Paper BB01

Solving Samurai Sudoku Puzzles – A First Attempt

John R Gerlach, CSG Inc., Raleigh, NC USA

ABSTRACT

Imagine a Sudoku puzzle that consists of a 9x9 matrix having about 30 out of 81 cells populated. Now, imagine five such Sudoku puzzles such that a center puzzle is joined to four others at their respective 3x3 corner sub-matrices. These five Sudoku puzzles define a Samurai Sudoku puzzle. The solution to the puzzle is the same: each puzzle having unique values per row, column, and sub-matrix. However, there is an obvious interdependence among the five puzzles that poses a new challenge. This paper explains an expanded version of a SAS solution that used a dynamic cube to solve a regular Sudoku puzzle, by incorporating five cubes to solve the Samurai puzzle.

PREFACE

This SAS solution for solving Samurai Sudoku puzzles was actually written several years ago, not long after the author presented his solution for solving Sudoku puzzles at the first SAS Global Forum (SGF) Conference, in 2007, as part of a round table discussion that included: Richard DeVenezia, Larry Hoyle, Talbot Katz, and Rick Langston.

By pirating the author's SAS programs for solving regular Sudoku puzzles, a new program emerged that solves "easy" Samuari Sudoku puzzles. Afterwards, the author decided to put aside his solution anticipating that other colleagues would bring forth more rigorous SAS solutions. However, no SAS papers have been presented at SAS conferences since SGF 2007, at least not addressing the Samurai puzzle. Thus, the author decided to resurrect his SAS solution and to share it with the SAS community hoping to invoke renewed interest in this challenging problem.

It is assumed that the reader is already familiar with solving regular Sudoku puzzles. Also, as a prerequisite to this paper, it is recommended to read the author's paper, *Solving Sudoku Puzzles – Using a Cube to Solve the Square*, which is available on the Web. Finally, the reader should be very proficient in the Macro Language.

INTRODUCTION

Samurai Sudoku puzzles have an inherent interdependency among the five component puzzles that makes it far more difficult to solve than the single 9x9 puzzle. On the other hand, there's information in one 9x9 corner puzzle that may help solve the center puzzle, which may help solve another corner puzzle, which may help solve the whole puzzle. There's more information, but greater complexity. The real challenge is trying to maintain information about eligible values for all the cells – All 405 cells (81 cells times 5 puzzles), double counting those cells in the 3x3 sub-matrices that intersect with the Center matrix.

There is still one simple rule: Unique values for each row, column, and 3x3 sub-matrix. However, the values of a 3x3 corner sub-matrix is influenced by the adjoining matrix. Consequently, the Center matrix becomes pivotal in solving the puzzle since it affects all four outer matrices.

THE GENERAL STRATEGY

Basically, the solution for solving Samuari puzzles is premised on the same idea for solving regular Sudoku puzzles, which is explained in the paper, *Solving Sudoku Puzzles – Using a Cube to Solve the Square*. However, this solution requires the task of initializing and updating five cubes dynamically, which is not easy, albeit well-suited for a computer program.

Each Sudoku puzzle is represented by the set of variables having the form XY*xy*, where *xy* denotes the coordinate location in the 9x9 matrix. Thus, there are 81 variables named:

XY11-XY19; XY21-XY29; . . . ; XY91-XY99

Each cube is represented by the set of variables having the form XYZ*xyz*, where *xyz* denotes the coordinate location of a cell in the cube. Thus, there are 729 variables named:

XYZ111-XYZ119; XYZ191-XYZ199; . . . ; XYZ911-XYZ919; XYZ991-XYZ999

The cube maintains information about the eligible values for a given cell in the square. Thus, the cell in the square located in the third row and third column (i.e. {3,3}) is represented by the variable XY33; and, its eligible values is

maintained in the Z orthogonal vector located at the third row and third column in the cube, which is represented by the variables: XYZ331, XYZ332, XYZ333, . . ., XYZ339. If the cell in the Square contains a value, then the vector will have all zeros except for the XYZ coordinate containing its value; otherwise, the vector contains missing values, since all possible values (1 through 9) are eligible. Consider the following mappings:

XY51 → Null → XYZ511 . . . XYZ5119 → Null

XY29 → 3 → XYZ293 *and* XYZ251 . . . XYZ258 → 0

The whole cube is initialized premised on the values at the onset of the puzzle and further refined based on the rules for solving the puzzle: a value can be used only in a row, a column, and a 3x3 sub-matrix. This process is explained in more detail in the author's previous paper. After initialization, the process of solving the puzzle begins by looking at each vector in the cube such that there is only one possible response, that is, only one missing value remaining in the Z-vector. Each discovered value prompts an update of the cube, which causes a chain reaction, affording new instances of valid values. Eventually, the puzzle is solved after about twenty iterations. Of course, for Samurai puzzles, this process is greatly compounded having five cubes, all of which require updating. Moreover, there is an exchange of information at the 3x3 corner matrices between the Center and Outer puzzles.

READING AND RENDERING THE PUZZLE

The following illustration depicts a portion of the data file that defines a Samurai puzzle (Appendix A contains the complete puzzle). It is reasonable that the data file would contain more than one puzzle; hence, the first field denotes the *i*th puzzle. The next field identifies the component matrix: Center (C), Upper Left (UL); Lower Left (LL); Upper Right (UR); and Lower Left (LL). Also, the record contains missing values and integers ranging from 1 to 9 denoting the *i*th row in the puzzle, as well as the answer key. The solution to the puzzle is for documentation and testing purposes only.

2	C	.	.	.	.	.	.	.	.	.	9	1	5	4	2	6	7	8	3
2	C	.	.	.	.	.	.	.	.	.	4	8	6	3	1	7	9	5	2
2	C	.	.	.	.	5	.	.	.	.	2	3	7	8	5	9	6	4	1
2	C	.	.	.	1	.	2	.	.	.	5	9	3	1	8	2	4	7	6
2	C	.	.	8	.	.	.	5	.	.	6	2	8	7	3	4	5	1	9
2	C	.	.	.	6	.	5	.	.	.	7	4	1	6	9	5	2	3	8
2	C	.	.	.	.	4	.	.	.	.	1	7	9	2	4	3	8	6	5
2	C	.	.	.	.	.	.	.	.	.	8	5	4	9	6	1	3	2	7
2	C	.	.	.	.	.	.	.	.	.	3	6	2	5	7	8	1	9	4
2	UL	.	.	6	.	.	.	7	.	.	9	8	6	4	2	1	7	5	3
2	UL	.	7	.	9	.	3	.	6	.	1	7	4	9	5	3	8	6	2
2	UL	3	.	.	.	8	.	.	.	9	3	5	2	7	8	6	1	4	9
2	UR	.	.	5	.	.	.	7	.	.	8	9	5	3	6	1	7	2	4
2	UR	.	6	.	9	.	7	.	3	.	2	6	4	9	8	7	1	3	5
2	UR	3	.	.	.	2	.	.	.	9	3	1	7	4	2	5	8	6	9
2	LL	.	.	3	.	.	.	.	.	.	2	4	3	8	6	5	1	7	9
2	LL	.	9	.	1	.	2	.	.	.	7	9	6	1	3	2	8	5	4
2	LL	5	.	.	.	7	.	.	.	.	5	1	8	9	7	4	3	6	2
2	LR	4	.	.	.	7	.	.	.	1	4	5	2	6	7	3	8	9	1
2	LR	.	8	.	5	.	9	.	6	.	7	8	1	5	2	9	4	6	3
2	LR	.	.	9	.	.	.	2	.	.	6	3	9	4	1	8	2	5	7

Table 1. Partial Listing of a Samurai Sudoku puzzle with solution.

The following Data step creates the data set SAMUARI which represents one puzzle and contains only five observations each representing one component matrix and having 81 variables each for the puzzle and answer key, respectively. The macro %square generates these variables following the naming convention XY*rc*, such that *r* and *c* denote the row and column coordinate. For example, the variable XY35 denotes the cell in the third row and the fifth column.

For the nine records that represent a puzzle, the two-dimensional arrays are populated one row at a time based on the value from the SUM statement (r+1). The DO loop takes a vector of input values and places them in their respective row and column. Because there are five component puzzles, the IF statement, along with the SAS modulus function, reinitializes the variable *r*, after processing a component puzzle.

```
data samurai(keep=matrix %square(xy) %square(ak));
   retain %square(xy) %square(ak);
   array grid{9,9} %square(xy);
   array akey{9,9} %square(ak);
   array vvector{9} v1-v9;
   array avector{9} a1-a9;
   infile samurai missover;
   input puzzle matrix $ v1-v9 a1-a9;
   if puzzle eq &puzzle.;
   r+1;
   do c = 1 to 9;
      grid{r,c} = vvector{c};
      akey{r,c} = avector{c};
      end;
   if mod{_n_,9} eq 0
      then do; r=0; output; end;
run;
```

Below is a partial print out of the SAMURAI data set, excluding the Answer Key (AK*nn*) variables. Notice how the suffix of the variables indicates the coordinate of a cell.

M A T	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
R	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	<etc>	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
I	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2		8	8	8	8	8	8	8	8	8	9	9	9	9	9	9	9
X	1	2	3	4	5	6	7	8	9	1	2	3	4	5	6	7	8	9		1	2	3	4	5	6	7	8	9	1	2	3	4	5	6	7	8	9
C	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.		.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
UL	.	.	6	.	.	.	7	.	.	.	7	.	9	.	3	.	6	.		.	9	.	2	.	7	.	.	.	.	8	.	.	.	.	.	.	.
UR	.	.	5	.	.	.	7	.	.	.	6	.	9	.	7	.	3	.	<etc>	.	.	.	7	.	3	.	4	.	.	.	.	.	.	2	.	.
LL	.	.	3	.	.	.	.	.	.	9	.	1	.	2	.	.	.		.	7	.	6	.	1	.	2	.	.	.	5	.	.	.	7	.	.
LR	.	.	.	.	.	.	9	.	.	.	.	.	8	.	1	.	4	.		.	8	.	5	.	9	.	6	.	.	.	9	.	.	.	2	.	.

Table 2. Partial Listing of a data set containing the Samurai Sudoku puzzle.

In order to get a good visual grasp of the puzzle, it seems appropriate to write the puzzle in its natural 'Star' format, such that the center and outer matrices are positioned accordingly, showing the interdependence between them. Thus, the SAMUARI data set must be converted into a reporting data set whose unit of analysis is simply the *i*th row of the Star Matrix. However, such a data set must contain variables having null values representing the gap, the area between the UL and UR matrices above the Center matrix.

In order to create the reporting data set, each observation representing a component matrix (UL, UR, LL, LR, C) must be transposed so that there are nine observations each having nine columns (C1-C9), thereby creating the several data sets that will be stitched together.

```
%* Transpose vectors representing component matrices ;;

   %macro TransposeMatrix(component);
      data &component.m;
         array square{9,9} %elements(square);
         array cols{9} c1-c9;
         set samurai(keep=matrix %elements(square)
            where=(matrix eq "&component.")) ;
         do r = 1 to 9;
            do c = 1 to 9;
               cols{c} = square{r,c};
               end;
            output;
            end;
```

```
      keep r c1-c9;
   run;
%mend TransposeMatrix;

%TransposeMatrix(UL)  ;
%TransposeMatrix(UR)  ;
%TransposeMatrix(LL)  ;
%TransposeMatrix(LR)  ;
%TransposeMatrix(C)   ;
```

In order to create a reporting data set from the component data sets, the process must be done in a piece-meal fashion, that is, it is necessary to rename variables appropriately such that the variables are named X1-X21. The %RenameSMVars macro performs the task of renaming variables accordingly.

```
%macro RenameSMVars(plus);
   %do i = 1 %to 9;
      c&i. = y%eval(&i.+&plus.)
      %end;
%mend RenameSMVars;
```

Having the utility macro on-hand, the following Data step creates the first of three data sets, specifically, the upper portion of the Samurai (star) puzzle, by merging the ULM(Upper Left Matrix) and URM(Upper Right Matrix) component data sets. Notice, after the sixth iteration of the Data step, the SET statement reads the CM (Center Matrix) data set. The variables in the ULM, URM, and CM data sets are renamed, accordingly, to map their original names to the appropriate X1-X21 variables names.

```
data upper;
   retain y10 y11 y12 .;
   merge ulm(rename=(%RenameSMVars(0)))
         urm(rename=(%RenameSMVars(12)));
      by r;
   if _n_ gt 6
      then set cm(rename=(%RenameSMVars(6)));
run;
```

Similarly, the following Data step creates a subset of the CM (Center Matrix) data set by accessing the last three observations (7-9) and reassigning the variable R denoting the *i*th row. The next Data step creates the data set BOTTOM using data sets by merging the LLM (Lower Left Matrix), LRM (Lower Right Matrix), and CM (Center Matrix) data sets. The variables in LLM, LLRM, and modified CM are renamed, accordingly, once again, to map their original names to the appropriate X1-X21 variable names.

```
data cm_;
   set cm(drop=r firstobs=7);
   r+1;
run;

data bottom;
   retain y10 y11 y12 .;
   merge llm(rename=(%RenameSMVars(0)))
         lrm(rename=(%RenameSMVars(12)))
         cm_(rename=(%RenameSMVars(6)));
      by r;
run;
```

After creating the UPPER and BOTTOM data sets, the following Data step creates the appropriate reporting data set that depicts the natural shape of a Samurai puzzle. Basically, the three input data sets, UPPER, CM, and BOTTOM, are concatenated in a seemingly strange, but effective, way. Notice that the reporting data set emulates a 21x21 matrix that encompasses the star design.

```
data star(keep=x1-x21);
   array y{*}   y1-y21;
   array x{*}$1 x1-x21;
   set upper cm(firstobs=4 obs=6 rename=(%RenameSMVars(6))) bottom;
   if _n_ le 6
      then do;
         do i = 1 to 21;
            if i in(10,11,12)
               then x{i} = '';
               else x{i} = put(y{i},1.);
            end;
         end;
      else if _n_ le 9
         then do;
            do i = 1 to 21;
               x{i} = put(y{i},1.);
               end;
            end;
         else if _n_ le 12
            then do;
               do i = 1 to 21;
                  if i le 6 or i ge 16
                     then x{i} = '';
                     else x{i} = put(y{i},1.);
                  end;
               end;
            else if _n_ le 15
               then do;
                  do i = 1 to 21;
                     x{i} = put(y{i},1.);
                     end;
                  end;
               else do;
                  do i = 1 to 21;
                     if i in(10,11,12)
                        then x{i} = '';
                        else x{i} = put(y{i},1.);
                     end;
                  end;
run;
```

Finally, the REPORT procedure generates the desired output that depicts the natural format of a Samurai puzzle, as shown below. The WIDTH option and respective Null label help create the desired report.

```
proc report data=star nowindows headline headskip;
   columns x1-x21;
   define x1  / display   width=2 '';   define x2  / display  width=2 '';
   define x3  / display   width=2 '';   define x4  / display  width=2 '';
   define x5  / display   width=2 '';   define x6  / display  width=2 '';
   define x7  / display   width=2 '';   define x8  / display  width=2 '';
   define x9  / display   width=2 '';   define x10 / display  width=2 '';
   define x11 / display   width=2 '';   define x12 / display  width=2 '';
   define x13 / display   width=2 '';   define x14 / display  width=2 '';
   define x15 / display   width=2 '';   define x16 / display  width=2 '';
   define x17 / display   width=2 '';   define x18 / display  width=2 '';
   define x19 / display   width=2 '';   define x20 / display  width=2 '';
   define x21 / display   width=2 '';
run;
```

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

Table 3. Listing of the Samurai Sudoku puzzle.

Notice that the above puzzle has adjacent 3x3 sub-matrices that have no values assigned. Also, the 9x9 Center Matrix has only eight values assigned to it. Thus, one might think that this puzzle poses quite a challenge. Yet, this puzzle is rated “Easy.” It is deferred to the reader to learn about the criteria that determines the level of difficulty of Samurai puzzles.

INITIALIZING THE CUBES

At the onset of the Samurai Sudoku puzzle, the majority of cells contain blanks and only a small percentage of the cells contain actual integers (1-9). The SAMURAI data set stores this information efficiently from which five cubes are initialized using the %cubes macro. This macro contains a Data step that creates a data set representing the cube for each component puzzle. For example, the data set CUBE_UR represents the 3-dimensional structure (XY and Z-vector) that defines the cube for the Upper Right component puzzle.

Initializing a cube involves the assignment of the Z-vector based on the value of the XY coordinate in the puzzle. This process requires knowing two things for any XY coordinate: 1) which sub-matrix where it resides; and, 2) the starting coordinate of a sub-matrix. For example, the coordinate {1,6} resides in sub-matrix 2 that has the starting coordinate of {1,4}. This information is obtained from two formats: blkf and $blkxy, respectively. The latter format is defined easily using the FORMAT procedure. The other format requires more effort, creating a CNTLIN data set, which is explained in the author’s previous paper.

For one observation in the SAMURAI data set, representing a two-dimensional array, the Data step creates a single observation data set representing its cube, a three-dimensional array. By traversing the two-dimensional array, the Z-vectors in the cube are assigned either a null value, a zero, or the integer value at the XY coordinate. Null values represent eligible values; whereas, zero values represent ineligible values that were determined by an existing integer value in the sub-matrix. Remember – An integer can be used only once in a sub-matrix.

```
proc format;
   value $blkxy 1 = '1,1'   2 = '1,4'   3 = '1,7'
                4 = '4,1'   5 = '4,4'   6 = '4,7'
                7 = '7,1'   8 = '7,4'   9 = '7,7';
run;

%macro cubes(component);
   data cube_&component.;
      array square{9,9} %elements(square);
      array cube{9,9,9} %elements(cube);
```

```
      set samurai(where=(matrix eq "&component."));
      do x = 1 to 9;
         do y = 1 to 9;
            if square{x,y} ne .
               then do;
                  xy   = put(x,1.) || put(y,1.);
                  blkx = input(scan(put(put(input(xy,blkf.),1.),$blkxy.),1),1.);
                  blky = input(scan(put(put(input(xy,blkf.),1.),$blkxy.),2),1.);
                  do z = 1 to 9;
                     if square{x,y} eq z
                        then cube{x,y,z} = z;
                        else cube{x,y,z} = 0;
                     end;
                  z = square{x,y};
                  do bx = blkx to blkx+2;
                     do by = blky to blky+2;
                        if square{bx,by} eq z
                           then cube{bx,by,z} = z;
                           else cube{bx,by,z} = 0;
                        end;
                     end;
                  do vy = 1 to 9;
                     if square{x,vy} eq z
                        then cube{x,vy,z} = z;
                        else cube{x,vy,z} = 0;
                     end;
                  do vx = 1 to 9;
                     if square{vx,y} eq z
                        then cube{vx,y,z} = z;
                        else cube{vx,y,z} = 0;
                     end;
                  end;
            end;
         end;
      keep xy:;
      drop xy;
   run;
%mend cubes;

%cubes(UL);     %cubes(UR);     %cubes(LR);     %cubes(LL);     %cubes(C);
```

THE RECIPROCATION PROCESS

Before proceeding with solving the puzzle, the cubes are refined by a process called reciprocation; that is, there is an exchange of information between the adjacent cubes. However, the primary facet of the expanded solution involves the transfer of information between the Center matrix and respective outer matrices called reciprocation. The reciprocation process is bi-directional, from the Center puzzle to an Outer puzzle and vice versa. Thus, since there is one Center puzzle and four Outer puzzles, there are eight transfers done. After each exchange, the component cube is updated.

The %Reciprocate macro contains only macro invocations, listed below. The macro produces reports *Before* and *After* the exchange of information between the corner matrices and the updating of the respective cubes for the Center and Outer puzzles. Notice the parameter values that indentify the puzzle and the beginning coordinate of the corner matrices.

%Rep_SubCube	Produces a report on the corner matrix of a component puzzle.
%Exchange	Exchanges information between the corner puzzles belonging to the Center and an outer puzzle.
%Update	Updates the component cube

```
%macro Reciprocate;
    %Rep_SubCube(UL,7,9,7,9);   %Exchange (UL,7,7,C, 1,1);  %UpdateCube(UL);  %Rep_SubCube(UL,7,9,7,9);
    %Rep_SubCube(C, 1,3,1,3);   %Exchange (C, 1,1,UL,7,7);  %UpdateCube(C);   %Rep_SubCube(C, 1,3,1,3);
    %Rep_SubCube(UR,7,9,1,3);   %Exchange (UR,7,1,C, 1,7);  %UpdateCube(UR);  %Rep_SubCube(UR,7,9,1,3);
    %Rep_SubCube(C, 1,3,7,9);   %Exchange (C, 1,7,UR,7,1);  %UpdateCube(C);   %Rep_SubCube(C, 1,3,7,9);
    %Rep_SubCube(LR,1,3,1,3);   %Exchange (LR,1,1,C, 7,7);  %UpdateCube(LR);  %Rep_SubCube(LR,1,3,1,3);
    %Rep_SubCube(C, 7,9,7,9);   %Exchange (C, 7,7,LR,1,1);  %UpdateCube(C);   %Rep_SubCube(C, 7,9,7,9);
    %Rep_SubCube(LL,1,3,7,9);   %Exchange (LL,1,7,C, 7,1);  %UpdateCube(LL);  %Rep_SubCube(LL,1,3,7,9);
    %Rep_SubCube(C, 7,9,1,3);   %Exchange (C, 7,1,LL,1,7);  %UpdateCube(C);   %Rep_SubCube(C, 7,9,1,3);
%mend Reciprocate;

%Reciprocate ;
```

REPORTING THE SUB-CUBE

In order to illustrate the effect of the reciprocating process between two 3x3 corner matrices belonging to the Center and Outer puzzles, as well as the updating of the cubes representing the component puzzles, the %Rep_SubCube renders a report showing a two-dimensional perspective of the cube that supports the 3x3 corner matrix. Actually, the Data step processes the whole cube creating a data set containing the X-Y coordinates juxtaposed with the Z-vector (i.e. variables Z1-Z9). However, because of the WHERE statement, the REPORT procedure shows only the information specific to the 3x3 corner matrix.

```
%macro Rep_SubCube(matrix,xstart,xend,ystart,yend);
   data rep;
      array cube{9,9,9} %elements(cube);
      array vector{9}   z1-z9;
      set cube_&matrix.(keep=%elements(cube));
      do x = 1 to 9;
         do y = 1 to 9;
            block = input(put(x,1.) || put(y,1.),blkf.);
            do z = 1 to 9;
               vector{z} = cube{x,y,z};
               end;
            output;
            end;
         end;
   run;

   proc report data=rep nowindows headline headskip;
      columns x y ('- Z -' z1-z9);
      define x       / order     width=1       'X';
      define y       / order     width=1       'Y';
      define z1      / display   width=1       '1'    spacing=6;
      define z2      / display   width=1       '2';
      define z3      / display   width=1       '3';
      define z4      / display   width=1       '4'    spacing=6;
      define z5      / display   width=1       '5';
      define z6      / display   width=1       '6';
      define z7      / display   width=1       '7'    spacing=6;
      define z8      / display   width=1       '8';
      define z9      / display   width=1       '9';
      break after x / skip;
      where (x between &xstart. and &xend.) and (y between &ystart. and &yend.);
      title2 "%upcase(&matrix.) 3x3 Sub-Matrix -- Cube";
   run;
%mend Rep_SubCube;
```

Consider the last set of invocations in the %Reciprocate macro. The two reports below show the *Before* and *After* effect of the %Exchange and %UpdateCube macros. Initially, the cube representing the 3x3 corner matrix of the Center puzzle shows mostly missing values and several zeros, denoting eligible and ineligible numbers, respectively. However, after the exchange and update processes, the second report shows many more zeros, revealing more ineligible numbers.

The Cube BEFORE the Exchange and Update Processes

		Z								
X	Y	1	2	3	4	5	6	7	8	9
7	1	.	.	.	0	.	.	.	.	.
	2	.	.	.	0	.	.	.	.	.
	3	.	.	.	0	.	.	.	0	.
8	1	.	.	.	.	.	.	.	.	.
	2	.	.	.	.	.	.	.	.	.
	3	.	.	.	.	.	.	.	0	.
9	1	.	.	.	.	.	.	.	.	.
	2	.	.	.	.	.	.	.	.	.
	3	.	.	.	.	.	.	.	0	.

The Cube AFTER the Exchange and Update Processes

		Z								
X	Y	1	2	3	4	5	6	7	8	9
7	1	.	0	0	0	.	.	0	.	.
	2	.	0	0	0	.	.	.	0	.
	3	.	.	0	0	0	.	.	0	.
8	1	0	0	.	.	.	.	0	.	0
	2	0	0	.	0	.	.	.	0	0
	3	0	0	.	.	0	.	.	0	0
9	1	.	0	.	.	0	.	0	.	.
	2	.	0	.	0	0	.	0	0	.
	3	.	.	.	.	0	.	0	0	.

Table 4. The Reciprocation Process.

It is the exchange of information at the corner matrices that influences the total cube. And, it is the continuous transformation of the cube that solves the puzzle.

THE SOLVER (DRIVER) MACRO

The %solver macro is the main driver of this SAS solution. Each iteration in the %DO %WHLE loop executes a set of macros that affect each of the five component puzzles. Each macro is discussed separately.

```
%macro solver;
   %let notsolved = 1;
   %do %while(&notsolved.);
      %solve(UL) ;   %Exchange(UL,7,7,  C,1,1) ;   %UpdateCube(UL) ;     %CheckComp(UL) ;
                     %Exchange(C ,1,1, UL,7,7) ;   %UpdateCube(C)  ;     %CheckComp(C)  ;

      %solve(UR) ;   %Exchange(UR,7,1,  C,1,7) ;   %UpdateCube(UR) ;     %CheckComp(UR) ;
                     %Exchange(C ,1,7, UR,7,1) ;   %UpdateCube(C)  ;     %CheckComp(C)  ;

      %solve(LR) ;   %Exchange(LR,1,1,  C,7,7) ;   %UpdateCube(LR) ;     %CheckComp(UR) ;
                     %Exchange(C ,7,7, LR,1,1) ;   %UpdateCube(C)  ;     %CheckComp(C)  ;

      %solve(LL) ;   %Exchange(LL,1,7,  C,7,1) ;   %UpdateCube(LL) ;     %CheckComp(LL) ;
                     %Exchange(C ,7,1, LL,1,7) ;   %UpdateCube(C)  ;     %CheckComp(C)  ;

      %solve(C)  ;                                 %UpdateCube(C)  ;     %CheckComp(C)  ;
      %notsolved;
      %end;
 %mend solver;
```

THE SOLVE MACRO

The %solve macro makes twenty attempts to solve the puzzle. For each iteration, it uses the cube to locate a valid value, that is, the only remaining value in the Z-vector.

```
%macro solve(component);
   data cube_&component.;
      array square{9,9}  %elements(square);
      array cube{9,9,9}  %elements(cube);
      array vector{9}    v1-v9;
      set cube_&component.;
      do run = 1 to 20;

         do x = 1 to 9;
            do y = 1 to 9;
               do z = 1 to 9;
                  vector{z} = cube{x,y,z};
                  end;
               if nmiss(of vector{*}) eq 1
                  then do;
                     do z = 1 to 9;
                        if vector{z} eq .
                           then leave;
                        end;
                     square{x,y} = z;
                     link cube;
                     end;
               end;
            end;

         do y = 1 to 9;
            do z = 1 to 9;
               do x = 1 to 9;
                  vector{z} = cube{x,y,z};
                  end;
               if nmiss(of vector{*}) eq 1
                  then do;
                     do x = 1 to 9;
                        if vector{x} eq .
                           then leave;
                        end;
                     square{x,y} = z;
                     link cube;
                     end;
               end;
            end;

         do x = 1 to 9;
            do z = 1 to 9;
               do y = 1 to 9;
                  vector{z} = cube{x,y,z};
                  end;
               if nmiss(of vector{*}) eq 1
                  then do;
                     do y = 1 to 9;
                        if vector{y} eq .
                           then leave;
                        end;
                     square{x,y} = z;
                     link cube;
                     end;
               end;
            end;

      nmiss = nmiss(of square{*});
      if nmiss(of square{*}) eq 0
         then do;
```

```
            output;
            stop;
            end;
      end;
   output;
   return;

cube:
   xy     = put(x,1.) || put(y,1.);
   blkx   = input(scan(put(put(input(xy,blkf.),1.),$blkxy.),1),1.);
   blky   = input(scan(put(put(input(xy,blkf.),1.),$blkxy.),2),1.);
   do z = 1 to 9;
      if square{x,y} eq z
         then cube{x,y,z} = z;
         else cube{x,y,z} = 0;
      end;
   z = square{x,y};
   do bx = blkx to blkx+2;
      do by = blky to blky+2;
         if square{bx,by} eq z
            then cube{bx,by,z} = z;
            else cube{bx,by,z} = 0;
         end;
      end;
   do vy = 1 to 9;
      if square{x,vy} eq z
         then cube{x,vy,z} = z;
         else cube{x,vy,z} = 0;
      end;
   do vx = 1 to 9;
      if square{vx,y} eq z
         then cube{vx,y,z} = z;
         else cube{vx,y,z} = 0;
      end;
   return;
   keep xy:;
   drop xy;
run;
%mend solve;
```

THE EXCHANGE MACRO

The %Exchange macro is a new facet of the SAS solution. Since Samurai Sudoku puzzles are interdependent, it is necessary to exchange information with respect to pair-wise cubes (e.g. Upper Left and Center puzzles). The macro contains six positional parameters, as follows:

- TARGET Target Component Puzzle
- TX Target X-coordinate
- TY Target Y-coordinate
- SOURCE Source Component Puzzle
- SX Source X-coordinate
- SY Source Y-coordinate

Once again, the task is performed in a single, rather complex, Data step that compares the 3x3 corner sub-matrices belonging to an Outer puzzle (e.g. Upper Left) and the Center puzzle. Consider the first macro invocation that compares the 3x3 corner matrices belonging to the Upper Left and Center puzzles.

```
%Exchange(UL,7,7,  C,1,1);

%Exchange(C ,1,1, UL,7,7);
```

Notice that the corner matrix for the Upper Left puzzle begins at the coordinate {7,7}; whereas, the corner matrix for the Center puzzle begins at the coordinate {1,1}. If a cell in the Outer puzzle is null and the respective cell in the Center puzzle contains a value, then that information is transferred to the Outer puzzle. Moreover, if the value of the Target cube is not zero, then the cell of the Target square (the actual puzzle) is assigned that value. The exchange process is done again in reverse order, comparing the Center puzzle with the Upper Left puzzle.

```
%macro Exchange(target,tx,ty, source,sx,sy);
   data cube_&target.;
      set cube_&target.;
      array &target.xyz{1:3,1:3,1:9} %EnumVarsXYZ(xyz,&tx.,&ty.);
      array &target.xy{1:3,1:3}      %EnumVarsXY (xy,&tx.,&ty.);

      array &source.xyz{1:3,1:3,1:9} %EnumVarsXYZ(&source.,&sx.,&sy.);
      array &source.xy{1:3,1:3}      %EnumVarsXY (&source.,&sx.,&sy.);

      set cube_&source.(keep= %EnumVarsXY (xy,&sx.,&sy.) %EnumVarsXYZ(xyz,&sx.,&sy.)
         rename=(%RenameXY(&source.,&sx.,&sy.) %RenameXYZ(&source.,&sx.,&sy.)));
      do x = 1 to 3;
         do y = 1 to 3;
            do z = 1 to 9;
               if &target.xyz{x,y,z} eq . and &source.xyz{x,y,z} ne .
                 then do;
                    &target.xyz{x,y,z} = &source.xyz{x,y,z};
                    if &target.xyz{x,y,z} ne 0
                       then &target.xy{x,y} = &source.xyz{x,y,z};
                    end;
               end;
            end;
         end;
      keep xy:;
%mend Exchange;
```

UPDATING THE CUBE

As the name implies the %UpdateCube macro simply updates the cube for a component puzzle, which is accomplished in a single Data step. Two data structures, 2-dimensional and 3-dimensional matrices, represent the square and the cube, respectively. Fundamentally, the current state of the square is used to update the cube; that is, if the *i*th cell contains a value, then the cube is updated regarding its respective vector, as well as the vectors belonging to the sub-matrix where it resides.

```
%macro UpdateCube(component);
   data cube_&component.;
      array square{9,9} %elements(square);
      array cube{9,9,9} %elements(cube);
      set cube_&component.;
      do x = 1 to 9;
         do y = 1 to 9;
            if square{x,y} ne .
               then do;
                  xy   = put(x,1.) || put(y,1.);
                  blkx = input(scan(put(put(input(xy,blkf.),1.),$blkxy.),1),1.);
                  blky = input(scan(put(put(input(xy,blkf.),1.),$blkxy.),2),1.);
                  do z = 1 to 9;
                     if square{x,y} eq z
                        then cube{x,y,z} = z;
                      * else cube{x,y,z} = 0;
                     end;
                  z = square{x,y};
```

```
                  do bx = blkx to blkx+2;
                     do by = blky to blky+2;
                        if square{bx,by} eq z
                           then cube{bx,by,z} = z;
                           else cube{bx,by,z} = 0;
                        end;
                     end;
                  do vy = 1 to 9;
                     if square{x,vy} eq z
                        then cube{x,vy,z} = z;
                        else cube{x,vy,z} = 0;
                     end;
                  do vx = 1 to 9;
                     if square{vx,y} eq z
                        then cube{vx,y,z} = z;
                        else cube{vx,y,z} = 0;
                     end;
                  end;
            end;
         end;
      keep xy:;
      drop xy;
   run;
%mend %UpdateCube;
```

THE NOT SOLVED MACRO

The %notsolved macro determines whether the puzzle has been solved. The macro contains a Data _null_ step that concatenates the five cubes, specifically those XY variables representing the Square. A simple SUM statement is used with the NMISS function that will aggregate the number of missing values in GRID, a 9x9 two-dimensional array. Finally, the Data step creates the macro variable NOTSOLVED. Obviously, the puzzle is solved when there are no missing values.

```
%macro notsolved;
   data _null_;
      array grid{9,9} %elements(square);
      set cube_ul(keep=%elements(square))
          cube_ur(keep=%elements(square))
          cube_lr(keep=%elements(square))
          cube_ll(keep=%elements(square))
          cube_c (keep=%elements(square)) end=eof;
      total + nmiss(of grid{*});
      if eof
         then call symput('notsolved',total);
   run;
%mend notsolved;
```

THE SOLUTION

Appendix A shows the solution to the puzzle. This puzzle took about a dozen iterations to complete. The report clearly shows the solution, even the adjoining 3x3 corner matrices. For example, the corner matrix connecting the Upper Left and Center puzzles contains the values: 9 1 5; 4 8 6; and, 2 3 7, which does not conflict with any other response in the rows and columns in the puzzles.

CONCLUSION

The proposed SAS solution exploits the notion of a dynamic cube that maintains information about eligible and ineligible numbers for each cell in a puzzle. For Samurai Sudoku puzzles, five cubes are employed to maintain this information. The solution is further extended by exchanging information at the adjoining corner matrices.

This solution does not employ the bootstrapping method explained in the author's previous paper, *Solving Sudoku Puzzles – Using a Cube to Solve the Square,* which was used to solve more advanced regular puzzles. Perhaps renewed effort should consider a bootstrapping component. Nonetheless, this solution solves easy Samurai Sudoku puzzles, which is a good first attempt.

REFERENCES

DeVenezia, Richard, et al.; "SAS and Sudoku – A Roundtable Discussion." *Proceedings of the SAS Global Forum Conference, 2007.*

Gerlach, John R.; "Solving Sudoku Puzzles – Using a Cube to Solve the Square." *Proceedings of the Southeast SAS User's Group Conference, 2007.*

ACKNOWLEDGMENTS

Special thanks to my wife Marcia who wanted to go to Vegas to see a Celine Dion concert in 2006. That trip prompted an incidental visit to a bookstore where the author found a pile of Sudoku books, which inspired the author to write several SAS papers on solving Sudoku puzzles. The trip to Vegas prompted several other trips: Atlanta (SESUG 2006), Orlando (SGF 2007), and San Francisco (SGF 2013).

CONTACT INFORMATION

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

Name:	John R. Gerlach
Enterprise:	CSG, Inc.
Address:	7780 Brier Creek Prkway, Suite 330
City, State ZIP:	Raleigh NC 27617
Work Phone:	609-672-5034
E-mail:	JRGerlach@optonline.net
Web:	www.csg-inc.com

APPENDIX A – SOLUTION TO SAMURAI SUDOKU PUZZLE

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