

Sudoku Solver and Generator Report

CSE1370 Advanced First-year Project

Semester 2, 2007

David Warner 20727356

24/10/2007

Supervisor: Reza Rafah

Table of Contents

| | |
|--|-----------|
| INTRODUCTION..... | 3 |
| WHAT IS SUDOKU?..... | 3 |
| EXAMPLE PUZZLE | 3 |
| VARIATIONS | 3 |
| MOTIVATION AND GOALS | 3 |
| SOLVING METHODS | 4 |
| BRUTE FORCE SOLVING METHOD | 4 |
| <i>Method 1</i> | 4 |
| <i>Method 2</i> | 6 |
| <i>Method 3</i> | 8 |
| LOGIC SOLVING METHODS | 9 |
| <i>Singleton Domains</i> | 10 |
| <i>Unique Numbers in Domains of a Row, Column or Box</i> | 11 |
| <i>'Pointing Pairs'</i> | 12 |
| <i>'Naked Pairs/Triples/Quads'</i> | 13 |
| <i>'Box/Line Reduction'</i> | 14 |
| <i>Hidden Pairs</i> | 15 |
| GENERATION | 16 |
| IMPLEMENTATION | 17 |
| LANGUAGE OF IMPLEMENTATION | 17 |
| SOLVER IMPLEMENTATION | 18 |
| <i>Measurements</i> | 18 |
| <i>Measurement Limitations</i> | 22 |
| GENERATOR IMPLEMENTATION..... | 24 |
| <i>Remarks</i> | 24 |
| <i>Generating a Grid</i> | 24 |
| <i>Different Grid Varieties</i> | 25 |
| <i>Measurements</i> | 27 |
| CONCLUSIONS..... | 29 |
| ACHIEVEMENTS..... | 29 |
| PROJECT STATUS | 29 |
| REFERENCES..... | 29 |

Introduction

What is Sudoku?

Sudoku is a logic puzzle that requires the player to fill a 9×9 grid with the digits 1 to 9 so that each row, column and 3×3 region contains each digit exactly once. Initially the grid is partially filled – the puzzles in the local papers *The Age* and *The Australian* tend to have 26 or 28 ‘hints’ (squares initially filled), but this number can be as low as 17. From this initial state, the player should be able to progressively deduce the values of the blank squares.

Example Puzzle

An example grid (found in *The Age*) and its solution:

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

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

Each row, column and 3×3 region contains the numbers 1 to 9 exactly once.

Variations

There are many variations of the traditional 9×9 grid, such as 4×4 grids (with 2×2 regions) and 16×16 grids (with 4×4 regions). These can be solved in exactly the same way as normal grids, just using smaller (or larger) domains (e.g. 1 to 16 instead of 1 to 9). Furthermore, there are variants that introduce extra constraints or have non-square regions. This report focuses on 9×9 grids, however.

Motivation and Goals

The aim of this CSE1370 project was to implement a solver (specifically, a step by step logic solver) and a generator. Although there are many such programs in existence, by creating one from scratch I sought to test and expand my programming knowledge.

Solving Methods

Brute Force Solving Method

A simple way to solve a Sudoku puzzle is to simply try filling each blank square with the numbers 1 to 9 until a valid solution is found. I ended up devising three different implementations of this method, as shown below.

Method 1

The first and most naïve solver starts at the top-left square and moves from left-to-right, top-to-bottom, filling each blank square with a number 1 to 9 until the grid is invalid (that is, a number is duplicated in a row, column or 3×3 region) or until the grid is filled and valid (that is, solved). If the grid is invalid, the solver will backtrack until it is valid and continue forward again.

To illustrate, this is the process on a 4×4 grid:

| | | | |
|---|---|---|---|
| | | | |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 1 | | | |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 1 | 1 | | |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 1 | 2 | | |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 1 | 2 | 1 | |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 1 | 2 | 2 | |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 1 | 2 | 4 | |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 1 | 3 | | |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 1 | 3 | 1 | |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 1 | 3 | 2 | |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 1 | 3 | 3 | |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 1 | 3 | 4 | |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 1 | 4 | | |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 2 | | | |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 3 | | | |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 4 | | | |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

Sudoku Solver and Generator Report

| | | | |
|---|---|---|---|
| 4 | 1 | | |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 4 | 2 | | |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 4 | 2 | 1 | |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 4 | 2 | 1 | 1 |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 4 | 2 | 1 | 2 |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 4 | 2 | 1 | 3 |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 4 | 2 | 1 | 4 |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 4 | 2 | 2 | |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 4 | 2 | 3 | |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 4 | 2 | 4 | |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 4 | 3 | | |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 4 | 3 | 1 | |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 4 | 3 | 1 | 1 |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 4 | 3 | 1 | 2 |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 4 | 3 | 1 | 2 |
| 1 | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 4 | 3 | 1 | 2 |
| 1 | 1 | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 4 | 3 | 1 | 2 |
| 1 | 2 | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 4 | 3 | 1 | 2 |
| 1 | 2 | 4 | 3 |
| 2 | 1 | 3 | 1 |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 4 | 3 | 1 | 2 |
| 1 | 2 | 4 | 3 |
| 2 | 1 | 3 | 2 |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 4 | 3 | 1 | 2 |
| 1 | 2 | 4 | 3 |
| 2 | 1 | 3 | 3 |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 4 | 3 | 1 | 2 |
| 1 | 2 | 4 | 3 |
| 2 | 1 | 3 | 4 |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 4 | 3 | 1 | 2 |
| 1 | 2 | 4 | 3 |
| 2 | 1 | 3 | 4 |
| 3 | 4 | 2 | 1 |

Performance Discussion

Even without other methods for comparison, this first implementation appears to be fairly inefficient. To find the solution of this simple 4x4 grid took 37 number placements. When solving a 9x9 grid, the number of number placements can reach the hundreds of thousands. While even this high number of 'guesses' is fast to execute on a modern computer, there are some obvious improvements that can be made to the algorithm. For instance, in the illustration's first two grids, the number 1 is placed immediately beside another number 1, clearly breaking the rule of Sudoku. Such improvements are covered in the implementations displayed below.

Method 2

The second implementation introduces the concept of 'domains'. Each square in a grid has a domain of up to 9 values (1 to 9) that is reduced according to numbers already present in the intersecting row, column and region.

| | | | |
|---|---|---|---|
| | | | |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

In this grid, we can reduce the domain of the top-left square to {1,4} since 2 and 3 already appear in the first column. Likewise, we can restrict the domain of the top-right square to {2}, since 3 and 1 occur in the rightmost column and 4 appears in the top-right region.

So initially, the domains of the grid are:

| | | | |
|-------|-------|-----|-----|
| {1,4} | {2,3} | {1} | {2} |
| {1} | {2} | 4 | 3 |
| 2 | 1 | 3 | {4} |
| 3 | 4 | 2 | 1 |

This solver acts in the same way as the previous implementation except that it restricts the domains of the grid before it starts, only using numbers present in the initial domains. While the domain restriction requires computation, it should result in significantly less backtracking to find the solution of a grid.

Illustrating the process:

| | | | |
|---|---|---|---|
| 1 | | | |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 1 | 2 | | |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 1 | 2 | 1 | |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 1 | 3 | | |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 1 | 3 | 1 | |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 4 | | | |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 4 | 2 | | |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 4 | 2 | 1 | |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 4 | 2 | 1 | 2 |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 4 | 3 | | |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 4 | 3 | 1 | |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 4 | 3 | 1 | 2 |
| | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

Sudoku Solver and Generator Report

| | | | |
|---|---|---|---|
| 4 | 3 | 2 | 1 |
| 1 | | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 4 | 3 | 2 | 1 |
| 1 | 2 | 4 | 3 |
| 2 | 1 | 3 | |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 4 | 3 | 2 | 1 |
| 1 | 2 | 4 | 3 |
| 2 | 1 | 3 | 4 |
| 3 | 4 | 2 | 1 |

| | | | |
|---|---|---|---|
| 4 | 3 | 2 | 1 |
| 1 | 2 | 4 | 3 |
| 2 | 1 | 3 | 4 |
| 3 | 4 | 2 | 1 |

Performance Discussion

In comparison to the first implementation, this version appears to be much better: instead of 37 number placements to find the solution, only 11 are required. There is some tradeoff in that the domains must be calculated once, initially, requiring some computation. However, the idea behind this implementation is that the reduced number of guesses will more than make up for that.

Method 3

My final brute force solver builds on the second implementation by restricting the domains of the grid every time a number is entered. This again should reduce backtracking considerably, but there will be some trade off for computing the domains.

To illustrate using a 9x9 grid:

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

The initial domains of the top 3 regions are:

| | | | | | | | | |
|-------------|-----------|-------------|-----------|-------------|-----------|---------|-----------|---------|
| {1,2,8,9} | {1,2,3,9} | {1,2,3,8} | {2,6,7,9} | {1,2,7,8,9} | 5 | {1,2,3} | 4 | {2,6} |
| 7 | {1,2,3,4} | {1,2,3,5,8} | {2,6} | {1,2,8} | {1,4,6,8} | 9 | {1,2,3,5} | {2,5,6} |
| {1,2,4,5,9} | {1,2,4,9} | 6 | {2,9} | {1,2,9} | 3 | {1,2,5} | 8 | 7 |

After the first box is set, the domains of the row, column and region are reduced accordingly:

| | | | | | | | | |
|-----------|---------|-----------|-----------|-----------|-----------|---------|-----------|---------|
| 1 | {2,3,9} | {2,3,8} | {2,6,7,9} | {2,7,8,9} | 5 | {2,3} | 4 | {2,6} |
| 7 | {2,3,4} | {2,3,5,8} | {2,6} | {1,2,8} | {1,4,6,8} | 9 | {1,2,3,5} | {2,5,6} |
| {2,4,5,9} | {2,4,9} | 6 | {2,9} | {1,2,9} | 3 | {1,2,5} | 8 | 7 |

This will happen every time a square is filled:

| | | | | | | | | |
|---------|----------|---------|---------|---------|-----------|---------|-----------|---------|
| 1 | 2 | {3,8} | {6,7,9} | {7,8,9} | 5 | {3} | 4 | {6} |
| 7 | {3,4} | {3,5,8} | {2,6} | {1,2,8} | {1,4,6,8} | 9 | {1,2,3,5} | {2,5,6} |
| {4,5,9} | {4,9} | 6 | {2,9} | {1,2,9} | 3 | {1,2,5} | 8 | 7 |

Performance Discussion

As predicted, this final implementation results in a significantly reduced search space.

Logic Solving Methods

The brute force solving methods, while reasonably quick on a modern computer and simple to implement, are not used by human solvers. Instead, human players use a variety of techniques to logically deduce the value of blank squares. These techniques are harder to program than the brute force methods, but since a human solver can use them to solve grids in a fairly short time, it should follow that a computer with its superior calculating power could use them to solve grids faster than with the brute force methods.

Furthermore, by using the same methods as a human solver, the program can provide a step-by-step account of how to solve a grid, which is much more useful to a player than simply being provided with the solved grid.

In this report I detail 6 different techniques for solving puzzles. There are many other, more advanced techniques, but the 6 methods described below are sufficient to solve many puzzles (only very difficult puzzles cannot be solved using them).

Singleton Domains

When a square has a domain with only 1 number in it, we can certainly set the value of that square to the remaining number.

For example, in the grid:

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

The highlighted square has only one number left in its domain (a four), since the numbers 1 and 8 are already present in its column, 5, 2, 6 and 3 are already present in its row, and 9 and 7 are already present in its 3×3 region. So we can be certain that the highlighted square can contain nothing other than a 4.

Unique Numbers in Domains of a Row, Column or Box

When a number appears only once in the domains of an entire row, column or 3×3 region, we can set the value of the square with that number in its domain.

To illustrate, using the grid:

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

The domains of column 8 (the column containing the red highlighted square) are:

| |
|-------|
| {1} |
| {7,9} |
| {7,8} |
| {4} |
| {8,9} |
| {2} |
| {5} |
| {6,9} |
| {3} |

The number 6 only appears once – in the 8th row – so we can set the value of that square (8,8) to 6 and adjust the surrounding domains accordingly.

The blue highlighted square contains a 2 (unique in the column) and the two green highlighted squares contain 1s (unique in their respective rows).

'Pointing Pairs'

If a number appears twice in a row or column, and those occurrences happen to be both in the same 3×3 region and be the only occurrences in that region, we can remove that number from any other domains in that row or column.

Using the same grid as before:

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

The two highlighted squares constitute a 'pointing pair' of 8s. This can be seen by looking at the domains of the top-left region and the third column:

| | | | |
|-----|-------|-----------|-------------|
| {2} | {3} | {4,7,8,9} | {4,7,8,9} |
| {1} | {4,7} | {4,7,9} | {4,7,9} |
| {6} | {5} | {4,7,8} | {4,7,8} |
| | | | {2} |
| | | | {3,5,6,7,8} |
| | | | {1,3,5,6} |
| | | | {3,4,6,7} |
| | | | {1} |
| | | | {4,5,6,8,9} |

Since the number 8 must be in either row 1 or 3 (demonstrated by the domains of the top-left region), we can remove the other 8s in the domains of the column.

The updated domains of column 3 are:

| |
|-----------|
| {4,7,8,9} |
| {4,7,9} |
| {4,7,8} |
| {2} |
| {3,5,6,7} |
| {1,3,5,6} |
| {3,4,6,7} |
| {1} |
| {4,5,6,9} |

'Naked Pairs/Triples/Quads'

When a pair of numbers are the only members of two domains in a row, column or 3×3 region, those numbers can be removed from the other domains in that row, column or region.

Take the domains of this row:

| | | | | | | | | |
|-----|-----|-------|-----|---------|-----|-----|-------|-------|
| {6} | {5} | {7,8} | {9} | {3,4,7} | {1} | {2} | {7,8} | {3,4} |
|-----|-----|-------|-----|---------|-----|-----|-------|-------|

The two highlighted squares constitute a 'naked pair'. The numbers 7 and 8 must be in either column 3 or column 8 (since setting the value of either square will reduce the other to having a singleton domain), so we can remove any 7s and 8s left in the row. In this case, we can remove the 7 from the domain of column 5:

| | | | | | | | | |
|-----|-----|-------|-----|-------|-----|-----|-------|-------|
| {6} | {5} | {7,8} | {9} | {3,4} | {1} | {2} | {7,8} | {3,4} |
|-----|-----|-------|-----|-------|-----|-----|-------|-------|

This idea can in fact be extended to groups of 3 and 4 numbers.

Take the following domains of a certain row:

| | | | | | | | | |
|-----|-------|---------|-----|-------------|-----------|-----|-------|-----------|
| {1} | {4,7} | {4,7,9} | {8} | {2,3,4,5,7} | {2,3,4,5} | {6} | {7,9} | {3,4,5,9} |
|-----|-------|---------|-----|-------------|-----------|-----|-------|-----------|

The three highlighted squares constitute a 'naked triple'. Importantly, the three squares needn't have all three numbers in their domains – just the union of the three squares must have 3 elements. The numbers 4, 7 and 9 can be removed from the other domains in the row:

| | | | | | | | | |
|-----|-------|---------|-----|---------|---------|-----|-------|-------|
| {1} | {4,7} | {4,7,9} | {8} | {2,3,5} | {2,3,5} | {6} | {7,9} | {3,5} |
|-----|-------|---------|-----|---------|---------|-----|-------|-------|

'Box/Line Reduction'

This tactic is similar to 'pointing pairs'. If a number appears only twice in the domains of a row or column, and those occurrences happen to be in the same 3×3 region, that number can be removed from the domains of that region.

Take the domains of the following region and row (the row corresponds to row 2 of the box):

| | | |
|-----|-------|-----------|
| {2} | {3} | {4,7,8,9} |
| {1} | {4,7} | {4,7,9} |
| {6} | {5} | {4,7,8} |

| | | | | | | | | |
|-----|-------|---------|-----|---------|---------|-----|-------|-------|
| {1} | {4,7} | {4,7,9} | {8} | {2,3,5} | {2,3,5} | {6} | {7,9} | {3,5} |
|-----|-------|---------|-----|---------|---------|-----|-------|-------|

The number 4 appears only twice in the row, so we can remove 4 from the domains of the square:

| | | |
|-----|-------|---------|
| {2} | {3} | {7,8,9} |
| {1} | {4,7} | {4,7,9} |
| {6} | {5} | {7,8} |

Hidden Pairs

When a pair of numbers occur exactly twice in the domains of a 3×3 region, the other numbers in the domains of those two squares may be removed.

Take the domains of this example region:

| | | |
|-----------|-----------|---------|
| {1,5,7,8} | {1,5,8,9} | {1,4} |
| {1,2,7} | {1,2,9} | {6} |
| {3} | {1,2,9} | {1,2,4} |

Notice that the numbers 5 and 8 appear only twice each in the region, in the top left and top middle squares. We can reduce the domains of those two squares:

| | | |
|---------|---------|---------|
| {5,8} | {5,8} | {1,4} |
| {1,2,7} | {1,2,9} | {6} |
| {3} | {1,2,9} | {1,2,4} |

Generation

Generation is the process of creating an empty Sudoku puzzle grid. A grid should have a unique solution if it is to be solvable by logic. The method Reza and I agreed on to create grids was to start with a filled grid and remove numbers until the number of filled squares reaches the desired figure (most puzzles in *The Australian* and *The Age* have 26 or 28 hints, so this seems to be a reasonable benchmark).

During the square-removal process, if the grid at any point could no longer be solved with logic, the generator should backtrack and try removing a different number. Thus the process should eventually terminate either when all the squares have been examined or when the desired number of hints is reached.

Implementation

Language of Implementation

For this project I chose to use Java as the programming language. This was simply due to it being the language I had been using most recently. A Constraint Programming language like Prolog would probably have been better suited to this task since such languages include the ability to do some of the solving work automatically. However, I have no experience in using this programming paradigm.

Additionally, Java has the benefit of being a portable language (provided the host machine has a Java Virtual Machine installed) – although the code was written in Windows, it should run on other platforms with minimal adjustment.

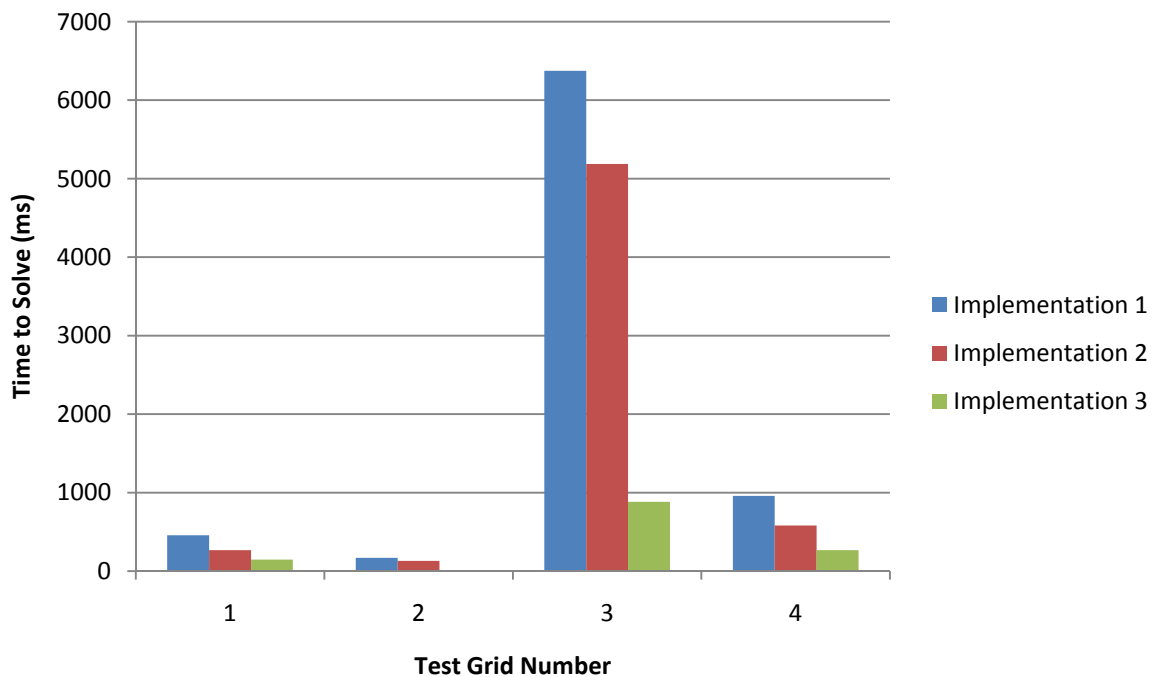
Solver Implementation

Of the 6 techniques shown above (Singleton Domains, Unique Domains, Pointing Pairs, Naked Pairs/Triples/Quads, Box/Line Reduction, Hidden Pairs), I implemented all but Hidden Pairs. In all the grids I tested, it seemed the Hidden Pairs technique was not required to for finding the solution (the other methods were sufficient).

Measurements

Brute Force

This graph gives the average performance of the three implementations on 4 different grids.



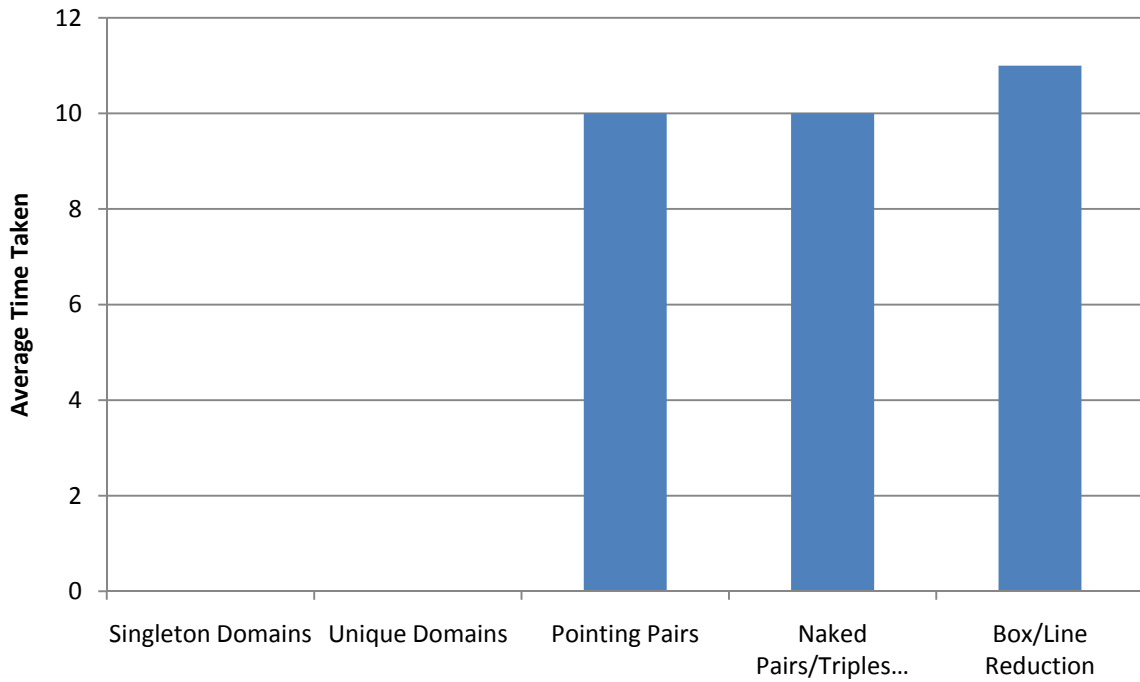
The performance is as expected: implementation 1 is the slowest, and implementation 3 is the fastest.

The exact values are:

| Grid Number | Implementation 1 (ms) | Implementation 2 (ms) | Implementation 3 (ms) |
|-------------|-----------------------|-----------------------|-----------------------|
| 1 | 457 | 268 | 147 |
| 2 | 170 | 131 | 12 |
| 3 | 6375 | 5187 | 882 |
| 4 | 959 | 582 | 267 |

Logic Solver

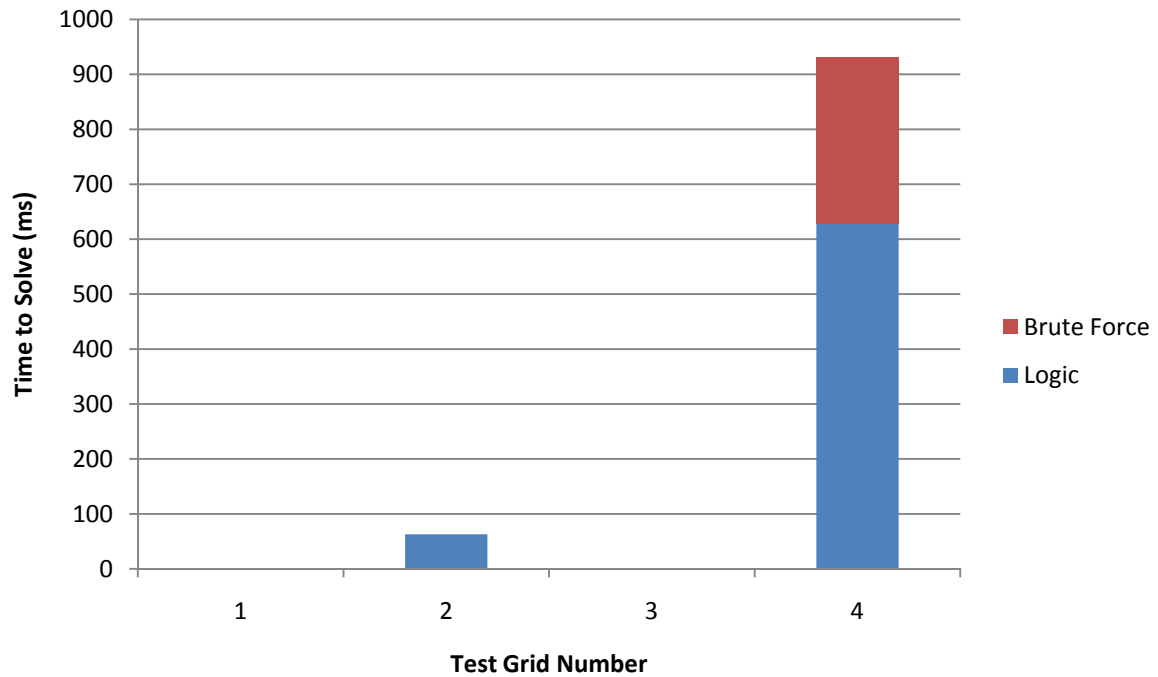
The first two most basic techniques, Singleton Domains and Unique Domains, are very fast to run in my implementation: both take 0ms according to my measurements (unfortunately I could not measure in units smaller than milliseconds). However, the three other techniques take significantly longer, averaging about 10ms on each execution.



Hybrid Solver

This solver implementation resorts to brute force if the logic techniques are not sufficient to solve the grid.

The following graph shows the performance of the hybrid solver over 4 different test grids.



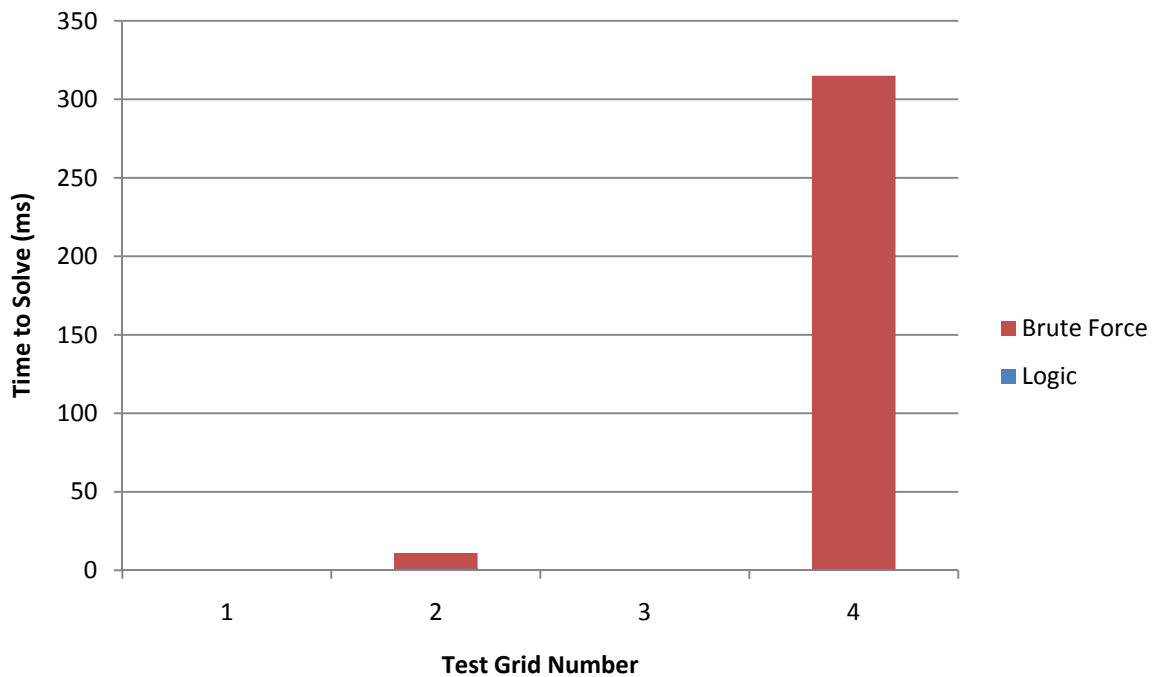
This table lists the number of 'number placements' (times a number is placed in a square by the solver) for each part of the solver across the 4 grids:

| Grid Number | Logic | Brute Force |
|-------------|-------|-------------|
| 1 | 53 | 0 |
| 2 | 53 | 146 |
| 3 | 55 | 0 |
| 4 | 10 | 5279 |

Hybrid Solver (Basic Logic Techniques Only)

This solver is the same as the version described above except that only the basic (Singleton Domains/Unique Domains) logic techniques are used before falling back to brute force. The rationale behind this is that those two techniques take very little time (0ms on average) compared to the other three techniques.

The following graph shows the performance of the solver over the same 4 test grids.



This table lists the number of 'number placements' (times a number is placed in a square by the solver) for each part of the solver across the 4 grids:

| Grid Number | Logic | Brute Force |
|-------------|-------|-------------|
| 1 | 53 | 0 |
| 2 | 4 | 146 |
| 3 | 55 | 0 |
| 4 | 10 | 5279 |

From the graph and the table, we can see that the trade-off seemed to result in performance gains for test grids 2 and 4: although the brute force time taken for each is longer, the overall time is shorter (since the computationally expensive logic techniques are not used).

Measurement Limitations

For the measurements I used a set of 4 test grids selected from various sources. These are shown below.

Test Grid 1 (28 hints, from The Age)

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

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

This grid can be solved using only the Singleton Domains/Unique Domains logic techniques.

Test Grid 2 (28 hints, from The Age)

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

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

This grid needs all 5 implemented logic techniques to be solved.

Test Grid 3 (26 hints, from The Australian)

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

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

This grid has a relatively low number of hints.

Test Grid 4 (29 hints)

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

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

This grid cannot be solved using the logic methods used by my solver.

I sought that the test grids might vary in nature, but 4 is still a very small sample of the over 5 billion possible grids. With more time, I would test many more grids.

Generator Implementation

Remarks

The generator demonstrates that the number of hints given in a puzzle is not necessarily a good measure of how difficult a puzzle is for a human player – the example Easy grid below has only 24 hints and yet requires no techniques other than the basic unique/singleton domain methods, while the example Hard grid has 26 hints and requires the other techniques to be used several times.

Generating a Grid

Generating the Filled Grid

Creating a filled grid turned out to be an easy task – simply running the brute force solver on an empty (0 squares filled) grid will produce a valid solved grid. However, in the original implementation the solver would produce the same grid every time (since it runs from left-to-right, top-to-bottom always). By instead taking a random path through the grid a random solved Sudoku grid may be produced.

First Square Removal Attempt

My first attempt to progressively remove squares from the filled grid did not fully succeed. My method was to remove a square, reset the domains of the row, column and 3×3 region intersecting that square and then check if the value of that square was still deducible by logic. This worked for hint-counts of about 35 and upwards, but never produced any grids with fewer hints. Furthermore, the grids produced were always ‘easy’ – that is to say, they never required the more advanced logic techniques (pointing pairs, etc.) to be solved.

Second Square Removal Attempt

My second approach was to simply remove numbers and check with the brute force solver whether the grid still had a unique solution – if it did, another number would be removed, if not, a different number would be removed. The process would repeat until the desired number of filled squares was reached.

This method worked reasonably well – it produced grids with a relatively small number of hints (down to about 25). Unfortunately, the time taken to produce a grid was extremely variable – anywhere from a few seconds to many minutes. The variation is caused by the random path that the generator uses – the decision as to which square to remove is entirely random. Thus the generator might ‘get lucky’ and happen to successively pick hints that maintain a unique solution, quickly finding an empty puzzle, or the generator might do the reverse and consistently pick squares which cannot be emptied without increasing the solution count. Furthermore, the brute force solver was being run again and again, increasing the time taken to find a grid.

Third Square Removal Attempt

This time, instead of using the brute force solver to check whether the generated grid had a unique solution, the logic solver was used to check if the grid was solvable. Since the logic solver is nearly always faster than the brute force solver, this should decrease the time taken to find a grid. However, the variability remains – the generator still removes squares at random.

Different Grid Varieties

Difficulty Level

The generator provides 2 different difficulty levels: easy and hard. Easy grids are solvable by using only the unique and singleton domain techniques (see the Logic Solver section of this report), while hard grids require at least one use of the other techniques.

Symmetry

The Sudoku puzzles published in *The Age* and *The Australian* have their hints distributed in a symmetric pattern across the board for aesthetic reasons. The generator provides the option of generating such grids. However, the number of possible empty grids that can be made from a filled grid is smaller if symmetry is required, so using the symmetrical option can slow down generation.

Example Generated Grids

Easy Grid (24 hints)

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

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

Hard Grid (26 hints)

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

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

Easy Symmetric Grid (27 hints)

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

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

Hard Symmetric Grid (30 hints)

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

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

Measurements

As mentioned above, even in its final implementation the generator is extremely variable as to how long it takes to generate a grid with a given number of hints as it clears squares in a random order. As such a simple average is not an appropriate measure. Instead, I ran the generator 10 times for various grid types and recorded the times.

Easy Grid (24 hints)

| Attempt | Time Taken |
|---------|------------|
| 1 | 168ms |
| 2 | 139ms |
| 3 | 1445ms |
| 4 | 142ms |
| 5 | 146ms |
| 6 | 127ms |
| 7 | 2260ms |
| 8 | 23741ms |
| 9 | 107ms |
| 10 | 114ms |

Although the numbers generally hovered around the 140ms mark, attempts 3, 7 and particularly 8 demonstrate the extreme variability of the generator.

Hard Grid (28 hints)

| Attempt | Time Taken |
|---------|------------|
| 1 | 6878ms |
| 2 | 11422ms |
| 3 | 25651ms |
| 4 | 16665ms |
| 5 | 10757ms |
| 6 | 17517ms |
| 7 | 26599ms |
| 8 | 12683ms |
| 9 | 24456ms |
| 10 | 43491ms |

The time taken to generate a hard grid is on average very high compared to generating an easy grid, but the time is still quite variable.

Easy Symmetric Grid (27 hints)

| Attempt | Time Taken |
|---------|------------|
| 1 | 207ms |
| 2 | 605451ms |
| 3 | 200ms |
| 4 | 68ms |
| 5 | 78ms |
| 6 | 72ms |
| 7 | 47ms |
| 8 | 53ms |
| 9 | 66ms |
| 10 | 53ms |

Attempt #2 is a good example of how the process may take a very long time to complete but that it will eventually terminate.

Hard Symmetric Grid (29 hints)

| Attempt | Time Taken |
|---------|------------|
| 1 | 3194ms |
| 2 | 770376ms |
| 3 | 12229ms |
| 4 | 17060ms |
| 5 | 12047ms |
| 6 | 3430ms |
| 7 | 14797ms |
| 8 | 26434ms |
| 9 | 11047ms |
| 10 | 16254ms |

Once again, the times vary significantly.

Conclusions

Achievements

Through generating this Sudoku solver and generator I feel I have improved my programming ability. This was perhaps the largest program in terms of time invested and lines of code written that I have created. The code is not of the highest quality, and it is severely lacking in documentation, but some of the problems posed by the project were a good challenge to solve. Writing the solver has demonstrated the advantages of 'smart' algorithms over naïve algorithms – evidenced in the measurements above. Finally, I have experienced participating in what could be called a small research project (useful for future work).

Project Status

The logic solver portion of the program is sufficient for solving many Sudoku puzzles. However, the implementation could likely be improved to execute faster. Furthermore, there are many logic solving techniques that have not been implemented.

The generator, on the other hand, is currently rather poorly implemented. It works in that it successfully generates grids of a given number of hints, but the variability of the time taken to generate is a significant weakness. Additionally, the generator currently only offers two levels of puzzle difficulty ('easy' and 'hard') – a scale of difficulties would be perhaps more useful.

References

Bau, D. (2006, September 4). *Sudoku Generator*. Retrieved October 24, 2007, from davidbau.com: http://davidbau.com/archives/2006/09/04/sudoku_generator.html

Stuart, A. (2006, November 21). *Sudoku Solver*. Retrieved October 24, 2007, from Scanraid Ltd: <http://www.scanraid.co.uk/sudoku.htm>