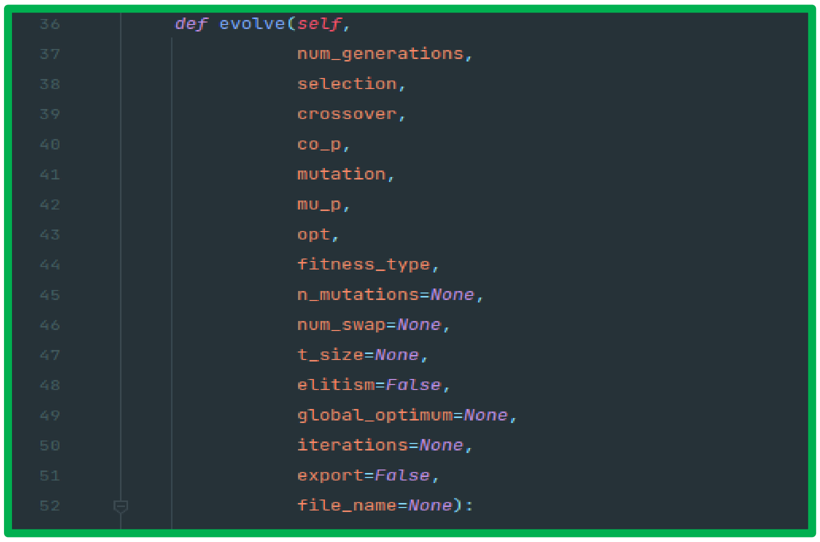
Master’s in Data Science and Advanced Analytics

Computational Intelligence for Optimization

2021/2022



```
def evolve(self,
           num_generations,
           selection,
           crossover,
           co_p,
           mutation,
           mu_p,
           opt,
           fitness_type,
           n_mutations=None,
           num_swap=None,
           t_size=None,
           elitism=False,
           global_optimum=None,
           iterations=None,
           export=False,
           file_name=None):
```

Sudoku Genetic Solver

Group W

Diogo Papeleira (20180416)

Henrique Barreiros (20211380)

Miguel Bernardo (20210580)

Introduction

Sudoku is a puzzle that was devised by an American architect in 1979. The most popular version of the game consists of filling in the empty positions (i.e. non-given numbers) of a 9 x 9 grid (fig. 1) with numbers from 1 to 9 so that there are no duplicate numbers in any row, column or 3 x 3 sub-grid (bound by thick lines).[1] There are over 6 sextillion ($6x10^{21}$) possible sudoku grids that yield a unique solution, thus the game can be considered a large combinatorial optimization problem.[1,2] In this work, we discuss the development of a genetic algorithm (GA) that is capable of searching for the solution of a given sudoku puzzle. Our aim is to have an algorithm capable of solving puzzles of various degrees of difficulty in useful time. The content of this report is organized as follows: in the methodology section, we explain the implementation and components of the GA; in the results and discussion section we present the obtained results and explain our choices concerning the algorithm's parameters to achieve the desired outcomes; lastly, the main results are summed up in the conclusion section. Moreover, the reader can consult the source code to see the full implementation and all the obtained results.

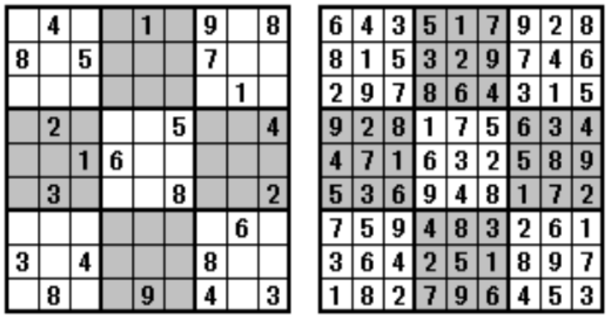


Fig. 1 - Sudoku puzzle with given positions (left) and its respective solution (right)

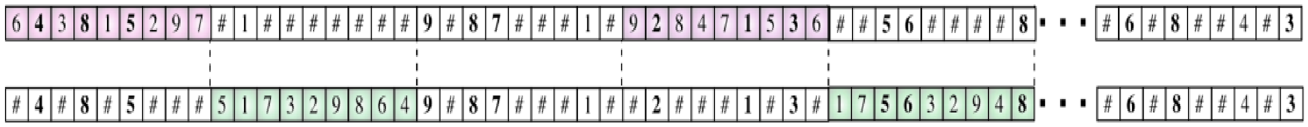


Fig. 2 - Representation of the sudoku grid in the GA implementation (each set of 9 digits represents a 3x3 sub-grid)

Methodology

We considered the following pseudo-code to adapt the GA for the sudoku problem:

```
1. Generate randomly a population P of n individuals (i.e. set of randomly filled sudoku puzzles)
2. Repeat until the predefined number of generations has been reached
    2.1. Calculate the fitness of each individual in P and save the best individual (if elitism == True)
    2.2. Create an empty population P'
    2.3. Repeat until population P' has n individuals
        2.3.1. Select two individuals from P using a selection algorithm
        2.3.3. Apply crossover to the selected individuals at 2.3.1. according to the crossover rate.
        Otherwise, individuals are reproduced (copied)
        2.3.4. Apply mutation to the obtained individuals at 2.3.3. according to the mutation rate.
        Otherwise, they are left unchanged
        2.3.5. Insert individuals into P'
    2.4. P' becomes the current population (P := P')
    2.5. Calculate the fitness of each individual of P' and print the fitness value of the best individual
    2.6. Replace worst individual with elite (if elitism == True)
3. Return the best individual in P
```

The GA implementation is divided into the following main components:

- **Encoding:** Conventional representation of the sudoku grid into an array of 81 digits, whereby each 3 x 3 sub-grid is merged in order from top to bottom and left to right (fig. 2). Non-given positions are encoded by zeros and their indexes are saved.[3]
- **Initialization:** A population of size *n* is created by the class Solver which fills in the non-given positions with randomly generated numbers from 1 to 9, hence the given positions are left untouched throughout the operations conducted by the evolution method (class function).
- **Fitness function:** The algorithm supports both minimization and maximization forms of optimization considering either the sum of non-unique numbers by row, column, and 3 x 3 sub-grid or the sum of unique numbers by row, column, and 3 x 3 sub-grid, respectively. Therefore, the global optimum is 243 for maximization problems or 0 for minimization problems. Moreover, an average variation of the same fitness functions was implemented, in this case the global optimum can be 81 or 0 respectively.
- **Selection:** Three selection methods with variations for the chosen fitness function were applied - Fitness Proportionate Selection (FPS), Tournament Selection, and Rank Selection.
- **Crossover:** Three crossover methods were implemented - One Point Crossover, Two Point Crossover, and Fitness Crossover. In all methods, the crossover operations, that have randomly chosen crossover points on both parents, are only allowed between 3 x 3 sub-grids. Thus, all the given positions that must remain fixed are preserved. The Fitness Crossover method applies One Point Crossover continuously to a given pair of parents until the average fitness of the generated children is higher than the average fitness of the parents or a predefined number of iterations is reached, whichever comes first.
- **Mutation:** Six types of mutations were implemented - One Point Mutation, *N* Point Mutation, Swap Mutation, Inner Swap Mutation, *N* Swap Mutation, Random Mutation. All mutation methods are only allowed to change non-given positions to avoid generating invalid individuals.

  <u>One Point Mutation</u> - Selects randomly a non-given (not fixed) position and replaces the current digit with a random number from 1 to 9.

  <u>*N* Point Mutation</u> - Replaces *n* randomly chosen not fixed positions with random numbers from 1 to 9. There's no limit for the selected number of points since the choice is made with replacement, so it's possible for the same non-given position to be mutated more than one time.

  <u>Swap Mutation</u> - Two non-given positions are randomly chosen and the current digits are swapped.

  <u>Inner Swap Mutation</u> - One sub-grid of an individual is randomly chosen and two non-fixed positions of the sub-grid are randomly selected and their respective digits swapped. Therefore, this restricted form of swap is only allowed within a sub-grid.

  <u>*N* Swap Mutation</u> - Swaps the digits of *n* randomly chosen non-given positions. Like the *N* Point Mutation, there's also no limit for the selected number of points because the choice is made with replacement.

  <u>Random Mutation</u> - This variation applies randomly either a Swap or Inner Swap Mutation to a given individual.
- **Statistical Analysis:** The fitness values of the best individuals of each generation were exported to .csv files, which are available in the project's repository, and were used to plot the corresponding fitness landscapes showing the aggregated results of 10 samples with a 95% confidence interval. Although more samples should be used for each experiment to draw more accurate results, we opted for 10 samples to reduce the computational time. Also, due to time constraints, we only attempted to solve one "easy" and one "very hard" puzzle instead of using several examples of comparable difficulty.

Results and discussion

Initially we had chosen an “easy” and a “very hard” (i.e. evil) puzzle from sudoku.com (fig. 3 - appendices), our approach consisted in finding the best selection, crossover and mutation methods for the “easy” puzzle and then, attempting to solve the “very hard” one. This way, if the algorithm succeeds with these extreme cases, it can be used for all puzzles. Furthermore, it should be noted that the puzzle’s degree of difficulty is related with the number of given digits (i.e. higher number of combinations to search when fewer numbers are given) and their placement.

Selection method comparison

Throughout the selection method comparison experiments, we used a population of 100 individuals, One Point Mutation (0.9 rate), and Two Point Crossover (0.9 rate). High rates for both genetic operators were found to be effective. Since it’s equivalent to sum the non-unique or unique numbers by row, column, and 3 x 3 sub-grid, we preferred to treat the sudoku puzzle as a minimization problem, considering the minimization of the sum of non-unique numbers. The results of fig. 4 (appendices) show that Tournament selection (tournament size of 10) is the best algorithm. Also, we note that Tournament selection with elitism does not considerably improve fitness.

Mutation method comparison

Using the previous parameters with Tournament selection, we proceeded to test the remaining mutations. As can be seen in fig. 5 (appendices), Inner Swap Mutation gives the worst results, leading the algorithm to premature convergence, while Random Mutation provides the lowest fitness values. In-between, we have all the other mutations. It seems that Two Swap and Three Point mutations are worse than their single counterparts because they tend to destroy good solutions.

Crossover method comparison

Using the parameters of the selection method comparison, we tested the remaining crossover methods. In this combination of parameters, Fitness Crossover (tolerance of 10 iterations) appears to be the best algorithm. However, One Point Crossover and Two Point Crossover are very close in terms of fitness results (fig. 6 - appendices).

Solving easy puzzle

Considering the obtained comparison results of the mutation and crossover algorithms, we decided to run experiments with a population size of 1000 individuals, reducing the tournament size to 4 (i.e. increase selective pressure over the individuals), and use all crossover algorithms with Swap Mutation (fig. 7 - appendices). One Point Crossover was the algorithm with the highest success rate (70 %) and Fitness Crossover was the worst algorithm (30 %). Additionally, we have decided to test One Point Crossover with Random Mutation, which achieved a 60 % success rate. Therefore, the best combination of One Point Crossover with Swap Mutation was chosen to tackle the “very hard” puzzle. These results suggest that Fitness Crossover, which imposes a restriction upon the GA search, reduces the number of local neighbors, thereby reducing the necessary diversity (i.e. accepting worse children) of the population to reach the global optimum.

Solving very hard puzzle

For this puzzle, we have increased the population size to 2000 individuals, reduced the tournament size even further to 2 individuals. We began by running the algorithm for 200 generations which reached 16 as the best fitness value. Then, to improve the algorithm, the fitness function was changed to the average sum of non-unique numbers by row, column, and 3 x 3 sub-grid. However, there was no noticeable improvement of fitness (5 as the best fitness value), both fitness functions yield similar results. Lastly, we have increased the number of generations to 500 using the average fitness function (fig. 8 - appendices). The best fitness value obtained was 2, corresponding to 4 repeated numbers (fig. 3 - appendices) within their respective columns. It's clear that the algorithm struggles to find the global optimum when the search space is significantly increased (i.e. from 40 to 25 given positions). At this point, we didn't decide to increase the population size or the number of generations because our objective was to have an algorithm that can solve the puzzle in useful time. Therefore, we feel that efforts should be devoted to implementing GAs that can go through the search space more intelligently and avoid being stuck in a local optimum. To that effect we would like to suggest the implementation of other fitness functions that combine both the sum of non-unique and unique numbers by row, column, and 3 x 3 sub-grid. For instance, the minimization of the following expression: *1 + sum (# non-unique numbers) / sum (# unique numbers)*.

Conclusions

Considering the limitations of the GA, we showed that our implementation should be capable of solving easy puzzles under 100 generations with populations of only 1000 individuals. The One Point Crossover (0.9 rate), Swap Mutation (0.9 rate), and Tournament Selection combination was found to be the best in terms of success rate (70%). However, when trying to solve the most difficult puzzles, the algorithm gets stuck in a local optimum, affording solutions with 4 repeated numbers after 500 generations with populations of 2000 individuals. Lastly, we have suggested the implementation of other fitness functions that might be able improve the ability of our algorithm to reach the desired goal.

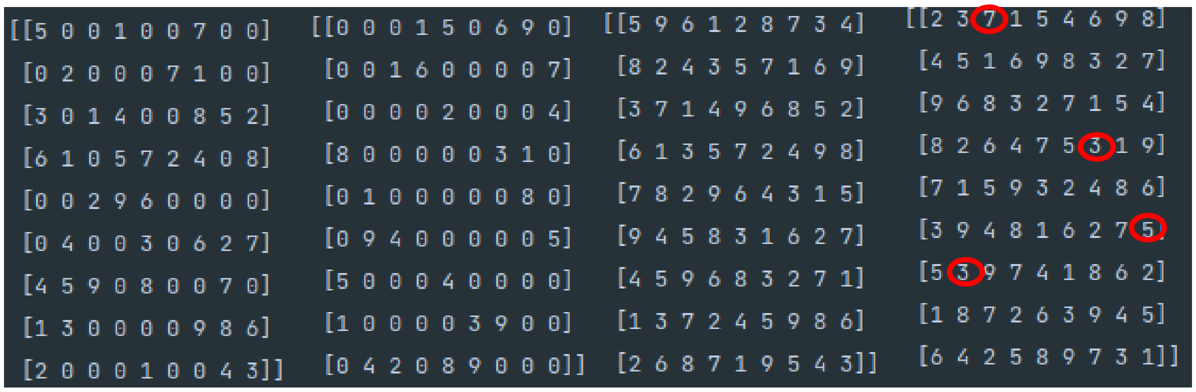
References

[1] Solving Sudoku with genetic operations that preserve building blocks (Sato Y. & Inoue H., 2010) Link: https://ieeexplore.ieee.org/document/5593375

[2] Will we ever run out of sudoku puzzles? (Conroy T.) Link: https://www.britannica.com/story/will-we-ever-run-out-of-sudoku-puzzles

[3] Solving, rating and generating Sudoku puzzles with GA (Mantere, T. & Koljonen, J., 2007) Link: https://ieeexplore.ieee.org/document/4424632

Appendices



```
[[5 0 0 1 0 0 7 0 0]   [[0 0 0 1 5 0 6 9 0]   [[5 9 6 1 2 8 7 3 4]   [[2 3 7 1 5 4 6 9 8]
 [0 2 0 0 0 7 1 0 0]    [0 0 1 6 0 0 0 0 7]    [8 2 4 3 5 7 1 6 9]    [4 5 1 6 9 8 3 2 7]
 [3 0 1 4 0 0 8 5 2]    [0 0 0 0 2 0 0 0 4]    [3 7 1 4 9 6 8 5 2]    [9 6 8 3 2 7 1 5 4]
 [6 1 0 5 7 2 4 0 8]    [8 0 0 0 0 0 3 1 0]    [6 1 3 5 7 2 4 9 8]    [8 2 6 4 7 5 3 1 9]
 [0 0 2 9 6 0 0 0 0]    [0 1 0 0 0 0 0 8 0]    [7 8 2 9 6 4 3 1 5]    [7 1 5 9 3 2 4 8 6]
 [0 4 0 0 3 0 6 2 7]    [0 9 4 0 0 0 0 0 5]    [9 4 5 8 3 1 6 2 7]    [3 9 4 8 1 6 2 7 5]
 [4 5 9 0 8 0 0 7 0]    [5 0 0 0 4 0 0 0 0]    [4 5 9 6 8 3 2 7 1]    [5 3 9 7 4 1 8 6 2]
 [1 3 0 0 0 0 9 8 6]    [1 0 0 0 0 3 9 0 0]    [1 3 7 2 4 5 9 8 6]    [1 8 7 2 6 3 9 4 5]
 [2 0 0 0 1 0 0 4 3]]   [0 4 2 0 8 9 0 0 0]]   [2 6 8 7 1 9 5 4 3]]   [6 4 2 5 8 9 7 3 1]]
```

Fig. 3 - Initial "easy" and "very hard" puzzles encoding (left) and the respective output solution (right)

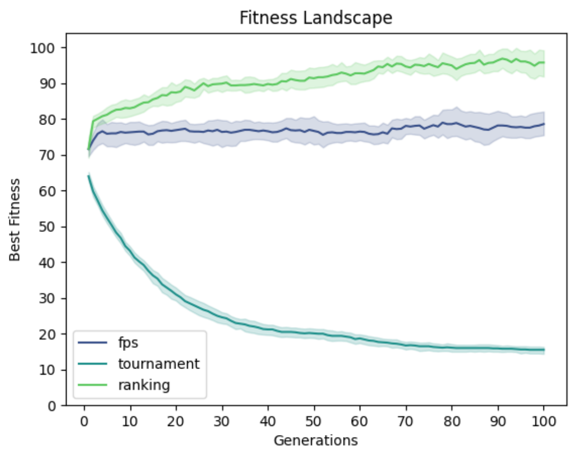


Fig. 4 - Selection method comparison

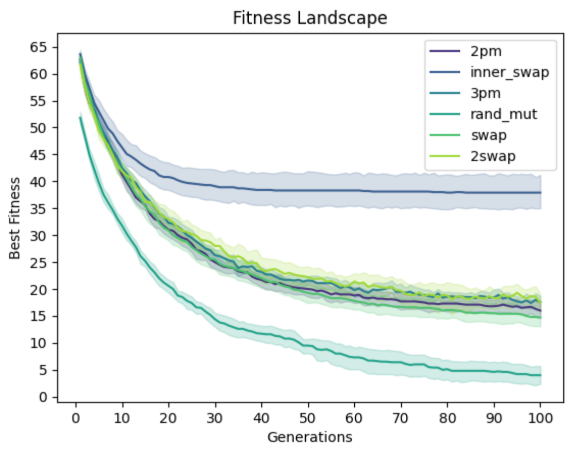


Fig. 5 - Mutation method comparison

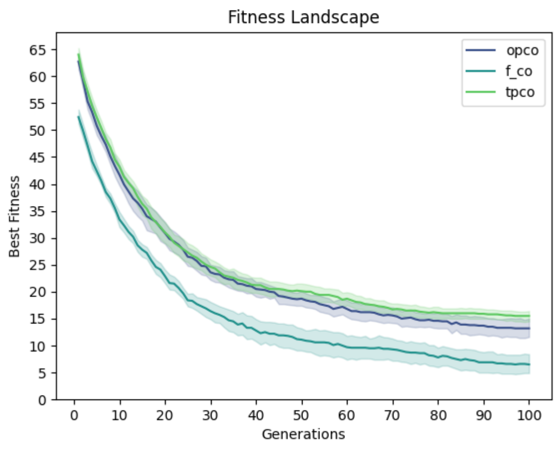


Fig. 6 - Crossover method comparison

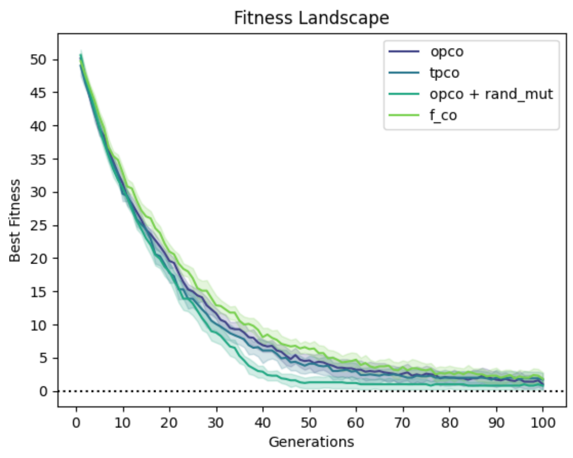


Fig. 7 – “Easy” puzzle solving combination comparison

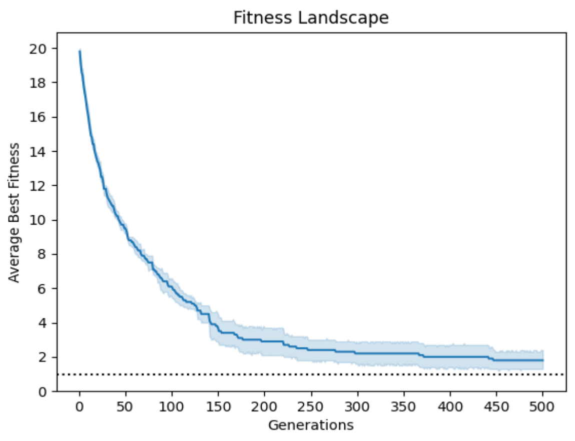


Fig. 8 – “Very hard” puzzle solving attempt