# Genetic Algorithms are Very Good Solved Sudoku Generators

Amit Benbassat
Sapir Academic College
M. P. Hof Ashkelon, Israel
amitbenb@mail.sapir.ac.il

## ABSTRACT

I present a simple and yet effective GA-based approach to content generation in the Sudoku domain. The GA finds multiple full boards which can act as solutions for Sudoku and Killer Sudoku puzzles. In this work I use a binning-based diversity maintenance approach in order to encourage GA to find more solution boards. resluts prove that though both approaches routinely manage to find multiple solution boards it is in fact the simple GA without any diversity maintenance that finds more such boards. Using a simpler approach to manipulate the fitness function to penalize previously found solutions improves the algorithm further.

## CCS CONCEPTS

**•Computing methodologies →Genetic algorithms; •Theory of computation →*Evolutionary algorithms;***

## KEYWORDS

evolutionary algorithms, puzzles, Sudoku, diversity

**ACM Reference format:**
Amit Benbassat. 2019. Genetic Algorithms are Very Good Solved Sudoku Generators. In *Proceedings of Genetic and Evolutionary Computation Conference Companion, Prague, Czech Republic, July 13–17, 2019 (GECCO '19 Companion),* 2 pages.
DOI: 10.1145/3319619.3326793

## 1 INTRODUCTION

Sudoku puzzles and Killer Sudoku puzzles have been maintaining their worldwide popularity for roughly two decades. The simple nature of the problem's definition coupled with the potential for solving difficulty make for an engaging puzzle game.

Below we quickly explain the Sudoku puzzle and show our approach to solving Sudokus as well as to problem generation.

## 2 SUDOKU

Sudoku is a 1-player puzzle game played on an $n \times n$ board. In puzzles solved by humans $n$ is typically 9 but can in principle be any square number. A Sudoku puzzle consists of this $n \times n$ partially filled by numbers in the range $\{1, 2...n\}$. The filled squares are called *Givens* and are designed to constrain the puzzle so that there is only one legal solution. The remaining squares are empty and to be filled by the solver in a fashion that is in line with board constraints.

In addition to the givens each Soduko puzzle solution has to comply by $3n$ additional constraints. $3n$ subsets of $n$ board positions must each contain each one of the numbers in $\{1, 2...n\}$ exactly once. These $3n$ subsets of the board consist of the following:

(1) The $n$ rows of the board.
(2) The $n$ columns of the board.
(3) The $n$ smaller squares of size $\sqrt{n} \times \sqrt{n}$ (called boxes), which make up the board.

In Figure 1 we see a $9x9$ Sudoku puzzle. This is considered an easy Sudoku because of the relatively high number of givens (30).

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

**Figure 1: A $9 \times 9$ Sudoku Puzzle with 30 givens. A solution must be found where each row, colomn, and $3 \times 3$ square contains a permutation of** $\{1, 2...9\}$

### 2.1 Killer Sudoku

Killer Sudoku is a popular variant of Sudoku where additional constraints are added to the $3n$ row, column and box constraints. There are typically less givens in a Killer Sudoku puzzle because of the extra constraints. An extra constraint in Killer Sudoku consists of a set of (typically adjacent) squares and a target sum for that set. There are many variants of Sudoku, however A Solved Killer Sudoku board looks the same as a solved Sudoku board. A solved Sudoku board can therefore be used as the basis for the generation of both types of puzzles.

## 3 GENERATING PUZZLES

In order to find Sudoku boards I first had to implement a GA-based solver for Sudoku. In this work I focus only on $9x9$ boards. In designing the GA I take a cue from previous research in solving Sudoku boards, most notably from Mantere and Koljonen [2] who used an optimized GA in order to both solve existing puzzles and create new ones. Following the example of Mantere and Koljonen I wish to find solved boards as the first step in finding new Sudoku Puzzles. But I also stray from their approach in some ways.

While previous work started with empty boards and ran a GA on these empty boards, the simulations I ran had the entire first row filled with givens. The reason for this is the observation that each solution board can be used to generate 9! solutions simply by permuting the locations of the numbers. My interest is in getting different solutions that cannot be reached that way from each other, and filling the first row with givens accomplishes that task.

I implemented a GA with individuals using a genome of 81 integers between 1 and 9 that represent the full Sudoku board. I then added attributes to the GA until it became an adequate Sudoku solver that I then used for content generation.

(1) The GA imposed a rule that kept all the rows as permutations of $\{1, 2...9\}$ that are in line with the givens in the row (thus reducing the number of constraints to be met from 27 to 18)
(2) A single genetic operator was implemented, a swap mutation that replaces the contents of two squares in the same row, neither of which is limited by givens (I used a 0.8 mutation rate).
(3) A fitness function of $100/(err + 1)$ where $err$ is the number of mistakes in a permutation constrains (number of repeated numbers).
(4) Tournament selection with tournament size 4.
(5) In some runs, hash-based bucketing diversity maintenance selection (also using tournament selection with tournament size 4) [1].
(6) Elitism with elite size 4.

This GA specification was sufficient to solve some Sudoku puzzles provided that the population size and generation limits were generous. I chose a high mutation rate since mutation is the only means of generating new solutions. The implementation of elitism prevents the high mutation rate from destroying the very best individuals.

## 4 RESULTS

To begin with, I ran 2 simulation sets of 20 runs each: One using a standard GA and the other using GA with bucketing. In both sets I used a population size of 1000 and a generation count of 2000. The idea was that the large population and long run would allow the diversity-maintaining bucketing technique to find more solutions.

The results were the complete opposite. It seems that due to the existence of many similar Sudoku solutions it was in fact the standard GA, unencumbered by the push for population diversity, which every time managed to find more solutions.

Bucketing GA found an average of 17.1 solutions (standard deviation of 2.05). Standard GA found an average of 25.9 solutions (standard deviation of 3.19).

### 4.1 Finding More Solutions by Manipulating Fitness

Clearly bucketing as it was implemented was inappropriate in this domain. Following this I attempted a simpler form of diversity maintenance inspired by clearing [3]. I implemented a generation counter that would activate every 100 generations, moving all found solutions into a special list which ensures their fitness will now be valued at 0.

A single standard GA simulation using the same parameters as above and augmented by this technique yielded 55 distinct solutions, far more than any previous attempt.

Next I attempted a larger simulation with a population size of 2000, and 4000 generations. This simulation yielded 228 distinct solutions.

## 5 CONCLUSIONS

I examined GAs as a content generation tool for Sudoku. We focused on attempting to find multiple distinct solved boards. It seems that due to the nature of the problem space a standard GA without any diversity maintenance manages to find multiple solutions with relative ease and in fact outperformed the more sophisticated diversity maintenance method.

I then decided to use a more straightforward approach to promote diversity and had success in roughly doubling the number of boards found by the same size of run. Increasing population size and number of generations increases the number of solution found.

This research is still ongoing. I believe that the GA system could use some improving and perhaps it will be able to find more solutions. I also wish to be able to implement the GA system on generating content to larger boards (I attempted to evolve boards for $n = 16$ but there was no success on that front as of yet).

Another part of process to be considered is turning the solved boards into challenging Sudoku puzzles. Perhaps it is possible to improve the method suggested by [2]. It may also be possible to generate puzzles for Sudoku variants. The system can be adapted to Killer Sudoku, for example, without requiring any changes other than to the fitness function so it takes the additional constraints into account.

## REFERENCES

[1] Amit Benbassat and Yuri Shafet. 2017. A Simple Bucketing Based Approach to Diversity Maintenance. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '17)*. ACM, New York, NY, USA, 1559–1564. DOI: http://dx.doi.org/10.1145/3067695.3082528
[2] Timo Mantere and Janne Koljonen. 2007. Solving, rating and generating Sudoku puzzles with GA. In *2007 IEEE congress on evolutionary computation*. IEEE, 1382–1389.
[3] Alan Pétrowski. 1996. A clearing procedure as a niching method for genetic algorithms. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*. IEEE, Nagoya, Japan, 798–803.