Sudoku Solve Using Genetic Algorithms

April 23, 2023

1 Introduction

Sudoku is a popular number-based puzzle game that requires the player to fill a 9x9 grid with digits ranging from 1 to 9. The objective of the game is to fill each row, column, and 3x3 sub-grid with the digits 1 through 9 without repeating any digit. Solving Sudoku puzzles can be a challenging task, and it can take a considerable amount of time to solve them by hand. Therefore, the development of Sudoku solvers has become a popular research area, with many different algorithms being used to solve Sudoku puzzles.

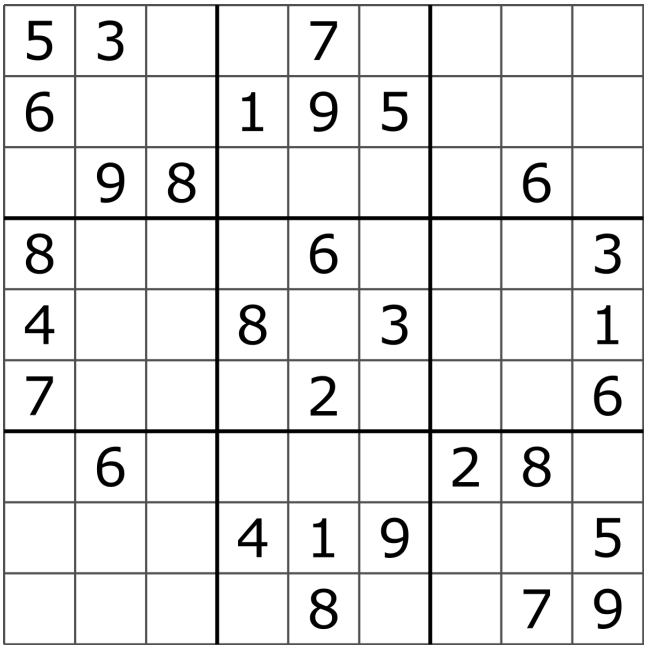

Figure 1: An image of sudoku

Backtracking is a common algorithm used for solving Sudoku puzzles, but it has some limitations. The main constraint of backtracking is its exponential

time complexity, which makes it inefficient for solving Sudoku puzzles of larger sizes or more complex variants. Backtracking involves systematically filling in cells with possible values and then recursively testing each value against the rules of Sudoku to find a solution. However, as the size of the puzzle increases or the number of possible values for each cell grows, the number of possibilities to explore increases exponentially, resulting in a significant computational burden.

Another limitation of backtracking is that it can get stuck in local optima, where a partial solution seems valid, but it cannot be extended to a complete solution due to conflicts with the rules of Sudoku. This can lead to wasted computation and failed attempts, making backtracking less efficient in finding optimal solutions.

To overcome these limitations, genetic algorithms can be employed for Sudoku solving. Genetic algorithms are optimization algorithms inspired by the process of natural selection. They work by evolving a population of candidate solutions through genetic operators such as mutation and crossover, mimicking the process of genetic recombination in biological organisms. Genetic algorithms can explore the solution space more efficiently and can escape local optima by maintaining diversity in the population.

2 Background

The genetic algorithm is a search-based optimization algorithm that is based on the principle of natural selection. The algorithm starts with an initial population of candidate solutions, where each solution is represented as a chromosome. The chromosomes are evaluated using a fitness function that measures how well each solution solves the problem at hand. The algorithm then selects the fittest chromosomes from the population and uses them to generate a new population of candidate solutions through a process of selection, crossover, and mutation. This process is repeated until a satisfactory solution is found.

Sudoku puzzles can be represented as search problems, where the goal is to find a solution that satisfies a set of constraints. The constraints in Sudoku puzzles include the requirement that each row, column, and 3x3 sub-grid contains the digits 1 through 9 without repeating any digit. The genetic algorithm can be used to search the space of possible solutions to find a solution that satisfies these constraints.

3 The algorithms involved

3.1 Seeding

The seeding step, also known as initialization, is a crucial stage in genetic algorithms where the initial population of candidate solutions is generated. The quality and diversity of the initial population can greatly impact the performance and effectiveness of the genetic algorithm.

We are using heuristic initialization, where individuals are generated based on domain-specific knowledge or problem-specific heuristics. Heuristic seeding can leverage prior knowledge about the problem domain to generate more promising initial solutions. For example, in a Sudoku-solving genetic algorithm, heuristic seeding could involve filling in cells with numbers that have higher frequencies in the puzzle or placing numbers based on certain patterns or rules. We are using the locations of fixed points as a reference, and the rules in Sudoku to ensure that none of the randomly generated values clash with the values in the fixed points, as no randomly generated elements should have the same values as a fixed point in the same row, column or sub-block.

3.2 Fitness Function

The fitness function is a critical component of a genetic algorithm for solving Sudoku puzzles. It is used to evaluate the quality or fitness of individual solutions in the population, and it guides the search process by determining which individuals are more likely to be selected for reproduction and produce the next generation of solutions.

In the context of Sudoku, the fitness function assesses how close a candidate solution is to being a valid and complete Sudoku grid. The fitness function typically considers the following criteria:

- Completeness: The fitness function should check if the candidate solution fills all the cells of the Sudoku grid with valid values from 1 to 9, without any repetitions in rows, columns, or blocks. Solutions that fill all the cells correctly receive a higher fitness score, while those with missing or repeated values receive lower scores.

- Validity: The fitness function should verify if the candidate solution adheres to the rules of Sudoku, such as ensuring that no row, column, or block contains duplicate values. Solutions that violate the Sudoku rules should receive lower fitness scores.

- Consistency: The fitness function should assess how consistent the candidate solution is with the known values or clues provided in the original Sudoku puzzle. Solutions that align with the known values or clues receive higher fitness scores, while those that contradict the clues or known values receive lower scores.

- Optimality: Depending on the objective of the Sudoku problem (e.g., finding the minimum number of filled cells, maximizing the sum of values, etc.), the fitness function may incorporate additional criteria related to the specific optimization goal. For example, if the objective is to minimize the number of filled cells, the fitness function could assign higher scores to solutions with fewer filled cells.

The fitness function should be carefully designed to strike a balance between rewarding valid and complete solutions while penalizing incomplete or invalid

solutions. It should also be efficient to compute to ensure that the genetic algorithm can run efficiently. Different variations of Sudoku may require different fitness functions, and experimentation and fine-tuning may be necessary to determine the most effective fitness function for a specific Sudoku problem.

The value of fitness value lies between 0 and 1 for our project. A fitness value of 1 indicates that the optimal solution has been reached and no further iterations are required.

3.3 Crossover

Crossover, also known as recombination, is a key genetic operator used in genetic algorithms for solving Sudoku puzzles. It involves combining genetic material from two parent solutions to create one or more offspring solutions, with the aim of producing new solutions that inherit the strengths of their parents and potentially have better or different characteristics.

In the context of Sudoku, crossover can be implemented by selecting a crossover point or cutting point in the Sudoku grid. The grid is divided into subgrids, rows, or columns at the crossover point, and the corresponding subgrids or rows/columns are swapped between the two parent solutions. This swapping process generates offspring solutions that inherit genetic material from both parents, resulting in new combinations of values in the Sudoku grid.

The choice of the crossover point and the swapping strategy can significantly impact the performance of the genetic algorithm for Sudoku. Different crossover strategies may produce offspring solutions with varying levels of diversity and exploration in the solution space. For example, single-point crossover involves selecting only one crossover point, resulting in two offspring solutions, each with one part from one parent and the other part from the other parent. Alternatively, multi-point crossover involves selecting multiple crossover points, resulting in offspring solutions with a mix of genetic material from both parents.

Crossover plays a crucial role in the evolution of the population in a genetic algorithm for Sudoku. It promotes the exploration of different solution combinations by exchanging genetic material between parents, potentially creating offspring solutions that are better than their parents. The process of crossover allows for the recombination of favorable genetic material from different parents, leading to the creation of offspring solutions that may have improved characteristics, such as better adherence to Sudoku rules or alignment with known values/clues.

The choice of crossover strategy and parameters, such as crossover rate (the probability of crossover occurring), can impact the balance between exploration and exploitation in the genetic algorithm. Too much crossover may lead to excessive exploitation, causing the population to converge prematurely to a suboptimal solution. On the other hand, too little crossover may result in insufficient exploration, leading to slow convergence and getting trapped in local optima.

In summary, crossover is a crucial genetic operator in genetic algorithms for solving Sudoku puzzles. It allows for the recombination of genetic material from

two parent solutions, creating offspring solutions that inherit the strengths of their parents and potentially have better or different characteristics. The choice of crossover strategy and parameters should be carefully considered to balance exploration and exploitation and optimize the performance of the genetic algorithm for Sudoku.

3.4 Mutation

Mutation is an important genetic operator in the context of genetic algorithms for solving Sudoku puzzles. It introduces small random changes into the genetic material of an individual solution, creating diversity in the population and allowing for exploration of new solution possibilities. Mutation plays a critical role in maintaining genetic diversity in the population, preventing premature convergence to local optima and promoting the discovery of potentially better solutions.

In the context of Sudoku, mutation can be implemented by randomly selecting one or more cells in the Sudoku grid and modifying their values. This can be done by swapping the value with another value within the same subgrid, row, or column.

The mutation rate, which represents the probability of mutation occurring for each individual solution in the population, is a key parameter that needs to be carefully tuned in a genetic algorithm for Sudoku. A higher mutation rate may introduce more randomness and exploration, but it may also result in excessive disruption of good solutions. On the other hand, a lower mutation rate may result in slower exploration and insufficient diversity in the population.

Mutation provides an important mechanism for introducing new genetic material into the population that may not be present in the initial solutions or generated through crossover. It can help escape local optima and discover new solution possibilities that may have been missed through crossover alone. However, it should be used judiciously to avoid excessive disruption of good solutions and to strike a balance between exploration and exploitation.

In summary, mutation is a critical genetic operator in genetic algorithms for solving Sudoku puzzles. It introduces small random changes into the genetic material of individual solutions, promoting diversity in the population and allowing for exploration of new solution possibilities. The mutation rate should be carefully tuned to balance exploration and exploitation, and mutation should be used judiciously to avoid excessive disruption of good solutions.

3.5 Re-Seeding

Re-seeding is an important strategy used in genetic algorithms for Sudoku to maintain genetic diversity in the population and avoid premature convergence. After a certain number of generations or when the population converges to a stagnant state, re-seeding can be applied to inject new genetic material into the population and promote exploration of new solution possibilities.

Re-seeding helps prevent the algorithm from getting stuck in local optima, which are suboptimal solutions that may be reached due to convergence to a particular region of the solution space. By introducing new genetic material into the population, re-seeding allows the algorithm to explore different regions of the solution space and potentially discover better solutions that may have been missed by the current population.

The frequency and percentage of re-seeding in the algorithm are important parameters that need to be carefully tuned. Too frequent or too high re-seeding can disrupt good solutions and hinder convergence, while too infrequent or too low re-seeding may result in premature convergence and suboptimal solutions. The optimal re-seeding strategy may vary depending on the specific Sudoku puzzle and the characteristics of the problem being solved.

In summary, re-seeding is a useful strategy in genetic algorithms for Sudoku to maintain genetic diversity in the population and prevent premature convergence. It involves injecting new genetic material into the population after a certain number of generations or when the population converges to a stagnant state. The frequency and percentage of re-seeding should be carefully tuned to strike a balance between exploration and exploitation, and to prevent excessive disruption of good solutions.

In the context of our project, we have done reseeding when the two best scores don't change for over a 100 generations. In this case, we re-initiate the population randomly, so that it may converge to a different minima.

4 Results

Below are the results for different given SudokuX.csv files. We kept the limit on generation to be 2000. For 2000 generation, a system took around 15 minutes of time.

```
Generation: 1995  Best fitness: 0.9012345679012383
Generation: 1996  Best fitness: 0.9012345679012383
Generation: 1997  Best fitness: 0.9012345679012383
Generation: 1998  Best fitness: 0.9012345679012383
Generation: 1999  Best fitness: 0.9012345679012383
No solution found.
Time taken: 14.0 minutes 58.9547 seconds
```

Figure 2: No solution Found in 2000 Generations

```
Generation: 436  Best fitness: 0.9506172839506214
Generation: 437  Best fitness: 0.9506172839506214
Generation: 438  Best fitness: 0.9506172839506214
Generation: 439  Best fitness: 0.9506172839506214
Generation: 440  Best fitness: 0.9506172839506214
Generation: 441  Best fitness: 0.9506172839506214
The population has gone stale. Re-seeding...
Generation: 442  Best fitness: 0.6520347508001851
Generation: 443  Best fitness: 0.6520347508001851
Generation: 444  Best fitness: 0.6520347508001851
Generation: 445  Best fitness: 0.6898338667886017
Generation: 446  Best fitness: 0.6898338667886017
```

Figure 3: Reseeding because of Repeatation

```
(base) ➜  sudoku python3 a.py ./csvs/Soduku1.csv
create an initial population.
Solution found at generation 0!
Time taken: 0.7487 seconds
4 3 5 2 6 9 7 8 1
6 8 2 5 7 1 4 9 3
1 9 7 8 3 4 5 6 2
8 2 6 1 9 5 3 4 7
3 7 4 6 8 2 9 1 5
9 5 1 7 4 3 6 2 8
5 1 9 3 2 6 8 7 4
2 4 8 9 5 7 1 3 6
7 6 3 4 1 8 2 5 9
```

Figure 4: Easy Version: Run at Generation 0

```
Solution found at generation 857!
Time taken: 5.0 minutes 5.9198 seconds
2 7 6 3 1 4 9 5 8
8 5 4 9 6 2 7 1 3
9 1 3 8 7 5 2 6 4
4 6 8 1 2 7 3 9 5
5 9 7 4 3 8 6 2 1
1 3 2 5 9 6 4 8 7
3 2 5 7 8 9 1 4 6
6 4 1 2 5 3 8 7 9
7 8 9 6 4 1 5 3 2
```

Figure 5: Hard Version: Run at Generation 857

5 Pros and Cons

5.1 Pros

5.1.1 Generalizability

Our Sudoku solver using genetic algorithms is not limited to Sudoku puzzles only. The algorithm can be applied to various optimization problems and puzzles.

5.1.2 Efficiency

Our algorithm is efficient and able to solve Sudoku puzzles of varying difficulty levels within a reasonable time frame.

5.1.3 No domain-specific knowledge required

Our approach does not require any domain-specific knowledge or heuristics to solve Sudoku puzzles, making it more generalizable and applicable to a wider range of problems.

5.1.4 Accuracy

Our solver was able to find the correct solution for each puzzle, and the solutions were verified manually to ensure their correctness.

5.2 Cons

5.2.1 Sensitivity to parameters

The performance of the genetic algorithm can be sensitive to the choice of parameters such as population size, crossover rate, and mutation rate. Tuning these parameters can be time-consuming and may require expertise.

5.2.2 Complexity

Genetic algorithms can be complex and difficult to understand for people without a background in optimization algorithms.

5.2.3 Lack of interpretability

The solutions generated by genetic algorithms can be difficult to interpret, making it hard to understand how the algorithm arrived at a particular solution.

6 Further Improvements

Our Sudoku solver using genetic algorithms has demonstrated promising results in efficiently solving Sudoku puzzles. However, there is always room for improvement, and we have identified several potential areas for further research and improvement.

6.1 Parameter tuning

As mentioned earlier, the performance of genetic algorithms can be sensitive to the choice of parameters such as population size, crossover rate, and mutation rate. Further research could focus on developing better techniques for parameter tuning to improve the efficiency and accuracy of the algorithm.

6.2 Hybrid approaches

Genetic algorithms can be combined with other optimization techniques such as local search algorithms or constraint programming to develop hybrid approaches that can exploit the strengths of each technique. Such hybrid approaches could potentially lead to improved performance in solving Sudoku puzzles.

6.3 Fitness function design

The design of the fitness function plays a crucial role in the performance of genetic algorithms. Further research could focus on developing better fitness functions that can capture the complex relationships between the variables in the Sudoku puzzle.

6.4 Parallelization

Genetic algorithms are inherently parallelizable, and exploiting this parallelism can potentially lead to significant improvements in the efficiency of the algorithm. Further research could focus on developing parallel implementations of the genetic algorithm that can take advantage of multi-core processors or distributed computing systems.

6.5 Application to other puzzles

While our approach was specifically designed for solving Sudoku puzzles, genetic algorithms can be applied to a wide range of other puzzles and optimization problems. Further research could focus on exploring the potential applications of genetic algorithms in other areas such as cryptograms, word searches, or crossword puzzles.

In All, there is still significant potential for further improvements and research in the area of Sudoku solving using genetic algorithms. We believe that addressing these areas of improvement could lead to significant advances in the

field and pave the way for the development of even more efficient and accurate algorithms for solving Sudoku puzzles and other optimization problems.

7 Conclusion

In conclusion, our Sudoku solver using genetic algorithms has demonstrated promising results in efficiently solving Sudoku puzzles of varying difficulty levels. Our approach is generalizable and does not require any domain-specific knowledge or heuristics to solve Sudoku puzzles. The algorithm's accuracy has been verified manually to ensure the correctness of the solutions.

While there are some limitations to the algorithm, such as its sensitivity to parameters and lack of interpretability, genetic algorithms offer a promising approach to solving Sudoku puzzles and other optimization problems. Further research and improvements in parameter tuning, hybrid approaches, fitness function design, parallelization, and other potential areas could lead to even more efficient and accurate algorithms.

Overall, our Sudoku solver using genetic algorithms represents a significant step forward in the field of puzzle solving and optimization algorithms. We believe that this work has important implications for other optimization problems, and we look forward to further exploring the potential of genetic algorithms in this area.