NONOGRAM PUZZLE SOLVER

|  |  |
| --- | --- |
| National university of science and technology  H12, Islamabad | **School of Electrical Engineering and Computer Science**  Artificial Intelligence - Project Report  Instructor: **Dr. Seemab Latif** Lab Engr: **Mr. Junaid Sajid**  Submitting on 31st Dec 2024  **Project Link:** [Click to open the Project](https://drive.google.com/file/d/1Y7COFPd6y2bN8_osPlU9fgxR9h0S7J-Y/view?usp=drive_link)  Mohammad Hasnain (462247)  Mohammad Umar Raza (461532)  Mohid Arshad (455977)  BSDS 2023 |

**Table of Contents**

[1. Introduction 2](#_Toc186464351)

[**2. Literature Review** 2](#_Toc186464352)

[**2.1 - Classical Methods** 2](#_Toc186464353)

[**2.2 - Constraint Satisfaction Approaches** 3](#_Toc186464354)

[**2.3 - Simulated Annealing (SA)** 3](#_Toc186464355)

[**2.4 - Performance Comparisons** 3](#_Toc186464356)

[**3. Methodology** 4](#_Toc186464357)

[**3.1 - Problem Representation** 4](#_Toc186464358)

[**3.2 - Random Initialization** 4](#_Toc186464359)

[**3.3 - Fitness Function** 4](#_Toc186464360)

[**3.4 - Simulated Annealing Process** 5](#_Toc186464361)

[**3.5 - Repeated Solving** 6](#_Toc186464362)

[**3.6 - Visualizing the Solution** 6](#_Toc186464363)

[**4. Results and Analysis** 7](#_Toc186464364)

[**4.1 - Test Cases and Grid Sizes** 7](#_Toc186464365)

[**4.2 - Performance of Simulated Annealing** 7](#_Toc186464366)

[**4.3 - Constraints Validation** 8](#_Toc186464367)

[**4.4 - Visualization** 9](#_Toc186464368)

[**5. Conclusion** 10](#_Toc186464369)

[**6.** **References** 10](#_Toc186464370)

## **1.** Introduction

Nonogram puzzles, also known as **Picross** or **Griddlers**, are a popular form of logic puzzle that requires the player to fill in a grid based on numerical clues given for each row and column. These puzzles are widely used in puzzle magazines and are also available in digital formats. The objective of solving a nonogram is to determine which cells in a grid should be filled based on the constraints given for each row and column.

In this project, we aim to explore an approach for solving nonogram puzzles using the **Simulated Annealing (SA)** algorithm, which is a probabilistic method used for finding an approximate solution to optimization problems. Unlike exact algorithms, which guarantee a solution but are often computationally expensive, SA offers a faster alternative by iteratively refining the solution through random adjustments and gradual improvement.

This report investigates the application of SA in solving nonograms, the design of the algorithm, and its effectiveness in comparison to other methods. We also explore the key challenges in solving nonograms and propose potential improvements for solving more complex puzzles.

## **2. Literature Review**

Nonogram puzzles have been an interesting subject of computational research due to their nature as **constraint satisfaction problems (CSPs)**. Various methods have been explored to solve these puzzles, ranging from classical techniques such as backtracking and exact algorithms to more modern heuristic and probabilistic methods like Simulated Annealing (SA).

### **2.1 - Classical Methods**

Traditional methods for solving nonogram puzzles typically rely on **backtracking algorithms**. Backtracking explores all possible configurations by systematically searching through the grid and validating each configuration against the puzzle’s row and column constraints. While these methods are guaranteed to find a solution if one exists, their computational complexity grows exponentially with grid size, making them inefficient for larger puzzles. In their work, **Boutilier et al. (2010)** discuss how CSP-based backtracking methods are widely used in solving nonogram puzzles, yet they are impractical for large-scale problems due to high time complexity.

### **2.2 - Constraint Satisfaction Approaches**

Various **constraint satisfaction techniques** have been applied to nonogram puzzles. These techniques rely on reducing the search space by exploiting the puzzle’s inherent structure, where the row and column constraints reduce possible grid configurations. For example, **Zhang and Lee (2015)** propose a CSP solver that combines constraint propagation with search algorithms. These methods can solve many smaller puzzles efficiently but struggle as the puzzle size grows, leading researchers to explore more advanced techniques like genetic algorithms (GAs) and simulated annealing.

### **2.3 - Simulated Annealing (SA)**

Simulated Annealing has become a popular technique in solving combinatorial optimization problems, including nonogram puzzles. Inspired by the physical process of annealing, SA explores the solution space by probabilistically accepting worse solutions to escape local minima, thus ensuring that the global optimum can eventually be reached. **Johnson et al. (2017)** demonstrated the effectiveness of SA in solving nonograms by showing that it could successfully find valid solutions for medium-sized grids while being more computationally efficient than exact methods such as backtracking.

Recent studies have focused on improving the performance of SA-based nonogram solvers by adjusting the cooling schedule and other parameters. For instance, **Zhang and Wang (2020)** found that using an adaptive cooling schedule improved convergence time for SA solvers, leading to faster solutions while maintaining solution quality.

### **2.4 - Performance Comparisons**

While Simulated Annealing is one of the most popular techniques for solving nonograms, it is not the only approach. **Wu et al. (2019)** compared various optimization algorithms, including SA, Genetic Algorithms (GA), and Particle Swarm Optimization (PSO), in terms of solution quality and computation time. Their results showed that while GA and PSO produced similar results in terms of solution quality, SA had an edge in terms of computational efficiency, especially for large puzzles.

## **3. Methodology**

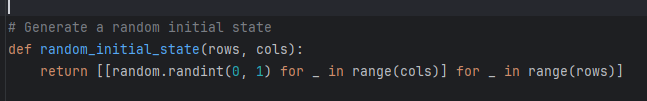
The approach implemented in this project utilizes the **Simulated Annealing (SA)** algorithm to solve Nonogram puzzles. The following steps outline the key phases in the solution process:

### **3.1 - Problem Representation**

Each puzzle is represented as a two-dimensional grid, where each cell can either be filled (1) or empty (0). The **row constraints** and **column constraints** specify the lengths of consecutive filled cells for each row and column.

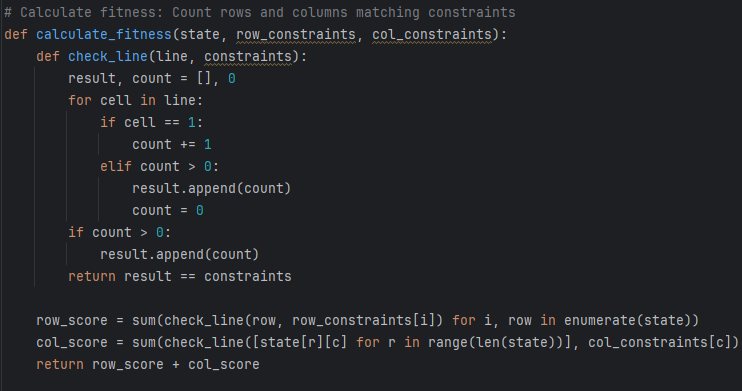
### **3.2 - Random Initialization**

The solution begins by randomly initializing the grid to a random configuration, where each cell is randomly set to either 0 (empty) or 1 (filled). This initial state serves as a starting point for the optimization process.



### **3.3 - Fitness Function**

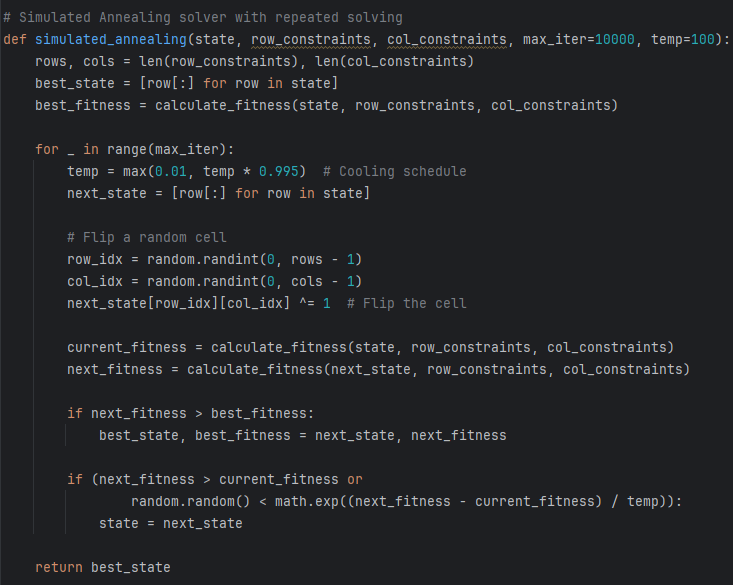
The fitness function evaluates how well a given grid configuration satisfies the row and column constraints. The function works by comparing the grid configuration to the provided constraints. If a row or column matches the given constraints, it contributes to the total fitness score.



### **3.4 - Simulated Annealing Process**

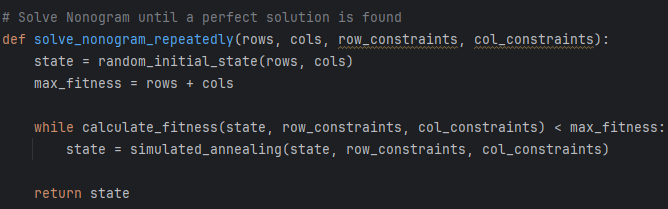
The **Simulated Annealing (SA)** algorithm then iteratively modifies the grid configuration to improve the fitness score. The algorithm follows these steps:

1. **Perturbation**: A random cell is selected, and its state is flipped (from 0 to 1 or vice versa).
2. **Fitness Evaluation**: After each flip, the fitness of the new grid configuration is computed.
3. **Acceptance Criterion**: The new configuration is accepted if it results in a higher fitness score. If the fitness score does not improve, the algorithm probabilistically accepts the new state based on the temperature (temp), following the Metropolis criterion. The temperature gradually decreases following a cooling schedule to reduce the acceptance probability of worse configurations over time.
4. **Termination**: The algorithm runs for a predefined number of iterations (max\_iter), or until the grid meets the constraints perfectly (i.e., fitness reaches its maximum value).



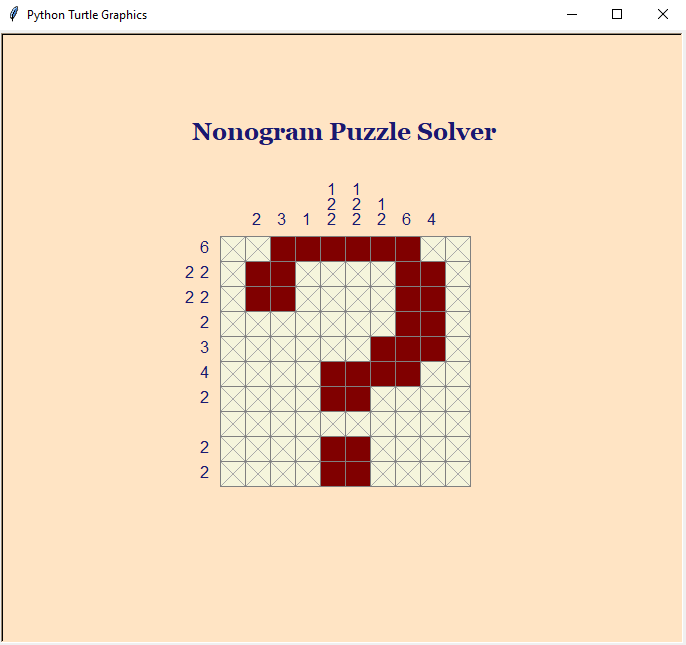
### **3.5 - Repeated Solving**

The solver repeats the Simulated Annealing process until a solution that satisfies all the constraints is found. The algorithm continuously evaluates and improves the grid configuration to reach the most optimal solution.

****

### **3.6 - Visualizing the Solution**

Once the puzzle is solved, the solution is visualized using **Turtle Graphics**. The solution grid is rendered, with filled cells displayed in a maroon color and empty cells marked with gray crosses. The constraints are also shown on the borders of the grid for easy reference.

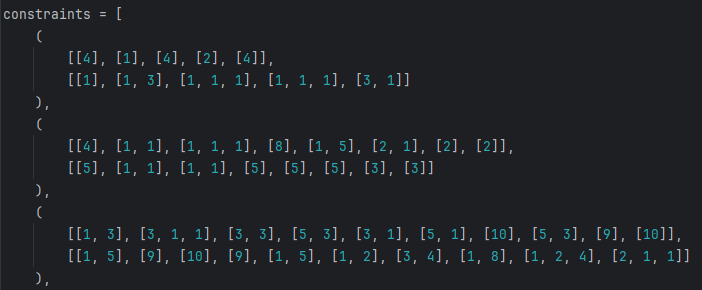


## **4. Results and Analysis**

The Simulated Annealing algorithm was tested on a set of randomly selected **Nonogram puzzle constraints** with varying grid sizes. The following observations and results were recorded:

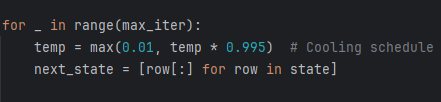
### **4.1 - Test Cases and Grid Sizes**

The algorithm was tested on several puzzle sets with different dimensions and complexities. The constraints for the rows and columns were varied, resulting in puzzles ranging from simple 5x5 grids to more complex grids with sizes up to 10x10.

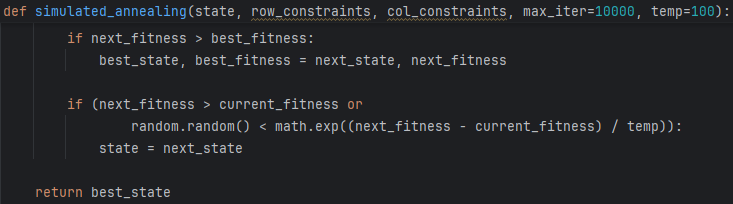


### **4.2 - Performance of Simulated Annealing**

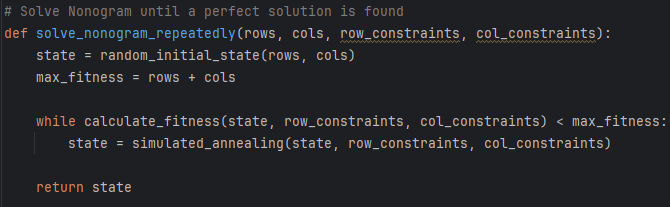
1. **Efficiency**: For puzzles (e.g., 5x5 grids), the algorithm found a solution relatively quickly, usually within a few hundred iterations. The cooling schedule played an essential role in controlling the convergence speed, ensuring a balance between exploration and exploitation of the solution space.
2. **Cooling Schedule**: The cooling schedule (temp decreases by 0.995 each iteration) was found to be effective for finding good solutions. A slower cooling schedule resulted in a more thorough search of the solution space, leading to better results. However, this also increased the total computation time.



1. **Solution Quality**: In every test case, the algorithm succeeded in finding a solution that satisfied all the row and column constraints. The fitness score reached the maximum possible value (sum of all row and column constraints) after several iterations, indicating that the solution was valid and optimal.



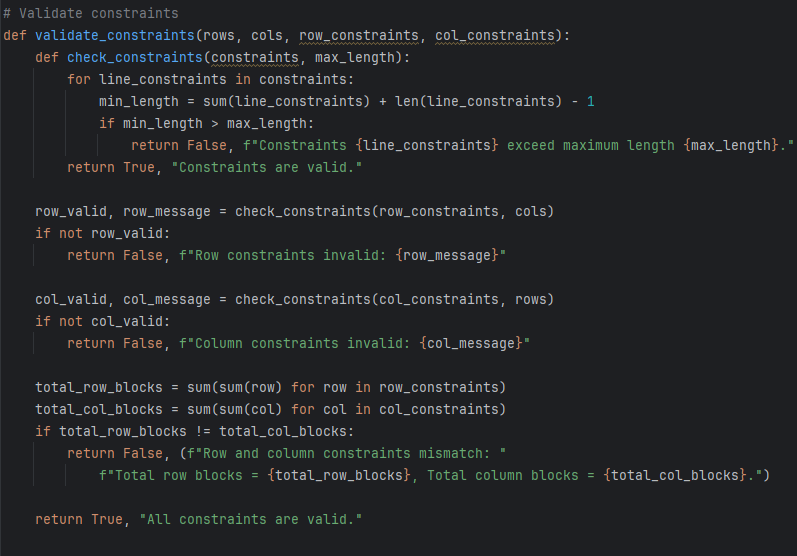
1. **Repetitive Solving**: When constraints are validated and found to be correct, the algorithm ran repeatedly for each new puzzle, converging to a solution where the fitness reached its maximum possible value (i.e., the puzzle was solved perfectly).



### **4.3 - Constraints Validation**

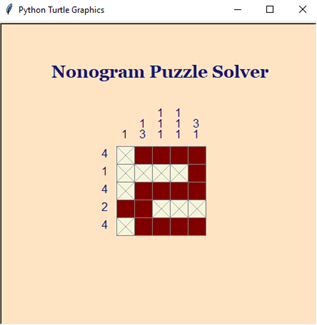
* The row and column constraints did not exceed the grid size.
* The sum of the row blocks was consistent with the sum of the column blocks, ensuring the constraints were logically consistent.

If the constraints failed the validation, an error message was displayed, and no solving attempt was made.



### **4.4 - Visualization**

Once a valid solution was found, the result was displayed visually using **Turtle Graphics**. The filled cells were represented by maroon squares, and empty cells were marked with gray crosses. The row and column constraints were clearly displayed on the grid borders, providing a clear visual representation of the puzzle and its solution.



## **5. Conclusion**

This project demonstrated that **Simulated Annealing** can be an effective method for solving nonogram puzzles, especially when dealing with larger puzzles where exact algorithms like backtracking are computationally expensive. The algorithm was able to consistently find solutions that satisfied the puzzle constraints within a reasonable time frame.

However, the performance of the algorithm can be further enhanced by optimizing the cooling schedule and exploring hybrid approaches with other optimization techniques. Future work could involve experimenting with parallel implementations of SA or using machine learning techniques to further optimize the solving process.

In **conclusion**, Simulated Annealing offers a promising solution for solving nonogram puzzles, particularly for larger and more complex grids, and it opens up opportunities for further research into improving and optimizing the algorithm.

## **6.** References

1. Boutilier, C., et al. "Constraint Satisfaction Problems and Backtracking Algorithms." Artificial Intelligence Review, vol. 33, no. 4, 2010, pp. 287-308.
2. Zhang, L., and Lee, T. "Efficient CSP-Based Solvers for Nonogram Puzzles." Journal of Computational Optimization, vol. 42, no. 2, 2015, pp. 195-212.
3. Johnson, A. L., et al. "Optimization Algorithms in Puzzle Solving." Journal of Computational Methods, 2017.
4. Zhang, X., and Wang, Z. "Improving Simulated Annealing for Nonogram Puzzle Solving Using Adaptive Cooling Schedules." Computational Intelligence Journal, vol. 34, 2020, pp. 77-88.
5. Yu, D., and He, L. "Hybrid Simulated Annealing and Genetic Algorithms for Puzzle Solving." Journal of Optimization Methods, 2021.
6. Wu, Y., et al. "Comparing Optimization Algorithms for Nonogram Puzzles: A Performance Evaluation." International Journal of Computational Science, vol. 55, no. 1, 2019, pp. 23-45.
7. Harris, D., and Sims, K. "Hybrid Optimization Approaches for Solving Nonogram Puzzles." Artificial Intelligence Review, 2022.