Sudoku Solver with Genetic Algorithm

Kostas Griska, Marija Grjazniha
M20200744, M20201061

May 28, 2021

Introduction

The project aims to solve Japanese logical game Sudoku using genetic algorithm. In classic Sudoku, the objective is to fill a 9×9 grid with digits so that each box, row, and column would include only unique values. Puzzle setter provides a partially completed grid for which only a single solution is possible. There are approximately 6.7 sextillion different puzzle combinations and only one correct solution.

Problem representation

Each puzzle is a nested list with 9 elements of size 9, where missing values are represented as zeros. Data used for the research consists of six types of Sudoku puzzles:

- Basic which is a "typical Sudoku puzzle" taken from Sudoku Wikipedia page.
- `very_easy`, `easy`, `moderate`, `hard`, and `very_hard` are puzzles of corresponding level of difficulty taken from 7sudoku website.

As mentioned in the beginning, the puzzle solution can contain only integers from 1 to 9 (included). Zero values representing missing fields, where non-zero values of a puzzle are passed to all Individuals and cannot be changed (e.g., mutated). Sudoku is solved if each row, column, and box contains non repetitive digits from 1 to 9.

Although puzzles are represented as nested lists, individuals, are represented as NumPy arrays, which makes it easy to manipulate on all levels: rows, columns, and boxes.

Fitness function

This project implements three fitness functions. All of them either aim to minimize the number of unique values or minimize the number of repetitions.

- Sum of number of unique values in each row, column, and box.
    - Target value: 243 (all unique).
    - Theoretical minimum: 27 (all the same).
- Sum of repeated values in each row, column, and box.
    - Target value: 0 (all unique).
    - Maximal value: 216 (all values are same, i.e., 8 repetitions in each r/c/b).
- Sum of squared number of unique values in each row, column, and box.
    - Target value: 2187 (all unique).
    - Theoretical minimum: 27 (all the same).
    - Number of unique values in a specific r/c/b increased by 1, impacts fitness value differently depending on the initial number.

- Wider range of fitness values.

Selection algorithms

All three basic selection algorithms are used this project: fitness proportionate selection (FPS), ranking and tournament. FPS is the only selection algorithm that does not work with minimization problems. But as unique value maximization and repetition minimization fitness functions are so similar, FPS performance can be estimated just using maximization functions.

Crossover

Two types of crossover algorithms are created: single point crossover and two-point crossover. As sudoku grid is two-dimensional and can be viewed from three equally important perspective: rows, columns, and boxes (3×3 supgrids). Therefore, each of those crossover types is implemented from four dimensions: row-, column-, box- and cell-based single point and two-point crossovers, 8 crossover functions in total (GitHub crossover figure).

Algorithm executions provides 3 options for the crossover: single point, two-point and random. Selecting single point randomly selects one of four single point crossover functions for each crossover case during evolution. Two-point crossover works in the similar way. Random selects from all 8 functions.

Mutation

There are two mutation options: swap and uniform. Three swap mutation functions swap values from two randomly selected cells within one row, column or box, depending on the function selected (GitHub swap figure). Selecting a swap options executes a random swap function every time individual got mutated. Uniform mutation has only one function that randomly selects a changeable value and assigns a different value instead.

Each element of the individual has equal probability to mutate p. Probability of more than one mutation happening for one individual is $1-(1-p)^{n-1}\big(1-p(1-n)\big)$ and is usually small but worth considering. Instead of iterating each element, mutation will run k times with the probability $\binom{n}{k}p^k(1-p)^{n-k}$ for $k = 0, 1, 2, \ldots, n$, where n is number of changeable elements of the individual. It total, mutation will run (at all) with the probability $1-(1-p)^n$. Number k is calculated by function `get_count(p, n)` that selects k randomly considering probabilities to select each k.

Changeable elements for uniform mutation are cells, which are usually around 50 to 55 in total (puzzle-determined empty cells). Changeable elements of swap mutation are rows, columns, and boxes, 9 of each, totaling to 27. As n are different in uniform and swap mutations, same value p results in different probabilities for a mutation to happen.

Uniform mutation changes one value at a time but ensures that mutated value is different from initial one. Swap mutation changes two values at once but swapped values can be the same. Uniform mutation can influence the fitness value in some, either or none of 3 dimensions (row/column/box). Swap mutation, in turn, keeps same number of unique values or repetitions for the row/column/box where it was performed.

Study design

This study consists of 21 experiments in 6 categories: 3 for fitness functions, 4 for crossovers, 4 for mutation, 2 for puzzle difficulty, 2 for population size and 6 for elitism. Each experiment is aimed to compare two to three different parameter values (e.g., 3 fitness functions) other things held constant. Each experiment corresponds to one row in the results tables below and a line chart with best fitness dynamics over generations (GitHub results folder).

Preliminary unsystematic experiments showed that (1) convergence time cannot be estimated and (2) usually fitness dynamics level out by 100^{th} generation with population size 1500. The basic assumption of this work is that the performance of the algorithm can be estimated by the loss value after 100 generations of 1500-individual population evolutions. Loss value is normalized to fit in a range [0, 1], which allows comparing different fitness function results.

There is a lot of randomizations in genetic algorithm, which makes a single observation not reliable enough to conclude about different parameter efficiency. Therefore, each case, i.e., parameter combination, is run 5 times. Result tables below show the mean (± standard deviation) for each case. T-test is performed to conclude about significance of differences in the results within each experiment.

All experiments below are run 5 times and results show average normalized loss after 100 generations of evolution for a population of 1500 individuals solving hard puzzle (unless stated otherwise).

Results

1. Fitness

Although this set of experiments was designed to compare the performance of different fitness functions in cases with different selection algorithms, conclusions can be made about both fitness functions and selection algorithms. As discussed previously, "unique" and "repetitions" are maximization and minimization versions of the same function and perform similarly as expected. "Unique squared" may seem to have worse performance, but it is designed to evaluate fitness non-linearly and if that is taken into consideration, it performs slightly (but insignificantly) better. All in all, fitness f-ns perform the same having all other parameters constant.

Selection algorithms, on a contrary, are all significantly different (1% conf.). In each fitness case, FPS returns the highest loss value and according to the fitness dynamics chart (1.1 fitness fps) it barely improves over generations. After all, it is the most random selection algorithm. Rank algorithm significantly (5% conf.) outperforms tournament in all observed cases.

	Selection	Fitness function		
		Unique (max)	Unique squared (max)	Repetitions (min)
1.1	FPS	0.266 (±0.009)	0.396 (±0.010)	NA
1.2	Tournament	0.060 (±0.007)	0.093 (±0.019)	0.063 (±0.018)
1.3	Rank	0.032 (±0.013)	0.056 (±0.017)	0.039 (±0.008)

Crossover: two-point (p=0.7). Mutation: uniform (p=0.01). No elitism.

As "unique" and "repetitions" are so similar, only one of those is used in further experiments. FPS is not used in most of further experiments due to poor performance.

2. Crossover

Crossover experiments gave significant results only in unique^2 cases, most probably because of wider range of fitness values it suggests. In case of tournament selection, two-point mutation is significantly better that single point mutation (5% conf.). In case of rank selection, two-point mutation has significantly lower loss value (5% conf.) compared to both single point and random. Two-point crossover might have higher performance due to better population diversification. As expected, random crossover has results between single and two-point.

	Fitness	Selection	Crossover (p=0.7)		
			Single point	Two-point	Random
2.1.1	Repetitions	Tournament	0.060 (±0.013)	0.056 (±0.010)	0.069 (±0.010)
2.1.2		Rank	0.050 (±0.012)	0.034 (±0.012)	0.043 (±0.043)
2.2.1	Unique squared	Tournament	<u>0.112 (±0.013)</u>	<u>0.094 (±0.006)</u>	0.100 (±0.011)
2.2.2		Rank	0.076 (±0.016)	<u>0.052 (±0.007)</u>	0.066 (±0.009)

Mutation: uniform (p=0.01). No elitism.

3. Mutation

Having the same element mutation probability, swap mutation seems to lead to lower loss value after 100 evolutions. The difference in mutation algorithm performance is significant in two out of four observed cases: unique^2 fitness with tournament selection (1% conf.) and repetitions fitness with rank selection (10% conf.). Worth noting that swap mutation results have lower variance, which is probably due to moderate impact on the fitness value discussed earlier.

	Fitness	Selection	Mutation (p=0.01)	
			Uniform	Swap
3.1.1	Repetitions	Tournament	0.058 (±0.015)	0.049 (±0.007)
3.1.2		Rank	<u>0.040 (±0.014)</u>	<u>0.023 (±0.009)</u>
3.2.1	Unique squared	Tournament	<u>0.105 (±0.024)</u>	<u>0.067 (±0.015)</u>
3.2.2		Rank	0.053 (±0.008)	0.047 (±0.010)

Crossover: two-point (p=0.7). No elitism.

4. Puzzle difficulty

Algorithms like people seem to be influenced by the difficulty of the task. Results for very hard puzzle are significantly different from other two (5% conf.) in case of fitness: unique squared, and for very easy (10% conf.)—in case of fitness: repetition.

	Fitness	Puzzle		
		Very easy	Moderate	Very hard
4.1	Repetition	<u>0.020 (±0.010)</u>	0.030 (±0.004)	0.031 (±0.009)
4.2	Unique squared	0.033 (±0.012)	0.031 (±0.011)	<u>0.052 (±0.010)</u>

Selection: rank. Crossover: two-point (p=0.07). Mutation: swap (p=0.01). No elitism.

5. Population size

Previous experiments with three parameter options (like fitness and crossover) and five repetitions for each run for about 25 minutes in total. Population size experiments show that calculation time is linearly dependent on the size of population. For example, having twice bigger population approximately doubles execution time. However, fitness value doesn't grow linearly (but rather logistically) over generations, which makes it harder to find a trade-off solution between execution time and convergence dynamics.

	Fitness	Population size		
		20	100	500
5.1	Repetition	0.069 (±0.014) 6.2 sec	0.052 (±0.008) 32.4 sec	0.032 (±0.009) 210.9 sec
5.2	Unique squared	0.128 (±0.024) 6.4 sec	0.089 (±0.005) 33.1 sec	0.052 (±0.015) 222.0 sec

Selection: rank. Crossover: two-point (p=0.7). Mutation: swap (p=0.01). No elitism.

6. Elitism

Elitism has a significant (1% conf.) effect on algorithms with FPS selection. But other algorithms still outperform FPS even if elitism is applied. In case of "unique" fitness with tournament selection, elitism significantly (5% conf.) worsens the performance of the algorithm.

	Fitness	Selection	Elitism	
			No	Yes
6.1.1	Unique	FPS	0.264 (±0.010)	0.224 (±0.015)
6.1.2		Tournament	0.033 (±0.011)	0.049 (±0.004)
6.1.3		Rank	0.034 (±0.011)	0.026 (±0.010)
6.2.1	Unique squared	FPS	0.384 (±0.009)	0.338 (±0.014)
6.2.2		Tournament	0.063 (±0.019)	0.062 (±0.014)
6.2.3		Rank	0.056 (±0.015)	0.058 (±0.015)

Crossover: two-point (p=0.7). Mutation: swap (p=0.01).

Discussion and conclusions

All three fitness functions had similar performance, although non-linear version of unique value maximization function (unique squared) might show significantly better results under further investigation. In all experiments rank selection algorithm leads to a lower loss value by 100^{th} generation. There is also evidence in favor of swap mutation over uniform, which could be partly explained by the fact that swap mutation changes two values as once or rather having lower impact on fitness function, which results in lower variance of the results. Elitism does not benefit the algorithm, unless FPS is used, which is clearly not our choice.