Solving Sudoku Puzzles with Genetic Algorithm

Rory Douglas
School of Computing and Mathematics
University of Derby
Derby, United Kingdom
r.douglas2@unimail.derby.ac.uk

***Abstract*—In this paper we investigate the feasibility of using genetic algorithms to solve a Sudoku. The genetic algorithm developed attempts to use a simple fitness function which checks whether numbers are repeated in rows and columns. While the algorithm was able to solve easy puzzles, it was unable to solve any of the more difficult puzzles.**

Keywords—Sudoku; Genetic Algorithm; Solver;

I. INTRODUCTION

Sudoku is a combinatorial logic puzzle consisting of a 9x9 grid. The objective of Sudoku is to fill the grid with numbers so each row, column and 3x3 sub-grid contains the digits 1 to 9 [4].

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

Fig. 1. Example of a Sudoku Puzzle.

As Sudoku has grown in popularity, a number of algorithms have been developed to solve them. Popular algorithms for solving Sudoku include backtracking and rule-based [2].

Backtracking is a very basic way of solving Sudoku, effectively based on brute force methods, however when a dead end is reached it will back track to an earlier guess and continue. Backtracking is potentially relatively slow, however it is guaranteed to find a result [2].

Rule-based works in a way similar to how most humans would attempt to solve a Sudoku, this involves testing the puzzle against certain rules to either fill in squares or remove candidate numbers [2].

A less popular algorithm for solving Sudoku is genetic algorithms. Genetic algorithm Sudoku solvers are based on the principles of evolution. A population of potential solutions, referred to as a chromosomes are generated when the algorithm begins. The fittest chromosomes from the population then breed to create a new generation. This continues until a solution is found. Breeding takes place through crossover between the two parent chromosomes, where parts of each parent are selected to form the child. There can also potentially be mutation, where the chromosome of the child changes regardless of the parents [1].

The aim of this paper is to investigate how suitable genetic algorithms are for solving Sudoku puzzles. To do this we will create a genetic algorithm capable of solving Sudoku.

II. LITERATURE REVIEW

In their paper "Solving, Rating and Generating Sudoku Puzzles with GA", Mantere & Koljonen developed a genetic algorithm capable of solving Sudoku puzzles. The algorithm created does not use many problem specific rules. They came to the conclusion that while their algorithm was able to solve puzzles, there were more efficient methods of solving Sudoku. They did however find genetic algorithms were ideal for rating the difficulty of puzzles [3].

Mantere & Koljonens solution will initially fill the puzzle with values leading to valid squares. Chromosomes are split into 9 sections consisting of 9 numbers, representing a 3x3 sub-grid in the Sudoku. Crossover involves selecting sections of a chromosome from each of the parents at random. Mutation occurs by swapping values within a section of a chromosome [3].

Das, Bhatia, Puri and Deep also investigated solving Sudoku in their paper "A Retrievable GA for Solving Sudoku Puzzles", however unlike Mantere & Koljonen, their initial

population is filled randomly, resulting in a much larger search space. Whereas in Mantere & Koljonen's solution crossover takes place between 3x3 sub-grids, Retrievable Genetic Algorithm (Ret-GA) performs crossover by rows. Further unlike Mantere & Koljonen, mutation is performed by generating random numbers for every value which is not part of the puzzle within a row [5].

Ret-GA was found to take more generations to solve puzzles on average, however was more likely to reach a solution than the pure genetic algorithm presented by Mantere & Koljonen [5].

III. Research Methodology

This section will provide an overview of the design of the algorithm and a plan for how to test the algorithm. The algorithm will be developed in C# and the user interface will use the Windows Runtime.

A. Algorithm

Like Mantere & Koljonen, the aim of this paper is to create a genetic algorithm capable of solving Sudoku puzzles with as few problem specific rules as possible. As such the genetic algorithm in this paper will be fairly similar to their design.

Chromosomes consist of 81 byte values. These are split into 9 sets of 9 byte values to represent 3x3 sub-grids. Chromosomes also feature a puzzle array of 81 bytes, this stores the original puzzle to ensure mutation never changes the original puzzle.

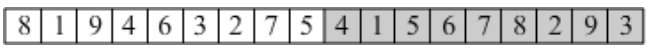

Fig. 2. Example of two sub-grids within a chromosome.

Crossover is performed by selected a 3x3 sub grid from either of the parent chromosomes based on the crossover rate. Mutation is performed by swapping two byte values within a 3x3 sub-grid.

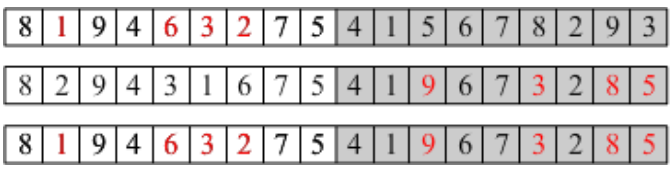

Fig. 3. Example of crossover.

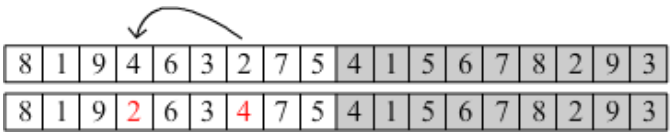

Fig. 4. Example of mutation.

Selection is performed using a combination of elitism and tournament selection. Based on the elitism chance the fittest chromosome from the parent population is added directly into the child population. Then tournament selection is used to select two parents. Using crossover and mutation a child is created and added to the child generation.

The fitness function in this algorithm simply checks whether the rows and columns have duplicates and increments a cost value for every time there is a duplicate. Squares are not checked as by design they will always be valid. Fitness will be measured between 0 and 1, with 1 being ideal fitness.

While more complex fitness functions were tested, including the one used by Mantere & Koljonen, these were not found to offer any noticeable performance increase. As noted by Mantere & Koljonen, the fitness function used for their algorithm was not ideal, however it was able to solve the puzzles they tested [3].

B. Test Plan

In order to test the algorithm, 2 puzzles of each of the difficulties: "Easy", "Medium", "Hard" and "Evil" will be generated using the online Sudoku generator at www.websudoku.com The algorithm will then attempt to solve each puzzle 50 times using the following settings:

Population Size	500
Tournament Size	3
Mutation Rate	10%
Crossover Rate	50%
Elitism	80%
Max Generations	50,000

Fig. 5. The settings used for testing the algorithm.

The average number of generations required to solve the puzzle will be recorded for each puzzle. These results will then be compared with the results from Mantere & Koljonen.

IV. Analysis

	Min	Max	Mean	Median
Easy #1	19	1958	254.16	113
Easy #2	8	2471	312.46	52

Fig. 6. The results of testing the algorithm.

The algorithm was able to solve both of the easy puzzles it was presented with. However as these results show, there is a reasonably large difference between the minimum and maximum generation counts for both puzzles. For the majority of attempts the algorithm would quickly reach a local optima of 98.76% complete, relying on a chance mutation in order to complete.

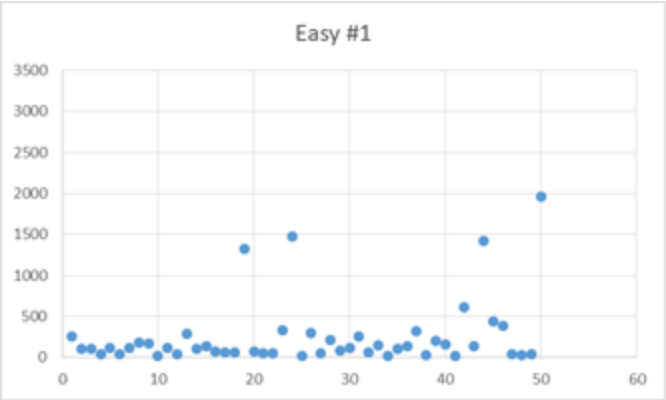

Fig. 7. The results of testing the first easy puzzle.

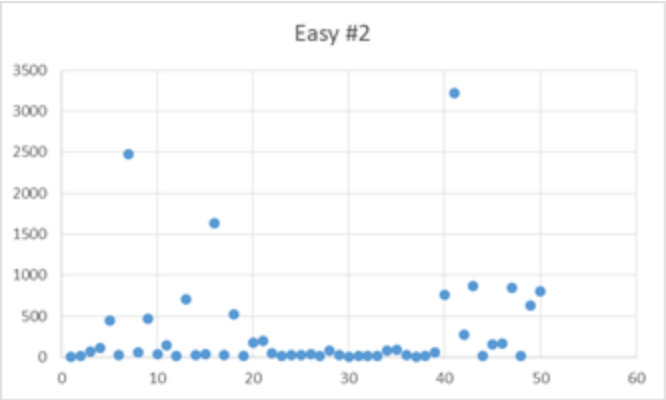

Fig. 8. The results of testing the second easy puzzle.

When solving the more difficult puzzles the algorithm would once again reach a local optima of 98.76%, however unlike the easy puzzles it would never reach 100%.

	Min	Max	Mean	Median
1 Star	20	8157	519	125
Easy	16	311	64	50

Fig. 9. Results for easy puzzles from Mantere & Koljonen's algorithm [3]

The above table shows Mantere & Koljonen's algorithm produced fairly similar results, despite the more complex fitness calculation. While their algorithm was able to solve more difficult puzzles, the maximum number of generations required to solve puzzles quickly climbs to 203,295 generations and the mean generations climbing to 29,078 generations [3].

V. Related Work

Mantere & Koljonen's implementation also faced issues with solving puzzles in a timely manner, however their algorithm was still able to complete the more difficult puzzles, albeit after a large number of generations.

Das, Bhatia, Puri and Deep noted that a due to the smaller search area when relying on intrinsically valid sub-grids, the algorithm would be more likely to quickly converge on a local optima. Their solution, implementing more problem specific rules for Sudoku was able to overcome a number of the issues with solving Sudoku using genetic algorithms.

VI. Future Work

An area of improvement could be to develop a more in depth fitness calculation. The current calculation has a local optima of about 98.76%. However the more complex algorithm used in Mantere & Koljonen's algorithm seems to produce fairly similar results on easy puzzles.

One area where the algorithm could be developed further is to add more problem specific rules, such as those used by Ret-GA.

The main area where this algorithm could be developed is to widen the search area. As squares must be valid, they are currently prescribed values at the beginning of execution from a limited number of values. This drastically limits the search space.

VII. Conclusions

The algorithm developed in this paper was able to solve simple Sudoku puzzles, however it is fairly safe to say it does not excel at solving Sudoku. Two major problems with the algorithm are the size of the search space and the fitness calculation.

With a more complex fitness calculation and larger search space it could be possible for this algorithm to solve more difficult puzzles.

VIII. References

[1] H. Mühlenbein, "Genetic Algorithms," 1997.

[2] P. Berggren and D. Nilsson, "A study of Sudoku solving algorithms," Royal Institute of Technology, Stockholm, 2012.

[3] T. Mantere and J. Koljonen, "Solving, Rating and Generating Sudoku Puzzles with GA," IEEE, New York, 2007.

[4] J. S. Provan, "Sudoku: Strategy Versus Structure," *American Mathematical Monthly*, pp. 702-707, October 2009.

[5] K. N. Das, S. Bhatia, S. Puri and K. Deep, "A Retrievable GA for Solving Sudoku Puzzles," 2012.