



A Perl-based Sudoku Puzzle Solver

Mark Haselup
 mhaselup@insilicodiscovery.com

Sudoku, a numerical form of the crossword puzzle, has taken the world by storm and appears in newspapers, books, online and even on parts of the British countryside*. The name is Japanese and is pronounced sue-do-koo (derived from su = number, doku = single) and refers to the fact that each number in the puzzle is constrained to a single use. Although the name comes from Japanese, the original creation of the puzzle is attributed to an American architect named Howard Garns (see the sidebar on page 23).

In theory, Sudoku puzzles can take many forms, but the most commonly published take the form of a 9 by 9 grid containing 9 subgrids of 3 by 3 cells, which I'll call a block. The grid is partially filled in with "clues" that are typically arranged in some kind of symmetrical fashion, as shown to the right.

By working within a simple set of rules, the reader has to establish the numbers in the vacant cells. The rules are:

- Each number (1 through 9, for a 9x9 grid) can only appear once in a given row.
- Each number can only appear once in a given column.
- Each number can only appear once in a given subgrid (one of the 9 blocks of 3 by 3 cells).

I implemented a Sudoku puzzle solver in Perl and gave it a web interface[†]. My program solves the puzzles iteratively without the use of logic to

* On June 29, 2005, a 275 foot Sudoku square appeared on a field at Hinton Farm, near Bristol, creating a world record. See <http://tinyurl.com/mu95u>

† <http://www.foureighty.net/sudoku/grid.htm>

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

This easy puzzle has many clues, and shows the symmetry typical of well-formed Sudoku puzzles. It was created by Mark Haselup.

determine the correct cell contents. There are programs that use logic to solve puzzles, however even those may resort to the kind of approach I show here if there is insufficient information available in the grid to constrain the puzzle to a unique solution (although a good puzzle should only have one solution).

I wrote this script to test out an idea I had about the similarity of this problem to solving mazes, or so called back-tracking algorithms. The principle of solving a maze is quite simple assuming that each branch in the maze offers only a left or right forks or is a dead end. From the starting point, I go as far as I can by taking the left most fork. If I encounter a dead end, I then re-track until I am at a branch where I can take the next left most fork, and proceed. Eventually I will traverse the maze if there is a solution.