Group 6 Assignment Report

An Evolutionary Algorithm for solving Sudokus

Ben Ertugrul Eduard Iorga Haoran Liu
Moritz Schlitter

October 3, 2025

1 Introduction

The recommended software for typesetting assignment reports is LaTeX. It will allow you to prepare high-quality documents, especially in the area of Computer Science. This document can serve as a template for reports. Each section begins with brief instructions in red text. All the instructions in red, as well as the dummy text, should be removed in the final version to submit. The LaTeX source of this file includes examples of using the most needed commands and environments. You can find plenty of other examples with explanations in many web forums and discussion groups on the Internet. The easiest way to edit your report is to use `https://www.overleaf.com/`. Overleaf does not require any setup on your computer, and it is free to create an account.

The book *Writing for Computer Science* [**?**] is a useful assistance on how to write properly and present your work when it comes to Computer Science topics. It is a strong recommendation to follow its guidelines and limit the usage of AI tools to generate text. Keep in mind that the examiner is an expert in Evolutionary Computation and therefore, any false information generated by an AI tool is easily notable. Such case may lead to failing the assignment.

The introduction should briefly introduce the assignment and its purpose.

—

- Context - Introduce Sudoku and Genetic algorithm very shortly

- This paper studies the efficiency of solving Sudoku puzzles with evolutionary algorithm approaches

- Different difficulties for 9x9, increased sizes

- Additionally, we want to test whether increasing the size of the Sudoku board leads to better performance of evolutionary algorithms compared to naive search algorithms like DFS.

- The performance is compared between evolutionary algorithms of different complexities as well as to Deapth-First Search - a naive optimization algorithm

- The objective is to draw a conclusion on which approach solves Sudoku more efficiently, both for regular 9x9 Sudoku boards as well as for bigger boards (16x16 and 25x25).

- The document is structured as follows: Section 1: Introduction and goal of the paper. Section 2: Describes the problem the paper solves in detail (Solving Sudoku) + motivation of the evolutionary approach used. Section 3: Describes the algorithm in detail. Section 4: Describes the experiments run with the algorithm. Section 5: Presents and analyses the results. Section 6: Conclusion.

Traditionally, a Sudoku is a logic based number placement puzzle. The objective of the puzzle is to fill a most commonly 9x9 sized grid with digits so that each column, row and 3x3 subgrid contain all of the digits from 1 to 9.[**?**]

Solving such a puzzle programmatically falls into the category of search and optimization problems. These types of problems can be approached in different ways. One possible approach is depicted by evolutionary algorithms or in this case more specifically, genetic algorithms (GA).[**?**]

Inspired by the process of natural selection, GAs use biologically inspired operators such as selection, crossover and mutation to generate solutions to optimization and search problems.

This paper studies the efficiency of solving Sudoku puzzles with GA approaches. The objective is to compare the performance of different implementations of GAs. These approaches will additionally be compared to the naive search algorithm deapth-first search (DFS). The puzzles explored will be of different difficulties for regular 9x9 grids and bigger 16x16 and 25x25 grids. This way the paper analyzes whether GAs are a viable option for efficiently solving Sudoku boards compared to naive search algorithms as the complexity of the puzzle increases.

This document is organized in the following way. Chapter one describes the objective of this study. Chapter two describes the problem of solving Sudoku puzzles and the motivation of using evolutionary algorithm approaches to solve this type of problem. Chapter three details the algorithm used in this study. The following chapter lays out the setup of the experiments run in this study. Lastly, chapter five present and analyses the results of this paper.

2 Sudoku Puzzle Problem

2.1 Problem Description

Sudoku is a Japanese logical game that is played on a 9×9 grid, that are further divided into 3×3 subgrids. The objective of the game is to fill the grid with digits from 1 to 9, ensuring that each row, column, and subgrid contains each digit exactly once[1]. The puzzle starts with some cells already filled in, and the player must use logic and respect the base rule of the game in order to finish the puzzle.

Sudoku puzzles can vary in difficulty based on how many numbers they start with, the arrangement of these numbers and even the varying size of the sudoku, since they can go beyond the standard 9×9 grid, similar to the 25×25 grid we used to test our algorithm with. Thus we can think of the sudoku problem as a graph coloring problem if it were to be expressed in a mathematical context. The 9×9 grid can be seen as graph that has 81 vertices, namely one vertex for each cell. Each vertex can be labeled with an ordered pair (x,y), where x and y are integers between 1 and 9 [2]. Two distinct vertices (x1,y1) and (x2,y2) are connected by an edge if and only if they are in the same row, column or subgrid:

- $x1 = x2$, which translates as same row
- $y1 = y2$, which translates as same column
- $\lfloor (x1-1)/3 \rfloor = \lfloor (x2-1)/3 \rfloor$ and $\lfloor (y1-1)/3 \rfloor = \lfloor (y2-1)/3 \rfloor$ which translates as same subgrid

As previously mentioned, the puzzle is completed when all the vertices have an integer between 1 and 9 assigned to them, in such a way that vertices that are joined by an edge don't have the same number assigned to them.

2.2 Evolutionary Approach

While Sudoku has a deterministic solution, evolutionary algorithms can be used in order to compare their performance with more traditional approaches such as the depth first search algorithm. As it is mentioned in[3], they managed to implement an evolutionary algorith that could solve very efficently easy Sudoku puzzles. The downside of their algorithm, was that it wasn't as efficient when it came down to solving medium or hard puzzles.

- The mathematical formulation considered in your study. Some problems have a clear mathematical model (e.g., Travelling Salesman Problem), while others do not (e.g., n-Queens). Based on the problem you chose, search the literature and find a proper way to present the problem.
- One paragraph that briefly presents at least 3 published academic works where any evolutionary approach is used to solve the problem. It would be wise to cite here works that influenced your algorithm. This practice saves you time from looking for additional academic resources. You can find more information about reading and searching in the literature in [**?**].
- The motivation behind the evolutionary approach you decided to develop. A good practice would be to align the motivation with some literature gap found in the academic works you presented above. However, this is not mandatory. You can motivate your selection on the characteristics of the algorithm making it proper for the problem.

Note: Change the section's title to match the name of the problem you chose for your assignment.

—

- This section presents the problem this paper solves with the use of evolutionary algorithm approach.

- Math in solving Sudokus? Maybe rather introducing quickly a sudoku, how to solve it and describing the steep increase of difficulty with increasing sizes of sudoku puzzles

- Cite 3 published papers. Cite where our idea is from[1]. Another Paper[4].

- Motivation behind evolutionary approach.

3 Genetic Algorithm

The third section should present the evolutionary approach you developed. You can divide this section into subsection. In any case, you should mention the following details:

Evolutionary approach. Clearly describe the algorithm you developed. You should clearly explain the evolutionary operators you used and what modifications you did to match the problem. It is extremely important to present also a pseudocode of your algorithm. An example is given in 1, below. For more insight into presentation of algorithms, you can advise [?].

To typeset pseudocode in LaTeX you can use one of the following options:

- Choose ONE of the (`algpseudocode` OR `algcompatible` OR `algorithmic`) packages to typeset algorithm bodies, and the algorithm package for captioning the algorithm.
- The `algorithm2e` package.

You can find more information here: `https://www.overleaf.com/learn/latex/Algorithms`

Algorithm 1 Example of an algorithm's pseudocode

```
Require: $n \geq 0$
Ensure: $y = x^n$
  $y \leftarrow 1$
  $X \leftarrow x$
  $N \leftarrow n$
  while $N \neq 0$ do
      if $N$ is even then
          $X \leftarrow X \times X$
          $N \leftarrow \frac{N}{2}$                ▷ This is a comment
      else if $N$ is odd then
          $y \leftarrow y \times X$
          $N \leftarrow N - 1$
      end if
  end while
```

Solution representation. Clearly describe the solution representation you used. You can use figures to improve the comprehensibility of this part.

Fitness function. It is also very important to mention the fitness function you used. In many cases, the objective function of the problem is not the same as the fitness function used in an evolutionary algorithm. An example, following the principles of [?], is given below.

$$F = \sum_{i=1}^{d} x_i^2 \tag{1}$$

where x_i is the i-th gene (i.e., decision variable) in the solution and d corresponds to the number of decision variables in the problem.

Note: Change the section's title to match the name of the algorithm you developed for your assignment.

3.1 Implementations

This section presents and compares different implementations of an evolutionary algorithm designed to solve Sudoku puzzles. Each implementation varies in its approach to selection, crossover, mutation strategies, and fitness evaluation.

The following sections document these processes and use the Sudoku board shown in 1 as a reference for visualizing the selection and mutation step.

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

Figure 1: Initial Sudoku puzzle

3.1.1 Implementation 1

Table 2 gives a brief overview of how the algorithm works. A detailed description of how the fitness function, selection, and mutation work, will be given afterwards.

Algorithm 2 Genetic Algorithm 1

```
Require: initial_sudoku, population_size, mutation_rate, max_generations
Ensure: /
  population ← PopulateRandomly(initial_sudoku, population_size)
  generation ← 0
  next_generation ← ∅
  while generation < max_generations do
      population ← sort_by_fitness(population, descending)
      if first(population).fitness = $max\_fitness$ then
          return first(population)
      end if
      top ← first 20% of Population
      for $i \leftarrow 1, population\_size * 0.8$ do
          parent1, parent2 ← sample(top)
          child ← crossover(parent1, parent2)        ▷ per-cell pick random parent
          child ← mutate(child, mutation_rate, initial_sudoku)        ▷ change
  $mutation\_rate$ amount of random cells
          next_generation ← next_generation + child
      end for
      for $i \leftarrow 1, population\_size * 0.2$ do
          parent1, parent2 ← sample(population)
          child ← crossover(parent1, parent2)        ▷ per-cell pick random parent
          child ← mutate(child, mutation_rate, initial_sudoku)        ▷ change
  $mutation\_rate$ amount of random cells
          next_generation ← next_generation + child
      end for
      generation ← generation + 1
      population ← next_generation
  end while
  population ← sort_by_fitness(population, descending)
  return first(population)
```

Fitness function The fitness function evaluates a Sudoku board by summing the number of unique digits in each row, column, and 3×3 block, with a maximum score of 243 for a valid solution.

Selection In the figure below, two parents (left and middle) of the current generation form an individual of the next generation (right). One area for improvement is evident in row 7: Parent 1 contains two occurrences of the digit 7 and Parent 2 contains three. The offspring also has three 7s, while it would have been possible to generate a row with all distinct digits, namely **286371945**.

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

$+$

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

$=$

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

Figure 2: Exemplary selection step according to Implementation 1

Mutation The 'Exemplary mutation step according to Implementation 1' on page 9 illustrates a mutation step with mutation rate $n = 2$; however, only a single tile (row 7, column 7) was actually modified. The other candidate position coincided with a given cell and therefore could not be changed. This behaviour highlights a weakness of the implementation: mutations can be wasted when selected positions are immutable, and a single mutation may decrease fitness by introducing additional conflicts (in this example the row contains three 7s, the column contains one 7 and the corresponding 3×3 block contains two 7s), which lowers the individual's score.

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

$\xrightarrow{\text{mutation}}$

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

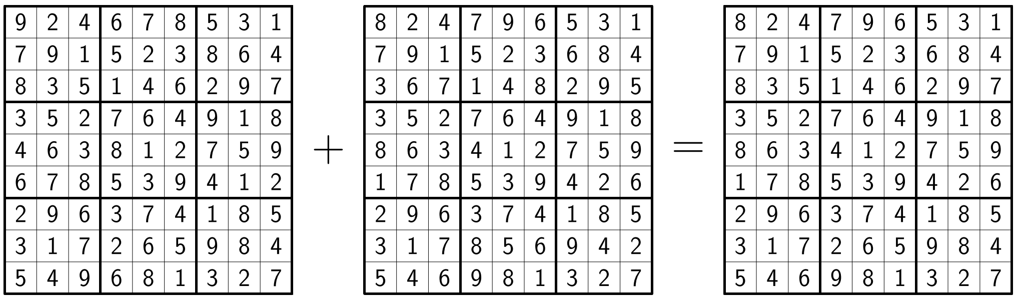
Figure 3: Exemplary mutation step according to Implementation 1

3.1.2 Implementation 2

1. Initialize the population with ***population_size*** number of random individuals that all respect the initially given sudoku tiles. The individuals create a complete board by iterating over the rows and filling in the empty tiles with random numbers that do not already exist in the respective row. The fitness function works counterintuitively, calculating the fitness as the sum of the number of duplicates in each column, and 3x3 subgrid. The minimum fitness is 0 (no duplicates).
2. The next generation starts with the top ***elites_size*** number of individuals from the current population. Then, a tournament selection is performed to select parents for the remaining individuals in the next generation. In each tournament, ***K*** individuals are randomly chosen from the population, and the one with the best fitness is selected as a parent. This selection process is repeated until we have ***population_size*** number of parents.
3. The crossover step is done by iterating over all parents in pairs, creating two children from each pair. For each row in the sudoku board, the child inherits that row either from the first or the second parent. This ensures that the children will still respect the uniqueness of numbers in each row.
4. For mutation, we iterate all rows of an individual and swap two random tiles that both haven't been given in the initial sudoku. We do this with the probability of ***mutation_rate*** for each row, once again ensuring the uniqueness of numbers in each row.
5. Once the new generation is created, we calculate the fitness of each individual and repeat the process from step 2. until a valid solution is found.

Selection In contrast to the selection step from the former implementation, this method doesn't introduce more duplicate numbers per row. Figure 4 shows an example where the very first row is taken from the first parent (left), while row 8 is copied from the second parent (center). Duplicate numbers can now only appear in a column or a 3×3 block, which is an improvement compared to Implementation 1.

Mutation The mutation of the individual that resulted from the previous selection is shown in the following figure 5. For this example the mutation rate was set to 20%, leading to mutations in row 1 and row 5.

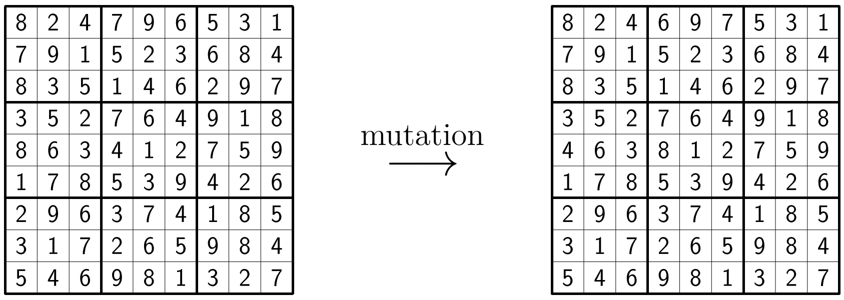


9	2	4	6	7	8	5	3	1
7	9	1	5	2	3	8	6	4
8	3	5	1	4	6	2	9	7
3	5	2	7	6	4	9	1	8
4	6	3	8	1	2	7	5	9
6	7	8	5	3	9	4	1	2
2	9	6	3	7	4	1	8	5
3	1	7	2	6	5	9	8	4
5	4	9	6	8	1	3	2	7

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

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

Figure 4: Exemplary selection step according to Implementation 2



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

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

Figure 5: Exemplary mutation step according to Implementation 2

3.2 Solution representation

The solution representation is an $n \times n$ matrix where each entry is a number in the range from 1 to n. The solution is valid if and only if every number in a row, a column, and a respective block is unique.

4 Experimental part

This section describes the setup of experiments [?]:

- Provide the details of the hardware and software that you used.
- Describe the steps you carried out during your experiments.
- Detail the data you used for the evaluation of your algorithm.

Hardware: Macbook pro M1 max Software: Python (VS Code) —- we have algorithm now we want hyperparamter -> bayesian optimization (paper) give us best population and mutation rate with these hp we run the algrothm easy, medium, hard and 16x16 and 25x25 we track execution time, number of generations to solution (or max of 10.000 gens with best fitness, no gen without improvement) Run default DFS and track time and solution

—- random boards for data using DFS we generate a solved board for the desired size randomly remove number and try to solve again. repeat, if we cannot solve it we restart 9x9 remove until difficulty is met (easy: range of 40.50, med: 30-39, hard 25-29) (cite something?) 16x16: from the internet? 25x25: from internet? ask Eddie (where from) big ones taken because generating them takes a lot of time, only test with one board because of complexity

—————————-

This section describes the setup of the experiments run to study the efficiency of solving Sudoku puzzles with GA approaches.

The test will be run on a Macbook pro M1 max.

4.1 Choosing hyperparameters

Chapter 3 discusses the algorithm used for the experiments in detail. However, before the algorithm can be run, the hyperparameters for ***population_size***, ***mutation_rate*** and ***elite_size*** need to be chosen. The solution quality of a stochastic algorithm strongly depends on choosing the correct hyperparameters. Previous studies show that configuring hyperparameters of GAs using Bayesian Optimization leads to significantly better results than choosing them at random while keeping the computational time low[5].

The Bayesian Optimization evaluates the optimal hyperparamters of the GA as follows...

4.2 Running the experiment

Next the GA is run 1000 times for a maximum of 10.000 generations for Sudoku puzzles of easy, medium and hard difficulty of 9x9 boards. Because of the drastic jump in complexity with 16x16 and 25x25 boards, the GA is only run ? times for these puzzles. For the evaluation of the performance the following properties of the experiment are tracked:

1. Number of successfully solved boards
2. Execution time
3. Number of average generations
4. Average execution time per Sudoku puzzle
5. Average generations per second

For failed runs the following properties are tracked:

1. Number of generations stuck at local minimum
2. Best fitness achieved
3. Average violations
4. Generations without improvement
5. Average generations without improvement

To compare the performance of the GA, a DFS algorithm is run on Sudoku boards of the same complexity. For the evaluation of the performance the following properties of the experiment are tracked:

1. Number of successfully solved boards
2. Execution time
3. Average execution time per Sudoku puzzle

4.3 Sudoku boards used in the experiment

Because of the high number of Sudoku boards needed to accurately evaluate the performance of the stochastic algorithm, the Sudoku grids are generated randomly as part of the experiment. The boards are generated in the following way. First, a solved board is generated using the DFS algorithm. Next, a random number is removed from the grid. The DFS algorithm tries to solve the puzzle again. If it is still solvable, repeat the previous steps until the requirement of given numbers is met for the different complexities of the Sudoku boards. For 9x9 grids, the easy difficulty gives between 40-50 numbers. Medium and hard puzzles have 30-39 and 25-29 givens respectively. (source required?)

Because of the increased complexity, run time and reduced success rate of solving 16x16 and 25x25 puzzles, only ? boards were taken from...

Problem with Implementation 1 This implementation leads to a situation where the diversity of the population decreases rapidly, resulting in premature convergence to suboptimal solutions. On most of the runs, the algorithm creates more than 100,000 generations before finding a valid solution. To counter this problem, we have tried to increase the mutation rate and the population size, as well as introducing new random individuals in each generation. However, these adjustments only led to marginal improvements in performance.

Problem with Implementation 2 TODO

5 Results and Analysis

This section should present the obtained results and provide an insightful analysis of them. You can present the results using graphs, tables, or any other visualization method suits your purpose. Do not forget to include proper captions [?] in any of these illustration methods you use. You do not need to provide any execution details as they are already presented in Sec. 4.

A good practise would be to compare your algorithm with a simpler approach, such as (a) a naive method, (b) a Hill Climbing approach, or (c) a simple evolutionary algorithm. In the third case, you can use the simpler version of the algorithm you developed, i.e., the original algorithm without your modifications. In that case, you should briefly describe the comparing method(s) in Sec. 4. Alternatively, you can use some reference results derived from the repositories you found some benchmark instances.

To display tables, the `booktabs` package might be useful. For example, Table 1 shows how you should increase the size of n, when running your code. You can advice [?] to see a few examples of proper tables.

Table 1: Example of comparison the developed algorithm's results with the best ones from a repository.

Instance	**Optimum (Repository xyz)**	**EA**	**time (s)**
st70	678.597	677.109	0.67
ei176	545.387	544.369	1.16
kroA100	21285.443	21285.443	1.69
rd100	7910.396	7910.396	2.14
Pr136	96772	96770.924	7.11
Pr144	58537	58535.221	7.97
a280	2856.769	2856.769	33.47

You can use different illustration methods to present different aspects of your analysis. Figure 6 gives an example using the `pgfplots` package.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

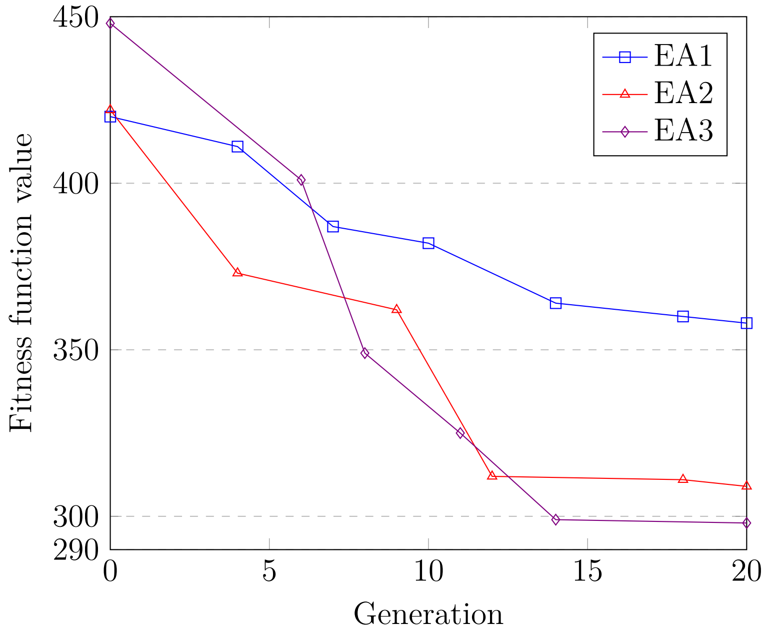


Figure 6: Example of convergence analysis.

6 Conclusions

In this section you should provide a concise summary of what has been done, the obtained results and some recommendations on how this study could be extended.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

As shown in [?], evolutionary algorithms can solve Sudoku puzzles efficiently.

References

[1] T. Mantere and J. Koljonen, "Solving, rating and generating sudoku puzzles with ga," *2007 IEEE Congress on Evolutionary Computation, CEC 2007*, pp. 1382–1389, 2007.

[2] "Mathematics - wikipedia." Accessed: 2025-10-02.

[3] "Evolutionary algorithms and sudoku." Accessed: 2025-10-02.

[4] P. R. Amil and T. Mantere, "Solving sudoku's by evolutionary algorithms with pre-processing," *Advances in Intelligent Systems and Computing*, vol. 837, pp. 3–15, 2019.

[5] C. Rüther and J. Rieck, "A bayesian optimization approach for tuning a genetic algorithm solving practical-oriented pickup and delivery problems,"