

Sudoku Puzzles – Using a Cube to Solve the Square

John R. Gerlach, MaxisIT, Inc

Abstract

A Sudoku puzzle consists of a 9x9 matrix into which several numbers, ranging from 1 to 9, have been placed leaving the majority of cells blank. The object of the puzzle is to complete the square such that each row, each column, and each 3x3 sub-matrix uses each digit *only* once. Simple rules to follow, indeed; yet, Sudoku poses a challenge that requires lots of patience and perseverance, and no math skills. This paper explains a SAS solution for solving Sudoku puzzles by using an abstract data structure, the cube, positioned behind the square.

Introduction

Consider the following Sudoku puzzle juxtaposed with its solution. Notice that each row, column, and 3x3 sub-matrix uses the digits from 1 through 9 *only* once. In fact, Sudoku puzzles are Latin Squares with the added restriction that each 3x3 sub-matrix uses each digit once, as well.

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

Typically, a Sudoku puzzle has about 30 of its 81 cells filled. Although the numbers 1 through 9 are used, any symbols (even fruit) could be used instead. Moreover, in a standard puzzle, if the values 1 through 9 were replaced by a new set of values (or fruit), such as the mapping: 1=3, 2=7, 3=5, 4=2, 5=8, 6=4, 7=9, 8=6, 9=1; the puzzles would be considered equivalent. Likewise, if the first three rows were swapped with the last three rows, or the first three columns with the middle three columns, etc., the puzzles are equivalent. Nonetheless, the number of basic Sudoku grids exceeds five billion. There are plenty of puzzles for the Sudoku fan to solve.

A valid Sudoku puzzle requires at least 17 cells populated in order to produce a unique solution. Conversely, a puzzle having all but four cells populated might prohibit a unique solution. For example, this occurs when two values can be applied to adjacent columns and adjacent sub-matrices. For more mathematical insight about Sudoku puzzles, read the article in the June 2006 issue of *Scientific American*.

Traditional Puzzle Solving

Sudoku puzzles are intended to be done using only logic and a pencil with a large eraser. Common techniques for solving these puzzles range from common sense to more obscure tactics in an attempt to conclude the only possible value for a given cell, of course, contingent on the values of adjacent and respective row / column cells.

Given the following Sudoku puzzle, note the following conventions concerning the Sudoku grid: the coordinate (1,1) denoting row 1 and column 1, begins in the upper left corner and coordinate (9,9) is caddy corner to (1,1), of course. The sub-matrices consist of 3 rows and 3 columns beginning from the upper left corner and proceeding left to right. For example, sub-matrix 6 consists of rows 4,5,6 and columns 7,8,9 and contains the values: 2, 5, 6, and 9.

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

Rather than expound on techniques, such as “Eliminating the Possibilities” and the inevitable “Trial and Error,” several imputations are shown so that the reader can appreciate the logic required to solve Sudoku puzzles and, more importantly, the SAS solution that solves such puzzles. The notation $N \rightarrow (x,y)$ denotes a proposed value that is assigned to a coordinate in the grid.

- 1 $\rightarrow$ (4,9) The number 1 is found in rows 5 and 6 and column 8. Thus, sub-matrix 6 contains the value 1 in cell (4,9).
- 9 $\rightarrow$ (2,7) The number 9 is found in rows 1 and 3. Thus, sub-matrix 3 contains the value 9 in cell (2,7).
- 4 $\rightarrow$ (1,4) The number 4 is found in rows 2 and 3. The number 4 is found in cells (6,2) and (7,5), thereby eliminating all cells in Row 1, except for cell (1,4). Why is cell (1,9) disqualified?
- 5 $\rightarrow$ (5,5) The number 5 is found in rows 4 and 6. Thus, sub-matrix 5 contains the number 5 at cell (5,5).
- 5 $\rightarrow$ (3,6) The number 5 is found in cell (2,9), thereby eliminating row 2. Where should the number 5 be placed in sub-matrix 2? In cell (1,5), since the number 5 was placed in cell (5,5) by the previous step.
- 1 $\rightarrow$ (2,4) The number 1 is found in columns 5 and 6, sub-matrices 5 and 8. Thus, cell (2,4) contains the value 1.
- 5 $\rightarrow$ (7,1) This one is more difficult. First realize that the sub-matrix 7 can hold the number 5 either in cell (7,1) or (7,2), since it is found in rows 8 and 9. Also, notice that the number 5 is found in rows 2 and 3 affecting sub-matrix 1, such that the number 5 can *only* be in cell (1,2), since that number is found already in column 3. Consequently, cells (7,1) and (1,2) both contain the number 5.

And so on, until the puzzle is completed, as shown below.

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

A SAS Solution

Sudoku puzzles offer quite a challenge for brain-teaser fans with lots of patience and plenty of erasers. But, to write a SAS solution that solves Sudoku puzzles systematically poses quite a different challenge. The proposed solution emulates the notion of constraint propagation; meaning, the event of placing a number in a cell invokes the process of updating an abstract data structure (the cube) in order to identify viable numbers for other cells. Most programming solutions use the backtracking method, which is little more than the trial and error method relying on the power of the computer.

The proposed SAS solution is now explained. The values in the square can be represented as a triple (x,y,z) such that X and Y represent the row/column coordinate and Z represents the vector of possible values at the XY coordinate. In fact, the Z coordinate is the possible value for the XY coordinate. Thus, the cube represents a collection of orthogonal vectors containing the possible values for each cell in the grid. There are two more important observations about the cube:

- There are n^2 triples of the form x,y,z where $1 \leq x,y,z \leq n$.
- All XY pairs are different, all XZ pairs are different, and all YZ pairs are different.

Keep in mind that the cube contains the possible values associated with each blank cell on the square.

Elements of the Square and Cube

In the SAS program, the Sudoku square is represented by the set of variables having the form $XYxy$, where xy denotes the coordinate location in the square. Thus, for a 9x9 matrix, there are 81 variables named:

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

Similarly, the cube is represented by the set of variables having the form $XYZxyz$, where xyz denotes the coordinate location of the cell in the cube. Thus, the cube contains 729 variables named:

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

So, for example, the variable $XYZ258$ represents the 2nd row, the 5th column, and the 8th cell in cube. Again, notice that the Z coordinate denotes the possible value that is assigned to the XY coordinate of the square.

The `%elements` macro below generates the enumerated variable names for the square and the cube by specifying the geometric figure, either a square or a cube. Notice that the macro takes advantage of the abbreviated form of enumerated variables in the SAS language by using the dash (e.g., $XY11$ - $XY19$).

```
%macro elements(type);
  %if %upcase(&type.) ne CUBE
    %then %do;
      %do x = 1 %to 9; xy&x.1-xy&x.9 %end;
    %end;
  %else %do;
    %do x = 1 %to 9;
      %do y = 1 %to 9; xyz&x.&y.1-xyz&x.&y.9 %end;
    %end;
  %end;
%mend elements;

%elements(square) ;

%elements(cube) ;
```

Mapping X-Y Coordinates to a Sub-Matrix

Part of the process of solving a Sudoku puzzles requires the ability to identify a sub-matrix, or block, for any XY coordinate. For example, the XY coordinate (4,7), that is, row 4 and column 7, maps to sub-matrix 6. To facilitate this process, the XY coordinate values are represented as two digit numbers, as follows: 11 - 19, 21 - 29, ..., 91-99 and the block values range from 1 through 9, since there are nine 3x3 sub-matrices. The following Data step generates a control input

data set used to create the input format called *blkf* that maps each XY coordinate to its respective sub-matrix. A partial listing of the control data set follows the Data step and the Format procedure.

```
data blocks;
  retain fmtname 'blkf' type 'I';
  do x = 1 to 9;
    select(x);
      when(1,2,3) block=1;
      when(4,5,6) block=4;
      when(7,8,9) block=7;
    end;
  do y = 1 to 9;
    start = put(x,1.) || put(y,1.);
    label = block;
    output;
    if mod(y,3) eq 0
      then block+1;
    end;
  end;
  keep fmtname type start label;
run;

proc format cntlin=blocks;
run;
```

Obs	fmtname	type	start	label
1	blkf	I	11	1
2	blkf	I	12	1
3	blkf	I	13	1
4	blkf	I	14	2
5	blkf	I	15	2
6	blkf	I	16	2
:	:	:	:	:
75	blkf	I	93	7
76	blkf	I	94	8
77	blkf	I	95	8
78	blkf	I	96	8
79	blkf	I	97	9
80	blkf	I	98	9
81	blkf	I	99	9

Another important facet of this SAS solution requires knowing the *starting* coordinate of a given sub-matrix. For example, the 4th sub-matrix in the Sudoku grid begins at coordinate (4,1); whereas, the 9th sub-matrix begins at coordinate (7,7). The following format *\$blkxy* maps easily maps each sub-matrix to its starting XY coordinate.

```
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;
```

In summary, the *blkf* format maps an XY coordinate to its respective sub-matrix, or block, and the *\$blkxy* format maps a sub-matrix to its starting XY-coordinate. The ability to impute these values becomes vital when solving the Sudoku puzzle with respect to the sub-matrices.

Reading Sudoku Puzzles

A typical Sudoku puzzle is a 9x9 matrix with about 60% of its cells left blank. Consider the following data records that emulate such a puzzle. Each record contains nine values representing a row in the matrix, such that the ‘blanks’ are denoted by a period, which translates to a numeric missing value in SAS. A data file could contain many puzzles consisting of $9n$ records (9 records per puzzle), as long as the data set contains a variable denoting the i th puzzle for subsequent processing. Regardless of which puzzle is being processed, the initial objective is to populate the eighty-one variables XY11 through XY99 denoting a 2-dimensional data structure.

The Data step below reads the aforementioned data file and generates one observation for each puzzle. The final data set contains the variables: PUZZLE, denoting the i th puzzle, and the eighty-one variables (e.g. XY48) denoting all the cells in the Sudoku square. Notice that the macro `%elements` is employed to generate the enumerated XY-variables. In order to map the variables P1-P9 to the 2-dimensional matrix, the row (R) denoting the n th record in the set of records and the loop control variable (C) are used. The variable PUZZLE is assigned appropriately using the SUM statement which is executed conditionally as determined by the modulus function in SAS.

Below is a listing of the SUDOKU data set that contains five Sudoku puzzles illustrating how each observation represents a single puzzle.

5

Initializing The Cube

Given a data set that contains a collection of Sudoku puzzles, we proceed to solve a puzzle. Imagine that you're looking at a transparent Sudoku grid such that you can see the cube behind the grid having likewise dimensions. Two ARRAY statements at the top of the following Data step define these two data structures: the square and the cube using the %elements macro to define the 2 and 3-dimensional arrays.

```
data sudoku1;
  array square{9,9} %elements(square);
  array cube{9,9,9} %elements(cube);
  set sudoku;
```

The following two DO-loops begin walking through the square, looking for non-missing values, in order to populate the cube. with as much information as possible. Notice that the cube contains three types of values:

- Z-value The value that has been assigned to the grid
- Zero The Z-value is no longer viable
- Missing A possible value for the grid.

For each non-missing value in the square, the XY coordinate is used to populate the cube, as follows:

- Populate the Z-vector in the cube with zeros, except for the XYZ coordinate that is assigned the value of the XY coordinate. For example, in the current puzzle, the XY coordinate (1,1) contains the value 9. Thus, the cube will be assigned the values zero for the Z-vector at (1,1), except for the cell (1,1,9), which contains the value nine.
- Populate the cube with respect to the sub-matrix. Recall that a sub-matrix can manifest a digit only once and the cube needs to contain that vital information. Again, the coordinate (1,1) contains the value 9; thus, the Z-vectors for cells (1:3,1:3,9) will be assigned the value zero, except for the coordinate (1,1,9).
- Populate the Y-vector in the cube based on the constant values of X and Z, assigning the values 0 or Z.
- Populate the X-vector in the cube based on the constant values of Y and Z, assigning the values 0 or Z.

```
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.),$blkxy.),1),1.);
        blkz = input(scan(put(put(input(xy,blkf.),1.),$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 = blkz to blkz+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;
    end;
  end;
keep xy;;
```

```
drop xy;
run;
```

Observe the following partial listings of the cube pivoting on the XY coordinate and the sub-matrix (Block 1). The first listing shows the Z-vector for the XY coordinate (1,1), which contains all zeros, except the last one, having the value 9. The second listing emphasizes a specific column (9) belonging to the Z-vector. Notice that except for the coordinate (1,1,9) the values are zero, which denotes that this value is no longer available for other cells in that sub-matrix.

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

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

Solving The Square

Once the cube has been populated with either Z-values, zeros, or missing values, the next step is to actually solve the puzzle using the method called constraint propagation. The following Data step processes a single observation with over 800 numeric variables including XY11-XY99, XYZ111-XYZ119, and XYZ911-XYZ999. By convention, it makes twenty attempts to solve the puzzle, and then terminates normally. But how does it solve the puzzle?

Consider an orthogonal Z-vector for a given XY coordinate on the grid. After initializing the cube, it so happens that there are instances where there is only one missing value in that vector; whereas, the others are either respective Z-values or zero. In such cases, the missing value is the only possible answer for that XY coordinate.

For each iteration the program walks through the grid (cell by cell), picks up the Z-vector for that cell from the cube, determines whether there is only *one* missing value, identifies the Z-value for that missing value, and assigns it to that XY cell on the grid. Then, in the event of finding a value for the grid, the Data step proceeds to update the cube, which is similar to the initialization of the cube. Finally, it checks to determine whether there are any more empty XY cells.

Now comes a peculiar feature of the solution. It becomes necessary to perform this process pivoting on YZX and XZY coordinates, as well as the obvious XYZ coordinates. It so happens that this process requires all three modes in order to solve more difficult puzzles, thereby optimizing the cube. It is left for the reader to study that aspect of the solution.

For easy Sudoku puzzles, the SAS solution solves a puzzle well within the twenty-iteration limit.

```
data sudoku2;
  array square{9,9} %elements(square);
  array cube{9,9,9} %elements(cube);
  array vector{9} v1-v9;
  set sudokul;
  do n = 1 to 20;

    * Process X,Y,Z Coordinates ;

    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;
    end;

    * Process Y,Z,X Coordinates ;

    * Process X,Z,Y Coordinates ;

    if nmiss(of square{*}) eq 0
      then do;
        output;
        stop;
      end;
    end;
  output;
  return;

cube:

  * Obtain the beginning X,Y coordinate of the sub-matrix ;

  * Assign Z the value of X,Y ;

  * Update the sub-matrix ;

  * Update the X-vector ;

  * Update the Y-vector ;

  return;

run;
```

Unfair or Unreasonable Puzzles

An unfair Sudoku puzzle occurs when there is insufficient information to solve the puzzle without resorting to relentless trial and error. Although logical techniques help to solve Sudoku puzzles, it becomes obvious that such techniques will not accomplish the task for unfair puzzles. The trial and error approach assigns a number to a cell hardly based on logic, then making more guesses along the way. Unfortunately, when the numbers don't pan out, you must back track to where you initiated the process (i.e., guess) or, at least, to a reasonable point in order to proceed. Since the objective here is to programmatically solve Sudoku puzzles efficiently, an alternative solution is now explained.

Using the SAS solution discussed so far, the following puzzle failed to be solved. In fact, it included only several more values to the puzzle, as illustrated below.

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

An additional module was written to solve unfair puzzles. The solution uses a bootstrapping method by first identifying possible values for the empty cells and storing such information in parallel macro variables: XCOORDS, YCOORDS, and VALUES, achieved by the following several steps. Once again, the `%elements` macro generates the appropriate enumerated variables for the two data structures. Also, notice that SQL step creates the data set TRIPLETS, containing the possible values (inherent in the XYZ coordinate), ordered by the frequency of occurrence based on the X-coordinate in descending fashion. This technique increases the likelihood that the considered triplet will have the maximum affect on the cube when attempting to solve the puzzle.

```
data xyz;
  array square{9,9} %elements(square);
  array cube{9,9,9} %elements(cube);
  set sudoku2;
  do x = 1 to 9;
    do y = 1 to 9;
      if square{x,y} eq .
        then do;
          do z = 1 to 9;
            if cube{x,y,z} eq .
              then output;
            end;
          end;
        end;
      end;
    end;
  keep x y z;
run;

proc sql noprint;
  create table triplets as
  select count(x) as count, *
  from xyz
  group by x
  order by count desc, y, z;
quit;

proc sql noprint;
  select x into :xcoords separated by ' '
  from triplets;
  select y into :ycoords separated by ' '
  from triplets;
  select z into :values separated by ' '
  from triplets;
quit;
```

The `%bootstrap` macro uses three parameters (XCOORD, YCOORD, VALUE), the singular form of the three aforementioned macro variables, which represents a single instance of the collection of possibilities emulating the trial and error approach. The macro is much like the previous module that solves most (i.e., fair) Sudoku puzzles, except that it uses the trial and error method and proceeds as usual by updating the cube. Consequently, the puzzle gets solved, as shown below.

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

Conclusion

Sudoku puzzles should be done using pencil and eraser. The SAS solution discussed in this paper affords an excellent opportunity to learn about related topics such as abstract data structures, design, and methodology. Because of the length and complexity, the full SAS solution was not shown; however, it is hoped that sufficient information was conveyed to appreciate the ideas that solved a rather nasty problem. Ironically, the author has never solved a Sudoku puzzle by hand.

Acknowledgement

I would like to express my appreciation to Maulik Shah, a vice-president at MaxisIT, Inc., who has been extremely supportive of this endeavor in recognizing the importance of professional development.

Bibliography

- Wilson, Robin. *How to Solve Sudoku, A Step-by-step Guide*. England: The Infinite Ideas Company Limited, 2005.
- Heron, Andrew and Edmund James. *Sudoku for Dummies, Volume 3*. England: John Wiley & Sons, LTD, 2005.
- Delahaye, Jean-Paul 2006. "The Science Behind Sudoku." *Scientific American* 294 (6): 80-87.

Author

John R. Gerlach
Senior Data Analyst
MaxisIT, Inc.

jgerlach@maxisit.com