Cracking the Sudoku: A Deterministic Approach

David Martin
Erica Cross
Matt Alexander

Youngstown State University
Youngstown, OH

Advisor: George T. Yates

Summary

We formulate a Sudoku-puzzle-solving algorithm that implements a hierarchy of four simple logical rules commonly used by humans. The difficulty of a puzzle is determined by recording the sophistication and relative frequency of the methods required to solve it. Four difficulty levels are established for a puzzle, each pertaining to a range of numerical values returned by the solving function.

Like humans, the program begins solving each puzzle with the lowest level of logic necessary. When all lower methods have been exhausted, the next echelon of logic is implemented. After each step, the program returns to the lowest level of logic. The procedure loops until either the puzzle is completely solved or the techniques of the program are insufficient to make further progress.

The construction of a Sudoku puzzle begins with the generation of a solution by means of a random-number-based function. Working backwards from the solution, numbers are removed one by one, at random, until one of several conditions, such as a minimum difficulty rating and a minimum number of empty squares, has been met. Following each change in the grid, the difficulty is evaluated. If the program cannot solve the current puzzle, then either there is not a unique solution, or the solution is beyond the grasp of the methods of the solver. In either case, the last solvable puzzle is restored and the process continues.

Uniqueness is guaranteed because the algorithm never guesses. If there

The UMAP Journal 29 (3) (2008) 381–394. ©Copyright 2008 by COMAP, Inc. All rights reserved. Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice. Abstracting with credit is permitted, but copyrights for components of this work owned by others than COMAP must be honored. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior permission from COMAP.

is not sufficient information to draw further conclusions—for example, an arbitrary choice must be made (which must invariably occur for a puzzle with multiple solutions)—the solver simply stops. For obvious reasons, puzzles lacking a unique solution are undesirable. Since the logical techniques of the program enable it to solve most commercial puzzles (for example, most "evil" puzzles from Greenspan and Lee [2008]), we assume that demand for puzzles requiring logic beyond the current grasp of the solver is low. Therefore, there is no need to distinguish between puzzles requiring very advanced logic and those lacking unique solutions.

Introduction

The development of an algorithm to construct Sudoku puzzles of varying difficulty entails the preceding formulation of a puzzle-solving algorithm. Our program (written in C++) contains a function that attempts to generate the solution to a given puzzle. Four simple logical rules encompass the reasoning necessary to solve most commercially available Sudoku puzzles, each more logically complex than the previous. The varying complexity establishes a somewhat natural system by which to rate the difficulty of a puzzle. Each technique is given a weight proportional to its complexity; then, difficulty is determined by a weighted average of the methods used. Our algorithm places each puzzle in one of four categories that we identify as Easy, Medium, Hard, and Very Hard.

The lowest level of logic is the most fundamental method used by our program (and humans!) in an attempt to solve a Sudoku puzzle. When a level of reasoning can no longer be used, the next level of logic is prompted. A successful attempt at this new level is followed by a regression back to the lowest level of logic employed. A failed attempt at the new stage initiates a further advance in logic. The procedure loops until the problem is completely solved or no more progress can be made. Consistency is guaranteed by the use of a check function, which verifies that each row, column, and box contains each of the digits 1 to 9 without duplication. If the techniques are inadequate to solve a puzzle, the loop terminates.

Our algorithm constructs Sudoku puzzles in a somewhat "backward" manner. First, a completed Sudoku is formulated using a simple random-number-based function, similar to many brute-force methods of solving the puzzles. Before puzzle generation begins, the user enters restrictions such as desired difficulty level and the maximum number of cells that are initially given. Creating a puzzle begins by randomly eliminating digits from one cell at a time. The elimination process continues until the conditions specified are met. After each removal, the program attempts to solve the existing puzzle.

A Sudoku puzzle cannot be solved in one of two scenarios:

- The puzzle no longer has a unique solution. The algorithm is deterministic and only draws conclusions that follow directly from the current state of the puzzle. In such a case, because an arbitrary decision must be made, the algorithm simply terminates.

- The logical methods available to the solver are not sufficient to solve the puzzle.

In either circumstance the program restores the last solvable puzzle and resumes the process.

Due to the undesirable nature of both ambiguous puzzles and puzzles that require guessing, the algorithm never guesses. If there exists a unique solution for a given puzzle, then failure to solve implies that the puzzle requires logical methods higher than those written into the program. This conclusion is appropriate, since demand is low for Sudoku puzzles requiring extremely sophisticated logical methods. Thus, our algorithm does not distinguish between puzzles with no solution and those requiring more-advanced logic.

Definitions

Cell: A location on a Sudoku grid identified by the intersection of a row and a column, which must contain a single digit.

Row: A horizontal alignment of 9 cells in the Sudoku grid.

Column: A vertical alignment of 9 cells in the Sudoku grid.

Box: One of the nine 3$\times$3 square groups of cells that together comprise the Sudoku grid.

Group: A row, column, or box on the Sudoku grid that must contain each digit from 1-9 exactly once.

Given: A cell whose answer is provided at the beginning of the puzzle.

Candidate: A possible solution for a cell that was not given.

Method: The technique used to eliminate candidates as possibilities and solve cells.

Unique: The puzzle is considered unique when it has a unique solution.

Difficulty: The level of skill needed to solve the puzzle, based on the complexity and frequency of the methods required to solve it.

Assumptions

- We work only with the classic Sudoku grid consisting of a 9$\times$9 square matrix.

- Guessing, while a form of logic, is not a deterministic method. Demand is low for Sudoku puzzles that require nondeterministic logic. All puzzles at Greenspan and Lee [2008] can be solved without guessing (or so the site claims).
- Demand is low for puzzles requiring extremely complicated logical methods. Our algorithm solves all Easy, Medium, Hard, and some Very Hard puzzles.
- The difficulty of a puzzle can be calculated as a function of the sophistication and frequency of the logical methods demanded.
- The ordering of a given set of puzzles by difficulty will be the same for the program as for humans, because the solver uses the same techniques employed by humans.

Model Design

The Solver

The program is based on simple logical rules, utilizing many of the same methods employed by humans. Like humans, the program begins solving each puzzle with the lowest level of logic necessary. When all lower methods have been exhausted, the next echelon of logic is implemented. After each step, the program returns to the lowest level of logic, so always to use the lowest possible level of logic. The procedure loops until either the problem is completely solved or the logical techniques of the program are insufficient to make further progress. The following techniques are included in the algorithm.

1. **Naked Single:** a cell for which there exists a unique candidate based on the circumstance that its groups contain all the other digits [Davis 2007]. In **1**, the number 1 is clearly the only candidate for the shaded cell.

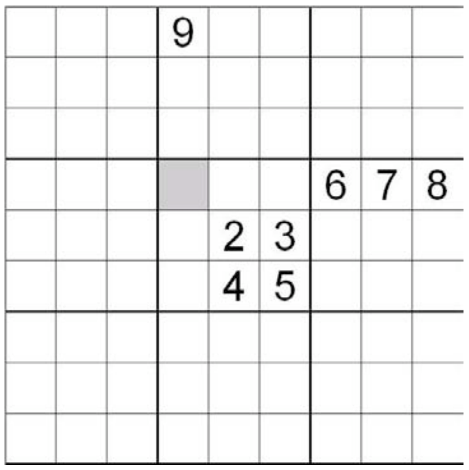

Figure 1. Naked Single.

2. **Hidden Single:** a cell for which there exists a unique candidate based on the constraint that no other cell in one of its groups can be that number [Davis 2007]. In **Figure 2**, the shaded cell must be a 1.

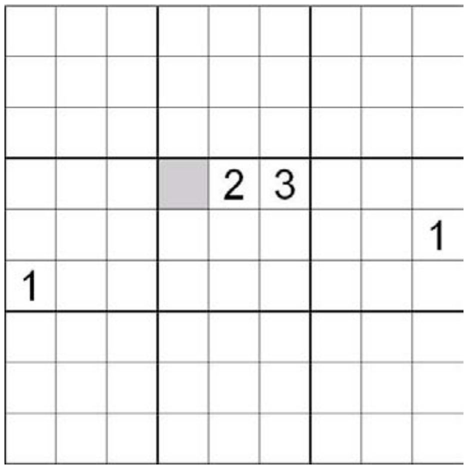

Figure 2. Hidden Single.

3. **Locked Candidate:**
   A. A method of elimination for which a number within a box is restricted to a specific row or column and therefore can be excluded from the remaining cells in the corresponding row or column outside of the selected box [Davis 2007]. In **Figure 3**, none of the shaded cells can be a 1.

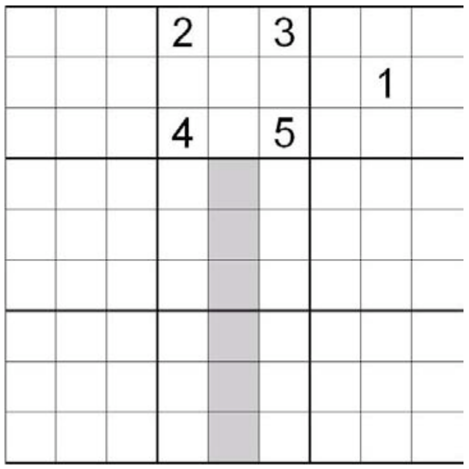

Figure 3. Locked Candidate (box).

   B. A method of elimination for which a number within a row or column is restricted to a specific box and therefore can be excluded from the remaining cells within the box. In **Figure 4**, again, none of the shaded cells can be a 1.

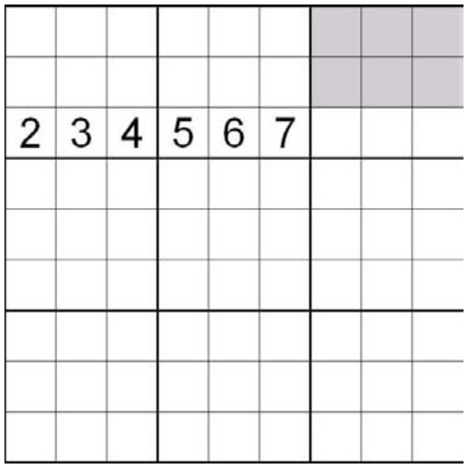

Figure 4. Locked Candidate (rows and columns).

4. **Naked Pairs:** This method of elimination pertains to the situation in which two numbers are candidates in exactly two cells of a given group. Consequently, those two numbers are eliminated as candidates in all other cells within the group [Davis 2007]. In **Figure 5**, none of the shaded cells can contain either a 1 or 2.

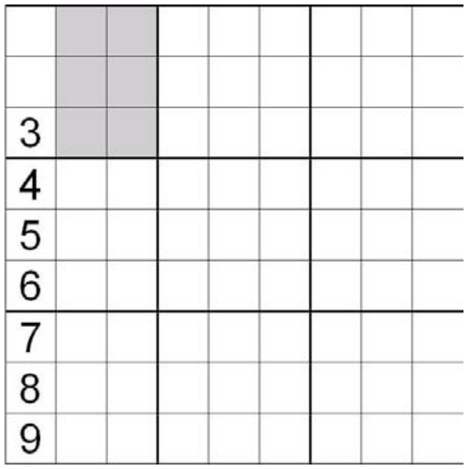

Figure 5. Naked Pairs.

The overall algorithm is represented by the diagram in **Figure 6**.

Difficulty

The algorithm is based on techniques commonly employed by humans; so, for a given set of puzzles, the ordering by difficulty will be about the same for the program as for humans, making the difficulty rating produced by the program of practical value.

As the solver works on a puzzle, it keeps track of the number of times that it uses each level of logic. Let $i \in \{1, 2, 3, 4\}$ correspond to a logic

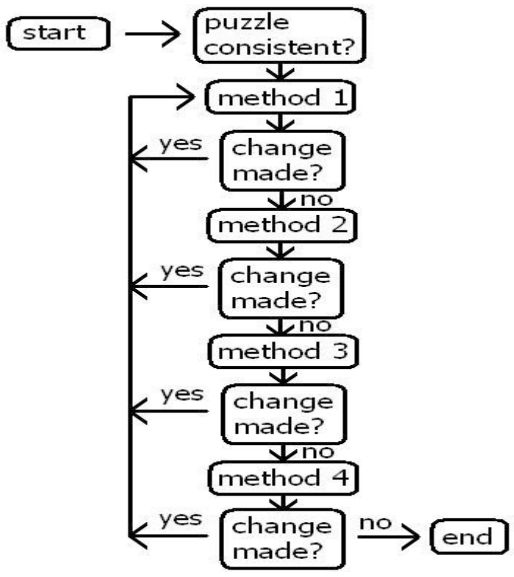

Figure 6. Puzzle Solver.

level discussed above (naked single, hidden single, locked candidate, naked pairs). Let n_i be the number of times that technique i is used. The difficulty rating can then be calculated by means of a simple formula:

$$D(n_1, n_2, n_3, n_4) = \frac{\sum w_i n_i}{\sum n_i},$$

where w_i is a difficulty weight assigned to each method. Naturally, the weight should increase with the complexity of the logic used in a technique.

As the proportion of changes $n_i / \sum n_i$ made by a method increases, the difficulty value approaches the weight assigned to that technique. In practical application, higher methods are used extremely rarely. Therefore, seemingly disproportionately high weights should be assigned to the higher methods for them to have an appreciable effect on difficulty. The choice of these values is somewhat arbitrary, and small changes are not likely to have an appreciable effect on the ordering by difficulty of a set of puzzles.

For our purposes, we used $w_1 = 1$, $w_2 = 3$, $w_3 = 9$, and $w_4 = 27$. In application, these values provide a nice spectrum ranging from 1 (only the first level of logic is used) to about 4 (the higher levels are used frequently). One of the hardest puzzles generated by the program required the use of techniques one, two, three, and four 42, 11, 10, and 2 times, respectively with corresponding difficulty rating

$$D = \frac{42 \cdot 1 + 11 \cdot 3 + 10 \cdot 9 + 2 \cdot 27}{42 + 11 + 10 + 2} \approx 3.37.$$

Difficulty categories can be determined by partitioning the interval [1,4] into any four subintervals. We determined the reasonable subintervals:

Easy , $D \in [1, 1.5)$. A typical Easy puzzle with a rating of 1.25 requires use of the second level of logic 7 times.

Medium , $D \in [1.5, 2)$. A typical Medium puzzle with a rating of 1.7 requires the use of the second level of logic 17 times and the third level once.

Hard , $D \in [2, 3)$. A typical Hard puzzle with a rating of 2.5 requires the use of the second level of logic 17 times, of the third level 4 times, and of the fourth level once.

Very Hard , $D \in [3, 4]$. The aforementioned puzzle, with 3.37, required the use of the second method 11 times, of the third method 10 times, and of the fourth method twice.

The Puzzle Creator

Rather than starting with an empty grid and adding numbers, the program begins with a completed Sudoku, produced by a random-number-based function within the program. The advantage is that rather than hoping to stumble upon a puzzle with a unique solution, the program begins with a puzzle with a unique solution and maintains the uniqueness.

Once a completed Sudoku grid has been created, the puzzle is developed by working backwards from the solution, removing numbers one by one (at random) until one of several conditions has been met. These conditions include a minimum difficulty rating (to ensure that the puzzle is hard enough) and a minimum number of empty squares (to ensure that the puzzle is far from complete). Following each change in the grid, the difficulty is evaluated as the program solves the current puzzle. If the program cannot solve the current puzzle, then either the puzzle does not have a unique solution or the solution is beyond the grasp of the logical methods of the algorithm. In either case, the last solvable puzzle is restored and the process continues (see **Figure 7**).

In theory, a situation may occur in which removing any number will yield a puzzle that is not solvable (by the algorithm) but has a unique solution. In such a case, the puzzle creator has reached a "dead end" and cannot make further progress toward a higher difficulty rating. To overcome this obstacle, the program, rather than generating a single puzzle as close as possible to a given difficulty rating, generates 1,000 puzzles and sorts them by difficulty. In this manner, the program produces a virtual continuum of difficulties ranging from 1.0 to whatever difficulty was requested (within the limits of the program, which cannot produce puzzles that are harder than about 4).

Uniqueness

Uniqueness is guaranteed because the algorithm never guesses.

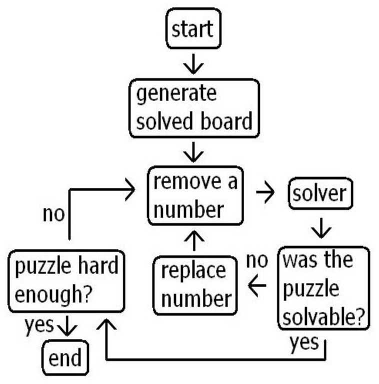

Figure 7. Puzzle Creator.

Difficulty

Since the logical techniques possessed by the program enable it to solve most commercial puzzles, we assume that demand for puzzles requiring logic beyond the current grasp of the solver is low. Therefore, there is no need to distinguish between puzzles requiring very advanced logic and those lacking unique solutions.

Model Testing (Relevance of the Difficulty Rating)

To determine the relevance of the algorithm to real-world Sudoku puzzles, we set our program loose on 48 randomly selected puzzles from three popular Websites [Greenspan and Lee 2008; ThinkFun Inc. 2007; and www.LearnToUseComputers.com]. Four puzzles were selected from each of four difficulty categories for each source. The difficulty levels assigned by our program and a summary of our results are in **Tables 1** and **2**.

All puzzles labeled by the source as Easy, Medium, and Hard (or an equivalent word) were solved successfully and rated. Some—but not all—of the Very Hard puzzles were solved successfully and rated; those beyond the grasp of our program were simply given a rating of 4.0, the maximum rating in the scale. Although the algorithm was able to crack exactly one half of the Very Hard puzzles attempted from both Greenspan and Lee [2008] and www.LearnToUseComputers.com [2008], it solved none of the Very Hard puzzles from ThinkFun Inc. [2007].

Under the suggested partition ($[1, 1.5)$, $[1.5, 2)$, $[2, 3)$, and $[3, 4]$), all of the puzzles labeled by the source as Easy (or equivalent) were awarded the same rating by our program. Agreement was excellent with ThinkFun Inc. [2007], with which our program agreed on 13 of the 16 puzzles tested (the four puzzles from that source with a rating of Very Hard were not solved

Table 1. Performance summary.

Label Awarded by Source	Websudoku	Puzzles	Sudoku Puzzles Online
Easy	Easy	Easy	Easy
	Easy	Easy	Easy
	Easy	Easy	Easy
	Easy	Easy	Easy
Medium	Easy	Easy	Easy
	Easy	Medium	Easy
	Easy	Medium	Easy
	Easy	Medium	Medium
Hard	Easy	Easy	Medium
	Easy	Medium	Medium
	Medium	Hard	Easy
	Medium	Hard	Medium
Very Hard	Hard	Not Solved	Hard
	Very Hard	Not Solved	Very Hard
	Not Solved	Not Solved	Not Solved
	Not Solved	Not Solved	Not Solved

Table 2. Difficulty rating.

		Websudoku	Puzzles	Sudoku Puzzles Online
Easy	Average Difficulty	1.0	1.1	1.0
	Proportion Solved	4 / 4	4 / 4	4 / 4
Medium	Average Difficulty	1.2	1.6	1.3
	Proportion Solved	4 / 4	4 / 4	4 / 4
Hard	Average Difficulty	1.6	1.9	1.7
	Proportion Solved	4 / 4	4 / 4	4 / 4
Very Hard	Average Difficulty	3.5	4.0	3.5
	Proportion Solved	2 / 4	0 / 4	2 / 4

by the algorithm and received a difficulty rating of 4.0 by default). The three puzzles for which the algorithm and ThinkFun Inc. [2007] disagreed were given a lower difficulty rating by the program. The program successfully solved four Very Hard puzzles from the other two sources but was apparently not too impressed, awarding half of those solved a mere Hard rating. In the Medium and Hard categories, puzzles from Greenspan and Lee [2008] and www.LearnToUseComputers.com [2008] were consistently awarded lower difficulty ratings than those suggested by the source.

Model Analysis

Complexity Analysis

The Solver

The puzzle-solving algorithm is surprisingly short, relying on a simple topology connecting a hierarchy of just four logical methods. At first glance, one might suspect that most puzzles would be beyond the scope of the four logical operations available to the solver. However, as seen above, the algorithm does not meet its match until it is presented with puzzles labeled as Very Hard.

The Puzzle Creator

At the start of the process, a mysterious procedure randomly creates a solved puzzle. This procedure is not complicated, and can be summarized as follows.

Going from the top of the puzzle to the bottom, and from the left to the right, a random digit is inserted into each empty box. The program checks for consistency. If insertion of the digit violates a constraint of the Sudoku puzzle, then another digit is attempted. After a fixed number of attempts have failed at a given cell (in fact, no digit may be possible if there remain no candidates for a given cell), the program removes both of the digits involved in the last contradiction. This allows the program to dismantle relationships that make a puzzle unsolvable. The process loops until the puzzle is both consistent and complete (no empty spaces).

The rest of the puzzle creation process is largely the inverse of the above procedure, except that rather than inserting numbers and checking for consistency and completeness, the program removes numbers and checks for solvability and uniqueness (which are equivalent for reasons discussed above) as well as constraints pertaining to difficulty rating and number of givens.

Sensitivity Analysis

The primary source of arbitrariness in the model is the method by which difficulty ratings are established. It requires the user to assign to each logical technique a weight proportional to its complexity and its contribution to the overall difficulty of the puzzle. We assigned weights of 1, 3, 9, and 27 to the levels of logic. The exact values are relatively unimportant, evidenced by the fact that two additional sets of weights produced exactly the same ordering by difficulty of a set of eight typical puzzles created by the program. **Table 3** summarizes the relative independence of the ordering on weight values.

Table 3. Sensitivity analysis.

LOGIC LEVEL	WEIGHTING SYSTEM			
	4^N	3^N	2^N	2N+1
1	1	1	1	1
2	4	3	2	3
3	16	9	4	5
4	64	27	8	7

PUZZLE	4^N	3^N	2^N	2N+1
A	1.00	1.00	1.00	1.00
B	1.47	1.32	1.16	1.16
C	1.96	1.61	1.29	1.27
D	2.66	2.00	1.45	**1.40**
E	3.52	2.29	1.48	**1.35**
F	3.87	2.61	1.66	**1.51**
G	5.38	3.06	1.70	**1.46**
H	5.78	3.28	1.78	1.52

Although all three of the exponential weight systems produce the same difficulty rating, the linear system does not. Because the featured system, which uses the weights of 1, 3, 9, and 27, agrees so well with ThinkFun Inc. [2007], it seems safe to say that the exponential weighting system makes much more sense (at least with the current hierarchy of logical techniques) than the linear.

Shortcomings of the Model

- We assume that four levels of logic are sufficient to solve any Sudoku puzzle, though other techniques exist.
- The model lacks the capacity to solve some "evil" puzzles featured on Greenspan and Lee [2008], due to the absence of more-complex methods within the program.
- Our model reports an error for Sudoku puzzles that either have no solution or multiple solutions but does not differentiate between the two.

Strengths of the Model

- Our model considers the fact that once a higher level of logic is used and a cell is filled, a human will return to attempting a solution with the simplest method of logic, and therefore so does our program.
- Utilizing a functional program, we were able to construct and evaluate the difficulty of a thousand Sudoku puzzles in a matter of minutes.
- The program uses deterministic logic in each method featured in the program and does not resort to guessing.
- The code can be easily expanded to include more advanced levels of logic such as naked triplets and quads, x-wings and swordfish, or coloring.
- The code could also be easily modified to do other types of Sudoku puzzles such as a 16×16 grid and other rules for the puzzle.

Conclusion

In spite of the seemingly small scope of the four logical operations available to the solver, the algorithm solved all Easy, Medium, and Hard puzzles from three popular Internet sources and one-third of their Very Hard puzzles. Therefore, a small set of logical rules is sufficient to solve nearly all commercially available Sudoku puzzles.

To expand the scope of the solver, the overall complexity of the algorithm need not be increased. Simply adding another logical technique to the loop can increase the solving power. A mere two or three additional methods would probably suffice to enable the program to solve all commercially available puzzles that do not require guessing.

Just For Fun

The puzzle in **Figure 8**, created by the program and given a difficulty rating of 3.52 (Very Hard), requires the use of methods one, two, three, and four 45, 12, 9, and 3 times, respectively. Have fun!

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

Figure 8. Difficulty 3.52.

References

Davis, Tom. 2007. The mathematics of Sudoku. `http://www.geometer.org/mathcircles/sudoku.pdf` . Accessed 15 February 2008.

Greenspan, Gideon, and Rachel Lee. 2008. Web Sudoku. `http://www.websudoku.com/` .

ThinkFun Inc. 2007. Sudoku 12. `http://www.puzzles.com/PuzzleLinks/Sudoku12.htm` .

www.LearnToUseComputers.com. 2008. MPS-Sudoku 2008. `http://www.sudokupuzzlesonline.com`, which defaults to `http://www.ltucshop.com/prodtype.asp?strParents=70&CAT_ID=71&numRecordPosition=1` .

Matt Alexander, George Yates (advisor), Erica Cross, and David Martin.