A Twodoku puzzle solver using Genetic Algorithm

Natural Computing Group 25

Leonidas Kaldanis *s1030408*
Foteini Papadopoulou *s1122141*
Laurens van Rongen *s1010314*

Radboud University

The code is available at https://github.com/foteinipapadopoulou/Twodoku_solver_GA

Introduction

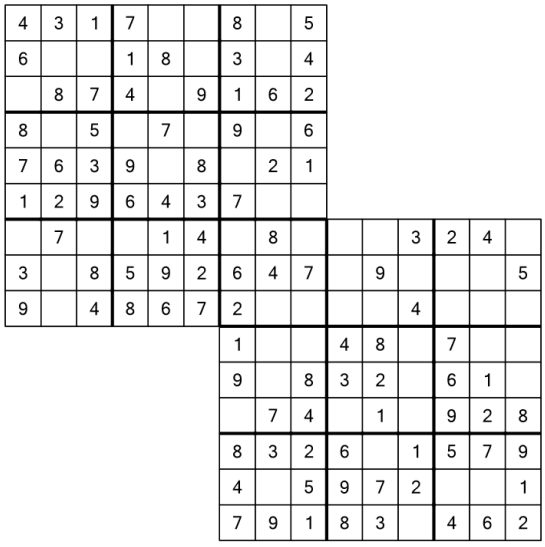
Sudoku, a logic-based puzzle game inspired by Leonhard Euler's 19^{th} century 'Latin Squares', first appeared in Japan in 1984. Modern Sudoku consists of a 9x9 grid, with each row, column, and 3x3 subgrid uniquely containing numbers 1-9 without repetition. Due to the computational complexity and the NP-hard nature problem, they have attracted attention in the fields of computational and algorithm design with different approaches being proposed to solve the traditional form of Sudoku by applying evolutionary algorithms [1, 2, 3, 4], ant colony optimization algorithms [5], cellular automata [6], employing various modifications and techniques. However, different variations of Sudoku puzzles, such as Double, Triple, or the more complex form Samurai Sudoku, have not been explored as extensively in terms of solving techniques. These variations often require additional rules and strategies to solve efficiently, making them more challenging than traditional Sudoku puzzles. Solving these kinds of complex variants of traditional Sudoku using Evolutionary algorithms is crucial as it involves computational techniques, that aim to enhance and optimize efficient solving methods. Numerous real-world problems, including resource allocation, task scheduling, and network optimization, share structural and constraint-based similarities with complex Sudoku variants like Twodoku. Exploring the application of evolutionary algorithms to such puzzles could provide practical and valuable insights to the research community.

Twodoku, or else called Double Sudoku, which is explored in this study, is a variant of the conventional Sudoku where two Sudoku puzzles overlap in the same grids. Usually, one 3×3 square grid overlaps with the two 9×9 Sudoku puzzles in the bottom right corner of the first one and in the top left corner of the second one, with the numbers of this block to satisfy the requirements in both Sudoku puzzles. The rules of the Twodoku align with the traditional one, where each row, column, and sub-block should contain each number from 1 to 9 once, with the overlapping grid to comply with these rules for both puzzles, making it an increased complexity variant of Sudoku. Figure 1 shows an example of an easy-level Twodoku puzzle with the fixed numbers on the left and the solved ones on the right.

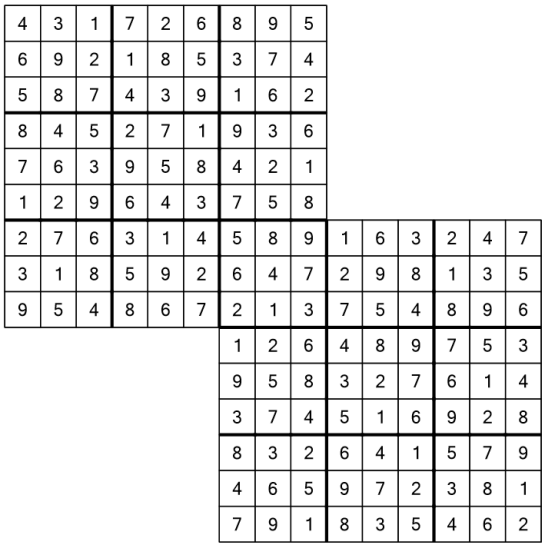
Despite important advancements in developing techniques and algorithms inspired by nature like Evolutionary algorithms to solve NP-hard problems as the traditional Sudoku, the application of these techniques to more complex Sudoku variants like Twodoku has not been extensively explored. This literature gap presents a significant opportunity for research and development in computational methods. Since Twodoku introduces additional degrees of complexity and constraints, it works as an excellent choice for testing the efficiency and adaptability of evolutionary algorithms. For this reason, our research aims to perform an analysis to assess the effectiveness of a hybrid genetic algorithm in solving Double Sudokus of three different levels, easy, medium, and hard. We also want to explore whether integrating techniques such as local search and elite population learning can enhance solution strategies and offer a flexible method suitable for a more complex structure such as the Double Sudoku, or else Twodoku. Specifically, our main research questions are the following:

- Main RQ: Can a genetic algorithm effectively solve Twodoku puzzles of different complexity levels (easy, medium, hard)?
  - RQ1: How does a specific range of mutation and crossover rates affect the tuning of an evolutionary algorithm for optimizing its performance on a medium-difficulty Twodoku puzzle?
  - RQ2: How does the integration of local search techniques and elite learning strategies affect the efficiency and speed of solving Twodoku puzzles?

The rest of the report is structured as follows: the next section presents studies related to solving Sudoku using Evolutionary algorithmic methods and various strategies. Subsequent section provides a detailed description of the methodology, dataset, and experimental setup, followed by the section with



(a) Unsolved Twodoku puzzle



(b) Solved Twodoku puzzle

Figure 1: An illustrative example of a Twodoku puzzle: the unsolved version on the left and the solved version on the right.

the findings and their analysis. Finally, last section offers concluding remarks, limitations of this study, and suggestions for future research.

Related Work

Various studies have been undertaken to develop and assess different solving strategies for the traditional Sudoku puzzle, leveraging evolutionary algorithms. Mishra et al. [7] have conducted a comprehensive analysis of the performance of different evolutionary algorithms, including Genetic Algorithms, Particle Swarm Optimization, Ant Colony Optimization, and Artificial Bee Colony Optimization, across various difficulty levels. Their study demonstrates that many of the tested methods achieve a 100% success rate with easy and medium Sudoku puzzles. However, despite some exceptions, the success rates in hard puzzles dropped greatly in many of the discussed evolutionary approaches.

Diving deeper into particular evolutionary methodologies, Gerges, Zouein, and Azar [4] have investigated what they refer to as the 'purge approach' to enhance the genetic algorithm's performance on traditional Sudoku puzzles, specifically to avoid being stuck at local optima. This approach mainly includes cross-overing the entire population with the most optimal chromosome, followed by a mutation of this chromosome. Then, the resulting individuals in the population are passed on to the next generation. Their heuristic algorithm, which uses modified crossover and mutation techniques, has been tested across different difficulty levels of Sudoku puzzles, with the results demonstrating the outperformance of this method compared to existing techniques.

Additionally, Pathak and Kumar [3] in their study, explore the role of entropy in an evolutionary algorithm to solve hard Sudoku puzzles. Their research introduces a hybrid approach to designing the fitness function by integrating problem-specific knowledge, which defines the uncertainty of a cell's position within the Sudoku. They have measured the efficacy and the success rate of their hybrid approach using the symmetric count metric they developed, which evaluates the state of the saturation point, where the fitness value is zero in the obtained solution. Their findings indicate a quicker convergence rate in most cases for solving hard and very hard puzzles that exceed the performance of the compared evolutionary algorithms. Sato and Inoue [8] proposes another technique of utilizing genetic operations that suggests 'effective building blocks' methodology to be used for genetic algorithms to solve Sudoku puzzles, showing further enhancement of the success rate of best solutions comparing different levels of Sudokus. Furthermore, Wang et al. [1] have adopted yet another approach to solving traditional Sudoku puzzles with genetic algorithms and the 'filtered mutations', as they named them. The filtered mutations method addresses the limitations of conventional genetic algorithm approaches, which usually involve random swap mutations by ensuring that at least one of the two numbers being swapped is acceptable for the targeted cell. Their method has been assessed in comparison with the conventional algorithm,

and the results have shown a competitive performance in solving Sudokus of different difficulty levels.

A recent study by Wang et al. [2] introduces a genetic algorithm named LSGA, which incorporates column and subgrid local search alongside elite population learning. The LSGA utilizes the local search strategy to enhance the convergence rate towards solutions, while the elite population learning aspect enables the population to evolve by integrating the historically optimal solutions. Their results showed that the LSGA not only exceeds the advanced approaches in terms of convergence rates but also in the capability to discover solutions across various difficulty levels of Sudoku puzzles.

Drawing inspiration from the aforementioned studies, and primarily from the recent advancement of the study by Wang et al. [2], we aim to develop a solver for the Twodoku variant utilizing the local search and the elite learning population strategy, discovering its efficiency in solving more complex Sudoku puzzles.

Methods

Our research is based on a hybrid evolutionary algorithm that integrates a genetic algorithm (Baseline) with a local search method and elite population learning strategy inspired by Wang et al. [2] to solve Twodoku puzzles of 3 different difficulty levels, named easy, medium, and hard. For each level, we sourced randomly Twodoku puzzles from the Sudoku-puzzles site, for the three selected levels. For the experiments, we leveraged 7 Twodokus, 3 easy, 2 medium, and 2 hard. Table 5 in Appendix present comprehensively the specific Twodokus utilized along with their number of given fixed numbers and the link that were sourced.

In order to address RQ1, our baseline algorithm was leveraged to explore the optimal mutation and crossover rates that allow the algorithm to find a solution over the first medium Twodoku puzzle, to establish a controlled and consistent environment for testing and comparison. The examined mutation rates are 0.05, 0.1, 0.15, and 0.2 and the cross-over rates are 0.1, 0.2, 0.3, and 0.4 with all their possible combinations. We have focused on a medium difficulty level so as to provide a balanced challenge that is neither too basic to simplify the algorithm's capabilities nor too advanced to complicate the initial analysis of the baseline. The utilized parameters for this initial exploration of the hyperparameters experiment can be found on Table 1.

Parameter	**Value**
Runs	15
Max Generations	5000
Population Size	150
Tournament Size	3

Table 1: *Hyperparameters and their values for the experiments to discover the optimal crossover and mutation rates for the baseline genetic algorithm for the medium-level Twodoku puzzle with ID 1.*

Since our objective is to tune the parameters and identify the most efficient mutation and crossover rates and their effect on the baseline algorithm configuration, statistical tests were not performed at this preliminary.

To address RQ2, we conduct experiments using three variants of our proposed genetic algorithm: the baseline model, a variant incorporating the local search technique, and another enhanced with elite population learning. After identifying the optimal hyperparameters for the baseline through the initial experiments described before, we applied these across all three model variants to evaluate their performance across different puzzle difficulties—easy, medium, and hard based on the success rate and the average generations count needed for each algorithm variation to find a solution. Due to computational constraints, 15 runs were performed on easy, medium, and hard puzzles. Furthermore, we employed Levene's statistical tests and t-tests to assess the significance of performance differences among the three models across the three difficulty levels. Table 2 presents the parameters applied to each difficulty level of the Twodoku puzzle experiments, along with their respective values.

Our implementation was mainly developed in Python, utilizing packages such as NumPy for numerical operations, SciPy for statistical tests, Pandas for data handling, and Matplotlib for generating plots. To ensure reproducibility across experiments, we set the random seed to 10.

In the following subsections, we provide a detailed step-by-step description of our genetic algorithm, including the integration of local search technique and elite population learning, as well as an overview of the algorithm structure.

Parameter	Value		
Difficulty Level	**Easy**	**Medium**	**Hard**
Runs	15		
Max Generations	5000		
Population Size	150		
Elite Population Size	50		
Tournament Size	3		
Cross-Over Rate	0.1		
Mutation Rate	0.1		

Table 2: *Hyperparameters and their values for the experiments conducted with the three variants of the genetic algorithm for each difficulty level of the Twodoku puzzle. The usage of the optimal mutation and cross-over rates were discovered in the initial stage of the experiments, discussed in the Results section.*

Initialization and Representation

The sourced Twodoku puzzles, which consist of two overlapping Sudokus within the same block, were initially presented in a conventional row-wise format, placed one after the other. The algorithm initializes by converting the provided Twodoku puzzle into a sub-block format, which better aligns with the operations of mutation and crossover discussed below. During the initialization phase, the algorithm generates an initial population of Twodoku solutions based on the given population size, where each candidate is randomly filled while respecting the fixed numbers from the original puzzle and complying with the subgrid Sudoku rule.

Tournament Selection

The implemented tournament selection in our genetic algorithm involves a traditional approach that includes random sampling of a subset of the current population depending on the given tournament size. Then, the sampled candidates are evaluated based on their fitness scores, with the low-scoring one being selected as the candidate to participate in the next, crossover and mutation processes.

Fitness Function

For the evaluation of each candidate in the population, a fitness function has been created and used to evaluate the upper, first 9 rows of the candidate grid, and the lower, last 9 rows, Sudokus of the Twodoku puzzle in sub-block format. The total fitness function combines the single fitness scores produced for the upper and lower Sudoku. The single fitness function quantifies the fitness of the given Sudoku grid by calculating the total number of repeated numbers across all its rows and columns, with the aim of our genetic algorithm to minimize these repetitions in order to optimize the grid's compliance with Sudoku rules. The single fitness function is mathematically defined as follows:

$$\text{Fitness} = \sum_{i=1}^{9} R_i + \sum_{i=1}^{9} C_i$$

, where $R_i = 9 - \#\text{distinct numbers in } i\text{-th row}$, $C_i = 9 - \#\text{distinct numbers in } i\text{-th column}$.

Crossover

For the crossover operation in our approach, we used a variation of the preserving building blocks method proposed in [8]. In this method, a crossover score is calculated using the blocks in each row for the first offspring and the blocks of each column for the second one. In order to compute the crossover score of a block-row, we check the rows of those three blocks and examine how many different numbers there are in each row. If a row does not contain any repeated number, meaning there are all the numbers between 1 and 9 only once, then its score is 9. Subsequently, the maximum score of a block-row is 27. Then this score is compared with the crossover score of the second parent for the same block-row and the parent with the highest score will lend this block-row to the first child. This procedure continues for all 3 block-rows, in order to create the first offspring. Similarly, the second offspring will be created by the crossover scores of the block-columns of the parents. Figure 2 visualizes the cross-over operation applied in the conventional form of Sudoku, as described above.

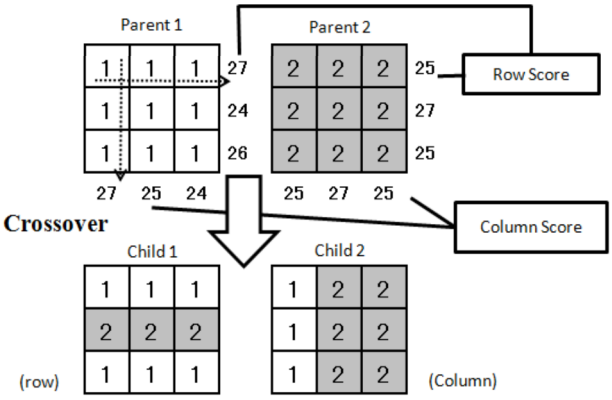


Figure 2: *Cross-over operation applied in a classic Sudoku. Image adapted from [8].*

In our case, for the Twodoku, we split the puzzle into two normal Sudokus, the upper one and the lower one, and we performed crossover independently for the two puzzles*.

Mutation

For the mutation operation of our genetic algorithm, the method iterates through each block of the candidate, deciding based on a given mutation rate whether to mutate that block. If the mutation is accepted, the operation finds the indices that the cells within a block are not filled with given numbers, based on a help array that contains mutable positions. If there are at least two mutable indices of cells, the function randomly selects two cells and swaps their values. Then, the mutated individual is returned if the mutation is approved, while in the opposite case, the initial candidate is returned.

Local Search

The local search algorithm we deploy is based on the one described in [2], with some modifications. In the initial algorithm, a set C contained all the illegal columns (columns that contain repeated numbers) and for every illegal column in C, the positions of the repeated numbers were saved. Then, for every illegal column in C, another column from C was selected at random and if there were repeated numbers in the same row and the value was not included in the other column, for both columns, they swapped the numbers. In that way, they ensured that both columns will have at least one less repeated number and in the position of the previously repeated number, there is a new number for the column. For example, in Figure 3a we see that column A contains two 1s and column B contains two 2s and there is a common row that they both contain the duplicates. Since 1 is not included in column B and 2 is not included in column A, the swap can take place. A similar procedure was executed for the subblocks, as shown in Figure 3.

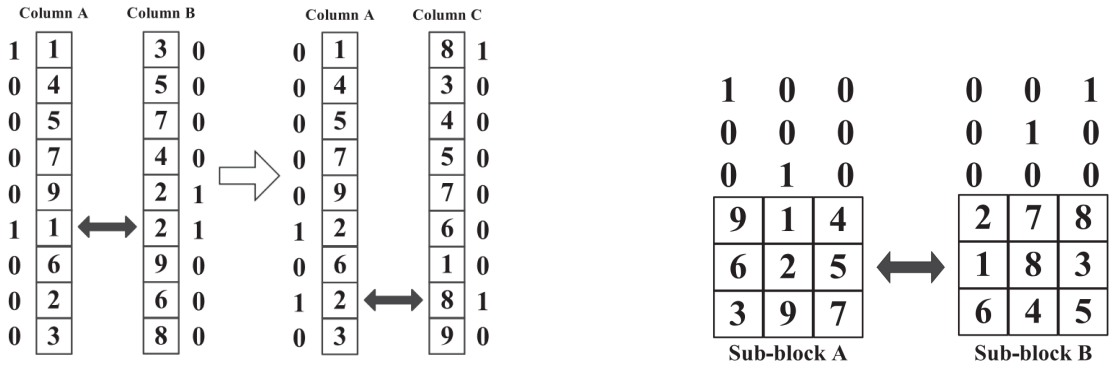
Since their initialization of the population did not allow for repeating numbers in a row, this swap between the columns and the subblocks maintained this rule. On the other hand, our initialization does not allow for repeating numbers in a subblock and thus a modification was made to their algorithm.

The first change we applied was the fact that we performed a local search on rows instead of subblocks. The second and most essential change was the fact that we checked to detect if the positions of the repeated numbers belonged to the same subblock. That way we maintain the rule of no repeating numbers in a subblock. Lastly, we decided to check all the combinations in C to make the swap.

Elite Population Learning

The elite population learning technique was developed in our algorithm following the suggested approach by Wang et al. [2], which helps to avoid the genetic algorithm to get stuck into local optima. This method leverages a queue structure elite population, which collects the best candidates of each generation and refreshes them with newly optimal individuals. As the authors of this study describe, the elite population learning replaces the worst individual candidates with a random candidate x_{random} from the

*A slightly different approach was implemented for the six middle block-rows / block-columns, due to the Overlapping Block Problem (OBP), discussed in a later section.



(a) Local search on columns

(b) Local search on subblocks

Figure 3: *Local search on columns(left) and sub-blocks(right) as described in [2]. Images adapted from [2]*

elite population with a probability P_b or it reinitializes by randomly filling the worst candidate, as in the initialization process. The probability P_b can be described mathematically as follows:

$$P_b = \frac{\text{Max}f(x) - f(x_{random})}{\text{Max}f(x)} \tag{3}$$

with $Maxf(x)$ indicating the fitness of the worst individual and $f(x_{random})$ being the fitness of the randomly selected candidate from the elite population. In that way, this approach enables the algorithm to overcome local optima by enabling the exploration of new populations while simultaneously exploiting the best candidates.

Overall Approach

A pseudocode of the overall approach followed to build the genetic algorithm solver for the Twodoku puzzles is presented in Appendix (Algorithm 1). Initially, the algorithm initializes the population, sorts and evaluates the initial population's fitness. The main loop of the algorithm runs until either a solution is found or the maximum number of generations is reached. If the fitness of the first individual for the sorted population is zero then the algorithms stop returning the solution and the generation number. Within the loop, a new population is made through tournament selection, cross-over, and mutation operations, and the population is replaced with the new one. Based on the experiment, local search and elite learning population techniques are applied to the population. If no solution is found until the maximum generations count is reached, the algorithm returns None.

The Overlapping Block Problem (OBP)

Implementing the methods of crossover proved to be a bit more challenging than it is in a typical Sudoku. Considering a Twodoku as a combination of an upper and a lower Sudoku, then the problem that is introduced is based on the common block the two puzzles share.

For example, to obtain the first offspring in the crossover operation, we compare the crossover scores of the two parents and select the parent with the higher score to pass its block-row to the child. However, when considering the block-row containing the overlapping block, there is a chance that when you compare the upper Sudoku, then parent 1 should be selected, but when you compare the lower Sudoku, parent 2 should be selected. The problem occurs with the selection of the overlapping block, since in one case the block from parent 1 should be inherited, whereas in the other case from parent 2.

To bypass this problem, we perform crossover the usual way for the block-rows/block-columns that do not contain the overlapping block and merge the scores of the upper and the lower Sudoku in the ones that it is contained. Then the comparison is made with a unified score and the offspring inherits all 5 blocks. In that way, we include the cases that the crossover scores are better for one specific parent for both upper and lower Sudoku, and we make this convention for the other two cases.

Results

To investigate RQ1, as described in the Methods section, we performed 15 runs on one of the medium puzzles to find approximately the optimal pair of cross-over and mutation rates and tune the baseline algorithm. In Table 3, we can see the different success rates for different pairs of mutation and crossover rates of the baseline Twodoku algorithm. The optimal pair appears to be the one for mutation rate and crossover rate equal to 0.1 with a success rate of 8/15 (53.3%).

Mutation Rate \ **Crossover Rate**	**0.1**	**0.2**	**0.3**	**0.4**
0.05	2/15	5/15	4/15	1/15
0.1	**8**/15	4/15	2/15	5/15
0.15	0/15	0/15	7/15	6/15
0.2	0/15	0/15	0/15	0/15

Table 3: *Success rates for different mutation and crossover rates of the baseline genetic algorithm solving the medium-level Twodoku.*

In Figure 4, the boxplot, which shows the distribution of the number of generations needed from the algorithm to find solution, contributes towards understanding why this pair is the most efficient out of the ones that were examined. For a mutation rate equal to 0.2, no crossover rate, out of the selected range, helped the algorithm find a solution. Additionally, this boxplot provides more information on the variability of those rates, but no specific pattern can be spotted between the values of the crossover rates and the mutation rates. However, interestingly, a mutation rate and crossover rate of 0.1 leads to the median being lower than 5000, which is not the case for the other combinations. This reinforces the idea that this combination is the best out of the ones that were tested.

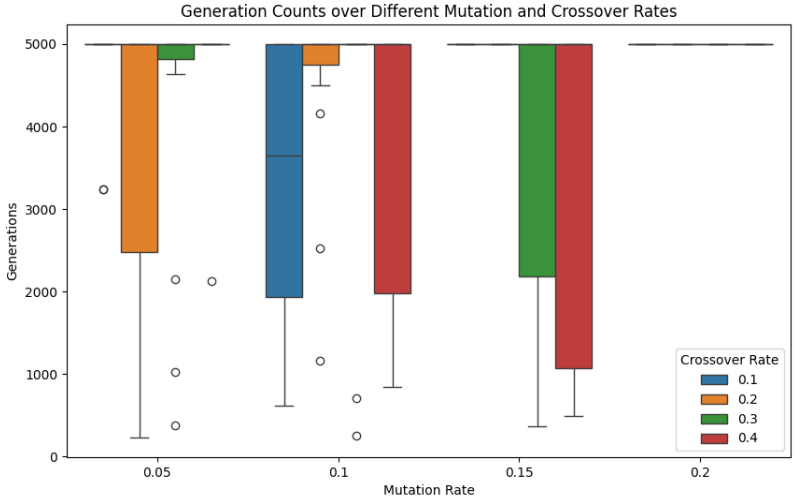
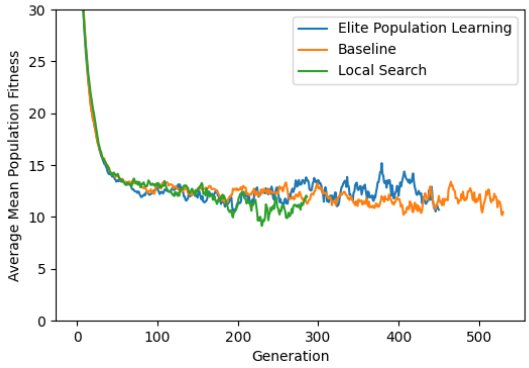


Figure 4: *Box plot showing the distribution of the number of generations needed for the baseline genetic algorithm to discover solutions across different mutation and crossover rates for the medium-level Twodoku. Each color represents a unique crossover rate, illustrating variations in mutation and crossover rates that impact the baseline algorithm's performance.*

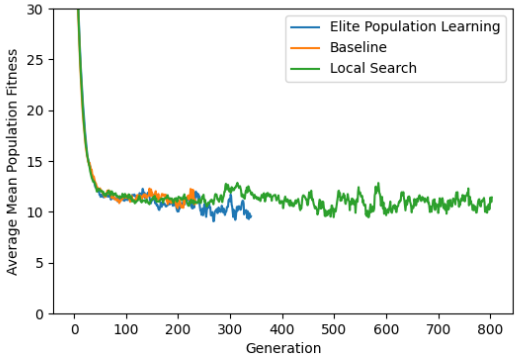
After obtaining the optimal mutation and crossover rate, the rates were tested on a baseline, a baseline with local search and a baseline with elite learning as can be seen in Table 4. The results for the easy puzzles show that solutions were found in all cases, whereas for medium puzzles a solution was found in half the cases, and only two times were solutions found for hard puzzles, both from the second hard puzzle. The average generation a puzzle was found differs quite a bit from puzzle to puzzle, and between baseline and using local search or elite learning. Using the statistical analysis technique specified in the Methods, no significant differences (p-value >0.05) were found between any average generation in the

	Metric	Easy 1	Easy 2	Easy 3	Medium 1	Medium 2	Hard 1	Hard 2
Baseline	*Succ_Count*	**15**/15	**15**/15	**15**/15	**7**/15	**15**/15	**0**/15	**1**/15
	Avg_Gen	188	110.67	35.8	1756.14	1329.01	-	6372
LS	*Succ_Count*	**15**/15	**15**/15	**15**/15	**6**/15	**15**/15	**0**/15	**0**/15
	Avg_Gen	115.80	202.80	44.27	2971.50	2500.87	-	-
EL	*Succ_Count*	**15**/15	**15**/15	**15**/15	**9**/15	**15**/15	**0/15**	**1/15**
	Avg_Gen	143.20	123.60	43.27	2959.89	2247.27	-	8465

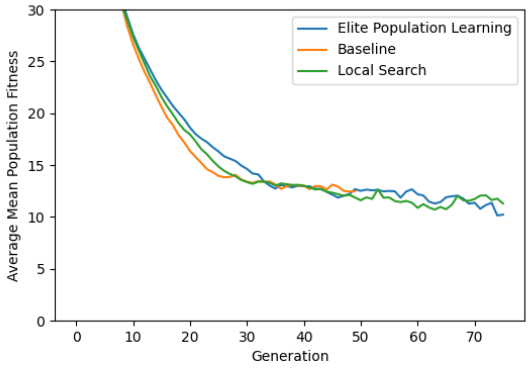
Table 4: *Comparison of the baseline, local search, and elite learning population variants of genetic algorithm on 3 Twodoku easy puzzles, 2 medium, and 2 hard based on success number of runs achieved solution(Succ_Count), and average generations needed to find solution(Avg_Gen). LS - Local Search, EL - Elite Learning*



(a) Easy 1



(b) Easy 2



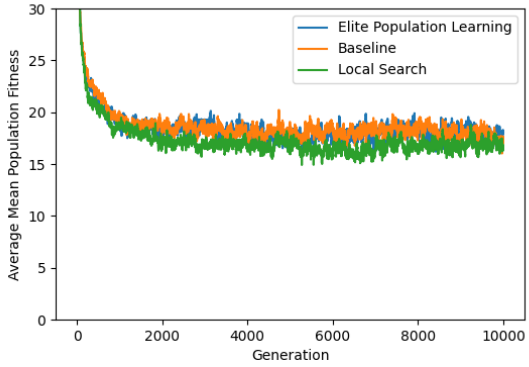
(c) Easy 3

Figure 5: *Average mean fitness of the population per generation for the easy puzzles for baseline, local search and elite population*

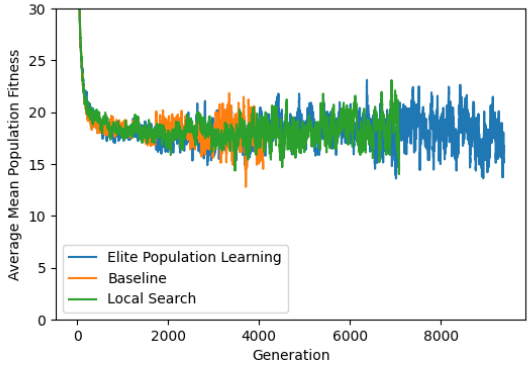
same puzzle for the different variations of the algorithm.

Between puzzles, there was quite a bit of variance, with the third easy puzzle taking the least generations to come to a solution at around 40 while the others were between approximately 100 and 200. The medium puzzles have an average between 1300 and 3000, while the hard puzzles either did not find a solution or the solution was found at approximately 6300 and 8500 generations.

Looking at fitness, Figure 5, Figure 6 and Figure 7 show the mean average population fitness for each

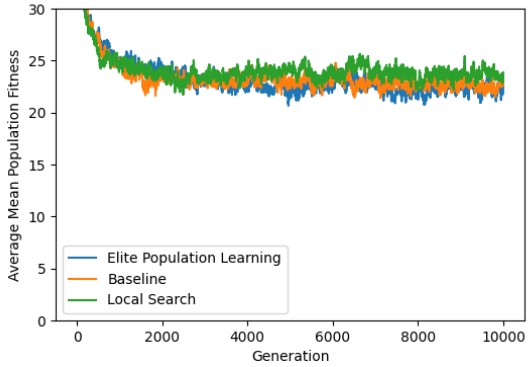


(a) Medium 1

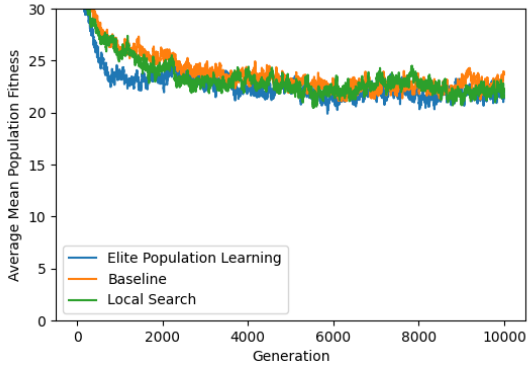


(b) Medium 2

Figure 6: *Average mean fitness of the population per generation for the medium puzzles for baseline, local search and elite population*



(a) Hard 1



(b) Hard 2

Figure 7: *Average mean fitness of the population per generation for the hard puzzles for baseline, local search and elite population*

of the puzzles. The lower bound for each figure increases with the puzzle difficulty, around 10-15 for easy, 15-20 for medium and 20-25 for hard. As for fluctuations, Easy 3 has few, while Hard 2 has more than Hard 1. The difference between techniques is also small, there doesn't seem to be much in terms of an identifiable pattern between techniques. None of the techniques seem to have a consistent advantage over the others.

Discussion

After examining various combinations of mutation and cross-over rates for fine-tuning our genetic algorithm model in the medium-level Twodoku, the results presented in Table 3 and Figure 4 illustrated that there is no specific pattern or trend in adjusting these specific parameters. The optimal combination of these rates using the success rates and generations counts as evaluation metrics has proven to be, for our model, a mutation and crossover rate of 0.1. Subsequently, applying this combination of parameters to our model and addressing RQ2, the results have shown that our genetic algorithm solver for Twodoku puzzles can solve the easy puzzles with significant confidence, and even the Medium puzzles in about half of the runs, within 5000 generations. Hard puzzles proved too complex for our genetic algorithm, for all three variations of our developed genetic algorithm, and as the generation where one solution was found was around 6000 to 8000, it can be assumed that it is not feasible to use our genetic algorithm on very difficult puzzles. Additionally, as can be seen in Table 4, the impact of the two selected variations for the Twodoku baseline genetic algorithm, local search, and elite population learning, was not proven statistically significant based on the average generation count needed to discover a solution. This is further corroborated with Figure 5, Figure 6 and Figure 7, where no significant consistent difference can be observed.

Conclusion

In this study, a genetic algorithm was developed to solve a more complex version of Sudoku, Twodoku, by utilizing two variants of existing genetic algorithms that solve the conventional Sudoku. By combining different approaches and adjusting them in order to be usable for a Twodoku, we developed an effective algorithm that performed well on easy and medium puzzles, though it struggled with the hard ones, for a number of cross-over and mutation rates combinations. Moreover, we tested techniques of local search and elite population learning, comparing their performance against the baseline, with the results showing there is no significant impact of them in the baseline algorithm.

Several limitations should be acknowledged to provide a comprehensive understanding of the findings, in the context of this study's objectives. Due to computational and time constraints, the total runs and maximum generations for each difficulty level of the puzzle were limited to 15, and 10,000 respectively. More runs could provide better insight into the true performance of the different methods.

Future work could include trials with various hyperparameters such as tournament size, population, and elite population size for each level of Twodoku puzzles, especially for the hard puzzles. Additionally, one could use enhancements related to constraint satisfaction (through filtered mutation methods [9]), evaluation of the children's fitness values after the cross-over operations, and implementation of different fitness function variations such as in [3] utilizing the role of entropy to better deal with the OBP problem. Lastly, the impact of the execution time for discovering a solution on each level could further enhance our knowledge of the implications for each model in solving this complex variant of Sudoku.

Author Contributions

The authors contributed equally to the conceptualization, development, and writing of this study. Specifically, Laurens van Rongen explored different variations of genetic algorithms that solve conventional Sudokus from the literature, while also contributing to the experimental design. Leonidas Kaldanis developed the code to adapt the genetic algorithm solver of traditional Sudokus to the Twodoku puzzle, while also exploring the OBP and which statistical tests to run. Foteini Papadopoulou explored the elite population learning from the respective study and contributed as well to developing the code for the initial Sudoku algorithm, the visualizations and the statistical tests, while also maintaining the Github page in a tidy manner for better reproducibility.

References

[1] Chuan Wang et al. "A Novel Evolutionary Algorithm With Column and Sub-Block Local Search for Sudoku Puzzles". In: *IEEE Transactions on Games* PP (Jan. 2023), pp. 1–11. DOI: 10.1109/TG.2023.3236490.

[2] Chuan Wang et al. "A Novel Evolutionary Algorithm With Column and Sub-Block Local Search for Sudoku Puzzles". In: *IEEE Transactions on Games* 16.1 (2024), pp. 162–172. DOI: 10.1109/TG.2023.3236490.

[3] Neeraj Pathak and Rajeev Kumar. "Entropy guided evolutionary search for solving Sudoku". In: *Progress in Artificial Intelligence* 12.1 (Mar. 2023), pp. 61–76. ISSN: 2192-6360. DOI: 10.1007/s13748-023-00297-7. URL: https://doi.org/10.1007/s13748-023-00297-7.

[4] Firas Gerges, Germain Zouein, and Danielle Azar. "Genetic Algorithms with Local Optima Handling to Solve Sudoku Puzzles". In: *Proceedings of the 2018 International Conference on Computing and Artificial Intelligence*. ICCAI '18. Chengdu, China: Association for Computing Machinery, 2018, pp. 19–22. ISBN: 9781450364195. DOI: 10.1145/3194452.3194463. URL: https://doi.org/10.1145/3194452.3194463.

[5] Huw Lloyd and Martyn Amos. "Solving Sudoku With Ant Colony Optimization". In: *IEEE Transactions on Games* 12.3 (2020), pp. 302–311. DOI: 10.1109/TG.2019.2942773.

[6] T. P. Chatzinikolaou, R. E. Karamani, I. A. Fyrigos, et al. "Handling Sudoku Puzzles with Irregular Learning Cellular Automata". In: *Nat Comput* 23 (2024), pp. 41–60. DOI: 10.1007/s11047-024-09975-4. URL: https://doi.org/10.1007/s11047-024-09975-4.

[7] D. B. Mishra et al. "Solving Sudoku Puzzles Using Evolutionary Techniques—A Systematic Survey". In: *Soft Computing: Theories and Applications*. Springer, 2017, pp. 791–802. DOI: 10.1007/978-981-10-5687-1_71.

[8] Yuji Sato and Hazuki Inoue. "Solving sudoku with genetic operations that preserve building blocks". In: *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*. IEEE. 2010, pp. 23–29.

[9] Zhiwen Wang, Toshiyuki Yasuda, and Kazuhiro Ohkura. "An evolutionary approach to sudoku puzzles with filtered mutations". In: *2015 IEEE Congress on Evolutionary Computation (CEC)*. 2015, pp. 1732–1737. DOI: 10.1109/CEC.2015.7257096.

Appendix

Puzzle ID	Given Numbers	Link
Easy 1	99	sudoku-puzzles.net/twodoku-easy/147/
Easy 2	99	sudoku-puzzles.net/twodoku-easy/
Easy 3	99	sudoku-puzzles.net/twodoku-easy/7/
Medium 1	67	sudoku-puzzles.net/twodoku-medium/660/
Medium 2	68	sudoku-puzzles.net/twodoku-medium/
Hard 1	48	sudoku-puzzles.net/twodoku-hard/94
Hard 2	49	sudoku-puzzles.net/twodoku-hard/941/

Table 5: *Table of Twodoku Puzzles with Puzzle ID, Given Fixed Numbers, and a Link to the Sudoku-puzzles site of the puzzle used for our hybrid genetic algorithm.*

Algorithm 1 Genetic Algorithm Solver Pseudocode for Twodoku

Initialize:
Initialization of population
Sorting - Evaluation of the initial population's fitness.
while max generations not reached and solution not found **do**
    if the best fitness is zero **then**
        return solution, generation number when solution found.
    end if
    Preparation of the new generation:
    Initialize an empty new population list.
    while the new population is not full **do**
        $parent1 \leftarrow$ tournament selection()
        $parent2 \leftarrow$ tournament selection()
        $child1, child2 \leftarrow$ crossover($parent1, parent2$)
        mutate($child1$) and mutate($child2$)
        Add $child1$ and $child2$ to the new population list
    end while
    Replace the old population with the new.
    if local search is enabled **then**
        Perform column local search
        Perform row local search
    end if
    if elite population learning is enabled **then**
        Preserve a subset of the best individuals
        Perform elite population learning
    end if
    Sorting — Evaluation of the population's fitness.
end while
if no solution was found **then**
    return None
end if