Solving Sudoku puzzles using Genetic Algorithms

Adil Rahman

ABSTRACT

A Sudoku puzzle is a combinatorial puzzle. The object of the puzzle is to fill in a 9x9 grid such that no row, column or 3x3 sub-grid contains duplicates.

This paper discusses the use of a Genetic Algorithm to solve a Sudoku puzzle. The latter half of this paper discusses the effects of the population size on the quality of solutions produced.

1. INTRODUCTION

A Sudoku puzzle is a popular combinatorial puzzle where the objective is to fill in every empty cell in the 9x9 grid with a value from 1 to 9, such that there are no duplicates in any rows, columns and sub-grids. When considered as a combinatorial optimization problem, it proves to be NP-Complete.

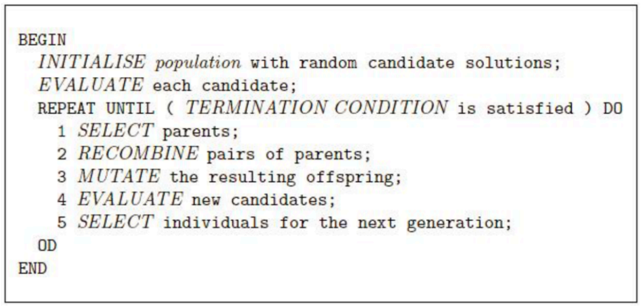
Text

Evolutionary Algorithms (EA) are a class of metaheuristic optimization approaches inspired by nature. These problems are used to find optimal solutions to problems that would otherwise be unfeasible to solve.

In a GA, an initial population of feasible solutions to a problem are produced. Parents are then selected from this population and recombined, producing children which may undergo some random mutation process. The quality of each child is evaluated in regard to the problem, with some bias being given toward the 'fitter' children to breed more solutions. This is analogous to Darwin's 'Survival of the Fittest' principle. This process is repeated until some termination criteria is met.

Although the implementation details differ, GAs tend to follow similar structures. As given in the assignment, the pseudocode for a GA is shown in in *Figure 1*.

Figure 1: Pseudocode for a general Genetic Algorithm



```
BEGIN
  INITIALISE population with random candidate solutions;
  EVALUATE each candidate;
  REPEAT UNTIL ( TERMINATION CONDITION is satisfied ) DO
    1 SELECT parents;
    2 RECOMBINE pairs of parents;
    3 MUTATE the resulting offspring;
    4 EVALUATE new candidates;
    5 SELECT individuals for the next generation;
  OD
END
```

In this report, I will discuss the design of my own implementation of a GA in regard to the Sudoku problem. I will also discuss the results that I had obtained from running experiments with the GA on various population sizes.

2. DESIGN

GAs tend to follow a similar structure, as mentioned in the previous section. However, the implementation details can differ.

In this section I will describe the design behind my implementation of the GA. This design can be broken down into eight components:

1. Solution Space and Representation
2. Fitness Function
3. Initialisation
4. Selection
5. Crossover
6. Mutation
7. Survivor Selection
8. Termination Criterion

The following subsections will describe each component in detail.

2.1 SOLUTION SPACE & REPRESENTATION

The solution space, also referred to as a search space, is the set of all feasible candidate solution. The solution space of the Sudoku puzzle is every possible grid permutation.

A solution representation refers to the encoding of a solution that is most appropriate for the GA. In essence, a Sudoku puzzle is a grid of values. Therefore, the most appropriate representation of the puzzle is a two-dimensional matrix. As the GA is written in python, a nested list was the most obvious implementation for this encoding; where each nested list represents a row in the grid. A two-dimensional array allows for the effective retrieval and manipulation of individual components of the puzzle, whist retaining each row and, effectively, each column.

2.2 FITNESS FUNCTION

A fitness function is a function that evaluates a candidate solution, returning a comparable metric of how 'fit' the solution is in comparison to another.

An optimal solution to a Sudoku puzzle is one where there are no duplicate values in any row, column or 3x3 sub-grid. Therefore, a solution with fewer duplicates than another can be presumed as a 'better' solution and closer to the global optimum.

The fitness function, f(x), that I have implemented simply calculates the sum of all the duplicates in every row, column and sub-grid. The worst fitness that any solution can have is 216. Therefore, $f(x) = y$, where $0 \leq y \leq 216$, with a lower value indicating a 'fitter' solution.

2.3 INITIALISATION

The initialization stage defines how an initial population is created in the GA.

In the GA implemented, a given Sudoku puzzle is read into the program, where all empty cells are represented as a 0 in the matrix. This illegal value is then replaced by a random integer between 1 and 9 to initialize a candidate solution.

During this stage, I add a hard constraint to initialize a better solution. The randomly generated numbers are added to each row only if the same number does not exist. This constraint ensures that there are no duplicates on each row and that each row has every number between 1 and 9. Adding this constraint does not change the complexity of the problem, with only the initialization stage becoming more cumbersome.

To create an initial population with **n** individuals, the above stage is simply repeated **n** times.

2.4 SELECTION

The selection stage defines the methods used to select parents to recombine and produce offspring. Parent selection has a direct effect on the quality of the solutions in the subsequent generation as well as the rate at which the GA converges. Although good parents are more likely to produce 'fitter' solutions, an elite population is more likely to result in premature convergence. Therefore, retaining solutions with generally bad fitness values is crucial to maintain an appropriate level of diversity in each generation.

Before selecting individual parents to mate, the GA uses a Greedy Selection to truncate the population, removing a specified proportion of the individuals with the highest fitness values. Defined by the *truncation rate* parameter, this stage creates a mating pool that is bias toward the fitter individuals in the population.

To select the two parents to crossover, the GA implements k-Tournament Selection. In each round of this selection strategy, **k** random individuals are selected from the mating pool; with the individual that has the lowest fitness value being selected. k-Tournament selection is executed for two rounds to select two parents for crossover.

Although truncating the population creates a bias mating pool, Tournament Selection introduces as stochastic element toward selecting the parent.

2.5 CROSSOVER

The crossover stage is analogous to reproduction and biological recombination. In this stage, two candidate solutions, referred to as the parents, produce two children; which are compositions of their parents.

The crossover operator that the GA incorporates is uniform crossover. This type of crossover treats each row as an individual entity, independently deciding whether or not a child will inherit it. This is in contrast to single or multi-point crossover, where the inherited traits are a collection of rows. Therefore, the uniform crossover operator will create a more diverse range of children than its alternatives; especially in regard to the fitness of their sub-grids.

To implement this operator, I effectively create a byte string of length 9 (the number of rows in the puzzle), which dictates which child inherits a particular row from which parent. To illustrate this mechanism, consider a byte string where the first bit value is 1. This will result in Child A inheriting the first row of Parent A. Subsequently, Child B will inherit the first row from Parent B.

The two children created are the transposed. These transposed candidate solutions are then crossed over in the same manner as their parents. This ensures there is also some diversification in regard to the columns, and effectively the sub-grids. I then transpose and return the final two children, returning the fixed positions to their original cells.

2.6 MUTATION

In the mutation stage, a mutation operator may or may not cause a random change to an individual, producing a new candidate solution. The result of this small change cannot be found in any parent. Therefore, it introduces a stochastic element that promotes diversity and allows the population to explore the solution space, giving the GA a chance to escape a local minimum.

The mutation operator the GA implements is a simple cell swap. Initially, a random integer between 0 and 8 is generated, representing the row. Two more integers, within the same range, are subsequently generated; which represents the columns within that row. Given that neither of the two coordinates point at a fixed position, and that the two coordinates or their corresponding values are not the same, the values are swapped. The hard constraint is honoured, as only two values in the same row are swapped, ensuring that there are no duplicates in the rows of any mutated child.

Two parameters are passed into the mutation function. The *number of swaps* made per mutation and the *mutation rate,* which defines the probability of a mutation taking place. This mutation function is applied to every produced child. However, only a proportion of children produced will be mutated, which is heavily influenced by the mutation rate.

2.7 SURVIVOR SELECTION

Survivor Selection refers to the strategy that determines which individuals are retained and passed on from generation N to generation N+1.

The GA employs components of elitism. An offspring population is initially created from a truncated copy of generation N, referred to as the mating pool. This offspring population has the same number of individuals as the generation before it.

Generation N+1 comprises of 70% of the fittest individuals from the offspring population and 30% of the population from generation N. Although this encourages elitism by removing the worst solutions, it allows the better solutions from the previous generation to propagate into the next.

2.8 TERMINATION CRITERION

The termination criterion is a set of conditions that determine when a GA program will terminate. It prevents a GA from continuing to produce newer generations when there is little or no improvement.

Once the best individual produced by any given generation has reached 0, the global optimum has been reached and a solution has been found. Therefore, the first termination condition is to terminate the program if the best individual in the latest generation has a fitness of 0.

From observing the pattern of the solutions produced by the GA, a general trend can be extracted. Initially, the GA progresses fast, saturating in later generations. However, although the fitness may converge and stay stable for an extended period of time, a fortunate crossover or mutation may occur, causing the EA to escape the local optimum. Some correlation between the population size and the number of generations that the GA has been stuck in a local optimum can be seen; with larger populations escaping the local optimum in fewer generations. Therefore, an appropriate termination criterion is to terminate the run when a specified number of candidate solutions has been produced by the GA. I have set this limit to 500,000 candidate solutions.

3 EXPERIMENTS

This section of the paper discusses the experiments performed on the GA that I have implemented.

The experiments, described in this section, are performed on three different Sudoku puzzles, each with a different number of fixed values, ranging from 23 to 29. In order to compare how the population size affects the quality of the solutions produced and rate of convergence of the GA, five trials were performed on each grid, with the populations 10, 100, 1000, 10000.

Each trial was executed with a different random seed. Although the seed was not stated explicitly, the default value for python's *random.seed()* function is the current timestamp. As each experiment was executed at different times, different seeds were used.

The results are displayed in *Table 1*:

Table 1: Results of running the GA on the different puzzles with different population sizes.

		Grid 1		Grid 2		Grid 3	
Pop	**Trial**	**G**	**BF**	**G**	**BF**	**G**	**BF**
10	1	6	50000	11	50000	13	50000
	2	16	50000	13	50000	10	50000
	3	8	50000	8	50000	12	50000
	4	12	50000	14	50000	12	50000
	5	10	50000	9	50000	19	50000
	Avg	**10**	-	**11**	-	**13**	-
100	1	14	5000	11	5000	12	5000
	2	16	5000	12	5000	13	5000
	3	12	5000	8	5000	14	5000
	4	13	5000	13	5000	11	5000
	5	10	5000	8	5000	14	5000
	Avg	**13**	-	**10**	-	**13**	-
1000	1	10	500	6	500	11	500
	2	12	500	2	500	5	500
	3	7	500	4	500	8	500
	4	9	500	7	500	15	500
	5	9	500	4	500	11	500
	Avg	**9**	-	**5**	-	**10**	-
10000	1	0	41	4	50	31	50
	2	0	46	2	50	32	50
	3	0	42	2	50	30	50
	4	2	50	0	41	32	50
	5	4	50	2	50	32	50
	Avg	**1**	-	**2**	-	**31**	-

Pop = Population Size, G = Generation, BF = Best Fitness

The figures in *Table 1* show that a larger population size will generally result in better solutions. With the exception of Grid 3 and a population size of 10000, the fitness of the best solutions returned by the GA at each generation improved as the population size increased, with the GA returning a correct solution four times with the population size set to 10000.

Figure 2. Population size = 10

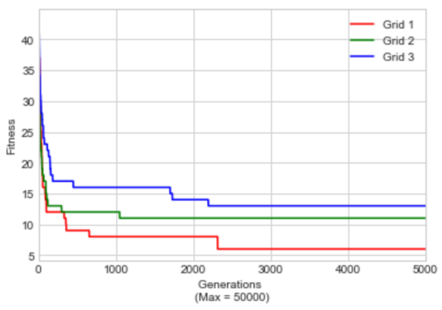


Figure 2 illustrates the different solutions produced at each generation for each Sudoku grid when the population size was set to 10. Each execution of the GA resulted in some stagnation at a local optimum before escaping toward a better solution. For all grids however, each execution resulted in premature convergence, with Grid 1 producing the best solution and Grid 3 producing the worst solution.

Figure 3. Population size = 100

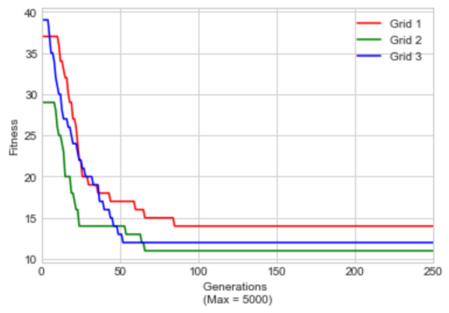


Figure 3 illustrates the different solutions produced at each generation for each Sudoku grid when the population size was set to 100. In comparison to a population size of 10, the GA converged quickly for all grids. For all girds, each execution converged prematurely, with Grid 2 producing the best solution and Grid producing the worst.

Figure 4. Population size = 1000

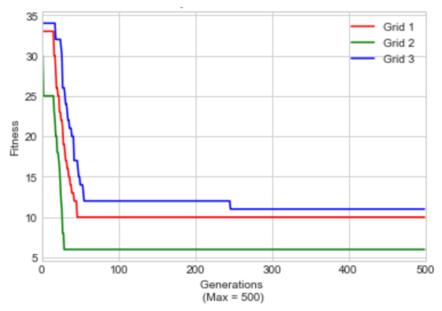


Figure 4 illustrates the different solutions produced at each generation for each Sudoku grid when the population size was

set to 1000. The GA converges quickly for all grids, with Grid 2 being trapped in a local optimum before escaping between generation 200 and 300. As with the previous results, each execution of the GA converged prematurely, with Grid 2 producing the best solution and Grid 3 producing the worst.

Figure 5. Population size = 10000

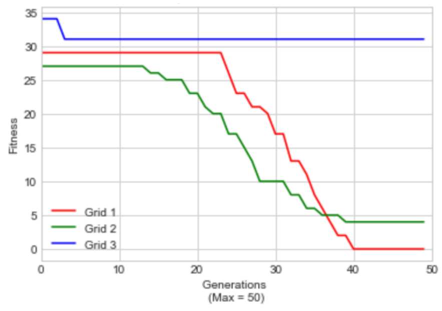


Figure 5 illustrates the different solutions produced at each generation for each Sudoku grid when the population size was set to 10000. Whereas the GA produced similar results for all grids in the previous population values, a population of 10000 produces a different pattern. Grid 3 immediately converges into a local optimum, producing solutions with a very high fitness value. After some stagnation, both Grid 1 and Grid 2 converge, with Grid 1 converging prematurely, albeit producing much better solutions that Grid 3, and Grid 2 finding an optimal solution around.

4 CONCLUSION

A Sudoku puzzle is a popular combinatory puzzle, where the objective is to fill in every empty cell in the 9x9 grid with a value from 1 to 9, such that there are no duplicates in any rows, columns and sub-grids. When considered as a combinatorial optimization problem, it proves to be NP-Complete.

A Genetic Algorithm is a subset of Evolutionary Algorithms, that adopts principles of genetics and natural selection in order to serve as a heuristic approach to optimization problems.

This report details the design of the GA I have implemented in regard to the Sudoku puzzle problem. I have then recorded the results of experiments which illustrate which population size is the most effective. From the results obtained, it can be concluded that a larger population size will generally result in better solutions.

5 QUESTIONS

In this section I will answer the questions specified in the assignment.

5.1 Which population size was the best?

Generally, the best population size was 10000. Only when the population size was set to 10000 did the GA solve some puzzles. However, although it generally produced good results for Grid 1 and Grid 2, it produced the worst results for Grid 3.

5.2 What do you think is the reason for your findings in 5.1?

A larger population size means that there is a greater chance for an optimal solution or solution close to the global optima to be produced during initialization. Therefore, a larger population size allows for a larger solution space and consequently allows for more diversity within the population. Therefore, the optimal solution is more likely to be produced in subsequent generations.

Although this population size produced the worst results for Grid 3, it can be argued that if the GA increased the number of candidate solutions in the second termination criterion, the GA would eventually escape the local optimum and converge closer to the global optimum.

5.3 Which grid was the easiest and which was the hardest to solve?

Grid 1 was the easiest to solve, producing an optimal solution three out of five times when the population was set to 10000. In contrast, Grid 3 was the hardest to solve, producing the worst solutions in every trial and was the only grid that the GA did not produce a solution for.

5.4 What do you think is the reason for your findings in 5.3?

Grid 1 was the easiest to solve as it had the greatest number of fixed values. Therefore, it had a smaller solution space than any other grid. A population size of 10000 proved to be sufficient enough to solve this puzzle, providing an optimal solution during the majority of the trials.

Grid 3 was the hardest to solve as it had the fewest number of fixed cells. It also contained a row with no fixed positions. This results in a much larger solution space. A population size of 10000 and a limit of 500000 candidate solutions produced was evidently insufficient to provide an optimal solution to this problem.

5.3 What further experiments do you think it may be useful to do and why?

The first experiment that I believe would be useful is to increase the maximum number of candidate solutions produced. In conjunction with a high population rate. It is more likely that more Grids, namely Grid 3, will converge at a better solution, if not the optimal one.

Another useful experiment would be to find the effect of the truncation rate on the quality of the solutions produced. Although a high truncation rate will be more bias toward better solution, an elite GA may produce more favorable results.

Different mutation rates should also be tested. Whereas a higher truncation rate creates a more elite population, a higher mutation rate will increase the stochastic nature of the evolutionary process. Experiments should be run to see the effect of the mutation rate on the quality of the solutions produced.

Finally, I would alter the ratio of the survivor selection strategy. Currently, 70% of the offspring produced from generation N and 30% of generation N are joined to produced generation N+1. Experiments to explore whether different proportions would yield better solutions should be carried out to find the optimal ratio, if one exists.