# An Empirical Analysis of Genetic Algorithm with Different Mutation and Crossover Operators for Solving Sudoku

D. Srivatsa, T. P. V. Krishna Teja, Ilam Prathyusha, and G. Jeyakumar(✉)

Department of Computer Science and Engineering, Amrita School of Engineering, Amrita Vishwa Vidyapeetham, Coimbatore, India
{cb.en.u4cse17459, cb.en.u4cse17465, cb.en.u4cse17424}@cb.amrita.students.edu, g_jeyakumar@cb.amrita.edu

**Abstract.** Prospective optimization tools such as Evolutionary Algorithms (*EAs*), are widely used to tackle optimization problems in the real world. Genetic Algorithm (*GA*), one of the instances of *EAs*, has potential research avenues of testing its applicability in real-world problems and improving its performance. This paper presents a study on the capability of the Genetic Algorithm (*GA*) to solve the classical Sudoku problem. The investigation includes various mutations and crossover schemes to unravel the Sudoku problem. A comparative study on the performance of *GA* with these schemes was conducted involving Sudoku. The findings reveal that *GA* is ineffective to deal with the Sudoku problem, as compared to other classical algorithms, as it often fails to disengage itself from some local optimum condition. On a positive note, *GA* was able to solve the Sudoku problems much faster, only the Sudoku had very few unfilled elements. A critical appraisal of the observed behavior of *GA* is presented in this paper, covering combinations of two mutations and three crossovers schemes.

**Keywords:** Genetic algorithm · Mutations · Crossovers · Puzzle · Sudoku · Local minima

## 1 Introduction

In the field of Computer Science, under Artificial Intelligence, Evolutionary Computing (*EC*) is a subfield of Soft Computing. *EC* has a family of algorithms called Evolutionary Algorithms (*EAs*) to resolve global optimization problems. Technically, *EAs* belong to the families of population-based, trial and error metaheuristic problem-solvers involving stochastic computation. *EAs* use biological evolutions such as reproduction, recombination, mutation, and selection. Natural selection (survival of the fittest) is espoused by individuals that consider fitness score provided by the fitness functions. The best individuals are selected for the iterative evolution process until a final solution is achieved with an appropriate fitness score.

Genetic Algorithm (*GA*), a component of *EA*, is used to generate high-quality accurate solutions to optimization and search problems [1, 2]. Parametric values, for the

Genetic algorithm, -several initial solutions, initial population, the maximum number of generations, mutation type, and crossover-type - can be initialized or provided by the user. *GA* encodes a population of solutions, embracing the natural evolution process, to arrive at an optimal solution in the population, with the help of the fitness function. The *GA* routine is executed until either the appropriate solution is achieved, or limited by the permissible maximum number of generations.

This paper aims at testing the suitability of applying *GA* to solve an *NP*-complete problem. The Sudoku problem was selected as a test case to solve Sudoku using *GA*.

A Sudoku puzzle is defined as a logic-based, number-placement puzzle. It can be of different sizes but the Sudoku used in this paper s $9 \times 9$ square, in which it has 9 rows, 9 columns, and 9 $3 \times 3$ grids. Initially, some cells of a Sudoku puzzle are provided with numerical digits in the range of 1 to 9. The puzzle is said to be solved, when each of the rows, columns, and grids, contain only single instances of numbers in the range of 1 to 9, such that no digit is repeated within the same row or column or grid. For a traditional Sudoku, the sum of digits in each row, column or grid should be equal to 45.

The remaining part of the paper is organized as follows: Sect. 2 discusses related works, while Sect. 3 presents the design of the experiment. Section 4 reports on the results and the observations, followed by closing remarks in Sect. 5.

## 2 Related Works

An insight into the assessment of *EA* in tackling Sudoku was presented in [3], which suggested that random mutation may have a significant impact on the performance of *EAs*. Coming to grips with Sudoku, using *GA,* by considering each $3 \times 3$ grid as a block was reported in [4]. The paper's authors treated each sub-block as a problem, and applied uniform crossover was to the particular block, in which the crossover positions were limited to the links between sub-blocks. Swap mutation technique was adopted such that it avoids duplication of numerals within a sub-block. This work also did a comparative study of disparity hypothesis and gene duplication using child. It was found The Sudoku was solved with a high degree of accuracy.

Code written in *C*++, to solve Sudoku, using *GA,* was explained by Weiss in [5]. He reported that *GA's* performance in Sudoku was impaired by slow convergence and inability to escape from local minima. In [6], descriptions on solving Sudoku, using different types of *GA* were delineated. A novel hybrid genetic algorithm (*HGA*) was generated, and the workings of *HGA*, interactive genetic algorithm (*IGA*), and classical *GA* were compared. He concluded that *HGA* gives better results than other *GAs.* However, the *HGA* is not that suitable for solving difficult Sudoku puzzles (the puzzles with more unfilled elements).

The authors in [7] described solving Sudoku, using *GA* mutation and crossover strategies. The Pencil and pen algorithm is one of the frequently proposed methods in the literature, for solutions of the Sudoku puzzle [7]. In *GA*, used in [7], to solve Sudoku puzzle one crossover type and two mutation types are used. The first mutation operation is applied to a gene (a cell in the matrix) of the chromosome. The second mutation is applied to each of the sub-square rows.

[8] discussed 3 objectives: (1) to check if the *GA* is efficient in solving the Sudoku problem. (2) Can *GA* be used to efficiently generate new puzzles for Sudoku? (3) Can *GA* be used to check the difficulty level of Sudoku problem?

The authors in [9] proposed a modified form of GA, Retrievable Genetic Algorithm (Ret-GA). Ret-GA was applied such that initially it creates a Blueprint Matrix (of the same size as the Sudoku puzzle), wherein '1' is assigned to a particular entry, only if the corresponding entry in the original Sudoku puzzle had an entered value (i.e. non-blank), otherwise that Sudoku location is assigned with value '0'. The initial population generated is subjected to Row-wise Uniform Crossover.

[10] described diverse algorithms - Cultural Genetic Algorithm (*CGA*), Repulsive Particle Swarm Optimization (*RPSO*), Quantum Simulated Annealing (*QSA*), and the Hybrid method that combines *GA* with Simulated Annealing (*HGASA*), which are used to solve the Sudoku puzzle. This paper concluded that *QSA* and *HGASA* are successful in solving the Sudoku puzzle. [11] solved the Sudoku puzzle using a novel multistage genetic algorithm (*MGA*). To solve a given puzzle, initially, a group table was constructed, with an initial random population. In every cycle, *GA* worked to find a better solution, and at each iteration, the group table was updated. Swap mutation technique was used, when mutation probability is satisfied.

Dealing with Sudoku through Variable Neighborhood Search (*VNS*) was shown in [12]. The basic idea is to explore a set of predefined neighborhoods, to provide a better diversification of successful solutions. A particular generation was selected based on the fitness function, invert an exchange strategy was undertaken to obtain the solutions. The performance was compared with Harmony Search, revealed that Harmony search is not as efficient as *VNS*. Finally, the paper concluded that *VNS* can produce competitive results at an easy level, and promising results in harder levels.

Authors in [13] introduced a heuristic to grapple with Sudoku, adopting modified crossover and mutation operators of *GA*. [14]. described a teaching strategy, engaging in-class exercise to introduce *GA* in Microsoft Excel to solve the Sudoku puzzle A permutation and row-crossover operators were designed for *GA* to solve Sudoku in [15]; the proposed algorithm was tested on different instances of Sudoku.

On a close review of different versions of *GA* available in the literature to solve Sudoku problems, the work presented in this paper proposes a study of the effect of using *GA* for solving Sudoku with combinations of different mutation and crossover operations.

## 3 Design of Experiments

As noted previously, a Sudoku puzzle is said to be solved, when each of the rows, columns, and grids, contain only single instances of numbers in the range of 1 to 9, such that no digit is repeated within the same row or column or grid. For a traditional Sudoku, the sum of digits in each row, column or grid should be equal to 45. The components of *GA* are designed as follows.

*Population for the GA* - The population for the GA to solve is the set of candidates each mean in Sudoku. A two-dimensional array was adopted to represent a Sudoku. Thus, a set of two-dimensional arrays in the population are used in this study.

*Fitness of Algorithm* - The fitness of each candidate has calculated an algorithm. This algorithm follows the following steps

(1) Calculate the number of occurrences of each number row-wise.
(2) Calculate the number of occurrences of each number column-wise.
(3) Calculate the number of occurrences of each number in all nine 3 × 3 grid.
(4) If the row-wise, column-wise, and 3 × 3 grid count is equal to 81 the puzzle is solved.

*Mutations* - In this experiment, two mutations were considered – Random Resetting and Swap Mutation.

- Random Resetting - If the value selected in each row is a fixed value then don't randomize it, otherwise insert a random value into it.
- Swap Mutation - Randomly select two genes, row-wise, and if both the elements do not belong to the set of fixed elements of the puzzle, then swap them; otherwise, continue, until both the selected elements are not fixed elements. The constraint is that the puzzle must contain at least two nonfixed elements in each row.

*Crossovers* - The three crossover operators deliberated in this study were a one-point crossover, two-point crossover, and uniform crossover.

- One-point crossover - In this procedure, a point is selected at random, which is said to be the crossover point; by dividing the chromosome, and swapping the genes of the chromosome concerning crossover point, new off-springs are produced.
- Two-point crossover - In this procedure, two points are selected at random, which are said to be the crossover points; genes of the parents between the two crossover points are swapped, and the genes, before the first crossover point, will be from the first parent, and genes from second crossover point, will be from the second parent. By combining these, new off-springs are generated.
- Uniform Crossover - In this procedure, the crossover points are randomly generated, and the genes of the parents are exchanged, at that particular point. No division of chromosome is done in this process, but the genes are replaced at the crossover points. There can be more than one crossover point in a chromosome.

## 4 Results and Discussions

The results of all the six combinations and mutations and crossovers used with *GA* to solve the Sudoku are explained in this section. The combinations of mutations and crossover are as follows: Random Resetting Mutation and One-Point Crossover, Random Resetting Mutation and Two-Point Crossover, Random Resetting Mutation and Uniform Crossover, Swap Mutation and One-Point Crossover, Swap Mutation and Two-Point Crossover and Swap Mutation and Uniform Crossover.

*The optimal score:* Each element in 9 × 9 Sudoku is to be counted one time and since there are 9 numbers in 9 rows, 9 columns, 9 grids, the optimal fitness score of Sudoku, of size 9 × 9, considered in the experiment, is 81, row-wise and column-wise. The optimal fitness score for the 3 × 3 grid is also 81. Summing up all the three fitness scores, a total value of 243 is the optimum score for the solution of Sudoku.

The *GA,* with each of the above-noted combinations of mutation and crossover, was applied to solve Sudoku puzzles, for 10 different runs and the average performance of *GA* is presented and discussed.

The results presented in Table 1 consist of the run number, whether or not the Sudoku puzzle was solved, Number of generations, Optimum score of each execution and the Time taken to complete execution. Table 1 depicts the performance of *GA* in solving Sudoku puzzles when Random Resetting Mutation and One-point crossover were used. An average number of generations obtained was 1091.6; the average optimum score was 244.40, and the average execution time was 54.71 s. The success ratio is 7:10 i.e., 7 out of 10 times the algorithm solved Sudoku perfectly.

**Table 1.** Sudoku results - *GA* with random resetting mutation and one-point crossover.

| Run no | Puzzle solved? | Generation | Optimum score | Time (sec) |
|---|---|---|---|---|
| 1 | NO | 2000 | 247 | 93.40 |
| 2 | YES | 255 | 243 | 20.08 |
| 3 | YES | 529 | 243 | 29.14 |
| 4 | YES | 182 | 243 | 16.73 |
| 5 | YES | 217 | 243 | 12.60 |
| 6 | NO | 2000 | 249 | 80.97 |
| 7 | YES | 663 | 243 | 37.2 |
| 8 | YES | 1565 | 243 | 82.01 |
| 9 | NO | 2000 | 247 | 97.83 |
| 10 | YES | 1505 | 243 | 77.21 |
| Average | | 1091.60 | 244.40 | 54.72 |

The results, using *GA* with Random resetting mutation and two-point crossover, are shown in Table 2. An average number of generations obtained was 1149.8, the average optimum score was 234.90, and average execution time was 47.15 s. The success ratio was 4:10, i.e. 4 out of 10 times, the *GA* algorithm solved Sudoku puzzles, successfully.

**Table 2.** Sudoku results - *GA* with random resetting mutation and two-point crossover.

| Run no | Puzzle solved? | Generation | Optimum score | Time (sec) |
|---|---|---|---|---|
| 1 | YES | 487 | 243 | 21.23 |
| 2 | YES | 133 | 243 | 13.89 |
| 3 | NO | 2000 | 253 | 71.86 |
| 4 | NO | 2000 | 249 | 82.25 |
| 5 | YES | 241 | 243 | 14.12 |
| 6 | NO | 2000 | 249 | 73.30 |
| 7 | NO | 2000 | 253 | 68 |
| 8 | YES | 512 | 243 | 35.30 |
| 9 | NO | 2000 | 249 | 81.39 |
| 10 | NO | 125 | 124 | 10.16 |
| Average | | 1149.80 | 234.90 | 47.15 |

Table 3 depicts the performance of *GA* when Random mutation and Uniform crossover was used. An average number of generations obtained was 1096.80, the average optimum score was 244, and the average execution time is 47.47 s. The success ratio was 8:10 implies that this combination of mutation and, crossover with *GA* was able to solve Sudoku in 8 out of 10 runs. The performance in solving Sudoku with Swap mutation and One-Point crossover is presented in Table 4. As seen from the results, the average number of generations obtained was 2000, the average optimum score was 263.20, and average execution time was 96.33 s. The success ratio was 0:10. It is worth noting that this combination of mutation and crossover does not support *GA* to solve the Sudoku problem in all cases of runs. This gives an insight that the impact of this combination to be studied further to understand the reason for such performance.

**Table 3.** Sudoku results - *GA* with random resetting mutation and uniform crossover.

| Run no | Puzzle solved? | Generation | Optimum score | Time (sec) |
|---|---|---|---|---|
| 1 | YES | 727 | 243 | 31.46 |
| 2 | YES | 405 | 243 | 19.84 |
| 3 | NO | 2000 | 247 | 80.65 |
| 4 | NO | 2000 | 249 | 93.90 |
| 5 | YES | 539 | 243 | 28.01 |
| 6 | YES | 1817 | 243 | 70.24 |
| 7 | YES | 414 | 243 | 19.85 |
| 8 | YES | 308 | 243 | 15.66 |
| 9 | YES | 1774 | 243 | 70.21 |
| 10 | YES | 984 | 243 | 44.19 |
| Average | | 1096.80 | 244 | 47.40 |

**Table 4.** Sudoku results - *GA* with random swap mutation and one-point crossover.

| Run no | Puzzle solved? | Generation | Optimum score | Time (sec) |
|---|---|---|---|---|
| 1 | NO | 2000 | 263 | 83.1 |
| 2 | NO | 2000 | 261 | 75.41 |
| 3 | NO | 2000 | 263 | 85.54 |
| 4 | NO | 2000 | 265 | 81.69 |
| 5 | NO | 2000 | 261 | 88.25 |
| 6 | NO | 2000 | 259 | 87.99 |
| 7 | NO | 2000 | 269 | 118.3 |
| 8 | NO | 2000 | 265 | 116.3 |
| 9 | NO | 2000 | 255 | 106.5 |
| 10 | NO | 2000 | 271 | 120.2 |
| Average | | 2000 | 263.20 | 96.33 |

Table 5 displays the performance of *GA,* with Swap mutation and Two-Point crossover. Results indicate that in all cases, *GA* uses the maximum generation of 2000, without reaching the solution. The average optimum score it obtained was 256, and average execution time was 95.06 s. The success ratio is 0:10. Table 6 delineates that the *GA* with Swap mutation and Uniform crossover also failed to attain the solution for Sudoku puzzle, in any of the runs, even with a maximum number of generations. The average optimum score was 260.66, and average execution time was 112.12 s.

The experimental results have established that the Swap Mutation routine is inadequate for *GA* to solve the Sudoku problem, irrespective of the associated crossover study. Hence, the comparative study was extended to probe the Random Resetting mutation, with all the three types of crossovers. The results were compared by the Success Ratio (*SR*), an average number of generations taken for successful runs (*AnG#*), and the Execution Time. The results are presented in Table 7.

**Table 5.** Sudoku results - *GA* with random swap mutation and two-point crossover.

| Run no. | Puzzle solved? | Generation | Optimum score | Time (sec) |
|---|---|---|---|---|
| 1 | NO | 2000 | 263 | 127.50 |
| 2 | NO | 2000 | 263 | 110.90 |
| 3 | NO | 2000 | 259 | 88.24 |
| 4 | NO | 2000 | 249 | 66.50 |
| 5 | NO | 2000 | 257 | 86.50 |
| 6 | NO | 2000 | 255 | 91.24 |
| 7 | NO | 2000 | 261 | 85.27 |
| 8 | NO | 2000 | 259 | 101.7 |
| 9 | NO | 2000 | 235 | 102.10 |
| 10 | NO | 2000 | 259 | 90.60 |
| Average | | 2000 | 256 | 95.06 |

**Table 6.** Sudoku results - *GA* with random swap mutation and two-point crossover.

| Run no | Puzzle solved? | Generation | Optimum score | Time (sec) |
|---|---|---|---|---|
| 1 | NO | 2000 | 261 | 123.50 |
| 2 | NO | 2000 | 265 | 126.80 |
| 3 | NO | 2000 | 259 | 94.54 |
| 4 | NO | 2000 | 261 | 103.90 |
| 5 | NO | 2000 | 261 | 112.70 |
| 6 | NO | 2000 | 255 | 105.80 |
| 7 | NO | 2000 | 255 | 103 |
| 8 | NO | 2000 | 261 | 125.70 |
| 9 | NO | 2000 | 263 | 112.20 |
| 10 | NO | 2000 | 265 | 113.10 |
| Average | | 2000 | 260.60 | 112.12 |

**Table 7.** Comparison of random resetting mutation.

| Crossover | *SR* | *AnG#* | *Time (Sec)* |
|---|---|---|---|
| One-point | 7 | 702.29 | 39.28 |
| Two-point | 4 | 299.61 | 18.94 |
| Uniform | 8 | 871 | 37.43 |

Test observations show that Random Setting mutation with Uniform crossover captured the top position, with higher *SR,* by yielding more number of successful runs. However, the Two-point crossover led the top position, for its speed, by achieving the solutions, with a minimal number of generations. Although its *SR* was 4, its *AnG#* was 299.61. These conflicting performances of Uniform and Two-point crossover need to be investigated further.

## 5 Conclusions

This paper presented evaluations of solving Sudoku problems by the Genetic Algorithm, using various combinations of Random mutations and crossover operations. Each blend of mutation and crossover was found to furnish different performance results. Empirical evidence of solving Sudoku problems, by various mixes of mutation and crossovers, has shown that deployment of Random mutation with Uniform crossover turned out more successful runs, whereas the same commingled with Two-Point crossover was effective in the speed of convergence.

## References

1. Janani, N., Shiva Jegan, R.D., Prakash, P.: Optimization of virtual machine placement in cloud environment using genetic algorithm. Res. J. Appl. Sci. Eng. Technol. **10**(3), 274–287 (2015)
2. Raju, D.K.A., Velayutham, C.S.: A study on GA based video abstraction system. In: Proceedings of World Congress on Nature Biologically Inspired Computing (2009)
3. Mcgerty, S., Moisiadis, F.: Are evolutionary algorithms required to solve sudoku problems? In: Fourth International Conference on Computer Science and Information Technology, pp. 365–377 (2014). https://doi.org/10.5121/csit.2014.4231
4. Sato, Y., Inoue, H.: Solving sudoku with genetic operations that preserve building blocks. In: Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games (2010). https://doi.org/10.1109/itw.2010.5593375
5. Weiss, J.M.: Genetic algorithms and sudoku. In: Midwest Instruction and Computing Symposium (MICS 2009), pp. 1–9 (2009)
6. Deng, X.Q., Da Li, Y.: A novel hybrid genetic algorithm for solving Sudoku puzzles. Optim. Lett. **7**, 241–257 (2013)
7. Kazemi, S., Fatemi, B.: A retrievable genetic algorithm for efficient solving of sudoku puzzles. Int. J. Comput. Inf. Eng. **8**(5) (2014)
8. Mantere, T., Koljonen, J.: Solving, rating and generating Sudoku puzzles with GA. In: IEEE Congress on Evolutionary Computation (2007)
9. Das, K.N., Bhatia, S., Puri, S., Deep, K.: A retrievable GA for solving sudoku puzzles. Technical report, Citeseer (2012)
10. Perez, M., Marwala, T.: Stochastic optimization approaches for solving sudoku. arXiv:0805.0697v1 (2008)
11. Chel, H., Mylavarapu, D., Sharma, S.: A novel multistage genetic algorithm approach for solving sudoku puzzle. In: Proceedings of 2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT) (2016)
12. Hamza, K.A., Sevkli, A.Z.: A variable neighborhood search for solving sudoku puzzles. In: Proceedings of the International Joint Conference on Computational Intelligence, vol. 1, pp. 326–331 (2014)
13. Gerges, F., Azar, D., Zoueii, G.: Genetic algorithms with local optima handling to solve sudoku puzzles. In: Proceedings of the 2018 International Conference on Computing and Artificial Intelligence (2018)
14. Ernstberger, K.W., Venkataramanan, M.A.: Announcing the engagement of sudoku: an in-class genetic algorithm game. J. Innov. Educ. **16**(3), 185–196 (2018)
15. Rodriguez Vasquez, K.: GA and entropy objective function for solving Sudoku puzzle. In: Proceedings of Genetic and Evolutionary Computation Conference Companion (2018)