;
    retain a1-a9;
    array a {9} $ 1;
    set original_&x1._&x3.; by row_no;
    if first.row_no then i=0;
    i+1;
    a{i}=v;
    if last.row_no;
run;

&star. ods listing close;
&star. ods html
file="&output\original_&x1._&x3.(&rotate._ROUND&round._OPE3_ORIGINAL&set3_no.).xls";
&star. proc print data=original_&x1._&x3.;
title "original_&x1._&x3."; run;
&star. ods html close;
&star. ods listing;

%mend operation_original;
```

```
*============================================================================;
%macro set_original(n1,n2,n3);
%* Set next candidate board situation to SAS data set ORIGINAL;
%let n2=%eval(&n2+1);

data original;
    set original_&n1._&n2.;
run;

%let &n3=&n2;
%let round=0;
%mend set_original;
*============================================================================;
%macro turn; %* Get back the current board situaton to original shape, if it was
rotated;
data original;
    keep  x row col;
    array a {9} $ 1;
    set original;
    do i=1 to 9; row=_n_; col=i;
                 x=a{i};
                 output;;
    end;
run;

data original;
    keep  x new_row new_col;
    rename new_row=row new_col=col;
    set original;
    new_row=col;
    new_col=10-row;
run;

proc sort; by row; run;
data original;
    keep a1-a9;
    array a {9} $ 1;
    retain a1-a9;
    set; by row;
    do i=1 to 9; a{col}=x; end;
    if last.row;
run;

%mend turn;
*============================================================================;
%macro reconstruct; %* Make four SAS data sets, CELL, ROW, COL     and BOX from
ORIGINAL;
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;
*/
%mend reconstruct;
*============================================================================;
%macro output;
%* When the solution was found, it is displayed in the OUTPUT window,
   otherwise the solving engine will be executed recursively by using
   rotated board situations one after another. In case no solution can
   be found after three rotations, only the intermidiate solution gained
   from Operations (1) and (2) is displayed;
options nonotes nosource;

%if &not_yet=0 %then %do; %*<-----; %* Go to display the solution in the OUTPUT
window;
%if &rotate=1 %then; %* Display now, because of no rotations done;
              %else %do i=1 %to %eval(5-&rotate); %turn; %end;
              %* Rotate clockwise to put it in the original posision;

proc printto;

data;
    endtime=DATETIME();
    put endtime= time.;
    call symput("endtime",endtime);
run;

data;
    time=&endtime-&starttime;
    put time= time.;
       time=round(time);
       call symput("time",time);
run;
```

```
proc print data=original;title1 "Problem # &Q. was successfully solved.";
                         title2 "No. of 90-Degree Clockwise Rotations: %eval(&rotate.-
1)";
                         title3 "No. of Repetitions of Running
'puzzle_solve_final.sas': &new_round";
                         title4 "Run Time: &time. Seconds"; run;

%abort;
                     %end; %*<-----;
                %else %do; %*======;
                         %if &rotate<4 %then
                         %do;
                          %* In case no solutons have been found, rotate the board
position 90 degrees clockwise;
                          %let rotate=%eval(&rotate+1);
                          %let rotate_sw=0;
                          %include &run_solve;
                         %end;

data original;
    set original_operation2;
run;

%turn

proc printto;

data;
    endtime=DATETIME();
    put endtime= time.;
    call symput("endtime",endtime);
run;

data;
    time=&endtime-&starttime;
    put time= time.;
       time=round(time);
       call symput("time",time);
run;

proc print data=original;title1 "Problem # &Q. was not be able to be solved.";
                         title2 "No. of Repetitions of Running
'puzzle_solve_final.sas': &new_round";
                         title3 "Run Time: &time. Seconds"; run;

%abort;
                     %end; %*======;
%mend output;
*============================================================================;
%macro all; %* Solving Engine;
%* Operation (1) starts;
%all_filter; %if &wrong=1 %then %goto another_one;
%solve(row_no);
%all_filter; %if &wrong=1 %then %goto another_one;
%solve(col_no);
%all_filter; %if &wrong=1 %then %goto another_one;
%solve(box_no);
%find_check;

%* Operation (2) starts;
%repeat;
%find_check;
%verify;

%another_one: %*label;
%* Operation (3) starts;
%once_again3;

%output;
%mend all;
*============================================================================;
```