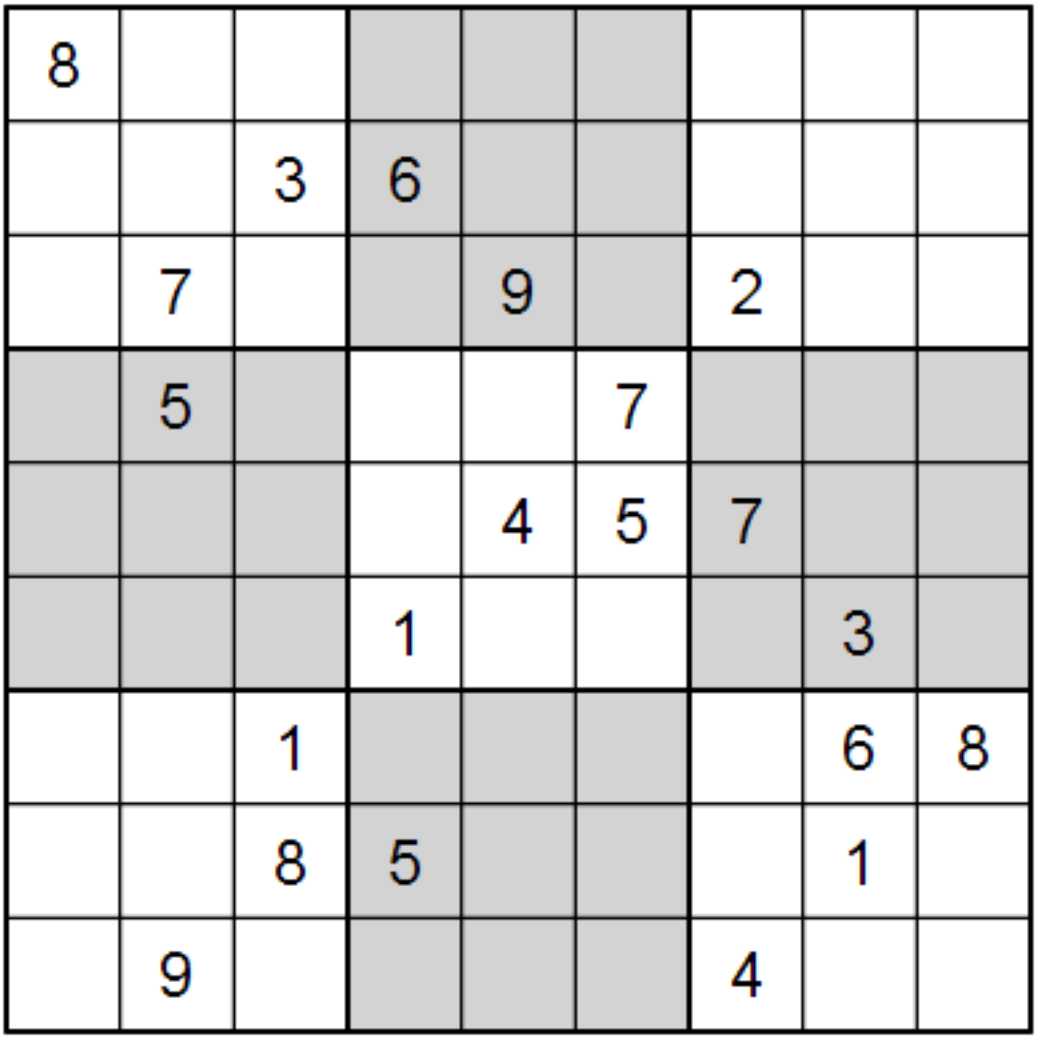
Solving the World's Hardest Sudoku



- Sudoku is a popular puzzle that appears in many puzzle books and newspapers.
- Given a 9 by 9 grid whose squares are either blank or contain a number between 1 and 9, the objective is to fill in the blank squares in such a way so that each row and column contains exactly one digit between 1 and 9. Additionally, each of the nine 3 by 3 subgrids which compose the grid (called blocks) must also contain exactly one digit between 1 and 9.
- The mathematician Arto Inkala published the "world's hardest Sudoku"

(shown above). The puzzle has been estimated to have a difficulty rating of 11 stars, whereas the most challenging Sudoku puzzles that are usually published are given a difficulty of 5 stars.

- Using Maple's built-in efficient SAT solver we can quickly solve this puzzle without any knowledge of Sudoku solving techniques. A SAT solver takes as input a formula in Boolean logic and returns an assignment to the variables that makes the formula true (if one exists). See the Satisfy command for more information.

Setting up the problem

- We'll use the Boolean variables $S[i, j, k]$ for $1 \leq i, j, k \leq 9$ to denote that the square (i, j) contains the digit k.
- We'll also use a few simple functions to compute the set of variables which appear in the same row, column, or block as the square (i, j) and concern the digit k:

```
rowVars := proc(i, j, k)
    return {seq(S[x, j, k], x=1..9)} minus {S[i, j, k]};
end proc:

colVars := proc(i, j, k)
    return {seq(S[i, y, k], y=1..9)} minus {S[i, j, k]};
end proc:

blockVars := proc(i, j, k)
    local block_x, block_y;
    block_x := 3*floor((i-1)/3)+1;
    block_y := 3*floor((j-1)/3)+1;
    return {seq(seq(S[x, y, k], y=block_y..block_y+2), x=block_x..block_x+2)} minus {S[i, j, k]};
end proc:

allVars := proc(i, j, k)
    return rowVars(i, j, k) union colVars(i, j, k) union blockVars(i, j, k);
end proc:
```

Generating the constraints

- To encode the rules of Sudoku in Boolean logic we want to encode both *positive* and *negative* constraints.
- Positive constraints say that each square must contain some digit.
- Negative constraints say that each square cannot contain more than one digit and that squares in the same row, column, or block do not contain the same digit twice.

Positive constraints

- For each square (i, j) at least one digit k appears in the square.
- These clauses have the form $S[i, j, 1] \vee S[i, j, 2] \vee \cdots \vee S[i, j, 9]$ for each $1 \leq i, j \leq 9$.

```
atLeastOneDigit := seq(seq(&or(seq(S[i,j,k], k=1..9)), j=1..9), i=1..9):
```

Negative constraints

- For each square (i, j) if the digit k appears in that square then no other digit appears in that square.
- These clauses have the form $S[i, j, k] \Rightarrow \neg S[i, j, l]$ (alternatively, $\neg S[i, j, k] \vee \neg S[i, j, l]$) where l is a digit with $k \neq l$.

```
atMostOneDigit := seq(seq(seq(seq(S[i, j, k] &implies &not(S[i, j, l]), l=k+1..9), k=1..9), j=1..9), i=1..9):
```

- For each square (i, j) if the digit k appears in that square then the digit k does not appear in any other square in the same row, column, or block.
- These clauses have the form $S[i, j, k] \Rightarrow \neg A$ where A is in the set $allVars(i, j, k)$.

```
distinctnessConstraints := seq(seq(seq(seq(S[i, j, k] &implies &not(A), A in allVars(i, j, k)), k=1..9), j=1..9), i=1..9):
```

Finding a solution

- We also need to specify the 21 numbers which are given to us as a part of the puzzle; if (i, j) contains k then we need to include the unit clause $S[i, j, k]$ in our SAT instance. The clauses resulting from the world's hardest Sudoku are as follows:

```
givenNumbers := S[1,9,8], S[2,1,9], S[2,6,5], S[2,7,7], S[3,2,8], S[3,3,1], S[3,8,3], S[4,2,5], S[4,4,1], S[4,8,6], S[5,5,4], S[5,7,9], S[6,5,5], S[6,6,7], S[7,1,4], S[7,5,7], S[7,7,2], S[8,2,1], S[8,3,6], S[8,4,3], S[9,3,8]:
```

- We use the Satisfy command from the Logic package which finds a satisfying assignment of a logical formula if one exists.
- We use the Usage command from the CodeTools package to measure how quickly the solution is found.

```
allConstraints := atLeastOneDigit, atMostOneDigit, distinctnessConstraints, givenNumbers:
satisfyingAssignment := CodeTools:-Usage(Logic:-Satisfy(&and(allConstraints))):
```

Visualization of the solution

- The following functions use commands from the plots and plottools packages to draw a Sudoku grid and fill in numbers on the grid.

```
with(plots):
with(plottools):

drawSudokuGrid := proc()
     local squares, blocks;
     squares := seq(seq(rectangle([i,j+1], [i+1,j], style=line, thickness=0), j=1..9), i=1..9);
     blocks := seq(seq(rectangle([3*(i-1)+1,3*j+1], [3*i+1,3*(j-1)+1], thickness=2, color=`if`(type(i
+j, even), "White", "LightGray")), j=1..3), i=1..3);
     return squares, blocks;
end proc:

drawDigit := proc(i, j, k)
     textplot([i+0.5, j+0.5, k], font=[Arial, roman, 18]);

end proc:

```

- The following code draws the Sudoku grid with the digits from the solution found by the SAT solver.

```
digits := Array(1..9*9):
i := 1:
for eq in satisfyingAssignment do
     if rhs(eq) then
          digits[i] := drawDigit(op(lhs(eq)));
          i := i + 1;
     end if;
end do:

display(drawSudokuGrid(), entries(digits, nolist), scaling=constrained, axes=none);
```