

Global and Multi Objective Optimisation

Genetic Algorithms with
Local Optima Handling
to Solve Sudoku Puzzles

Firas Gerges Germain Zouein Danielle Azar

[link to the article](#)

Edoardo Insaghi, UniTs, 2024

Goal of the Project

Write an evolutionary algorithm
able to solve Sudoku Puzzles
following the paper*

Evaluate how consistent the
algorithms is with respect to the
difficulty of the task at hand

* Genetic Algorithms with Local Optima Handling to Solve Sudoku Puzzles

Firas Gerges
Department of Computer Science
and Mathematics
Lebanese American University
Byblos, 1h401 2010, Lebanon
+961-76-029164
firas.gerges@lau.edu

Germain Zoueïn
Department of Computer Science
and Mathematics
Lebanese American University
Byblos, 1h401 2010, Lebanon
+961-76-700554
Germain.zoueïn@lau.edu

Danielle Azar
Department of Computer Science
and Mathematics
Lebanese American University
Byblos, 1h401 2010, Lebanon
+961-9-547254#2408
Danielle.azar@lau.edu.lb

ABSTRACT

Sudoku is a popular combinatorial number puzzle game and is widely spread on online blogs and in newspapers worldwide. However, the game is very complex in nature and solving it gives rise to an NP-Complete problem. In this paper, we introduce a heuristic to tackle the problem. The heuristic is a genetic algorithm with modified crossover and mutation operators. In addition, we present a new approach to prevent the genetic algorithm from getting stuck in local optima. We refer to this approach as the “purge approach”. We test our algorithm on different puzzles of different difficulty levels. Results show that our algorithm outperforms several existing methods.

CCS Concepts

•Mathematics of computing → Combinatorial optimization •
Theory of computation → Evolutionary algorithms •
Computing methodologies → Heuristic function construction •
Computing methodologies → Genetic algorithms •
Computing methodologies → Learning paradigms • Applied
computing → Enterprise computing

Keywords

Combinatorial Problems; Genetic Algorithm; NP-Complete Problem; Sudoku; Puzzles; Games.

1. INTRODUCTION

The Sudoku is a puzzle game based on a 9x9 grid in which a player inserts numbers from 1 to 9 in empty cells. The grid is divided into nine 3x3 sub-grid (each called a block). The game starts with a given Sudoku grid containing some filled cells (called clues). The player has to fill the rest of the cells in such a way that no row, column or sub-grid contains duplicates. The number of clues determines the difficulty level of the Sudoku instance. Figure 1 shows an example of a Sudoku puzzle

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
ICCAI 2018, March 12–14, 2018, Chengdu, China
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-6419-5/18/03...\$15.00
<https://doi.org/10.1145/3194452.3194463>

generated via www.websudoku.com.

Nowadays, Sudoku puzzles are widely spread as a single player logic game online and in newspapers. Solving this game can be seen as a combinatorial optimization problem [1]. Given its growing popularity of the problem, there have been many attempts to develop algorithms to solve it. In this paper, we present a genetic algorithm to solve Sudoku puzzles in an efficient way.

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

Figure 1. An example of an instance of a Sudoku puzzle.

2. LITERATURE REVIEW

Many papers tried to tackle the game through different techniques. In [1], Douglas presents a genetic algorithm to solve the game. The algorithm is tested on three problems. It solves two and fails to solve the harder one. Results show a big standard deviation in terms of generations needed to solve the problem. The work is an improvement trial over [2] where Mantere and Koljonen introduce another Genetic Algorithm which is tested on 10 difficulty ratings (including one "empty" Sudoku board). Results show that GA can always solve an empty puzzle with an average of 1400 iterations. Easy puzzles were all solved while higher difficulty levels had lower success. The number of iterations required also varied greatly, from 770 for the easiest puzzles to 25,000 for the hardest. The results also suggest that GA can be used to rate Sudoku puzzles, by checking the percentage of puzzles solved as well as the number of iterations used. Another genetic algorithm (GA) is presented in [3] where crossover is used to avoid collisions. In [4],

The Game of Sudoku

- Played on a 9x9 grid. The grid is further divided into 9 3x3 subgrids
- Fill the grid with numbers ranging from 1 to 9
- Cannot repeat the same number in the same row, column, or subgrid
- Each Sudoku puzzle consists of an incomplete matrix, with some fixed numbers that cannot be changed

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

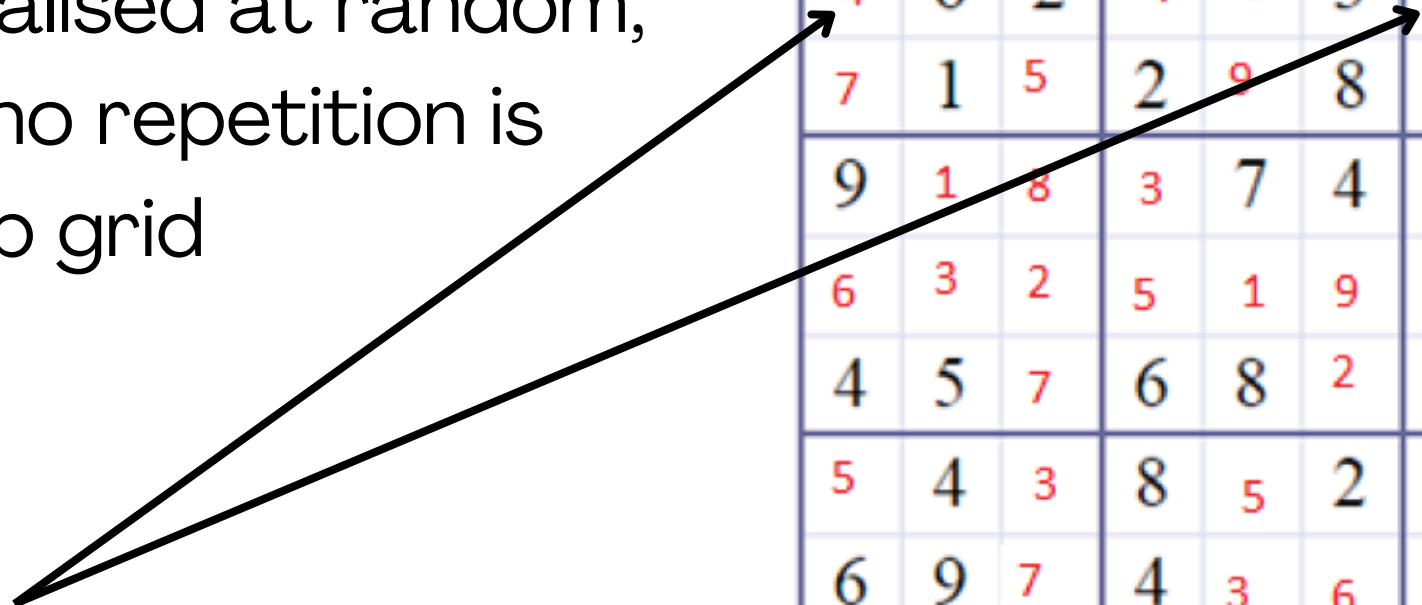
1	3	7	4	8	2	6	5	9
8	5	9	6	7	1	3	2	4
6	2	4	5	9	3	7	8	1
2	9	6	1	5	4	8	3	7
3	4	8	2	6	7	1	9	5
7	1	5	9	3	8	4	6	2
9	6	3	7	4	5	2	1	8
4	8	1	3	2	9	5	7	6
5	7	2	8	1	6	9	4	3

Chromosome and Initialisation

We can represent each candidate as a 9x9 matrix of integers, plus additional information on which numbers in the grid are fixed

Non fixed numbers are initialised at random, under the constraint that no repetition is allowed within each 3x3 sub grid

Repetition of the same number on the second row



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

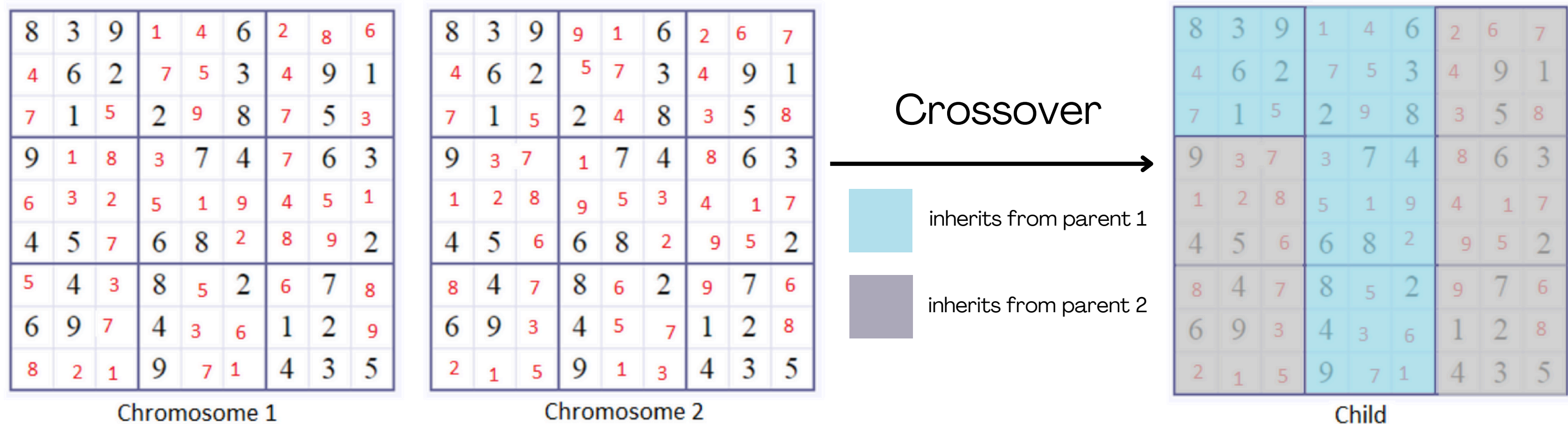
Fixed Values

Randomly
Initialised
Values

Mutation and Crossover

The Mutation Operator randomly swaps two non fixed numbers within the same 3x3 subgrid of a given chromosome

Crossover between two parent chromosomes results in a new individual that randomly inherits each 3x3 subgrid from one of the two parents with probability 0.5



Fitness Function and Selection

Given a candidate solution X , we define the Penalty as:

$$\text{Penalty} = \sum_{i=1}^9 \sum_{j=1}^8 \sum_{L=j+1}^9 (x_{ij} == x_{iL}) \longrightarrow \text{Number of repetitions in each row} \\ + \sum_{i=1}^8 \sum_{j=1}^9 \sum_{L=i+1}^9 (x_{ij} == x_{Lj}) \longrightarrow \text{Number of repetitions in each column}$$

There is no need to check for duplicates within each 3x3 subgrid by construction

The fitness function is then computed simply as **1 / (Penalty + 1)**

Selection of the parent chromosomes happens via **K-Tournament Selection**:

K individuals are chosen at random from the population, and the fittest is kept. This procedure is repeated for the second parent chromosome.

Elitism and the Purge Operator

The best individual of each generation in terms of fitness is always added to the next generation without going through mutation.

After q iterations of GA without improvement, each individual performs crossover with the best individual of the current generation. The resulting candidates will compose the next generation for the algorithm.

The authors justify this procedure empirically, stating that in their results it contributes to avoid local optima in the fitness function.

Pseudocode and Parameters

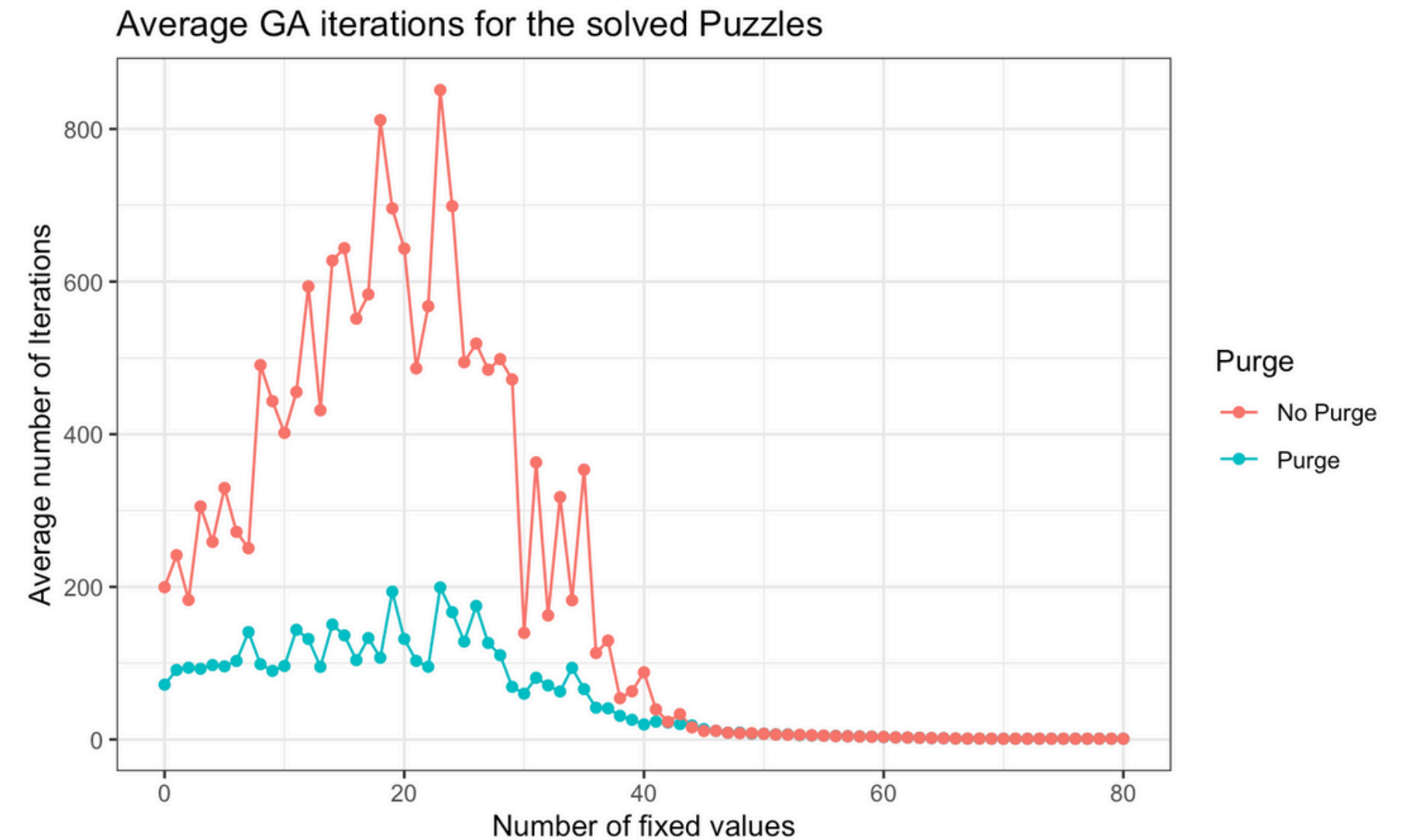
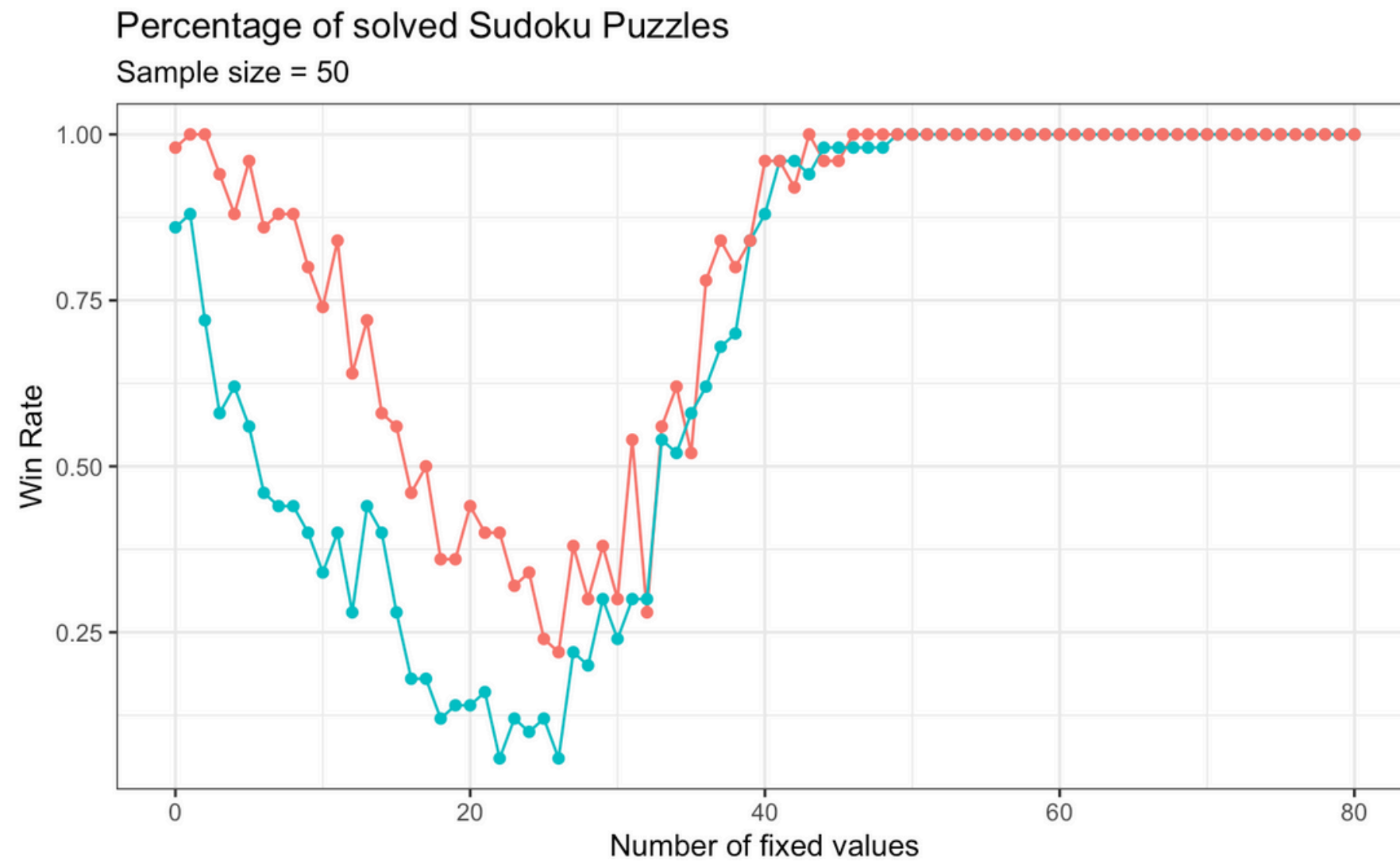
Sudoku-GA:

```
Population P = Initialize Random Population
Generation G = P
Generation newG= null
While Penalty(Best) != 0 and iteration <
max_iteration:
    While |newG| < |G| :
        Parent1 = K-tournament (G)
        Parent2 = K-tournament (G-{Parent1})
        Child = Crossover (Parent1, Parent2)
        Mutate(child,  $\mu$ )
        Mutate(Parent1,  $\mu'$ )
        Mutate(Parent2,  $\mu'$ )
        bestParent = Best (Parent1, Parent2)
        newG.add (bestParent)
        newG.add (child)
    newG.add (BEST_OF_G)
    G = newG
    Purge (G)
```

- Size of the population **n** = 700
- Number of individuals selected for K-tournament **K** = 7
- Child mutation probability μ = 0.05
- Parent mutation probability μ' = 0.5
- Number of iterations until Purge Operator is applied ϱ = 200
- Maximum number of GA iterations: 2000

Experiments and Results

The algorithm is tested on 50 randomly initialised Sudoku Puzzles,
with the number of fixed values ranging from 0 to 80



Experiments and Results

How does the population evolve
through time on average?

average fitness

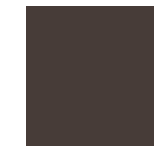
best fitness

Purge Operator

Number of Fixed Values

10 20 30 40

T

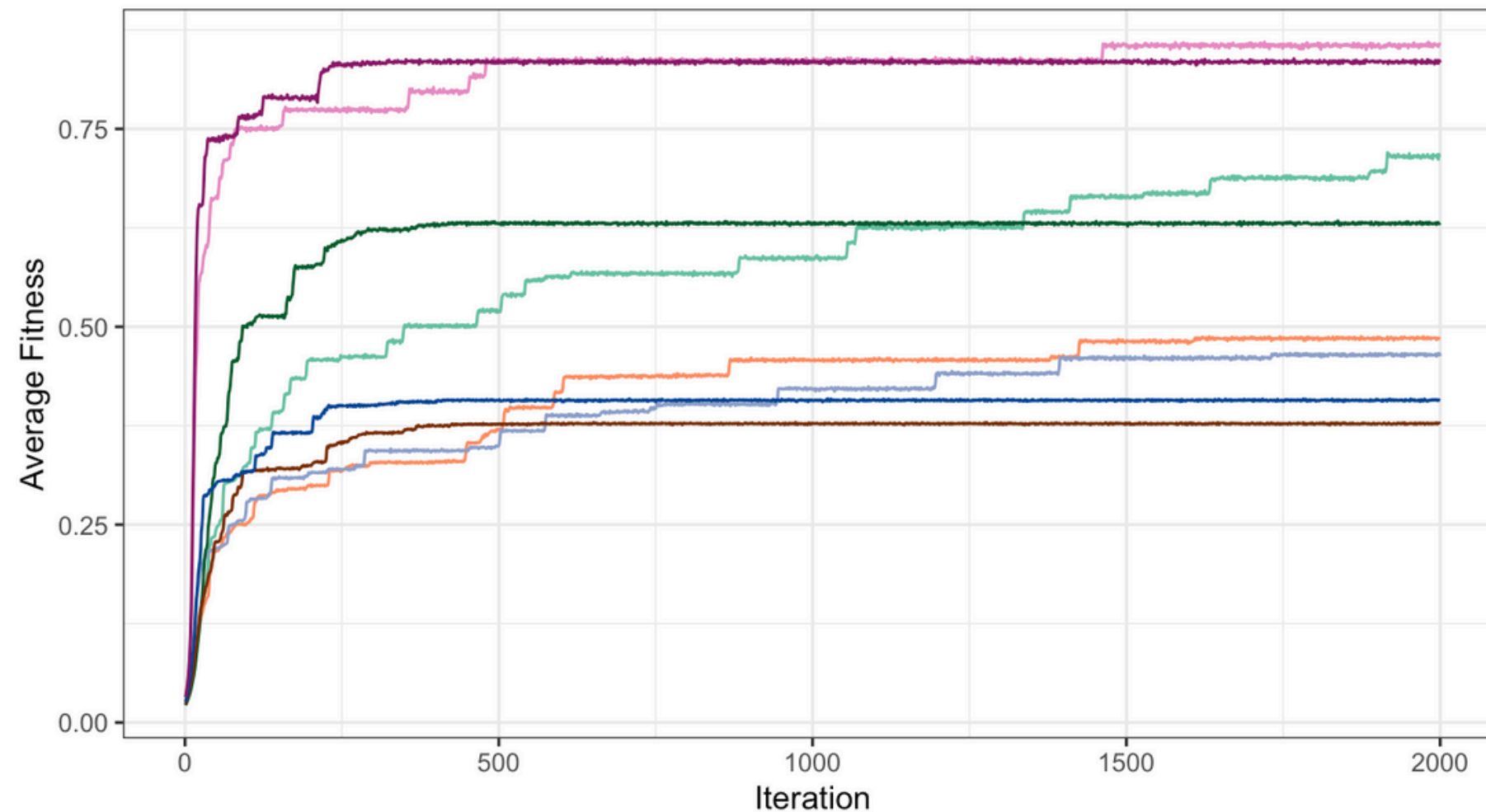


F



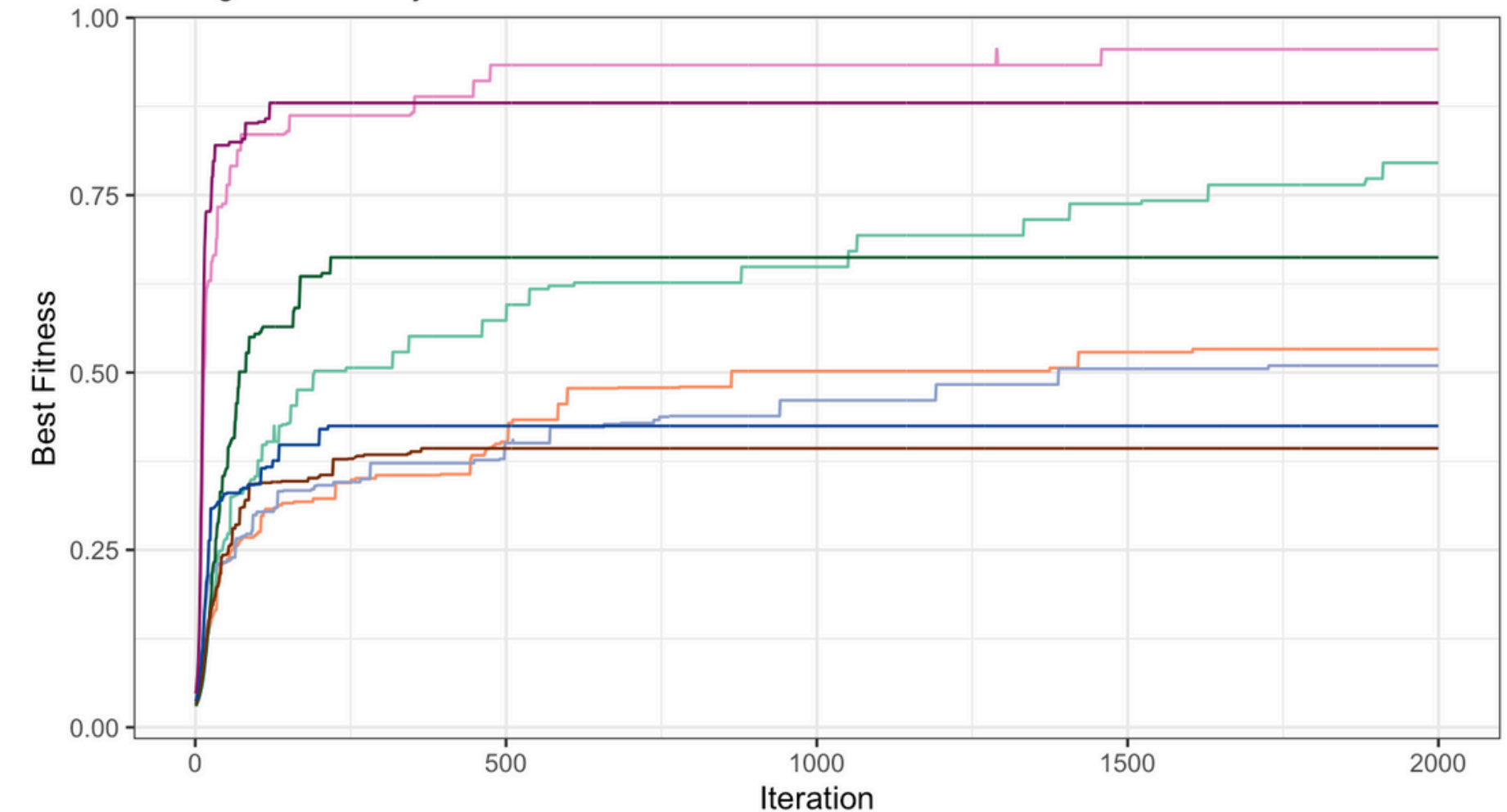
Average Fitness as a function of the GA steps

Average over 30 trajectories



Best Fitness as a function of the GA steps

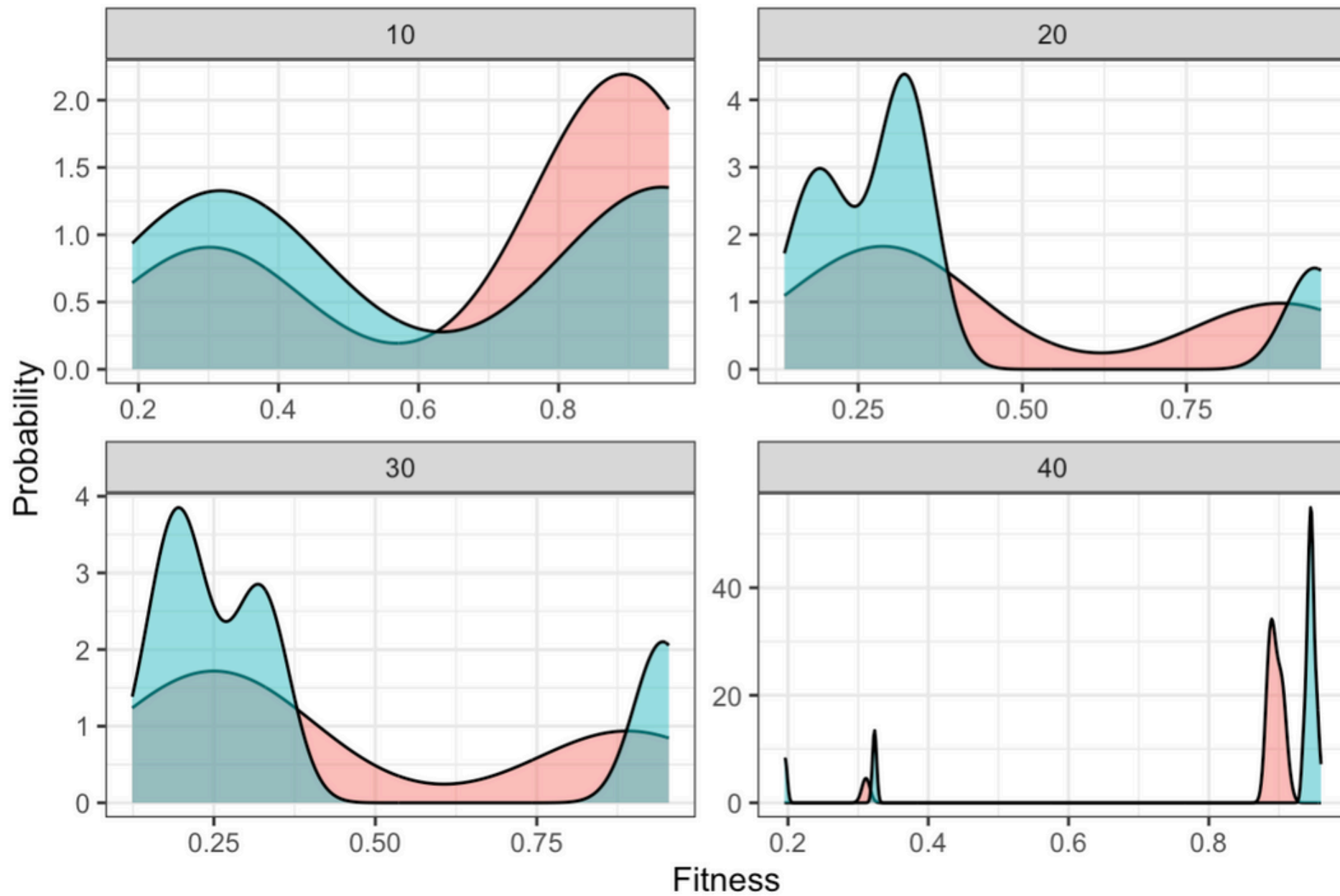
Average over 30 trajectories



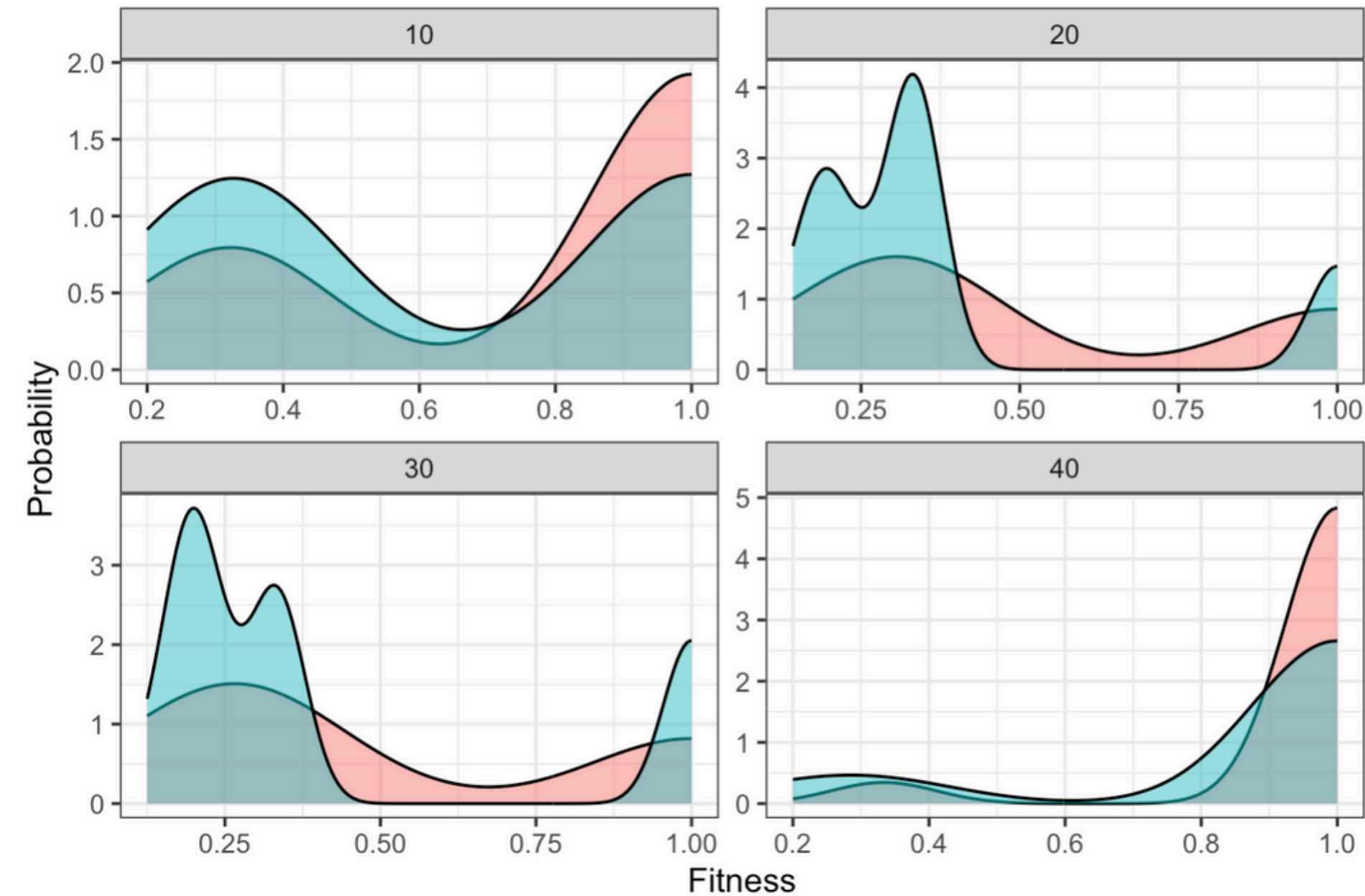
Experiments and Results

Distribution of the **Average** and **Maximum** fitness in the population at the end of GA

Distribution of Average fitness after 2000 GA steps



Distribution of Maximum fitness after 2000 GA steps



Thanks for the attention!