Comparison of Sudoku Solving Algorithms

Calum Harvey (Student ID: 170349985)

Computer Science

Supervisor: Dr Jason Steggles

Word Count: 1324

Abstract

Declaration

“I declare that this dissertation represents my own work except, where otherwise stated.”

Acknowledgements

Table of Contents

[1 Introduction 6](#_Toc33797528)

[1.1 Sudoku 6](#_Toc33797529)

[1.2 Motivation 6](#_Toc33797530)

[1.3 Aim 7](#_Toc33797531)

[1.4 Objectives 7](#_Toc33797532)

[2 Background Research 8](#_Toc33797533)

[2.1 Puzzle Generation 8](#_Toc33797534)

[2.1.1 Pre-generated Puzzles 8](#_Toc33797535)

[2.2.2 Generating Puzzles 8](#_Toc33797536)

[2.2 Algorithms 9](#_Toc33797537)

[2.2.1 Backtracking 9](#_Toc33797538)

[2.2.2 Simulated Annealing 9](#_Toc33797539)

[2.2.3 Genetic Algorithm 9](#_Toc33797540)

[2.2.3 Hill Climb or Tabu Search 9](#_Toc33797541)

[2.3 Existing Systems 9](#_Toc33797542)

[3 Algorithms 13](#_Toc33797543)

[4 Sudoku Generation 13](#_Toc33797544)

[5 Test Bed 13](#_Toc33797545)

[6 Testing 14](#_Toc33797546)

[7 Evaluation 15](#_Toc33797547)

[8 Conclusion 16](#_Toc33797548)

[References 17](#_Toc33797549)

# 1 Introduction

## Sudoku

Sudoku is a logic-based number placement puzzle game that has grown in popularity since it first appeared in French newspapers in the late 19th century. It usually consists of a 9x9 board containing 81 individual cells which are further partitioned into 9 3x3 smaller boxes that need to be filled. The aim is to fill these cells with a number 1 to 9 with each cell containing a single integer. There are 3 constraints on the board that must be met, each row, each column and each 3x3 smaller box must contain the numbers from 1-9 only once. When a sudoku is created a number of the cells are pre-defined by the puzzle creator to ensure that the puzzle only has one unique solution. The difficulty of the puzzle is determined by the number of pre-filled cells in the grid, more is easier; less is harder.

There have been various algorithms implemented to solve the Sudoku problem. The way a human solves the easier problems revolves around using the numbers already in the board and using logic to determine the missing numbers in each cell. When the problem becomes harder and requires the person to start guessing numbers, simple algorithms such as backtracking can be used to come to a solution in a shorter time than the human. The problem with the brute-force backtracking is the efficiency of the algorithm especially as the number of empty cells starts to increase as the Sudoku problem gets harder.

The answer to solve this efficiency problem could be the use of the stochastic algorithm optimization. Where backtracking searches through all the possible solutions to find the optimal result, stochastic algorithms can reduce the number of searches by stochastically iterating through the solution space for a puzzle only taking improvements to the potential solution. This allows the algorithm to move forward towards the optimal solution without having to check every possible outcome of the puzzle with the complexity of the problem being almost irrelevant to the efficiency of the algorithm.

## Motivation

The problem with all these algorithms is since all stochastic algorithms are different it can be very hard to tell which one is the most efficient to use on a Sudoku puzzle and more importantly are different algorithms better for different difficulties of problem. On an easier puzzle a backtracking algorithm might be the most efficient as there are fewer empty cells and the backtracking can cycle through all the possibilities quicker than a stochastic algorithm can come to an optimum solution. However, the more complex problems might require a more efficient stochastic algorithm but its unclear which one is the best to choose.

This paper explores a number of the different stochastic algorithms that can be employed to solve Sudoku problems along with backtracking to allow for a base brute-force case to also be examined alongside the more efficient algorithms. These can then be compared against each other in terms of their speed and number of iterations taken to reach the optimum solution. These can also be compared at a number of different difficulties of puzzle across a wide set of problems to ensure the comparison is as accurate as possible.

## Aim

To explore backtracking, stochastic and constraint satisfaction problem sudoku solving algorithms and compare and contrast them using a range of difficulty of puzzles.

## 1.4 Objectives

1. Explore current methods of sudoku solving and select three
2. Develop test bed to allow comparison of algorithms
3. Establish test data for comparison of algorithms
4. Implement algorithms into the test bed

1. Evaluate implemented sudoku algorithms at multiple complexities
2. Explore state of the art sudoku solving tools and evaluate selection

# 2 Background Research

This chapter addresses the background research done into Sudoku puzzles and the solving of them. It details the similar research that has already been carried out for the algorithms are being compared and research into design decisions that will be made.

## 2.1 Puzzle Generation

### 2.1.1 Pre-generated Puzzles

The simplest way for an algorithm to be tested on a valid sudoku board is to take already generated puzzles from the internet or Newspaper and implement them in a format that can be read by the algorithm and then solved [1]. This is used is research when a single algorithm’s performance is being tested against a limited selection of puzzles and a vast number of these puzzles it not required [2].

Choosing pre-generated puzzles guarantees that each puzzle will only have one solution and the difficulty of the puzzle will be guaranteed for each one, allowing for more emphasis on the algorithms. However, my aim is a comparison of algorithms and a more accurate comparison requires a big sample size. This is possible with pre-generated puzzles if I can get them automatically integrated from the generation website to my implementation but the complexity of that is similar to generating my own puzzles.

### 2.2.2 Generating Puzzles

There are two main advantages of generating puzzles as test data, the first is that you can control the difficulty of the puzzles that are being created which allows you to test algorithms against a very specific difficulty which allows for more accurate data. This is done by regulating the number of cells that are filled in the 9x9 board with more cells filled making the puzzle easier and having less making it harder.

The second is that these generated puzzles can be created in huge volumes due to them being created by the system which is infinitely faster than manually entering them from a third party. Also, when these puzzles are created the format that they are in is controlled by the system and therefore allows them easily to be parsed in and out of algorithms to be tested.

This is why many papers into the subject of sudoku solving, also incorporate the generation of the puzzles as it not only improves the research that they are doing but also allows for really good test data. This is shown [3] where the solving of sudoku puzzle using genetic algorithms are being looked at but also generation is incorporated into it.

## 2.2 Algorithms

The algorithms most commonly looked at are the stochastic algorithms which are all the ones being compared apart from backtracking. Simulated Annealing, Genetic Algorithm and Hill Climb are all similar in design but have a different take on the stochastic approach.

### 2.2.1 Backtracking

### 2.2.2 Simulated Annealing

### 2.2.3 Genetic Algorithm

### 2.2.3 Hill Climb or Tabu Search

## 2.3 Existing Systems

Explanation of all types of algorithm as well as explanation of how ones that will be implemented will be.

Talk about generating sudoku and the steps that have to be completed for this to occur:

Get completed board and remove cells checking that there is still one solution.

Research into how information will be displayed at the end and comparisons done (GUI?)

Big Emphasis on how similar systems implement their algorithms and how they do their comparisons. How they are all linked together and how mine differs from that. Also show how they display their data at the end and how I intend on doing it.

Look into using images of sudoku puzzle to explain stochastic. Potentially create own puzzle and annotate it to avoid having to reference as much

Backtracking:

With / without recursion

Stochastic:

Simulated annealing

Genetic

3rd algorithm -

Constraint Problem

Introduce concept but probably won’t be implemented

Implementation Technologies:

Visual studio code – good python integration, easy to use

Different programming language options:

Java – wanted to learn something new with my project

C++ - very good as very fast and allows for more accurate time taken readings, libraries for algorithms not great

Python – one area of interest for me is data analysis and python is used for this. Have used it briefly for a small project so not starting from nothing. Since used for data analysis as really good libraries for stochastic algorithms allowing programming to take less time and is guaranteed to work.

C – too low level for what I want to do. Algorithms hard to implement from scratch

Existing systems:

Use links in plan document to github comparing algorithms but in C++ and does not compare the same ones that I am doing.

Look for more similar systems – research paper comparing stochastic algorithms but not with backtracking at different difficulties of puzzle.

# 3 Algorithms

# 4 Sudoku Generation

# 5 Test Bed

# 6 Testing

# 7 Evaluation

# 8 Conclusion

# References

[1] Qqwing.com. (2020). *QQWing Sudoku*. [online] Available at: https://qqwing.com/ [Accessed 25 Feb. 2020].

[2] Mangwani, J., & Prateek, P. (2012). Using Progressive Stochastic Search to solve Sudoku CSP. <https://pdfs.semanticscholar.org/2596/8256c1dd61c5c5ff13b3ecc2fb146259f171.pdf>

[3] T. Mantere and J. Koljonen, ‘Solving, rating and generating Sudoku puzzles with GA’, in 2007 IEEE Congress on Evolutionary Computation, 2007, doi: 10.1109/cec.2007.4424632.