SOLVING NONOGRAMS WITH THE USE OF ARTIFICIAL INTELLIGENCE

Ekaterina Balashova, Uyen Nguyen Thai, Phuong Nguyen Lam

VinUniversity
Hanoi, Vietnam

ABSTRACT

This paper focuses on solving nonograms or picross puzzles using artificial intelligence (AI) techniques. Nonograms are grid-based puzzles where numbers indicate the lengths of consecutive blocks of filled cells. The goal is to deduce the pattern of filled and empty cells to complete the grid. We approach this problem from the perspective of AI researchers, aiming to develop intelligent systems capable of solving complex logical puzzles. We formulate nonograms as Constraint Satisfaction Problems (CSPs) and employ optimization techniques to improve efficiency and effectiveness. We experiment with various solvers, including brute force, backtracking search, and a minimum remaining values approach. We assess the performance of these solvers using metrics such as puzzle-solving ability, invalid puzzle detection, time complexity, and error susceptibility. The results indicate that the minimum remaining values solver with forward checking performs the most efficiently. The paper concludes by highlighting future possibilities, such as finding efficient solutions for partial assignment validation and exploring advanced filtering techniques to further enhance the solving process.

1. INTRODUCTION

[1] In our project, "Solving Nonograms with the use of Artificial Intelligence," we are tackling the problem of finding solutions to nonograms or picross puzzles using artificial intelligence techniques. Nonograms are grid-based puzzles where numbers are provided for each row and column, indicating the lengths of consecutive blocks of filled cells. The goal is to deduce the pattern of filled and empty cells to complete the grid.

Finding solutions for game-related problems – from Pacman to Go – is a common application for AI. Moreover, approaching this problem from the perspective of AI researchers provides a valuable learning experience for us as students. By working on a game we are familiar with as players, we can gain insights into the challenges of designing intelligent systems and develop a deeper understanding of AI principles. By addressing the problem of solving nonograms with AI, we aim to contribute to the development of intelligent systems capable of solving complex logical puzzles in the modern world.

2. METHOD

CSP involves defining variables, domains, and constraints, and then finding a solution that satisfies all the constraints. In the context of nonograms, variables represent the cells of the grid, domains define the possible values (filled or empty), and constraints are the rules defined by the given row and column numbers.

By formulating nonograms as CSPs, we can leverage various optimization techniques to improve the efficiency and effectiveness of solving these puzzles. For example, we can enhance constraint propagation algorithms to efficiently propagate constraints and reduce the search space. Additionally, we can explore different heuristics for variable and value selection to guide the search towards promising solutions.

From the lecture, we are attempting to develop solvers using Backtracking Search with various speed up strategies such Order, Filtering and Structure and Iterative min-conflicts algorithm. We are developing these solvers using Python and import puzzle information by using this given open source library: orsonhart/pycross: A Picross game developed using Pygame (github.com)

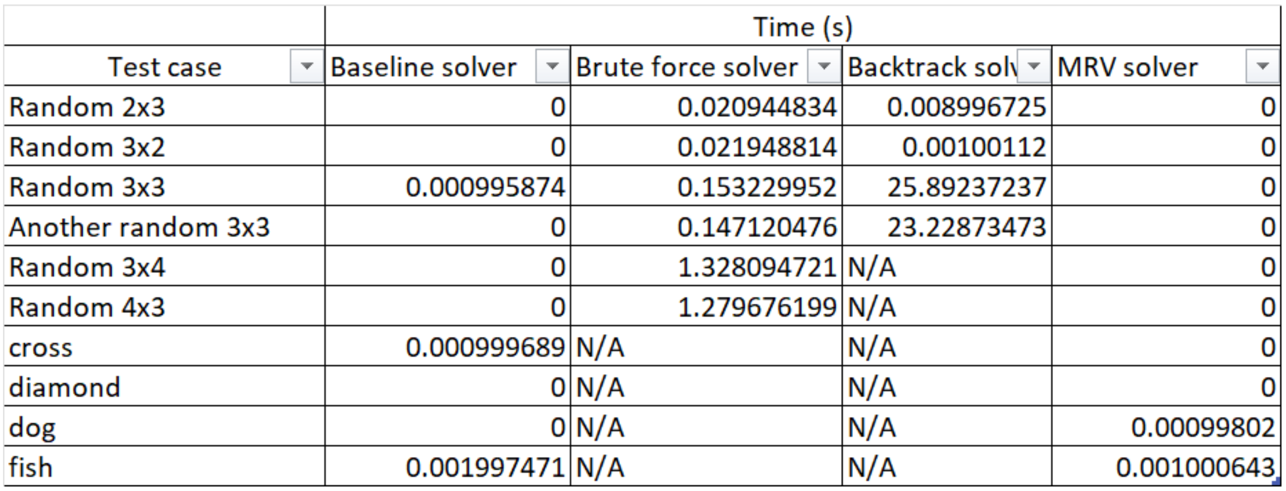
Using the CSP approach, we aim to develop an AI system that can solve nonograms accurately and efficiently. Through experimentation and evaluation, we can assess the performance of the CSP-based approach and potentially identify areas for further improvement.

3. EXPERIMENTS

In this project [1], we extended the open source library for the nonogram game user interface and puzzle generations. We aimed to integrate our own developed solvers with this library to perform the solving progress interactivly. Through testing and experiments, we would aim to develop effective algorithms to solve nonogram puzzles.

-**Baseline solver**: The library that we are working on has a built-in solver that we decided to use as a base for comparison with our developing solvers to see their advantages and disadvantages. From what we gathered while analyzing this algorithm, it tries to imitate the human solving strategy. It also

[1]Our public code is available at `https://github.com/TraMiu/Nonogram-COMP2050`



	Time (s)			
Test case	Baseline solver	Brute force solver	Backtrack solv	MRV solver
Random 2x3	0	0.020944834	0.008996725	0
Random 3x2	0	0.021948814	0.00100112	0
Random 3x3	0.000995874	0.153229952	25.89237237	0
Another random 3x3	0	0.147120476	23.22873473	0
Random 3x4	0	1.328094721	N/A	0
Random 4x3	0	1.279676199	N/A	0
cross	0.000999689	N/A	N/A	0
diamond	0	N/A	N/A	0
dog	0	N/A	N/A	0.00099802
fish	0.001997471	N/A	N/A	0.001000643

Fig. 1: Result of four solvers

provided us with the idea of reformulating the problem using whole rows as variables.

- **Experiment Setup**: Each experiment involved an auto generated nonogram puzzle with size ranging from 5x5 to 50x50. For each experiment, we would test our solvers and the built-in solver to collect outcome data and performance metrics

-**Performance metrics**: our performance metric include:

-Puzzle solving ability: keep track of number of puzzles each solver was able to solve and compare them

-Invalid puzzle detecting ability: keep track of the ability of the solvers to detect unsolvable puzzles

-Time complexity: keep track of the time each algorithm takes to solve each puzzle, measuring in units of millisecond (ms). Note: some changes in the measuring unit may occur in future to effectively measure our fully developed solvers.

- **Error susceptibility**: measuring the frequency of errors such as timeout error occurring given a puzzle and solver.

4. RESULTS

In this project, we have attempted and compared 4 different solutions: brute force, backtracking search, least remaining values with simple filtering, and the human strategy imitation built into the library. We have also worked with two different formulations of picross puzzles as CSP's: the simpler and most intuitive one that takes single squares as variables and given numbers as constraints, and the more sophisticated one that takes whole rows and columns as variables. The solutions that we considered (excluding the built-in one) are as follows:

- **Bruteforce**: With this solver, we formulate the puzzle as CSP's where individual cells are the variables with boolean domain. Then, we generate boolean strings with length equal to the number of cells in the puzzles and check whether that assignment is consistent with the clues. The complexity of this approach is $O(2^n)$ where n is the number of cells in the puzzles. After testing, we conclude that this approach is inefficient as it can only solve puzzles with 12 as the maximum number of cells without freezing. Nevertheless, this approach gave us an insight that it is important to choose suitable formulation for CSP's as it as an inefficient approach can result in exponential complexity.

- **Simple Backtracking**: We decided to attempt solving nonograms with backtracking search using single cells as variables because we considered it to be a good starting point that we could later improve on by using filtering techniques and exploiting problem structure. However, we encountered a significant difficulty when trying out this approach: while validating a fully assigned row or column in a nonogram is a very simple problem, no such thing can be said about partial or incomplete assignment validation: for example, if the row numbers are "3, 4, 2", the "3, 2" partial assignment has multiple possibilities of being valid or invalid. Despite our best efforts, we were unable to come up with an efficient solution for this problem, so the backtracking search ended up being unable to backtrack and therefore extremely slow and can only work on small puzzles.

- **Minimum Remaining Values**: This solution is the most efficient one that our group was able to come up with. We were inspired by the idea of treating rows and columns of the nonogram grid as variables that the built-in solver employed. At the start, this solver generates the domains for each row and column using a simple combinatorics method described by Harder (2022). After that, the minimum remaining values principle is used to order the variables: the algorithm iteratively chooses an unassigned variable with the smallest domain and recursively tries out all its possible values. For filtering, we decided to use forward checking: very time a row is assigned, it removes all conflicting values from the domains of all columns, and vice versa.

5. POSSIBILITIES FOR FUTURE DEVELOPMENT

The problem of solving nonograms still has some potential for future development. The main questions that are yet to be answered are the following:

- Is there an efficient solution for partial assignment validation? This problem is the main bottleneck not allowing for an efficient solution of nonograms using the naive formulation. We have an idea that the possible solution to it may lie in the realm of dynamic programming.

- Can more advanced filtering techniques be used to further speed up the final solution? Forward checking does not provide early detection for all failures. The efficiency might be improved by employing arc consistency.

6. CONCLUSION

In this project, we introduced the concepts of CSP in an attempt to solve the nonogram puzzle. Despite being an NP-Complete problem, most of the nonogram puzzles can be solved in polynomial time due to their small sizes. Our testing puzzles is an extensive collection of built-in and randomly generated puzzles which is diverse in terms of size and types (unique solution or multiple solutions). The solvers included in the project would be a valuable resource for students, nonogram lovers, and programmers to understand more about algorithms and concepts in CSPs. By examining this project, researchers may get inspirational ideas to develop solvers for similar problems such as sudoku or crossword puzzles.

7. CONTRIBUTION

Name	Contribution
Nguyen Thai Uyen	Researching, deciphering the existing code, working on the interface
Balashova Ekaterina	Researching and creating new solvers
Nguyen Lam Phuong	Researching, writing testbench and writing the report

Table 1: Contribution of members

8. REFERENCES

[1] H. de Harder, "Solving nonograms with 120 lines of code," https://towardsdatascience.com/solving-nonograms-with-120-lines-of-code-a7c6e0f627e4, October 2022, Medium.